석사학위논문

DCL 기반의 서블릿을 사용한 웹 서비스의 성능향상

Performance Upgrade of Web Services using DCL-based Servlet

한국방송통신대학교 평생대학원

정 보 과 학 과

김 대 중

2006년

DCL 기반의 서블릿을 사용한 웹 서비스의 성능향상

Performance Upgrade of Web Services using DCL-based Servlet

지도교수 곽 덕 훈

이 논문을 석사학위 논문으로 제출함

한국방송통신대학교 평생대학원

정 보 과 학 과

김 대 중

2005년 11월

김 대 중의
석사학위 논문을 인준함

심사위원장 이 언 배  인

심사위원 곽 덕 훈  인

심사위원 이 병 래  인

한국방송통신대학교 평생대학원

2005년 12월

논 문 요 약

DCL 기반의 서블릿을 사용한 웹 서비스의 성능향상

김 대 중

한국방송통신대학교 평생대학원

정 보 과 학 과

(지도교수 곽 덕 훈)

웹 서버 어플리케이션에 있어서 동적인 콘텐츠를 생성하는 기술은 매우 다양하다. 이러한 기술들은 ASP나 PHP의 경우와 같은 인터프리터 기반의 기술과 JSP나 ASP.NET과 같은 컴파일러 기반의 기술이 있으며, 이들 각각은 그들의 특징에 포함된 단점으로 인하여 시스템 자원의 효율을 높이는데 있어서 한계점을 가진다.

시스템 자원을 최대한 효과적으로 사용하는 방법은 네이티브 바이너리 코드로 개발된 어플리케이션을 사용하는 것이다. 본 논문은 동적인 콘텐츠를 생성하는 어플리케이션을 위하여 운영체제의 동적공유객체(DSO: Dynamic Shared Object)를 사용하는 기술인 DCL HTTP 서버 확장(DHE: DCL HTTP Server Extension)에 대하여 다룬다.

DHE는 DHE 서블릿과 DHE 서블릿 컨테이너로 이루어져 있다. 서블릿은 DHE 환경에서 HTTP 요청에 대한 구체적인 처리를 하도록 개발된 DSO이고

서블릿 컨테이너에 의하여 실행된다. 서블릿 컨테이너는 이미 널리 사용되고 있는 웹 서버 소프트웨어인 Apache나 IIS의 플러그인(plug-in)으로 개발되며 웹 서버간의 API의 차이점을 흡수하여 서블릿이 다양한 웹 서버 소프트웨어에 이식이 가능하도록 하는 독립된 환경을 제공한다.

DHE 서블릿은 표준 C언어 함수호출 규약을 제공하고 DSO를 개발할 수 있는 모든 프로그래밍 언어를 사용해서 개발할 수 있다. DCL(Daejung's Class Library) 프로젝트를 통해 개발된 C++ 클래스 라이브러리를 사용할 경우 운영체제 이식성과 더불어 객체 지향적 방법을 통한 생산성 향상을 얻을 수 있다.

DHE의 유효성을 검증하기 위해 동일한 알고리즘이 적용된 PHP, ASP, ASP.NET, JSP 서버 어플리케이션과의 성능비교 실험을 실시하였다. 실험의 결과는 DCL 기반의 서블릿을 사용한 DHE 환경이 가장 적은 가상메모리를 사용하고 있었고 200라인(27.8Kbytes)이상의 문자열을 생성하는 실험에서 단위 시간당 처리된 HTTP 요청의 개수가 JSP에 비하여 3배 이상, ASP, PHP에 비하여 6배 이상의 결과를 얻었다.

- 목 차 -

- 표 목 차 -

- 그 림 목 차 -

제1장 서 론

동적인 콘텐츠를 생성하는 주요 웹 서버 어플리케이션 기술은 ASP, PHP와 같이 스크립트를 위한 인터프리터를 웹 서버의 프로세스에 플러그인(plug-in) 형태로 상주시키는 방법과 JSP, 자바 서블릿(Java Servlet)과 같은 자바기술을 적용한 형태로 발전하고 있다. 기존의 이러한 방식들의 공통적인 단점은 각각의 실행환경에 의한 특징 때문에 시스템 자원의 효율을 향상시키는데 있어서 한계점을 가진다는 것이다. ASP, PHP와 같은 인터프리터 기반의 방식은 HTTP 요청마다 스크립트의 번역이 반복되고, 자바기술을 적용한 경우에는 실행환경이 JVM에 제한되며 대체적으로 JVM이 시스템의 메모리 자원을 많이 필요로 하는 단점이 있다.

웹 서버 어플리케이션의 느린 실행은 그것이 데이터베이스에 접근할 때 데이터베이스 서버까지 부하를 가중시킨다는 점이다. 어플리케이션이 데이터베이스 서버에 연결된 이후에는 연결이 종료되기 전까지는 어플리케이션의 실행이 빠르든 느리든 데이터베이스 서버는 단위 트랜잭션을 유지하게 된다. 따라서 어플리케이션의 실행이 느리게 되면 데이터베이스 서버는 어플리케이션의 트랜잭션을 위해 할당한 자원을 더욱 오랫동안 유지해야 하기 때문에 결국엔 데이터베이스 서버의 가용성까지 떨어뜨린다.

이와 같은 방식으로 개발된 웹 사이트에서 가용성 증대를 위한 해결은 대체적으로 하드웨어 자원에 투자함으로써 해결하고자 하는 경향이 있다. 단일 CPU는 다중 CPU로, 단일 서버는 다중 서버로 서비스 하게 함으로써 해결하고자 한다. 그러나 이러한 접근의 결정적인 단점은 초기 투자비용뿐만 아니라 막대한 유지보수 비용의 증가를 초래할 수 있다는 점이다.

소프트웨어적인 방법을 통하여 웹 서버 어플리케이션의 실행 속도를 높이기 위한 최선의 방법은 웹 서버의 API를 사용하여 서버의 플러그인 형태로 어플리케이션을 개발하는 것이다[1]. 대부분의 웹 서버들은 서버의 기능을 확장하기 위한 방법으로 C언어 기반의 API(Application Programming Interface)를 제공 한다. 그러나 웹 서

버의 API는 웹 서버 소프트웨어에 종속되기 때문에 서로 다른 서버 소프트웨어간의 이식이 매우 어렵다는 단점이 있다.

본 논문에서는 DCL(Daejung's Class Library) 프로젝트를 통해 개발된 DCL HTTP 서버 확장(DHE: DCL HTTP Server Extension)이란 웹 서버 어플리케이션 기술을 제안한다. DHE는 특정 웹 서버 소프트웨어에 종속되지 않는 서버환경을 위해 새로운 인터페이스를 정의하고 이것과 관련된 기본 구성요소를 DHE 서블릿과 DHE 서블릿 컨테이너로 구별하였다. 서블릿은 DHE에서 실제 어플리케이션으로 HTTP 요청에 대한 구체적인 처리를 담당하고 서블릿 컨테이너는 웹 서버의 플러그인 모듈로 서블릿의 실행환경을 제공한다.

DHE 서블릿은 C/C++와 같은 컴파일러 언어로 작성되어 네이티브 바이너리 코드(native binary code)로 만들어진 동적공유객체(DSO: Dynamic Shared Object)로 PHP, ASP, JSP와 같은 기존의 방법들과 비교하여 최소의 시스템자원 환경에서 최대의 실행속도를 기대할 수 있다. 그러나 이것은 웹 서버 프로세스에 적재된 후 실행 기회가 주어지면 어떠한 방법으로든 통제되지 않기 때문에 DHE에 관한 연구는 서블릿의 실행환경뿐만 아니라 서블릿의 디버깅을 위한 개발환경에 많은 부분이 할애 되었다.

제2장 관련연구 및 배경지식

2.1 Apache HTTP 서버

Apache 프로젝트는 상용 웹 서버 수준의 강력하고 풍부한 기능을 가진 소스코드 수준의 공개형 HTTP 서버를 만들어 내기 위한 목적으로 시작되었다. 이 프로젝트는 전 세계에 퍼져있는 개발자들이 인터넷과 웹 환경을 통한 자발적인 참여에 의하여 계획되어 개발 및 관련 문서에 대한 작업이 이루어지고 있다. Apache 웹 서버의 특징으로는 소스코드 전체의 공개, 다양한 운영체제 플랫폼에 대한 이식성, 서드파티(third-party) 확장모듈, 모듈 작성을 위한 API의 제공, 모듈의 추가와 제거를 통한 다양한 서버의 구성이 가능하다는 점 등이 있다[2].

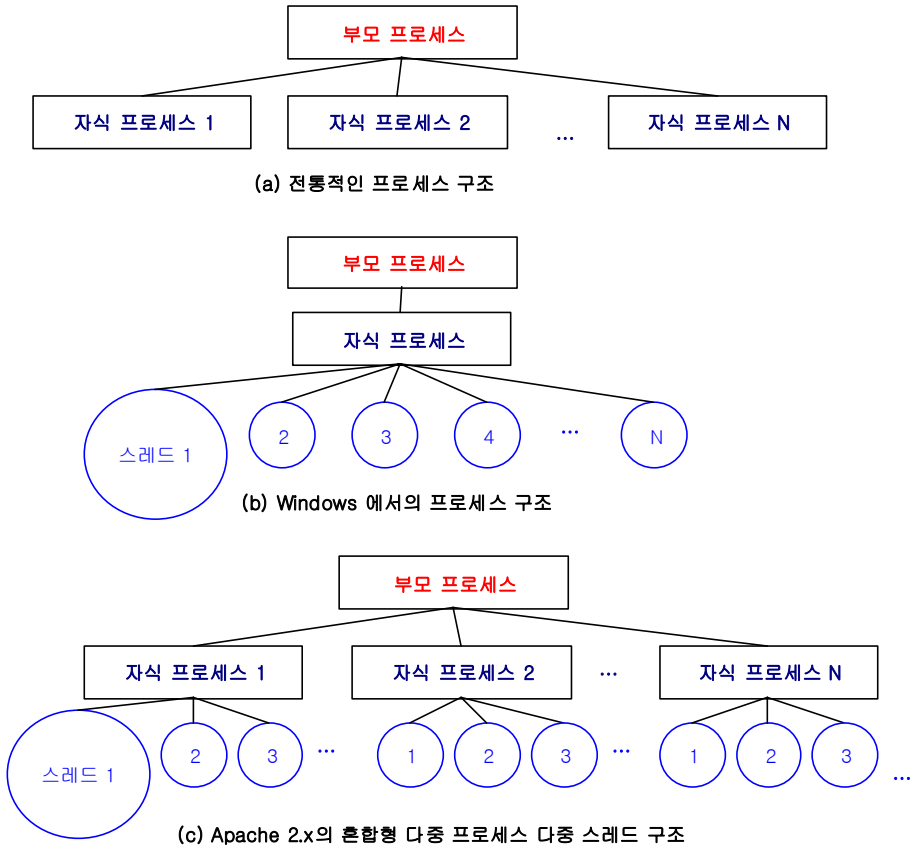
2.1.1 프로세스 구조와 프로세싱 모델

전통적인 Apache 서버의 구조는 단일의 부모 프로세스와 재사용되는 자식 프로세스들로 구성된다. 부모 프로세스는 서버가 시작될 때 구성(configuration)을 읽어들이고 자식 프로세스의 풀(pool)을 관리한다. 부모 프로세스는 자식 프로세스를 위한 풀의 크기를 자동으로 조정하는데 이 때문에 유휴(idle) 프로세스로 인한 시스템 자원의 사용을 최소화 한다. 자식 프로세스는 단일의 소켓 연결을 사용하여 HTTP 요청을 처리하거나 잠든(sleep) 상태에서 대기한다.

Apache의 전통적인 프로세스 모델은 fork 시스템 호출[1]을 제공하는 UNIX와 같은 프로세스 기반의 운영체제에 적당하다. 그러나 Microsoft의 Windows와 같은 스레드 기반의 운영체제에서는 프로세스를 생성하는데 필요한 오버헤드가 너무 크기 때문에 스레드 기반으로 구성된다. <그림 2-1>은 Apache 서버의 프로세스 구조를 표시하고 있다. 부모 프로세스는 서버가 시작될 때 구성을 읽어 들이고 구성 파일의 유효성을 검사한 후 실제 작업 프로세스(worker process)인 자식 프로세스를 생성

[1] UNIX의 fork 시스템 호출은 현재의 프로세스 환경을 복사하여 새로운 프로세스를 생성한다.

한다. 서버가 동작 중일 때 부모 프로세스의 가장 중요한 역할은 예외적인 상황에서 자식 프로세스가 종료되었을 때 새로운 자식 프로세스를 생성하여 서버의 서비스를 지속시키는 것이다.



<그림 2-1> Apache의 프로세스 구조

다중 스레드 방식이 적용된 경우 작업 프로세스인 자식 프로세스는 실제 서비스를 담당하는 스레드에 대하여 스레드 풀(pool)을 유지한다. <그림 2-1>의 (b)는 Windows에서의 프로세스 구조를 보여주고 있는데 엄격히 따지면 Windows 운영체제에서의 프로세스는 부모-자식의 관계는 아니다.

Apache 2.0부터는 서버가 HTTP 요청을 받아들이고 이 요청을 처리하기 위해 분

배하도록 하는 다중처리모듈(MPM: Multiple Processing Module)을 사용한다. MPM은 서버의 다른 부분으로부터 운영체제 플랫폼 의존적인 API로 구현된 부분들을 은닉한다. MPM의 사용으로 인해 서버는 특정 사이트의 요구조건에 특화 되어 구성될 수 있다. 가령 확장성(scalability)이 필요한 사이트는 worker와 같은 스레드 MPM을 사용하고 기존의 1.3.x 버전과의 호환성이 필요한 사이트는 prefork를 사용할 수 있으며 perchild는 서로 다른 사용자 계정으로 여러 호스트의 서비스가 가능하도록 한다.

POSIX 스레드 라이브러리(pthread)를 사용할 수 있는 UNIX 운영체제에서는 혼합형 다중 프로세스 다중 스레드(hybrid multi-process multi-threaded) 구조가 사용될 수 있으며 worker나 perchild와 같은 MPM이 여기에 해당된다.

2.1.2 Apache 확장모듈

Apache 웹 서버의 기능은 Apache 확장모듈을 통해 확장할 수 있다. Apache의 모듈은 서버를 컴파일할 때 httpd 실행파일에 정적으로 링크하거나 httpd 실행파일과 분리하여 동적공유객체(DSO: Dynamic Shared Object)로 컴파일할 수 있다. DSO 모듈은 서버를 컴파일할 때 컴파일하거나 서버와 별도로 컴파일할 수 있지만 운영체제 플랫폼이 DSO를 지원하여야 하고 정적으로 컴파일된 mod_so 모듈을 포함하고 있어야 한다. Apache의 확장모듈이 DSO로 구현되어 있다 하더라도 이것의 적재는 서버의 시작 시점에서 이루어진다. 이것은 Apache 서버에서의 모듈이 Apache 서버의 어플리케이션이기보다 서버를 이루는 하나의 구성요소이기 때문이다. Apache 서버의 모듈로 사용할 DSO 파일을 쉽게 만들기 위해 apxs(Apache eXtension Tool)라는 지원 프로그램이 있다. 이 프로그램은 Apache의 소스 트리(source tree) 밖에서 DSO로 사용할 모듈을 컴파일하고 설치한다.

Apache 서버의 모듈은 module이라 불리는 단 하나의 구조체가 구현되어 있어야 한다. 이 구조체는 Apache의 다른 구성 요소들이 모듈에 접근할 때 사용되며 모듈에 대한 정보와 모듈이 제공하는 함수들의 포인터로 구성되어 있다. Apache 1.3.x와 2.x는 module 구조체가 서로 다르기 때문에 컴파일된 모듈은 서로 다른 버전에서

같이 사용할 수 없다.

PHP, Perl과 같은 HTTP 서버 어플리케이션 환경을 위한 스크립트 엔진은 Apache 서버와 동일한 프로세스 환경에서 동작하도록 하는 플러그인(plug-in) 모듈인 Apache 확장모듈로 구현되며, DHE 서블릿 컨테이너도 마찬가지 이다.

2.2 Microsoft IIS 5.0

IIS(Internet Information Services)는 Microsoft의 Windows 2000, 2003, XP와 같은 Windows NT 기반의 운영체제의 인터넷 서비스를 위한 구성요소로 HTTP 서비스뿐만 아니라 FTP, NNTP, SMTP에 대한 서비스도 함께 제공한다. IIS는 Windows NT 기반의 운영체제에 밀접하게 구현되어 있으며 가장 최근에 발표된 IIS 6.0은 Windows 2003 서버의 커널과 더욱더 밀접하게 연관되어 있다[4].

본 절에서는 Windows 2000과 함께 발표된 IIS 5.0을 중심으로 IIS의 HTTP 서비스를 위한 DHE 서블릿 컨테이너 개발과 관련된 내용을 중심으로 살펴본다.

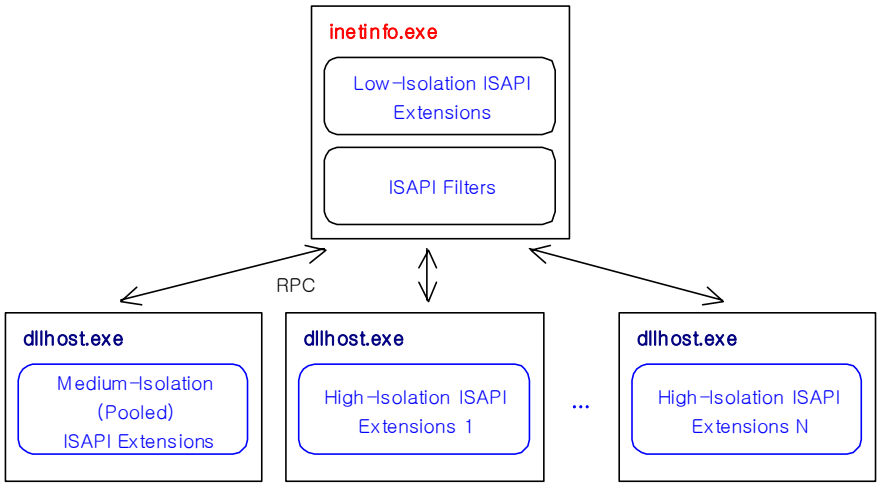
2.2.1 응용프로그램과 프로세스 구조

IIS에서 응용프로그램이란, 웹 사이트의 정의된 디렉토리(directory) 안에서 실행되는 모든 파일을 의미한다. 응용프로그램을 만들 때는 IIS의 스냅인(snap-in)을 사용하여 웹 사이트에서 응용프로그램의 시작위치 디렉토리를 지정한다. 웹 사이트의 시작위치 디렉토리 아래에 있는 모든 파일과 디렉토리는 다른 시작위치 디렉토리를 만날 때까지 그 응용프로그램의 일부로 인식된다. 이런 방식으로 디렉토리 경계를 사용하여 응용프로그램을 정의한다.

IIS 5.0의 응용프로그램의 보호는 정의된 응용프로그램을 서로 다른 프로세스에 격리시키는 것으로 세 가지 수준이 있다. 첫 번째, 높은 수준(High-Isolation)의 경우는 응용프로그램 각각을 inetinfo.exe와는 별도의 프로세스인 dllhost.exe에 격리되어 실행 되도록 한다. 두 번째, 보통 수준(Medium-Isolation)은 응용프로그램을 풀링된(pooled) 프로세스에서 실행되도록 설정하는 것으로, 이러한 경우 모든 응용

프로그램은 하나의 dllhost.exe 프로세스에서 실행된다. 세 번째는 가장 낮은 수준(Low-Isolation)으로 응용프로그램을 웹 서버의 프로세스환경 내에서 실행하도록 하는 것으로, 이 경우는 웹 서버 프로세스인 inetinfo.exe를 실질적으로 보호하지 못한다. 서버 구성에 있어서 응용프로그램의 보호 수준은 이 세 가지를 단독으로 사용하거나 혼합하여 사용할 수 있다[5].

IIS 5.0의 HTTP 서비스는 응용프로그램의 보호를 위한 프로세스 격리를 위하여 <그림 2-2>와 같이 다중 프로세스 구조일 수 있다.



<그림 2-2> IIS 5.0의 응용프로그램의 보호와 프로세스 구조

2.2.2 ISAPI 확장모듈

ISAPI 확장모듈(ISAPI Extensions)은 IIS에서 실행되는 실제 어플리케이션으로 IIS가 제공하는 모든 기능들에 접근할 수 있다. ASP의 페이지를 처리하는 asp.dll은 대표적인 ISAPI 확장모듈이다. ISAPI 확장모듈은 Windows의 DLL로 구현되어 IIS에 의해 제어되는 프로세스에 적재된다.

Apache 서버의 확장모듈과 달리 ISAPI 확장모듈은 별도의 라이브러리를 필요로 하지 않는데 모듈을 적재한 프로세스가 EXTENSION_CONTROL_BLOCK 이라 불

리는 구조체를 통하여 모듈이 필요한 함수를 제공하기 때문이다.

ISAPI 확장모듈은 HTTP 요청이 있는 경우에 적재되고 요청에 대한 처리가 완료되면 제거된다. 만약 IIS의 응용프로그램을 캐시되도록 설정한 경우에는 서버가 종료될 때까지 적재된 상태를 유지한다.

2.3 웹 서버 어플리케이션 기술

동적인 콘텐츠를 위한 초기의 서버 어플리케이션은 별도의 프로세스로 실행되는 CGI의 구현이 있었으나, 최근에는 서버 프로세스를 위한 플러그인 모듈로 구현되어 서버사이드 스크립트의 실행환경을 제공하는 형태로 발전하고 있다.

서버사이드 스크립트(SSS: Server-Side Script)는 웹 서버에서 번역되어 실행되는 스크립트를 의미한다. SSS는 텍스트 편집기를 사용하여 간단히 작성할 수 있고 스크립트 파일이 URI(Uniform Resource Identifier)의 경로에만 위치하면 별도의 작업 없이도 쉽게 그 결과를 확인할 수 있다는 장점 때문에 웹 사이트 개발에 매우 널리 사용되고 있다. SSS에서 사용되는 언어는 스크립트의 번역 방식에 따라 인터프리터 방식과 컴파일러 방식이 있을 수 있지만 모두 HTML과 프로그램 코드가 혼용되어 작성될 수 있다는 공통점이 있다.

2.3.1 PHP

PHP[7]는 공개용 소스코드(open source code)기반의 SSS 실행 환경이자 인터프리터 언어로 UNIX를 비롯한 다양한 운영체제에 이식될 수 있다. PHP의 실행은 CGI 형태로도 사용할 수 있지만 Apache를 비롯한 많은 웹 서버에서 플러그인 형태로 설치되어 사용된다.

PHP의 시작은 1995년 Rasmus Lerdorf 라는 개발자가 자신의 온라인 이력서 페이지의 방문자들을 관리하기 위한 수단으로 개발했다. PHP라는 이름은 초기의 버전을 "Personal Home Page Tools"라고 불렀던 사실에서 유래 되었고, 현재는 PHP를 재귀적으로 사용한 "PHP Hypertext Preprocessor"를 의미한다.

가장 최근에 발표된 PHP5에서 사용하는 Zend 엔진 2.0은 객체핸들의 개념을 사용한 새로운 객체모델 사용하고 있으며, 객체지향의 특징을 더욱 발전시켰다. PHP5의 객체지향 언어의 특징은 사적(private)인 멤버변수와 메소드, 정적(static) 변수와 메소드, 클래스 상수, 추상(abstract) 클래스와 추상 메소드, 인터페이스, final 메소드와 final 클래스, 객체가 파괴될 때 실행되는 __destructor 메소드 등이 있다. Zend 2의 가장 주목할만한 점은 오버로딩(overloading)을 통하여 자바 클래스, COM/DCOM 그리고 .NET과 같은 외부 객체와 상호 작용할 수 있으며, 이 외에도 try, throw, catch를 사용한 구조적 예외처리의 기능도 추가되었다[8].

PHP에서 데이터베이스 관련 함수는 대부분의 PHP 함수와 마찬가지로 해당되는 DBMS(Database Management System)의 클라이언트 API를 PHP의 함수로 포장(wrapping)하는 형태이다. 따라서 이 함수들을 사용하여 개발된 어플리케이션은 특정 DBMS에 의존되어 이식이 불가능하다. 이에 대한 노력으로 PHP로 작성된 ADOdb[2]와 같은 클래스 라이브러리가 있으나, 라이브러리가 PHP로 작성되어 있기 때문에 어플리케이션의 스크립트가 번역될 때마다 라이브러리도 같이 번역되는 단점이 있다.

2.3.2 ASP와 ASP.NET

ASP(Active Server Page)는 IIS에서 인터프리터를 사용하는 어플리케이션 환경이다. ASP에서 사용되는 프로그래밍 언어는 VBScript, JScript, Perl외에 다양한 언어가 사용될 수 있으며, 프로그램 스크립트는 “<%” 과 “%>” 사이에 기술한다. 스크립트에서 사용될 언어는 스크립트의 처음에 “<%@ language=VBScript %>”, “<%@ language=JScript %>”, “<%@ language=PerlScript %>” 와 같이 명시한다.

ASP는 Windows 운영체제와 밀접하게 연관되어 있으며 스크립트 내에서 ActiveX와 같은 COM(Component Object Model) 객체의 사용이 매우 자유롭다. 그러나 Windows에서의 이러한 장점은 비록 Perl을 사용하는 ASP 스크립트라 하더라

[2] http://adodb.sourceforge.net/

도 다른 운영체제에 이식하는 것을 어렵게 하는 단점으로 작용한다.

ASP.NET은 Microsoft Windows의 새로운 프로그래밍 모델인 .NET Framework의 한 부분으로 동적인 웹 어플리케이션을 위한 기술이다. ASP.NET 페이지의 스크립트는 Visual Basic, C#, C++, JScript, J# 등 다양한 언어로 작성될 수 있으며 DLL로 컴파일되어 실행된다.

IIS에서 ASP.NET 페이지를 위한 실행 환경은 5.0 버전과 6.0 버전에서 서로 다르다. 6.0 버전은 HTTP 서비스의 어플리케이션 풀(pool)인 W3wp.exe에 포함되어 있고 5.0 버전에서는 inetinfo.exe와는 격리된 aspnet_wp.exe 프로세스에서 실행된다[9].

2.3.3 JSP와 자바 서블릿

JSP(JavaServer Page)는 자바진영(Java World)을 대표하는 선 마이크로시스템(Sun Microsystems)이 제시한 동적인 웹 페이지 개발을 위한 스크립트 기술로 자바 서블릿 기술의 확장이다. JSP는 자바와 자바 서블릿 기술을 기초로 개발된 기술이기 때문에 자바 언어의 특성이 그대로 적용이 되고 자바 서블릿에서 제공하는 API를 모두 사용할 수 있다.

자바 서블릿(Java Servlet)에서 “Servlet”은 “Server”와 조각을 의미하는 “-let” 접미사의 합성어로 자바 언어를 기초로 하는 웹 서버의 기능을 확장하도록 하는 기술이다. 자바 서블릿 컨테이너는 자바 서블릿의 실행을 위해 필요한 요소로 IIS, Apache와 같은 서버의 확장모듈로 구현되거나 BEA WebLogic Application Server, Tomcat, IBM WebSphere, Sun Java System Application Server와 같은 전용 컨테이너가 있다.

JSP의 실행에는 컴파일러와 인터프리터 모두가 관여한다. 최초 요청된 .jsp 파일은 자바 페이지 컴파일러에 의하여 .java 확장자를 갖는 서블릿 코드로 컴파일되고 이것은 다시 .class 파일로 컴파일된다. JSP 컨테이너는 .jsp와 대응되는 .class를 호출하여 HTTP 요청에 대하여 응답한다.

2.3.4 CGI와 FastCGI

CGI(Common Gateway Interface)[11]는 HTTP 서버에서 외부 어플리케이션과의 인터페이스를 위한 하나의 표준이다. CGI 어플리케이션은 HTTP 서버와는 분리된 별도의 프로세스로 실행되며 HTTP 서버 프로세스와 CGI 프로세스간의 IPC(Inter-Process Communication)는 주로 파이프가 사용된다. CGI는 인터페이스만 정의하고 있기 때문에 CGI 프로그램은 C/C++, Perl, FORTRAN, TCL, Python, Visual Basic, UNIX Shell Script 등과 같이 운영체제에서 실행될 수 있는 모든 언어를 사용하여 작성될 수 있다.

표준 CGI의 단점은 HTTP 요청마다 CGI 프로세스의 생성에 따르는 오버헤드가 발생한다는 점이다. 어플리케이션마다 별도의 프로세스로 격리하는 것은 보안적인 측면과 어플리케이션의 안정성에 장점으로 작용하지만, 과다한 CGI 프로세스의 생성은 서버의 가용성을 떨어뜨리게 한다.

FastCGI[12]는 CGI의 프로세스 생성에 따르는 오버헤드를 줄이기 위해 새롭게 제안된 방법이다. C/C++와 같이 네이티브 바이너리 코드를 생성하는 어플리케이션에 있어서 기존의 CGI는 실행파일(Windows는 *.exe, UNIX는 a.out)을 사용하지만 FastCGI에서는 DSO를 사용하여 어플리케이션의 서비스가 완료된 후에도 한번 생성된 CGI 프로세스를 종료하지 않는다. FastCGI에서 어플리케이션은 CGI와 마찬가지로 격리된 별도의 프로세스로 실행되기 때문에 어플리케이션의 개수가 많아질수록 시스템의 메모리 사용은 증가한다.

이 외에도 CGI와 FastCGI 어플리케이션은 프로세스간의 메시지 전달을 위하여 별도의 IPC 오버헤드가 발생하는 단점이 있다.

2.4 동적공유객체

현대의 많은 운영체제는 특별한 형식의 실행코드 조각을 담고 있는 파일인 동적공유객체(DSO: Dynamic Shared Object)를 응용프로그램의 실행시간에 적재하여 사용할 수 있도록 하는 기능을 제공하는데, 이것은 운영체제마다 약간의 차이점이 있긴

하지만 기본적인 역할은 거의 동일하다. 본 절에서는 UNIX의 공유객체와 Windows의 DLL에 대하여 살펴본다.

2.4.1 UNIX의 공유객체

UNIX에서 DSO는 두 가지 형태로 구분될 수 있다.

첫 번째는 공유 라이브러리(shared library) 또는 DSO 라이브러리라고 부르며 libfoo.so나 libfoo.so.1.2와 같이 lib로 시작한다. 이들 파일은 보통 /usr/lib나 /lib 디렉토리에 있고 실행모듈(어플리케이션이나 DSO)을 컴파일할 때 –lfoo와 같이 링커 옵션을 주어 실행모듈과 연결한다. 실행모듈이 실행될 때에는 UNIX의 런타임 링커(ld.so)[3]가 모듈의 의존관계를 분석하여 참조된 공유객체를 반복적으로 연결하여 필요한 심볼(symbol)을 찾고 재배치하게 된다. 모듈의 의존관계 분석에 사용되는 디렉토리의 검색순서는 모듈을 링크할 때 경로를 직접 지정하지 않는 한 일반적으로 <표 2-1>과 같다.

<표 2-1> UNIX의 공유 라이브러리 검색순서

①	LD_LIBRARY_PATH 환경변수에 포함되어 있는 디렉토리
②	(운영체제에서 지원할 경우)/etc/ld.so.cache에 포함되어 있는 디렉토리
③	기본 라이브러리 디렉토리(/lib, /usr/lib)

두 번째는 공유객체(shared object) 또는 DSO 파일이라고 부르며 파일의 확장자는 자유롭다. 이들 DSO 파일은 dlopen 시스템 호출에 의해 프로세스의 주소공간으로 읽어 들인다. 이 함수를 호출할 때 사용하는 DSO의 경로명이 '/' 문자로 시작하는 절대경로가 아니면 <표 2-1>의 검색순서에 의해 DSO를 검색한다. dlopen이 성공하면 dlsym 시스템 호출을 사용하여 함수나 데이터의 주소를 직접 찾아서 DSO를

[3] ld.so는 /usr/lib 또는 /lib 디렉터리에 위치하며 모든 UNIX 실행모듈에 연결되어 모듈이 실행될 때 필요한 시작코드의 일부이다.

사용하게 되고 사용을 마치면 dlclose로 닫는다.

2.4.2 Windows의 DLL

Windows에서 DSO는 DLL(Dynamic-Link Library)로 불려진다. DLL은 다른 어플리케이션이나 DLL에 의해 사용될 함수나 데이터를 포함하고 있는 모듈이다. DLL은 익스포트(export)된것과 내부적(internal)으로 사용되는 두 종류의 함수를 정의 할 수 있으며 내부함수(internal function)는 오직 DLL 내부에서만 사용된다. DLL은 그것의 기능성이 더 쉽게 수정되고 재사용되기 때문에 모듈화된 어플리케이션을 위한 방법을 제공한다.

동적링킹은 모듈이 포함하고 있는 유일한 정보가 로드타임(load time)이나 런타임(run time)에 필요한 DLL의 익스포트된 함수의 위치에 대한 정보만이 되는 것을 허용하며 이것은 필요한 라이브러리 함수의 복사하는 정적링킹(static linking)과는 다르다.

동적링킹은 로드타임 동적링킹(load-time dynamic linking)과 런타임 동적링킹(run-time dynamic linking)의 두 가지가 있다. 시스템이 하나의 프로그램을 시작할 때 사용하는 것이 로드타임 동적링킹이다. 로드타임 동적링킹을 사용하는 모듈(실행파일이나 DLL)은 사용하고자 하는 DLL의 임포트(import) 라이브러리에 대한 정적인 링크가 필요하다. 링커는 파일 안에 포함되어 프로세스에 의해 사용되는 DLL의 이름 정보를 사용하여 <표 2-2>의 검색순서에 의해 DLL를 검색한다.

<표 2-2> Microsoft Windows의 DLL 검색순서

①	어플리케이션이 적재된 디렉토리
②	GetSystemDirectory에 의해 얻어진 디렉토리
③	GetWindowsDirectory의 얻어진 디렉토리
④	어플리케이션의 현재 디렉토리
⑤	환경변수 PATH에 포함되어 있는 디렉토리

만약 시스템이 필요한 DLL을 찾지 못하면 에러 메시지에 관한 다이얼로그 박스를 표시하고 프로세스는 종료된다. 다른 경우, 시스템은 DLL을 프로세스의 가상 주소 공간에 대응시키고 DLL의 참조 카운터를 증가시킨다.

시스템은 DLL에 DllMain 함수가 기술되어 있는 경우 이 함수를 호출한다.

DLL은 초기화되는 동안 프로세스의 가상주소 공간에 사상되고(mapped) 필요할 때만 물리적인 메모리로 적재된다.

런타임 동적링킹을 위한 DLL의 적재는 LoadLibrary나 LoadLibraryEx 함수를 사용한다. 이들 함수를 사용할 때 DLL의 경로를 표시하는 문자열이 순수한 파일명으로만 구성되면 <표 2-2>의 DLL 검색순서에 의해 DLL을 검색한다. DLL이 적재된 후에는 익스포트된 함수의 주소를 얻기 위해 GetProcAddress를 호출하고, 이 함수의 리턴 값을 사용하여 DLL의 함수를 사용할 수 있으며 이것은 로드타임 동적링킹에 필요한 임포트 라이브러리의 필요성을 제거한다.

FreeLibrary나 FreeLibraryAndExitThread 함수는 프로세스에서 더 이상 DLL을 사용하지 않는 경우에 호출할 수 있다. 이 함수는 프로세스에 사상된 참조 카운터의 값을 감소시키고 참조 카운터의 값이 0(zero)이 되었을 경우 프로세스의 사상을 제거(unmapped)한다.

2.5 DCL 프로젝트

DCL(Daejung's Class Library) 프로젝트는 GNU/Linux를 비롯한 UNIX 운영체제와 Microsoft의 Windows 운영체제에서 시스템 프로그래밍에 사용될 수 있으면서 이식성이 있는 C++ 라이브러리 시스템 구축을 위한 개인 프로젝트이다[15].

■ DCLCore

DCLCore는 DCL 프로젝트의 중심을 이루고 있는 라이브러리 패키지로 String을 비롯한 Vector, List, HashMap과 같은 콜렉션(collection) 클래스들과 날짜 및 시간을 위한 클래스, Regex 클래스, 스트림 클래스, 데이터베이스 연결을 위한 클래스,

File, Thread, Mutex와 같은 시스템 객체에 관한 클래스를 포함하고 있다.

DCLCore의 가장 중요한 부분은 라이브러리 차원에서 어플리케이션의 실행시간에 디버깅을 위한 환경을 제공한다는 것이다. 이 환경은 프로그램 작성에 있어서 ASSERT, TRACE[4]를 사용할 수 있도록 할 뿐만 아니라 동적 메모리 할당과 해제에 관하여 라이브러리 차원의 추적이 가능하도록 하고 있다. 만약 어플리케이션이 다중 스레드 어플리케이션일 경우 스레드별 추적도 가능하다[16].

■ DCLNet

DCLNet은 socket을 비롯한 IPC관련 API들에 대한 클래스들과 DHE에 대한 명세, DHE 서블릿을 개발하는데 필요한 서블릿 클래스, HTTP 관련 클래스, HTML 관련 클래스들을 포함하고 있는 패키지이다.

■ DHE

DHE(DCL HTTP Server Extension)는 Apache, IIS 웹 서버에서 DSO로 개발된 서블릿을 실행 하도록 해주는 웹 서버 어플리케이션 기술로 자세한 내용은 3장에서 다룬다. DHE의 명세는 어플리케이션인 서블릿을 구현하는데 필요한 프로그래밍 언어를 명시하고 있지는 않지만 C++를 사용하게 될 경우 DCLCore, DCLNet을 사용할 수 있다. 이들을 사용한 서블릿 개발환경은 5장에서 자세히 설명한다.

■ DDBC

DDBC(DCL Database Connectivity)는 DBMS와 운영체제 사이에서 어플리케이션의 이식성과 성능의 극대화를 목표로 개발된 데이터베이스 드라이버 시스템으로 데이터베이스 클라이언트 어플리케이션을 서로 다른 DBMS 접속 환경으로부터 분리해 준다. DDBC 인터페이스는 DCLCore에 포함되어 있고 ANSI SQL92에서 정의한 대부분의 데이터 타입을 지원하며 Informix, InterBase, MySQL, Oracle의 드라이버가 개발되어 있다[18].

[4] ASSERT는 조건이 거짓일 경우 관련 메시지를 출력한 후 프로그램의 실행을 중단하고, TRACE는 프로그램의 정상적인 출력 외에 디버깅을 위한 제 삼의 출력을 사용한다.

제3장 DCL HTTP 서버 확장

3.1 기본개념

DCL HTTP 서버 확장(DHE: DCL HTTP Server Extension)은 HTTP 서비스에서 운영체제와 서버 소프트웨어간의 이식성이 있으면서 서버의 API를 사용하는 플러그인 모듈의 성능을 가지는 어플리케이션 환경을 위하여 다음 세 가지의 개념에 기초한다.

첫 번째, 운영체제 이식성과 성능의 목표를 동시에 달성하기 위해 DHE의 어플리케이션인 서블릿은 네이티브 바이너리 코드를 생성하는 컴파일러 언어를 사용하여 개발된 DSO의 형태를 취한다. 서블릿(Servlet)은 자바의 서블릿과 마찬가지로 “Server”와 접미사 “-let”의 합성어로 서버조각을 의미한다.

두 번째, 웹 서버간의 이식성을 위하여 기존 서버들의 API와는 독립적인 새로운 인터페이스를 정의하고 HTTP 서버의 플러그인 모듈로 구현되어 동작하는 서블릿 컨테이너(Servlet Container)라는 개념을 사용한다. 서블릿 컨테이너는 하나 이상의 서블릿을 위한 풀(pool)로 서블릿의 실행환경을 제공한다.

세 번째, 서블릿과 서블릿 컨테이너는 모두 웹 서버의 작업 프로세스(worker process)내에서 동작하도록 한다. 따라서 FastCGI의 경우와 같은 별도의 IPC 오버헤드는 발생하지 않는다.

DHE를 이해하기 위해 기존의 널리 알려진 기술들과 비교할 수 있다. Apache 서버에서 .php 요청이 있는 경우, 서버는 HTTP 요청을 PHP의 실행 환경인 mod_php.so에 이를 전달하게 되고 PHP의 실행환경은 스크립트를 실행하여 그 결과를 User-Agent에 되돌린다. IIS의 ASP의 경우는 .asp 요청을 asp.dll에 전달하고 asp.dll은 스크립트를 실행하여 그 결과를 User-Agent에 되돌린다.

이들과 마찬가지로 DHE의 실행 환경인 DHE 서블릿 컨테이너는 실제 어플리케이션인 DHE 서블릿을 적재하고 이것에 실행요청을 하여 그 결과를 User-Agent에 되

돌리도록 하게 함으로써 HTTP 요청에 응답한다.

<표 3-1> 웹 서버 어플리케이션 기술의 비교

서버기술	언어	번역 및 실행	이식성	DB 인터페이스
PHP	PHP	Interpreter	○*	PHP lib
ASP	VBScript, JScript, Perl	Interpreter	△**	ADO
ASP .NET	VB, JScript, C#, C++	Compiler	X	ADO
JSP (Java Servlet)	Java	Compiler, Interpreter (JVM)	○	JDBC
FastCGI***	(C++)	(Compiler)	○	-
DHE(DCL)	C++	Compiler	○	DDBC

* PHP에서 제공하는 라이브러리 함수는 데이터베이스 의존적이기 때문에 DBMS 이식성이 있는 어플리케이션의 개발이 쉽지 않다.

** ASP의 경우 Perl을 사용하여 스크립트를 작성할 경우 이식성이 있으나 ADO와 같은 ActiveX를 사용하면 이식성이 없다.

*** FastCGI는 CGI의 발전된 형태로 인터페이스만 제공하고 사용 언어는 정의하고 있지 않다. 이 표에서는 C++가 사용될 수 있음을 보인다.

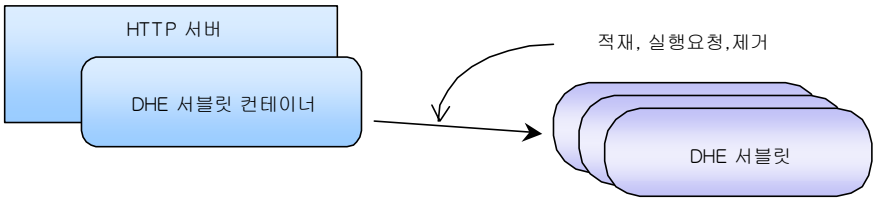
3.2 구성요소

DHE이 실행환경을 이루는 기본 구성요소는 HTTP 서버, DHE 서블릿 컨테이너 그리고 DHE 서블릿의 세 가지이고 DHE 관리서버는 서버의 작업 프로세스가 다중 프로세스 방식일 경우 선택적으로 사용된다.

3.2.1 HTTP 서버

HTTP 서버는 Apache1, Apache2, IIS와 같은 웹 서버를 말한다. DHE는 독립적

인 서버로 구현되지 않고 기존에 널리 알려진 웹 서버들의 API를 확장하는 형태를 취한다. 이렇게 하는 이유는 기존의 웹 서버들이 이미 널리 사용되고 있고 이들을 사용해서 개발된 기존의 웹 어플리케이션을 그대로 사용할 수 있도록 함과 동시에 또 다른 개발환경을 제공하고자 위함이다.



<그림 3-1> DHE 기본 구성요소

3.2.2 DHE 서블릿

DHE 서블릿은 HTTP 요청을 분석하고 이에 대한 실질적인 서비스를 담당하는 부분으로 운영체제의 DSO로 구현된 모듈이다.

하나의 서블릿은 단 한 개의 DCL_HTTP_SERVLET 인터페이스를 구현해야 한다. 이 인터페이스를 통하여 서블릿의 생명주기 동안 단 한번의 초기화 함수가 불려지고 한번 이상의 HTTP 서비스 요청 요구를 받게 된다. 적재된 서블릿은 제거되기 직전에 서블릿 컨테이너에 의하여 크린업(cleanup)함수가 호출되어 서블릿이 사용하고 있는 시스템 자원을 반환하도록 한다.

서블릿은 서블릿 컨테이너로부터 제공받은 서버 API 때문에 다양한 웹 서버로부터 API 독립성을 보장 받는다. 따라서 동일한 운영체제하에서는 서로 다른 웹 서버라 할지라도 별도로 컴파일할 필요가 없다. 예를 들어, Windows 운영체제를 위하여 컴파일된 서블릿은 Windows의 Apache와 IIS에서 모두 사용될 수 있다.

하나의 웹 서버 프로세스는 동일한 서블릿을 중복하여 적재하지 않는다. 이 때문에 다중 스레드로 서비스 하는 웹 서버에서 하나의 서블릿은 둘 이상의 HTTP 요청

에 관한 스레드 상태에 놓일 수 있다.

서블릿 컨테이너로부터 전달 받은 HTTP 요청은 CGI 인터페이스와 관련한 변수와 서버의 환경변수를 제외하고는 가공되지 않은 상태이다. 이 때문에 서블릿은 QUERY_STRING, Cookie 뿐만 아니라 HTTP POST 메소드에 의해 제출된 데이터(submitted data)를 디코드(decode)해야 한다.

3.2.3 DHE 서블릿 컨테이너

DHE 서블릿 컨테이너는 웹 서버가 제공하는 API의 프로세스 환경 내에서 동작하는 웹 서버의 어플리케이션으로 IIS에서는 ISAPI 확장모듈의 형태를 취하고 Apache 서버에서는 Apache 확장모듈로 구현된다.

서블릿 컨테이너의 첫 번째 역할은 서블릿을 적재(load)하여 실행요청(invoke)을 하고 제거(unload)하는 것이다.

서블릿을 적재하는 과정에는 서블릿의 엔트리포인트가 적법하게 작성되었는지를 검사한다. 서블릿은 운영체제의 동적공유객체(DSO)이기 때문에 DSO 적재를 위한 함수 호출 중에는 서블릿 내부에서 참조한 외부의 다른 모듈에 있는 심볼을 찾지 못하는 등의 에러가 발생할 수 있다. 만약 서블릿의 적재와 검사에서 에러가 발생하면 서블릿 컨테이너는 클라이언트인 User-Agent에 이에 관한 내용을 보고한다.

서블릿의 적재가 성공적으로 완료되면 서블릿을 초기화 하기위해 서블릿의 엔트리포인트를 통하여 초기화 함수를 호출한다. 초기화까지 성공적으로 마친 서블릿은 서블릿 컨테이너가 관리하는 서블릿 풀(pool)에 이를 삽입하고 서블릿의 엔트리포인트의 멤버인 HTTP 서비스 함수를 통하여 HTTP 요청을 전달한다.

서블릿의 엔트리포인트를 이루는 함수들은 그 함수 내에서 에러가 발생할 수 있는데 서블릿에 의하여 에러 메시지가 서블릿 컨테이너에 전달되면 서블릿 컨테이너는 이를 User-Agent에 다시 전달한다.

서블릿의 적재와 서비스 실행요청, 그리고 제거의 과정에는 DHE.INI에 설정된 서블릿의 정책에 관한 설정이 사용된다. 한번 초기화된 서블릿은 서블릿 풀에 캐시되어 DSO의 적재와 관련된 오버헤드가 제거된 상태에서 HTTP 서비스를 수행할 수

있다. 만일 DHE.INI의 설정이 서블릿의 서비스를 금지시키면 서블릿 컨테이너는 파일 시스템에 서블릿이 존재하더라도 User-Agent의 HTTP 요청을 거절한다.

서블릿 컨테이너의 또 다른 중요한 역할은 웹 서버마다 각기 다른 API 차이점을 흡수하여 서블릿에게 특정 웹 서버로부터 독립적인 API 환경을 제공하는 것이다. 3.3절에서 설명하고 있는 인터페이스 중에 DCL_HTTP_SERVER_API가 여기에 해당된다. 서블릿 컨테이너는 이 인터페이스를 구현하여 서블릿으로 하여금 HTTP 요청을 완료하기 위해 웹 서버의 API에 접근 하도록 한다.

마지막으로, 서블릿 컨테이너는 실행시간에 서블릿 컨테이너의 설정에 관한 변경이 가능하도록 제어 인터페이스를 제공한다. 이 인터페이스는 단일 프로세스/다중 스레드 웹 서버뿐만 아니라 다중 프로세스/다중 스레드 웹 서버에서도 동일한 형태의 제어가 가능하도록 한다.

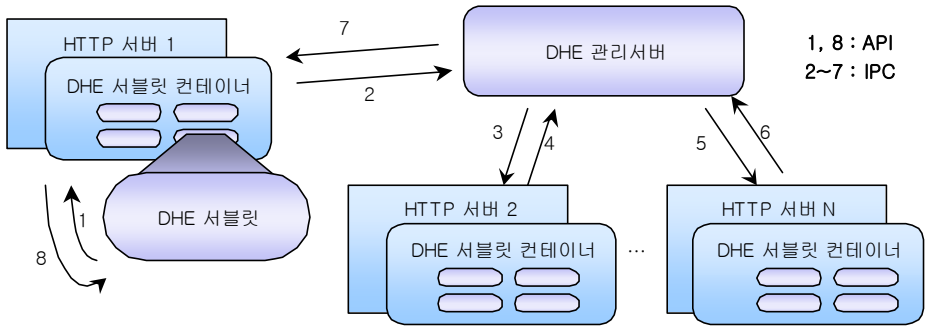
3.2.4 DHE 관리서버

Microsoft Windows 운영체제에서 Apache 서버나 단일 응용프로그램 상태에서 서비스하는 IIS의 HTTP 서비스와 같이 단일 프로세스/다중 스레드 방식의 프로세싱 환경에서 동작하는 웹 서버의 경우에는 서블릿 컨테이너의 제어는 비교적 간단히 구현될 수 있다. 이것은 서블릿과 서블릿 컨테이너가 동일한 프로세스 환경에서 동작하기 때문에 서블릿 컨테이너는 서블릿의 제어요청에 대하여 자신에 대해서만 처리하면 되기 때문이다. 그러나 다중 프로세스 프로세싱 환경에서의 서블릿 컨테이너는 서블릿의 제어 요청을 다른 프로세스에도 전달해야 한다.

<그림 3-2>는 다중 프로세스 환경에서 DHE 관리서버와의 관계를 표시하고 있다. 서버의 관리 목적으로 작성된 서블릿은 자신이 속해있는 서블릿 컨테이너에게 DCL_HTTP_SERVER_API의 멤버인 pfnServerControl 함수를 사용하여 제어요청을 한다. 이때 서블릿 컨테이너는 자신이 관리서버에 의해 관리되고 있으면 IPC채널을 통하여 제어요청을 전달하고 관리서버 역시 자신이 관리하고 있는 서블릿 컨테이너에 전달하여 그 결과를 되돌린다.

관리서버가 적용될 수 있는 운영체제는 다중 스레드가 지원되는 것만 가능하다.

이것이 적용될 경우, 서블릿 컨테이너는 HTTP 서버의 생명주기 동안 관리서버와 메시지교환을 위해 IPC연결을 확립하고 있게 되며 관리서버의 비동기적인 제어요청에 응답하기 위해 별도의 스레드를 생성한다.



<그림 3-2> DHE 관리서버

3.3 DHE 인터페이스

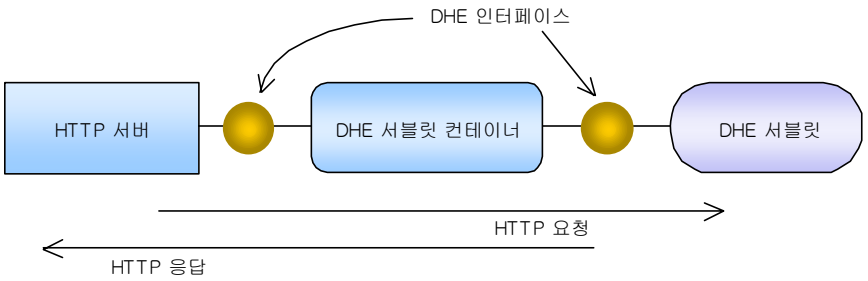
3.3.1 역할과 특징

DHE 인터페이스의 역할은 HTTP 서버, 서블릿 컨테이너 그리고 서블릿 간의 컴파일타임 의존관계를 제거하고 각 구성요소 사이에서 메시지를 전달하는 것이다.

특히, 컴파일타임 의존관계를 제거하는 역할은 매우 중요하다. 이것은 HTTP 서버의 API가 C언어를 사용하고 있고, API명세를 기술하는 헤더파일에 C++의 예약어를 식별자로 사용할 경우 C++ 컴파일러를 사용하여 컴파일하는 것이 불가능 한 경우가 있는데 Apache 서버가 이에 해당된다.

DHE의 인터페이스는 크게 두 가지로 구분할 수 있다. 첫 번째는 서블릿 컨테이너에서 구현되어 서블릿에게 제공되는 것으로 서블릿이 HTTP 서버의 기능에 접근하도록 하는 콜백함수(callback function)가 있다. 두 번째는 서블릿을 구현할 때 사용하는 것으로 서블릿의 엔트리포인트(entry point)로 익스포트(export)되어 서블릿 컨테이너로 하여금 서블릿을 실행하도록 한다. 이들 인터페이스는 C언어의 구조체로

되어 있으며 typedef문을 사용하여 별도로 정의되어 있는 함수 포인터 멤버를 포함하고 있다.



<그림 3-3> DHE 인터페이스

DHE 인터페이스는 표준 C언어로 기술되어 있기 때문에 서블릿 개발을 위해 반드시 C/C++을 사용할 필요는 없으며 표준 C언어의 함수 호출규약을 제공할 수 있고 운영체제의 DSO를 개발 할 수 있는 언어는 무엇이든 가능하다. 다만 서블릿의 DCL_HTTP_SERVLET 인터페이스의 DSO 엔트리포인트는 반드시 표준 C언어 네임 맹글링(name mangling)을 사용하여야 한다.

DHE 인터페이스를 사용하여 서블릿을 개발 할 때에는 인터페이스 명세 외에는 다른 어떠한 임포트(import) 라이브러리도 필요가 없다. 서블릿 컨테이너가 서블릿을 호출할 때에는 인터페이스를 통해 서버의 API에 접근할 수 있는 메소드와 데이터를 함께 제공하기 때문이다. 이러한 특징이 서블릿을 개발하는데 있어서 C/C++ 이외의 언어를 사용할 수 있도록 한다.

3.3.2 콜백함수

■ DCLHttpWriteClient

```
typedef BOOL (* DCLHttpWriteClient)(
                DCL_HTTP_HCONN      hConn,          /* (IN) */
                const void*         pvBuffer,       /* (IN) */
```

```
    UINT32*                 pnLength        /* (IN, OUT) */
    );
```

버퍼의 데이터를 클라이언트로 전송한다. *pnLength에는 정상적으로 전송된 데이터의 크기가 설정된다. 리턴 값이 FALSE이면 클라이언트와의 연결에 에러가 발생한 것이다.

■ DCLHttpReadClient

```
typedef BOOL (* DCLHttpReadClient)(
    DCL_HTTP_HCONN      hConn,          /* (IN) */
    void*               pvBuffer,       /* (OUT) */
    UINT32*             pnLength        /* (IN, OUT) */
    );
```

HTTP POST 메소드에 의해 클라이언트로부터 전송되어진 데이터를 읽는다. 함수의 요청이 성공하면 *pnLength에 읽은 데이터의 크기가 설정된다.

리턴 값이 TRUE이고 *pnLength가 0이면 더 이상 읽혀질 데이터가 없는 경우이고 FALSE는 클라이언트와의 연결이 종료되었거나 에러가 발생한 것이다

■ DCLHttpSendResponseHeader

```
typedef BOOL (* DCLHttpSendResponseHeader)(
    DCL_HTTP_HCONN      hConn,          /* (IN) */
    UINT32              uStatusCode,    /* (IN) */
    const char*         pszHeader,      /* (IN) */
    UINT32      uLength /* (IN) number of bytes in header */
    );
```

클라이언트에 HTTP Response-Header를 전송한다. uStatusCode는 HTTP Status-Code를 지시하고 서블릿 컨테이너는 이 값을 사용하여 Status-Line을 완성한다. pszHeader는 헤더를 포함하고 있는 문자열을 가리킨다. 이 문자열은 HTTP 프로토콜 명세에서 지시하는 형식이어야 하며 맨 마지막엔 CRLF문자가 추가되어 있어야 한다. 다음은 이 문자열의 예이다.

Cache-Control : no-cache CRLF

Content-Type : text/html CRLF
CRLF

이 함수는 헤더 데이터에 대하여 NULL('\0') 종결 문자열을 파라미터로 받지만 NULL문자를 제외한 문자열의 길이를 제공해야 한다.

■ DCLHttpGetRequestHeader

```
typedef BOOL (* DCLHttpGetRequestHeader)(
        DCL_HTTP_HCONN      hConn,      /* (IN) */
        const char*         pszName,    /* (IN) NULL => all */
        char*               pchBuffer,  /* (OUT) */
        UINT32*             pnLength    /* (IN, OUT) */
        );
```

HTTP Request-Header의 값을 얻어온다. pszName이 NULL이면 전체 헤더를 얻으며 각각의 헤더 필드는 CRLF로 구별된다.

함수 호출 후 리턴 값이 FALSE이고 *pnLength가 0보다 크면 버퍼의 크기가 작음을 의미한다. 이 경우 *pnLength는 필요한 버퍼의 크기를 의미한다. 리턴 값이 FALSE이고 *pnLength가 0이면 pszName이 없거나 값이 없는 경우이다.

■ DCLHttpGetCgiVariable

```
typedef BOOL (* DCLHttpGetCgiVariable)(
        DCL_HTTP_HCONN      hConn,      /* (IN) */
        const char*         pszVarName, /* (IN) NULL => all vars */
        void*               pvBuffer,   /* (OUT) */
        UINT32*             pnLength    /* (IN, OUT) */
        );
```

CGI와 호환된 이름을 사용하는 변수의 값을 얻어온다. pszVarName이 NULL이면 웹 서버가 제공하는 모든 변수들의 이름과 값을 얻으며 각각은 CRLF로 구별된다. 이때 각각의 변수는 "name = value CRLF" 형태이다.

함수 호출 후 리턴 값이 FALSE이고 *pnLength가 0보다 크면 버퍼의 크기가 작음을 의미한다. 이 경우 *pnLength는 필요한 버퍼의 크기를 의미한다. 리턴 값이

FALSE이고 *pnLength가 0이면 pszVarName이 없거나 값이 없는 경우이다.

■ DCLHttpWriteStream

```
typedef void (* DCLHttpWriteStream)(
        void*               hStream,        /* (IN) */
        const void*         pvData,         /* (IN) */
        UINT32              uLength         /* (IN) */
        );
```

서블릿과 서블릿 컨테이너 사이에 스트림 데이터를 주고받을 때 사용하는 함수 이다. hStream 값은 데이터를 받는 쪽에서 제공한다.

■ DCLHttpServerControl

```
typedef void (* DCLHttpServerControl)(
        DCL_HTTP_HCONN      hConn,          /* (IN) */
        const char*         psControl,
        UINT32              uLength,
        DCLHttpWriteStream  pfnWriteResult,
        void*               hResultOutputStream
        );
```

실행시간에 서블릿 컨테이너의 설정을 변경하는데 사용된다.

3.3.3 엔트리포인트 함수

■ DCLHttpServletInitialize

```
typedef BOOL (* DCLHttpServletInitialize)(
        const DCL_HTTP_SERVLET_CONFIG* pConfig,
        void*               hReportError
        );
```

DCL_HTTP_SERVLET의 멤버 함수로 서블릿에서 구현되어 있어야 한다. 서블릿 컨테이너에 의해 서블릿이 성공적으로 적재되면 최초에 단 한번 불려 진다. 서블릿의 초기화 중에 에러가 발생하면 서블릿 컨테이너에게 에러 메시지를 전달하기 위하

여 DCL_HTTP_SERVER_API::pfnReportError에 hReportError를 적용하여 사용할 수 있으며 FALSE를 리턴해야 한다.

■ DCLHttpServletCleanup

```
typedef BOOL (* DCLHttpServletCleanup)(
                void*                   hReportError
                );
```

DCL_HTTP_SERVLET의 멤버 함수로 서블릿에서 구현되어 있어야 한다. 서블릿 풀에서 서블릿이 제거되기 직전에 단 한번 호출된다.

■ DCLHttpServletService

```
typedef BOOL (* DCLHttpServletService)(
                const DCL_HTTP_SERVLET_CONTEXT* pContext,
                void*                   hReportError
                );
```

HTTP 요청이 있을 때마다 호출되며 둘 이상의 스레드 상태에 있을 수 있다.

3.3.4 데이터 구조

■ DCL_HTTP_SERVER_API

서블릿 컨테이너가 서블릿을 초기화 할 때 서버 API들에 대한 함수 포인터를 제공한다.

```
struct _DCL_HTTP_SERVER_API {
        UINT32                  uSize;          /* (IN) size of this structure */
        UINT32                  uVersion;       /* (IN) version info of this spec */
        DCLHttpWriteClient              pfnWriteClient;
        DCLHttpReadClient               pfnReadClient;
        DCLHttpSendResponseHeader       pfnSendResponseHeader;
        DCLHttpGetRequestHeader         pfnGetRequestHeader;
        DCLHttpGetCgiVariable           pfnGetCgiVariable;
        DCLHttpWriteStream              pfnReportError;
```

```
    DCLHttpServerControl        pfnServerControl;
};
```

■ DCL_HTTP_SERVLET_CONFIG

DCL_HTTP_SERVLET::pfnIntialize가 호출될 때 사용된다. 서블릿은 이 구조체의 값들을 사용하여 자신을 초기화 한다.

```
typedef struct _DCL_HTTP_SERVLET_CONFIG {
    const char*                 pszFileName;
    const char*                 pszConfigDir;
    const char*                 pszTempDir;
    const DCL_HTTP_SERVER_API* pSAPI;
} DCL_HTTP_SERVLET_CONFIG;
```

pszFileName 절대경로를 포함하는 서블릿의 파일명이다. 서블릿은 웹 서버와 동일한 프로세스 환경에 있기 때문에 프로세스의 실행 디렉토리를 변경하는 chdir과 같은 시스템 호출은 사용하지 말아야 하며, 서블릿이 위치한 디렉토리를 알고자 할 때에는 pszFileName으로부터 얻어와야 한다.

pszConfigDir 서블릿이 중앙 집중식으로 관리될 경우 서블릿들의 초기화 파일들이 위치할 디렉토리를 가리킨다.

pszTempDir 서블릿에서 임시 파일을 사용할 경우 사용할 수 있는 디렉토리이다.

pSAPI 서블릿 컨테이너에 의해 제공되는 서버의 API로 서블릿은 이 값을 보관하고 있어야 한다.

■ DCL_HTTP_SERVLET_CONTEXT

HTTP 요청에 의해 DCL_HTTP_SERVLET::pfnHttpService가 호출될 때 사용된다.

```
typedef struct _DCL_HTTP_SERVLET_CONTEXT {
    UINT32              uSize;          /* (IN) size of this structure */
    UINT32              uVersion;       /* (IN) version info of this spec */
    DCL_HTTP_HCONN      hConn;          /* (IN) */
    const char*         pszRemoteAddr;  /* (IN) REMOTE_ADDR */
```

```
    UINT32              uRemotePort;        /* (IN) REMOTE_PORT */
    UINT32              uRequestMethod;     /* (IN) RFC2616/5.1.1 */
    const char*         pszRequestMethod;/* (IN) REQUEST_METHOD */
    const char*         pszPath;            /* (IN) URI's abs_path */
    const char*         pszQueryString;     /* (IN) QUERY_STRING */
    const char*         pszContentType;     /* (IN) CONTENT_TYPE */
    UINT32              uContentLength;     /* (IN) CONTENT_LENGTH */
} DCL_HTTP_SERVLET_CONTEXT;
```

uSize 이 구조체의 크기를 바이트 단위로 나타낸다.

uVersion dcl/.Config.h에 DCL_HTTP_SERVER_VERSION로 정의되어 있으며 DHE 명세의 버전을 의미한다.

hConn pfnReportError를 제외한 모든 DCL_HTTP_SERVER_API의 멤버를 호출할 때 사용한다.

pszRemoteAddr User-Agent의 주소를 담은 문자열이다. CGI 변수에서는 REMOTE_ADDR로 표현한다.

uRemotePort User-Agent가 사용한 포트번호 이다. CGI의 REMOTE_PORT와 같다.

uRequestMethod HTTP Request-Method를 숫자로 표현한다. HTTP가 확장되어 요청된 메소드를 지시하는 문자열이 OPTIONS, GET, HEAD, POST, PUT, DELETE, TRACE, CONNECT 중의 하나가 아니면 이것의 값은 HTTP_METHOD_UNKNOWN이다. 이 경우 서블릿은 pszRequestMethod의 값을 검사하여야 한다.

```
enum HTTP_REQUEST_METHOD {
        HTTP_METHOD_UNKNOWN     = 0,
        HTTP_METHOD_OPTIONS     = 1,
        HTTP_METHOD_GET         = 3,
        HTTP_METHOD_HEAD        = 4,
        HTTP_METHOD_POST        = 5,
        HTTP_METHOD_PUT         = 6,
        HTTP_METHOD_DELETE      = 7,
        HTTP_METHOD_TRACE       = 8,
        HTTP_METHOD_CONNECT     = 9
```

};

pszRequestMethod HTTP Request-Method에 해당하는 문자열이다.

pszPath URI(Uniform Resource Identifier)에서 절대경로(abs_path)에 대한 값이다.

pszQueryString URI의 QUERY_STRING을 나타낸다.

pszContentType HTTP POST 메소드를 통해 전달 받는 데이터의 MIME 타입을 의미한다.

uContentLength POST 데이터의 크기이다.

■ DCL_HTTP_SERVLET

서블릿의 DSO 엔트리포인트를 구성하는 인터페이스이다. 서블릿 컨테이너가 서블릿을 적재한 직후에 각 멤버의 값을 검사한다.

```
typedef struct _DCL_HTTP_SERVLET {
        /* DCL common members     */
        UINT32              uSize;              /* (OUT) size of this structure          */
        UINT32              uDCLVersion;        /* (OUT) DCL_VERSION, non DCL(0) */
        const char*         pszBuildTimeStamp;          /* (OUT) __TIMESTAMP__ */
        UINT32              uBuildFlag;         /* (OUT) release(0), debug(1)            */
        UINT32              uModuleType;        /* (OUT)                                 */
        const char*         pszDescription;     /* (OUT) module description              */
        /* private members */
        UINT32              uVersion;           /* (OUT) version info of this spec */
        DCLHttpServletInitialize    pfnInitialize;
        DCLHttpServletCleanup       pfnCleanup;
        DCLHttpServletService       pfnHttpService;
} DCL_HTTP_SERVLET;
```

uSize 이 구조체의 크기를 바이트 단위로 나타낸다.

uDCLVersion 서블릿이 DCL을 사용해서 개발되었을 경우 dcl/.Config.h에 정의된 DCL_VERSION을 나타낸다. 다른 경우 이 값은 0이다.

pszBuildTimeStamp 서블릿이 컴파일된 __TIMESTAMP__ 값이다.

uBuildFlag 서블릿이 DCL을 사용해서 개발되었을 경우 Debug로 컴파일되면 1을 그렇지 않으면 0이다.

uModuleType DCL에서 정의한 모듈의 종류를 나타내며 dcl/.Config.h에서 이 값은 DCL_HTTP_SERVLET_MODULE로 정의(define)되어 있다.

pszDescription 서블릿에 대한 간략한 설명을 포함하는 문자열이다.

uVersion 이 값은 dcl/_Config.h에 DCL_HTTP_SERVER_VERSION으로 정의되어 있으며 DHE의 버전을 의미한다.

pfnInitialize 서블릿 컨테이너가 서블릿을 적재한 후 서블릿에 초기화 기회를 주기위해 최초에 한번 호출된다.

pfnCleanup 서블릿이 닫히지 직전에 단 한번 호출된다.

pfnHttpService HTTP 요청이 있을 때마다 호출된다. 이 함수는 웹 서버가 다중 스레드로 서비스가 가능할 경우 2개 이상의 스레드에 의하여 호출 될 수 있으며, 서블릿에서 전역적으로 사용되는 객체들에 대한 동기화에 대하여 주의하여야 한다.

제4장 서블릿 컨테이너의 구현

4.1 서블릿 컨테이너의 설정과 제어

4.1.1 DHE.INI

DHE.INI는 DHE의 실행환경에 대한 구성(configuration)을 위하여 서블릿 컨테이너에 의해 사용되는 파일이다. 이것은 서블릿 컨테이너의 설정과 서블릿 서비스에 관한 정책을 저장하고 있는 파일로 기본적으로 “dhe.ini” 라는 이름을 갖는다(디버그 버전의 서블릿 컨테이너에서는 “dhed.ini” 이다). 만약 HTTP 서버의 서비스에 관여하는 작업 프로세스가 다중 프로세스일 경우, 서블릿 컨테이너는 관리서버로부터 초기화 되고 통제된다.

```
 1: ;dhe.ini
 2:
 3: [SERVER]
 4: CONTROL_ALLOWS = 127.0.0.1
 5: CONTROL_PASSWORD =
 6:
 7: LOG_TYPES = error,warning,information
 8:
 9: SERVLET_CONFIG_DIR = D:/HOME/dcl/examples/dhe/conf/
10: SERVLET_TEMP_DIR = D:/HOME/dcl/examples/dhe/log/
11:
12: [SERVLET]
13: ; path,          filename,          enable,          cacheable
14: ;-----------------------------------------------------------------
15: /varinfo.dhe,D:/HOME/dcl/examples/dhe/varinfo/Release/varinfo.dhe,true,false
16: /src2html.dhe,D:/HOME/dcl/examples/dhe/src2html/www/src2html.dhe,true,true
```

<그림 4-1> dhe.ini

<그림 4-1>에서 보이고 있는 DHE.INI 파일은 텍스트 파일로 만들어 지며 ‘;’나

‘#’으로 시작되는 라인과 비어 있는 라인은 무시된다.

4라인의 CONTROL_ALLOWS은 서블릿 컨테이너의 설정을 변경하게 될 경우 허용되는 클라이언트의 IP주소 값으로 ‘,’로 구분되는 둘 이상의 값이 될 수 있다.

5라인의 CONTROL_PASSWORD는 서블릿 컨테이너의 제어를 위하여 서블릿 컨테이너에 접속할 때 사용되는 패스워드로, 값이 설정되면 패스워드의 MD5값을 갖고 그렇지 않으면 비어 있다.

7라인의 LOG_TYPES은 서블릿 컨테이너가 dhe.log에 저장하는 로그의 종류를 결정한다.

9, 10라인은 서블릿이 초기화 될 때 서블릿에게 전달하는 값으로 서블릿은 이 값을 사용하여 자신의 초기화 파일을 결정하거나 임시파일을 만들 수 있다.

12라인부터는 서블릿의 실행 정책에 대한 내용으로 각각의 의미는 다음과 같다.

path URI에서의 절대경로에 해당한다.

filename 서버의 파일시스템에 위치한 서블릿의 실제 파일명이다.

enable 서블릿 서비스의 가능(enable)을 나타낸다. User-Agent가 이 값이 false인 서블릿의 서비스를 요청하면 서블릿 컨테이너는 에러를 되돌린다.

cacheable 서블릿의 cache 설정을 true와 false로 지정한다. 이 값이 true이면 서블릿은 한번 초기화 되어 서블릿 컨테이너의 서블릿 풀에 보관되기 때문에 반복적인 런타임 링킹과 초기화가 제거된다. false의 경우 기본동작은 HTTP 요청마다 서블릿을 적재하고 서블릿의 실행이 마치면 제거된다. 그러나 다중 스레드 방식의 웹 서버에서는 하나의 스레드에서 서블릿의 실행 중에 또 다른 스레드의 요청이 있게 되면 서블릿은 재사용된다.

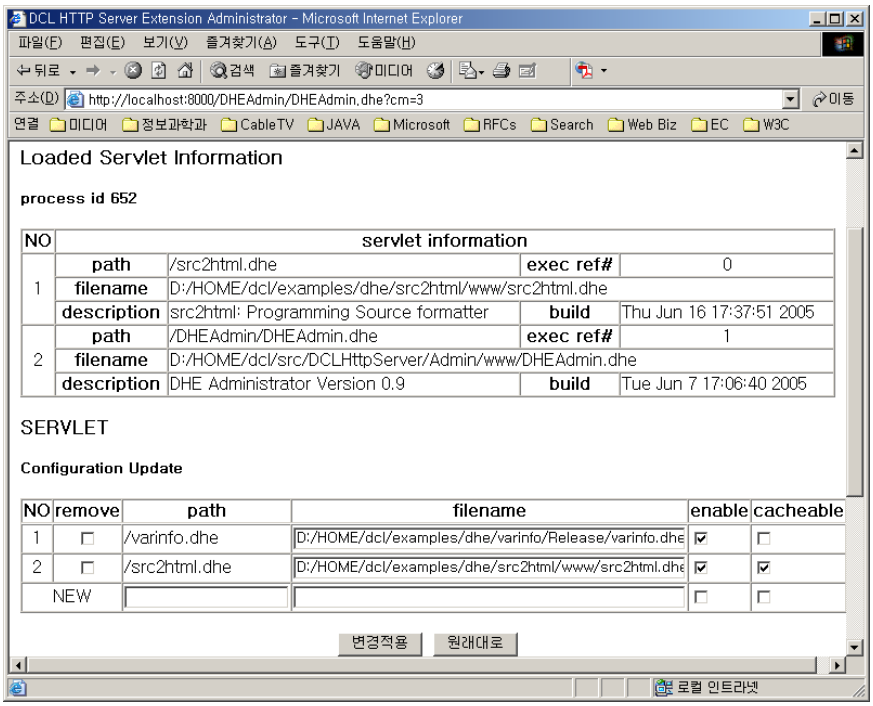
4.1.2 DHEAdmin.dhe

DHEAdmin.dhe은 서블릿 컨테이너를 실시간으로 제어하도록 개발된 서블릿으로 DCL_HTTP_SERVER_API 인터페이스의 pfnServerControl을 사용하여 구현되어 있다. 서블릿 컨테이너는 이것을 통한 제어요청의 결과가 성공할 경우 필요하면 그

내용을 DHE.INI에 기록하여 구성을 유지한다.

DHEAdmin.dhe을 사용하면 DHE.INI의 모든 설정들에 대한 변경이 가능하지만 이것을 사용하는 가장 중요한 목적은 DHE.INI에서 서블릿의 정책을 표시하고 있는 [SERVLET] 섹션과 관련이 있다

<그림 4-1>의 16라인을 보면 src2html.dhe는 cacheable 부분이 true로 되어 있다. 이러한 경우, User-Agent에 의하여 이 서블릿이 요청되어 적재되면 다른 조치를 취하지 않는 한 웹 서버 프로세스가 종료될 때까지 서블릿 컨테이너에 의하여 캐시된다. 서블릿 컨테이너를 실시간으로 제어할 수 있는 방법이 없는 상황이라면 src2html.dhe를 변경해야 할 경우 웹 서버를 종료하지 않고는 서블릿을 교체할 수 없게 된다.



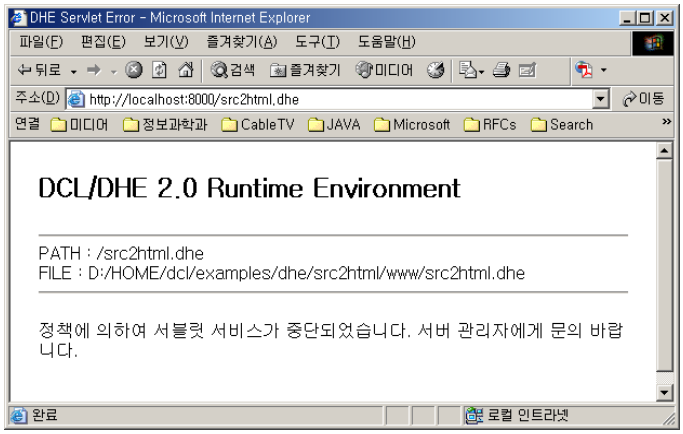
<그림 4-2> DHEAdmin.dhe

따라서 웹 서버의 실행시간에 서블릿 컨테이너를 제어할 방법이 반드시 필요한 것

이다. <그림 4-2>는 src2html.dhe를 호출 한 후 DHEAdmin.dhe을 통해 적재된 서블릿의 상태를 보여주고 있다. 이것을 실행하고 있는 서버는 Windows에서 Apache2인데 서버의 프로세스 ID는 652이다. 만약 GNU/Linux와 같은 UNIX 운영체제에서 Apache 웹 서버를 실행하면 1개 이상의 프로세스의 상태들에 대한 내용을 얻게 된다.

이 상태에서 서블릿 컨테이너의 풀에 캐시되어 있는 src2html.dhe를 제거하려면 cacheable의 체크박스의 체크를 제거하고 변경사항을 적용하면 된다. 서블릿 컨테이너는 DHEAdmin.dhe의 이 요청을 수행하는데 있어서 src2html.dhe의 실행참조카운터를 확인하여 참조카운터가 0이면 즉시 서블릿 풀에서 제거한다.

<그림 4-3>은 src2html.dhe의 enable과 cacheable을 모두 false로 적용한 후 src2html.dhe 요청에 대한 서블릿 컨테이너의 서비스 거부 메시지이다.



<그림 4-3> 서블릿 컨테이너의 서비스 거부

4.2 Apache 1.3.x, Apache 2.x

Apache 웹 서버에서 DHE 서블릿 컨테이너의 구현은 Microsoft의 Windows와 UNIX계열의 운영체제에서 서로 다르다. Windows에서 Apache1과 Apache2는 단일 프로세스 다중 스레드 상태에서 서비스를 하고, UNIX에서는 다중 프로세스 상태에

서 소켓을 공유하며 서비스를 하게 된다. 이러한 차이점 때문에 서블릿 컨테이너의 서블릿 제어와 관련된 부분은 운영체제에 직접적으로 의존되어 구현되게 된다. 하지만 Apache 서버의 버전과 운영체제별 구현의 차이점에도 불구하고 설치 방법은 Windows와 UNIX 모두 동일하다.

Apache 서버의 서블릿 컨테이너는 Apache 서버의 모듈에 관한 이름 규칙 권고에 따라 mod_dhe.so이다. Apache1과 Apache2 버전 모두 mod_dhe.so를 사용하는데 이것은 파일의 이름만 이렇게 사용할 뿐 같은 운영체제 일지라도 구현은 전혀 다르다.

Apache 서버에서 서블릿 컨테이너 모듈의 설치는 modules 하위 디렉토리에 mod_dhe.so를 복사하고 httpd.conf 를 수정하는 것으로 완료된다.

```
 1:  LoadModule dhe_module modules/mod_dhe.so
 2:  LoadModule dhed_module modules/mod_dhed.so
 3:
 4:  # Apache1 에서는 다음의 주석을 풀어준다.
 5:  #AddModule mod_dhe.c
 6:  #AddModule mod_dhed.c
 7:
 8:    <IfModule mod_dhe.c>
 9:    AddHandler dhe-handler .dhe
10:  DHE_INI      conf/dhe.ini
11:  DHE_LOG    logs/dhe.log
12:  </IfModule>
13:
14:  <IfModule mod_dhed.c>
15:  AddHandler dhed-handler .dhed
16:  DHE_INI      conf/dhed.ini
17:  DHE_LOG    logs/dhed.log
18:  </IfModule>
```

<그림 4-4> httpd.conf에서 DHE의 설정

<그림 4-4>는 DHE 서블릿 컨테이너를 위한 httpd.conf의 추가 내용으로 DHE_INI와 DHE_LOG 지시자가 생략되면 각각 “conf/dhe.ini”, “logs/dhe.log”가 기본값으로 설정된다. AddHandler 지시자는 파일의 확장자가 .dhe일 경우 HTTP 요청을 DHE

서블릿 컨테이너에 전달해 준다.

DCL 프로젝트의 다른 것들과 마찬가지로 DHE 서블릿 컨테이너도 디버그 버전과 릴리즈 버전이 존재하는데 이는 디버그 버전으로 빌드된 서블릿은 디버그 버전의 서블릿 컨테이너에 의해서만 서비스가 가능하도록 했기 때문이다. <그림 4-4>는 디버그 버전의 서블릿 컨테이너를 위한 설정내용을 포함하고 있는데 이것은 Windows 버전에서는 그대로 적용될 수 있으나 Linux에서는 둘 중 하나만 설정해야 한다. Linux에서 Apache 서블릿 컨테이너는 공유 라이브러리의 심볼 충돌에 관한 문제 때문에 같은 프로세스에서 디버그 버전과 릴리즈 버전이 동시에 사용될 수 없다.

Apache 서버에서 서블릿 컨테이너의 초기화와 종료에 관한 내용은 logs/error에 기록된다.

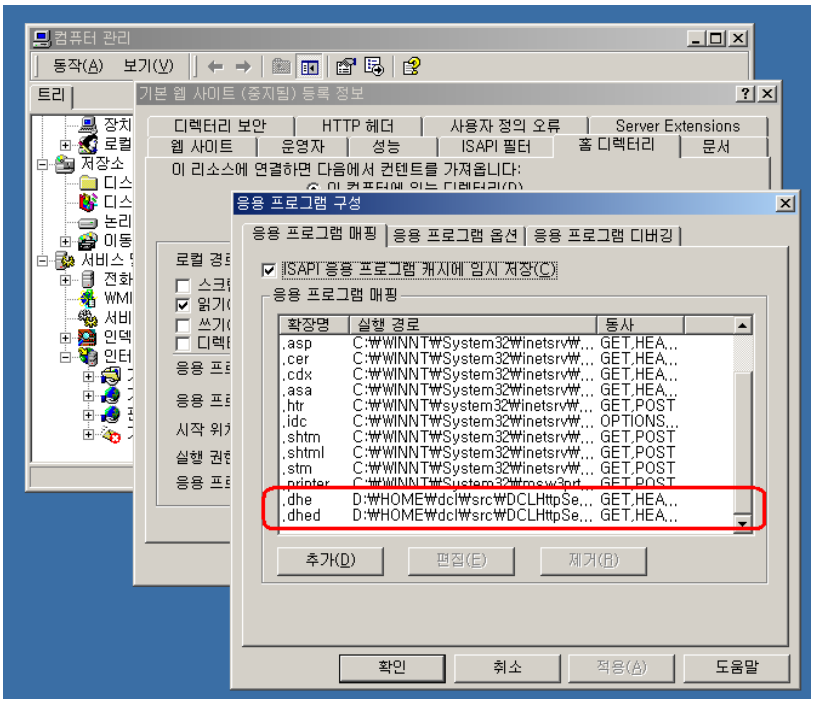
4.3 Microsoft IIS 5.0

IIS에서 사용되는 DHE 서블릿 컨테이너는 “dhe.dll”(디버그 버전은 “dhed.dll”)이다. 서블릿 컨테이너의 설치는 서블릿 컨테이너를 위한 레지스트리 내용을 <그림 4-5>의 예와 같이 설정해 주고 <그림 4-6>와 같이 IIS에 응용프로그램을 등록하면 DHE 서블릿을 사용할 수 있다.

IIS에서 서블릿 컨테이너의 초기화와 종료에 관한 정보는 Windows의 이벤트 로그(event log)의 응용프로그램 부분에 보고 된다.

```
1:  Windows Registry Editor Version 5.00
2:
3:  [HKEY_LOCAL_MACHINE\SOFTWARE\Kim Daejung\DCL\DHE]
4:  "INI"="D:/HOME/dcl/src/DCLHttpServer/IIS/DHE for IIS/dhe.ini"
5:  "LOG"="D:/HOME/dcl/src/DCLHttpServer/IIS/DHE for IIS/dhe.log"
6:
7:  [HKEY_LOCAL_MACHINE\SOFTWARE\Kim Daejung\DCL\DHED]
8:  "INI"="D:/HOME/dcl/src/DCLHttpServer/IIS/DHE for IIS/dhed.ini"
9:  "LOG"="D:/HOME/dcl/src/DCLHttpServer/IIS/DHE for IIS/dhed.log"
```

<그림 4-5> IIS를 위한 서블릿 컨테이너의 레지스트리 내용



<그림 4-6> DHE를 위한 IIS 5.0 응용프로그램 구성

4.4 명령행

명령행 서블릿 컨테이너는 Windows의 콘솔이나 UNIX의 쉘 상태에서 서블릿을 실행하도록 하는 유틸리티 성격의 프로그램으로 DHE.INI를 사용하지 않으며 서블릿 컨테이너의 로그는 표준에러(stderr)를 통해 출력된다.

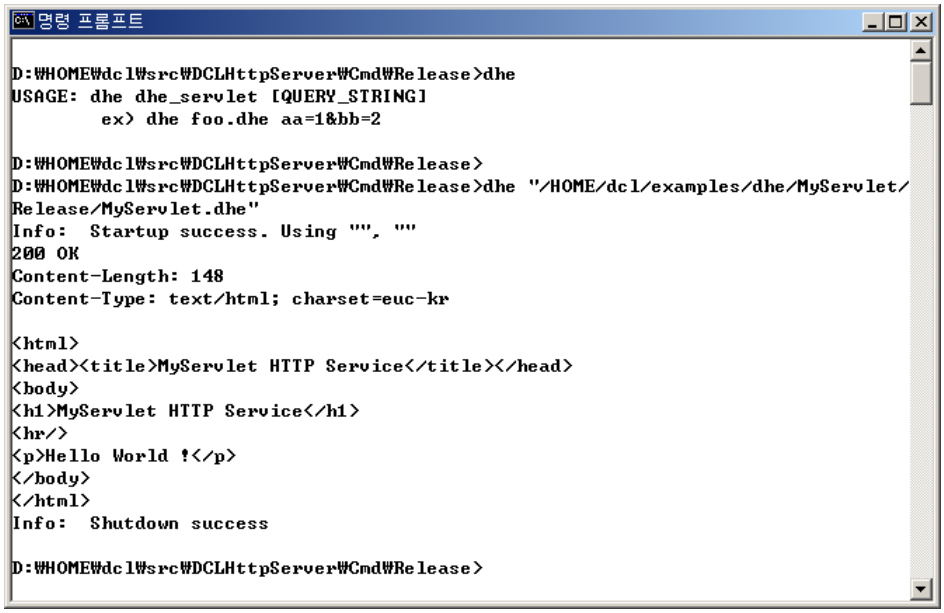
<그림 4-7>은 Windows의 콘솔에서 명령행 서블릿 컨테이너를 사용하여 서블릿을 실행한 결과를 보여주고 있는데 서블릿이 생성한 HTML 뿐만 아니라 HTTP 응답헤더도 함께 출력되는 것을 볼 수 있다.

명령행 서블릿 컨테이너 역시 Apache나 IIS를 위한 컨테이너와 마찬가지로 디버그 버전과 릴리즈 버전이 있는데, <표 4-1>은 Windows와 UNIX에서 명령행 컨테이너의 실행파일명을 보여주고 있다.

디버그 버전의 명령행 컨테이너는 Microsoft Visual C++의 디버거와 같이 소스 프로그램의 실행 상태를 추적하면서 디버깅을 하도록 하는 환경에서 유용하게 사용될 수 있다. DHE 서블릿은 DSO이기 때문에 이러한 디버거를 사용하여 직접 실행하는 것은 불가능한데 명령행 컨테이너를 사용하여 서블릿을 디버그 모드로 실행하도록 하는 것을 가능하게 한다.

<표 4-1> 명령행 서블릿 컨테이너의 파일명

운영체제	릴리즈 버전	디버그 버전
Windows	dhe.exe	dhed.exe
UNIX	dhe	dhed



<그림 4-7> 명령행 서블릿 컨테이너의 실행

제5장 DCL의 서블릿 개발환경

DHE는 서블릿에 대한 명세(specification)와 서블릿 컨테이너와 같은 실행 환경을 제공하지만 구체적인 서블릿을 구현하기 위해서는 별도의 많은 코드가 필요하다. DCL 프로젝트의 라이브러리는 범용의 시스템 프로그래밍을 위한 것으로 서블릿의 개발에도 사용할 수 있다. 특히, DCLNet에서는 서블릿 인터페이스를 구현한 클래스와 HTTP, HTML 관련 클래스가 구현되어 있고 서블릿 클래스를 사용한 프레임워크를 제공한다.

본 장에서는 서블릿을 개발하는데 직접적으로 관련이 있는 클래스들과 데이터베이스 관련 클래스, 그리고 서블릿의 디버깅을 위한 환경에 대하여 설명한다.

5.1 서블릿 프레임워크

■ class HttpServlet

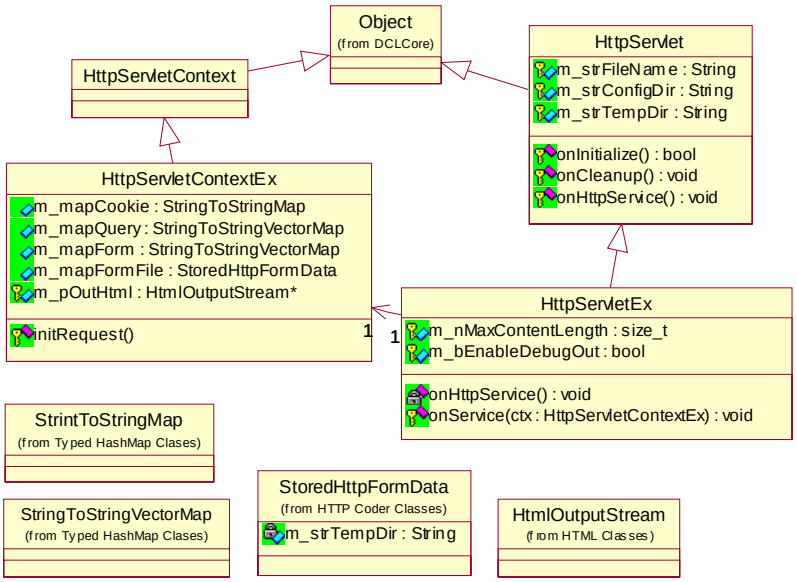
DCLNet을 사용하여 서블릿을 개발할 때에는 HttpServlet 클래스를 상속하여 구현한 클래스가 있어야 하고 실행시간에는 그 클래스의 인스턴스(instance)가 단 한 개 있어야 한다. HttpServlet 클래스는 모든 서블릿 클래스의 추상클래스로 서블릿 구현을 위한 프레임워크를 구성할 수 있도록 한다.

멤버변수 m_strFileName, m_strConfigDir, m_strTempDir은 onInitialize가 호출되기 전에 프레임워크에 의하여 초기화된다.

이 클래스는 세 개의 가상(virtual) 메소드를 가지고 있는데 서블릿은 이들을 오버라이드(override)하여 구현한다. 이들 중 onInitialize는 HttpServlet 객체가 생성된 직후 이를 초기화 하기위해 호출되는데 성공하면 true를 그렇지 않으면 false를 리턴해야 한다. 만약 false를 리턴하게 되면 서블릿의 서비스는 사용되지 않는다.

HTTP_SERVLET_INSTANCE(ServletClass, Description) 매크로는 서블릿의 기능에 접근하도록 하는 인터페이스인 DCL_HTTP_SERVLET과 HttpServlet 클래스

를 연결해 줌과 동시에 서블릿의 DSO 엔트리포인트를 만들어 준다.



<그림 5-1> 서블릿 프레임워크

HTTP_SERVLET_INSTANCE의 ServletClass는 HttpServlet 클래스로부터 상속된 클래스를 말하고 Description은 DCL_HTTP_SERVLET::pszDescription에 대응된다. 다음은 이 매크로의 사용법을 보이는 간단한 예제이다.

```
// MyServlet.cpp
#include <dcl/net/HttpServlet.h>
__DCL_USING_NAMESPACE
class MyServlet : public HttpServlet
{
        …
};
HTTP_SERVLET_INSTANCE(MyServlet, “MyServlet HTTP Service”)
```

■ class HttpServletEx

일반적인 서블릿을 개발하고 할 때에는 이 클래스를 사용한다. 이 클래스의 주 목

적은 __DCL_DEBUG가 정의(define)되어 컴파일된 디버그 버전의 서블릿이 생성한 HTML 스트림에 __DCL_TRACE의 출력을 추가하는 기능이다. 이것은 디버그 버전의 HttpServletEx::onInitialize 내부에서 m_bEnableDebugOut을 true로 설정하고 HttpServletContextEx의 getOutputStream이나 createOuputStream을 최소한 한번 호출하여 HttpServletContextEx의 객체 내부에서 텍스트 콘텐츠를 위한 HtmlOutputStream 객체가 활성화 되었을 때 가능하다.

이 클래스로부터 파생된 클래스는 HttpServletEx의 순수가상함수(pure virtual function)인 onService를 오버라이드 해야 하며 HttpServlet::onHttpService는 사용할 수 없다. 이 클래스의 onHttpService의 구현 내부에서는 onService를 호출하기 전에 HttpServletContextEx::initRequest를 호출하여 HTTP 요청에 관한 파라미터들을 초기화 한다.

m_nMaxContentLength는 POST 데이터의 최대 크기를 나타내며 0(zero)일 경우 제한을 두지 않는다. POST 데이터가 m_nMaxContentLength보다 크게 되면 POST 데이터는 디코드되지 않는다.

HttpServlet::m_strTempDir은 HttpServletContextEx 객체가 생성될 때 사용되며 POST 데이터에 “file”이 있는 경우 이것을 저장할 임시파일의 디렉토리로 사용된다. 만약 이 디렉토리를 바꾸고자 하면 onInitialize를 오버라이드 하여 이 값을 바꾸면 된다.

이 클래스는 생성된 스트림의 CONTENT_TYPE이 “text/html”일 경우 HTML 스트림의 마지막에 “</body></html>”을 추가한다.

■ class HttpServletContext

DCL_HTTP_SERVLET_CONTEXT를 캡슐화하며 서블릿에 HTTP 서비스 실행환경을 제공한다.

■ class HttpServletContextEx

이 클래스는 HttpServletEx와 함께 사용된다. HTTP 요청에 동반한 Cookie,

QUERY_STRING, POST 메소드에 의한 데이터 등은 HttpServletEx::onService가 호출되기 전에 디코드되어 m_mapCookie, m_mapQuery, m_mapForm, m_mapFormFile 멤버에 각각 보관하며, 출력을 위한 HtmlOutputStream 객체를 유지한다.

5.2 HTTP 관련 클래스

■ class URLEncoder, class URLDecoder

URI(RFC2396)에 포함될 수 없는 문자에 대하여 문자열 변환을 한다.

■ class HttpCookieDecoder

HTTP 요청에서 User-Agent로부터 받은 HTTP Request Header중에 Cookie를 디코드하여 StringToStringMap에 보관한다.

■ class HttpQueryStringDecoder

StringToStringVectoMap 객체에 URI의 QUERY_STRING 부분의 문자열을 디코드하여 보관한다. QUERY_STRING에는 같은 이름을 가지는 값을 가질 수도 있고 값이 없는 경우도 있을 수 있다.

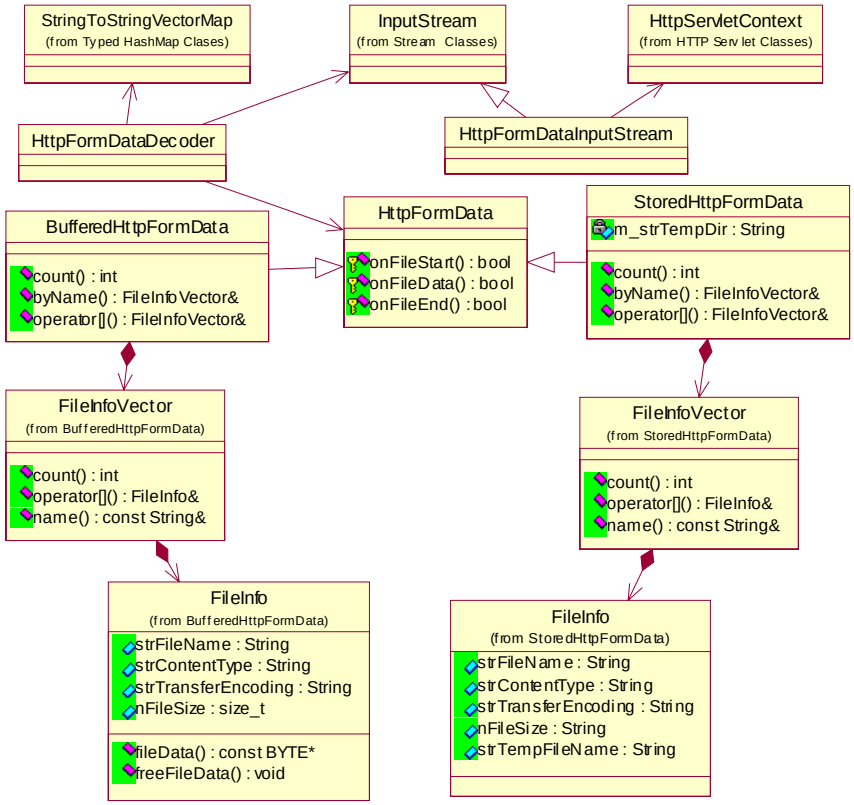
■ class HttpFormData

POST 메소드에 의한 데이터 중 CONTENT_TYPE이 “multipart/form-data” MIME 타입으로 된 데이터에 포함된 파일데이터의 디코드된 결과를 보관하기 위한 콜렉션 클래스이다. 이 클래스의 객체를 적용하여 디코드하면 파일 데이터는 버려지므로 데이터를 사용하려면 이 클래스로부터 파생된 StoredHttpFormData나 BufferedHttpFormData 클래스를 사용하여야 한다.

■ class StoredHttpFormData

“multipart/form-data”에 포함된 파일 데이터를 주어진 디렉토리의 임시파일에 저

장한다. 이들 임시파일은 객체가 파괴될 때 삭제되므로 파일을 계속 사용하려면 다른 디렉토리로 옮기든지 아니면 파일의 이름을 바꾸어야 한다.



<그림 5-2> HTTP POST 데이터의 디코드

HTML의 form 엘리먼트 블록 내부에 올 수 있는 input 엔티티는 동일한 이름이 올 수 있는데 이 때문에 같은 name을 갖는 파일이 둘 이상이 있을 수 있다. StoredHttpFormData 클래스는 이러한 문제를 해결하기 위해 멤버클래스로 구현된 FileInfoVector와 FileInfo를 두고있다. FileInfoVector와 FileInfo를 사용할 때 주의할 점은 그것의 참조를 사용해야 한다는 것이다. 이들 클래스는 사용자에 의하여 객체가 생성되는 것을 허용하지 않기 때문이다. 다음은 이 클래스를 사용하는 간단한

예이다.

```
if (!ctx.m_mapFormFile.isEmpty())
{
        String toDir = File::getDirName(m_strFileName);
        for(int i = 0; i < ctx.m_mapFormFile.count(); i++)
        {
                StoredHttpFormData::FileInfoVector& v = ctx.m_mapFormFile[i];
                out << v.name() << ":";
                for(int j = 0; j < v.count(); j++)
                {
                        StoredHttpFormData::FileInfo& info = v[j];
                        out << "\n\tfilename: " << info.strFileName
                                << "\n\tfilesize: " << info.nFileSize
                                << "\n\tContent-Type: " << info.strContentType
                                << "\n\tContent-Transfer-Encoding: "
                                << info.strTransferEncoding
                                << "\n\ttemp filename: " << info.strTempFileName
                                << "\n";
                        File::rename(
                                info.strTempFileName,
                                File::makeFileName(toDir, info.strFileName)
                                );
                }
        }
}
```

■ class BufferedHttpFormData

StoredHttpFormData와 유사하지만 파일의 데이터를 메모리에 보관한다. 객체가 파괴되기 전에 BufferedHttpFormData::FileInfo::freeFileData 메소드를 호출하여 데이터를 위해 할당된 버퍼를 반환할 수 있다.

■ class HttpFormDataDecoder

“multipart/form-data”의 MIME 타입의 스트림을 디코드한다. form 엘리먼트 블록 내의 input 인라인 엘리먼트 중 type이 “file”인 경우에는 HttpFormData 콜렉션에 저장하고 그 외의 모든 경우는 StringToStringVectorMap에 저장한다.

디코드하는 과정에서 IOException* 예외가 발생 할 수 있으며 이 외의 데이터 에러와 같은 경우의 경고메시지는 HttpFormDataDecoder::warnings을 통해 확인할 수 있다.

■ class HttpFormDataInputStream

HttpFormDataDecoder가 사용하는 스트림을 위하여 HttpServletContext에 대하여 InputStream 인터페이스를 구현한다.

■ class HttpHeader, class HttpSetCookie

HTTP Response-Header를 표현하는 문자열을 생성한다.

■ class StringToStringMap

DCLCore에 포함되어있는 클래스로 해시함수를 사용하여 문자열을 연관짓는다. 이 클래스는 User-Agent로부터 전송 받은 HTTP Request-Header에서 Cookie의 디코드된 결과를 저장하는데 사용된다.

■ class StringToStringVectorMap

하나의 이름에 하나 이상의 문자열을 포함하는 StringVector를 연관짓는다. 이 클래스는 QUERY_STRING과 FORM_DATA의 디코드된 결과를 저장하는데 사용된다.

다음은 QUERY_STRING을 디코드하는 예제이다.

```
StringToStringVectorMap map;
HttpQueryStringDecoder::decode(
        map,
        “a=1&b=1&a=3&a=4&a=&b=2”
        );
out << “map[“\a\”][0] = “ << map[“a”][0]
        << “\nmap[“\a\”][1] = “ << map[“a”][1]
        << “\nmap[“\a\”][2] = “ << map[“a”][2]
        << “\nmap[“\a\”][3] = “ << map[“a”][3]
        << “\nmap[“\b\”][0] = “ << map[“b”][0]
        << “\nmap[“\b\”][1] = “ << map[“b”][1]
```

```
        << "\n";
```

결과

```
map["a"][0] = 1
map["a"][1] = 3
map["a"][2] = 4
map["a"][3] =
map["b"][0] = 1
map["b"][1] = 2
```

5.3 HTML 관련 클래스

■ class HtmlEntityEncoder

HTML Entity Sets에 정의된 문자들에 대하여 문자들을 변환하여 문자열을 만들거나 OutputStream에 출력한다. 이들의 대표적인 문자는 '>', '<', '&', ' ' 등이 있는데 이들을 각각 ">", "<", "&", " "로 변환된다.

■ class HtmlOutputStream

DCLCore의 ByteBufferOutputStream으로부터 파생된 클래스로 출력데이터에 포함되어 있는 LF(line-feed, ASCII 10)를 CRLF(ASCII 13, ASCII 10)으로 변환한다. 서블릿은 HttpServletContext::write를 호출하여 User-Agent에 데이터를 전송할 수 있지만, 작은 HTML 조각을 HttpServletContext::write를 통하여 직접 출력하는 것은 효율적이지 못하다. HtmlOutputStream 클래스는 HTML 조각들을 위한 스트림 버퍼를 유지한다.

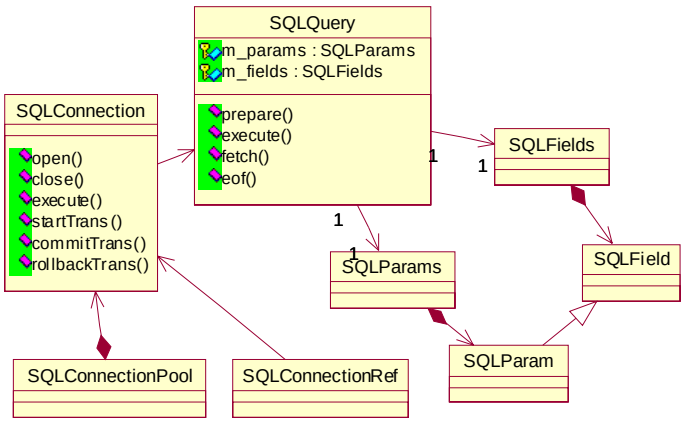
■ class HtmlTemplate

이 클래스는 HTML 페이지를 표현하는데 있어서 디자인 부분과 프로그램에 의하여 변화하는 데이터의 분리를 가능하게 해주는 클래스이다[19]. 이 클래스는 DCL의 데이터베이스 관련 클래스와 함께 사용될 수 있는데 사용에 대한 예는 5.4에서

소개한다.

5.4 데이터베이스 연결성

DCL은 C++의 추상클래스(abstract class)를 인터페이스로 하여 어플리케이션의 실행시간에 표준화된 방법을 통하여 서로 다른 DBMS에 접근할 수 있도록 하는 방법을 제공하는데 이것이 DDBC(DCL Database Connectivity)이다[18]. 본 절에서는 데이터베이스 어플리케이션을 위한 클래스들에 대하여 간략히 요약하고 데이터베이스와 HtmlTemplate 사용에 대한 간단한 예를 보인다.



<그림 5-3> DCLCore의 데이터베이스 클래스

■ class SQLConnection

클래스의 생성자에서 명시한 드라이버로부터 연결 객체를 얻어와서 이를 추상화한다. 객체가 성공적으로 구성이 되면 open 메소드를 사용하여 서버에 연결하고 close로 연결을 닫는다. 이 클래스는 서버와의 연결 세션을 유지하는 것 외에 트랜잭션을 관리하고 파라미터와 리턴 값이 없는 INSERT, UPDATE, DELETE 문과 DDL(Database Definition Language)을 실행할 수 있다.

■ class SQLConnectionPool

다중 스레드 어플리케이션에서 동일한 드라이버와 연결문자열을 갖는 연결객체들의 풀(pool)을 유지한다. SQLConnection 객체는 둘 이상의 스레드에서 공유될 수 없기 때문에 다중의 스레드가 같은 서버에 연결하도록 하려면 각 스레드마다 새로운 연결을 만들어야 한다. 또한, 사용이 완료된 연결은 시스템 자원을 위해서 연결을 닫을 수도 있고 나중에 다시 연결하는 오버헤드를 피하기 위하여 유휴(idle)연결 상태를 유지하고 있을 수도 있다. 이 클래스는 이러한 일련의 처리를 자동화 한다.

■ class SQLConnectionRef

SQLConnectionPool::getConnection을 사용하여 얻은 연결객체는 사용이 완료된 후에는 반환하여야 한다. 이 클래스의 주요 사용목적은 C++의 예외 메커니즘과 관련이 있다. 어플리케이션의 실행 중에 예외(Exception) 발생할 경우 컴파일러는 예외 핸들러(handler)를 찾기까지 스택에 있는 객체를 파괴한다. C++에는 자바 언어의 finally[5]와 같은 방법을 제공하지 않는데 연결객체를 얻어온 위치에서 이 클래스를 사용하게 되면 예외가 발생했을 경우 SQLConnectionPool에 연결객체를 정상적으로 반환할 수 있다.

■ class SQLQuery

SELECT, INSERT, UPDATE, DELETE, EXECUTE PROCEDURE, Oracle의 PL/SQL과 같은 대부분의 질의를 사용할 때 사용하며 prepare, execute, fetch 세 가지의 주요 멤버함수와 파라미터(SQLParam) 및 리턴된 레코드의 필드(SQLField)에 대한 컬렉션을 유지한다. prepare는 한번 준비된(prepared) SQL에 서로 다른 파라미터를 적용하여 실행(execute)할 수 있도록 하는 메커니즘을 제공하고 fetch를 사용하여 하나의 레코드를 사용할 수 있도록 한다.

■ class SQLField

[5] Java, Object Pascal, C#과 같은 언어의 finally 블록은 예외가 발생하든 그렇지 않든 반드시 실행되어야 하는 코드를 기술할 수 있다.

필드 데이터에 대한 구체적인 접근 방법을 제공한다.

■ class SQLParam

SQLField로부터 파생된 클래스로 파라미터를 설정하고 값에 대한 접근을 제공하는 클래스이다.

■ class SQLFields, class SQLParams

SQLQuery 객체의 멤버이고 각각 SQLField, SQLParam 객체에 대한 컬렉션을 유지한다.

■ HtmlTemplate를 적용한 데이터베이스 예제

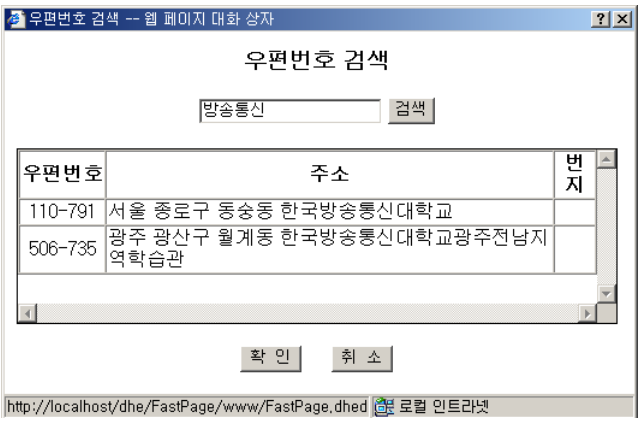
HtmlTemplate는 실행시간에 주어진 템플릿 파일을 파싱(parsing)하여 객체가 구성된다. <그림 5-4>는 우편번호를 검색한 결과를 보여주기 위한 HTML 템플릿으로 전체 템플릿과 "<!—BEGIN ZIPCODE -->"와 "<!—END ZIPCODE -->" 블록의 서브 템플릿으로 구성된다.

```
62: <table id="zipcodeTable" border=1 cellspacing=0 style="width:100%; text-align:center">
63: <col span="1" width="15%" style="text-align:center"/>
64: <tr>
65:         <th>우편번호</th>
66:         <th>주소</th>
67:         <th>번지</th>
68: </tr>
69: <!-- BEGIN ZIPCODE -->
70: <tr onclick="selectItem(this);">
71:         <td>$ZIPCODE</td>
72:         <td style="text-align:left">$ADDRESS</td>
73:         <td>$BUNJI</td>
74: </tr>
75: <!-- END ZIPCODE -->
76: </table>
```

<그림 5-4> 템플릿 파일의 예: zipcode_search.html

```
534:        HtmlTemplate& ZIPCODE = (*this)["ZIPCODE"];
535:        SQLQuery& q = m_pOwner->session()->m_query;
536:        q.execute(
537:            "SELECT ZIPCODE"
538:            ", CONCAT(SIDO, ' ', GUGUN, ' ', DONG) AS ADDRESS"
539:            ", BUNJI\n"
540:            "FROM DCLWC_ZIPCODE "
541:            "WHERE DONG LIKE '%" + (*it).value[0] + "%'"
542:            );
543:        q.fetch();
544:        if (q.eof())
545:            erase("ZIPCODE");
546:        else
547:        {
548:            while(!q.eof())
549:            {
550:                ZIPCODE.assign(q.fields(), " ");
551:                append("ZIPCODE", ZIPCODE);
552:                q.fetch();
553:            }
554:        }
```

<그림 5-5> 템플릿과 데이터베이스 질의: zipcode_search.cpp



<그림 5-6> HtmlTemplate을 적용한 데이터베이스 질의결과

<그림 5-6>은 <그림 5-4>의 템플릿과 <그림 5-5>의 C++ 프로그램 코드가 포함된 서블릿을 실행한 결과이다. 템플릿의 $ZIPCODE, $ADDRESS, $BUNJI는 질의

결과 레코드에서 각각의 필드에 대응된다.

PHP, ASP, JSP와 같은 기존의 스크립트 기술은 HTML과 서버의 프로그램을 단일의 파일로 작성할 수 있도록 하는 장점이 있기는 하지만 다른 한편으로는 하나의 페이지를 위하여 개발자가 모든 것을 알아야 하기 때문에 인력의 배치를 어렵게 하고 생산성을 떨어뜨리게 할 수도 있다.

이 예제에서 주목하여야 할 점은 동적인 웹 페이지를 구성하는데 있어서 보여주는 부분과 서버 프로그램을 완벽히 분리했다는 사실이다. 이렇게 함으로써 디자인 전문가는 보여주는 부분만 관여를 하고 프로그래머는 서버 프로그램을 위하여 집중할 수 있도록 할 수 있기 때문에 결과적으로 생산성은 향상된다. 또한, 서버 프로그램이 변경되지 않는 상태에서도 템플릿의 변경이 용이하므로 유지보수 비용도 절감된다.

5.5 서블릿의 실행시간 디버깅

스크립트 언어를 사용하든 컴파일러 언어를 사용하든 동적인 HTML 페이지를 생성하는 프로그램을 작성하는데 있어서 어려움을 겪게 되는 것은 어떠한 코드가 제대로 실행되었는지, 또는 그 코드를 실행하기 위한 조건이 제대로 이루어 졌는지에 대하여 알기 위한 방법이 쉽지 않다는 것이다. 이러한 이유는 스크립트의 실행이 서버에서 이루어지고 웹 브라우저에 나타난 것은 그것의 결과이기 때문이다.

결국엔 스크립트의 실행 중간에 변수들의 값을 확인하는 출력 함수를 삽입하게 되고 스크립트가 완성되면 이러한 부분들을 제거하게 된다. 그러나 이러한 작업은 매우 번거로울 뿐만 아니라 복잡한 페이지의 경우 이 값들의 위치를 찾아내기 위해 헤매는 해프닝의 원인이 된다.

DCL은 소스코드 차원에서 디버깅을 위한 코드 삽입을 자동화 하여 소프트웨어 개발 및 테스트 단계에서 필요한 정보를 생성하도록 하는 실행시간 환경을 제공한다 [16]. 이것을 사용하면 별도의 디버거를 사용하지 않아도 프로그램의 실행에 관련된 변수들의 값을 쉽게 확인할 수 있을 뿐만 아니라 동일한 소스코드를 릴리즈로 빌드하게 되면 디버깅을 위해 소스에 삽입되었던 코드들은 제거되는데, 이는 DHE 서블

릿의 개발에서도 사용할 수 있다.

5.5.1 AssertException

DCL의 __DCL_ASSERT의 동작은 두 가지로 지정할 수 있다. 첫 번째는 표준 C 언어 런타임 라이브러리에 포함되어 있는 assert처럼 표현식에 대한 문자열을 출력하고 abort 시스템 호출을 사용하여 프로세스를 종료하는 것이다. 그러나 이 형태는 서블릿에 적용될 수 없다. 왜냐하면 서블릿은 이미 웹 서버 프로세스의 일부분이기 때문에 서블릿에서 abort 해 버리면 웹 서버의 서비스가 중단되기 때문이다

두 번째는 AssertException 예외를 던지는 형태이다. 서블릿에서 발생한 모든 __DCL_ASSERT 실패는 abort 하지 않고 AssertException 예외를 발생한다. 이렇게 함으로써 서블릿의 실행 중에 예외가 발생하더라도 웹 서버의 프로세스 실행에는 영향을 주지 않는다.

디버그 버전의 서블릿에서 __DCL_ASSERT를 실패하면 AssertException 예외가 발생한다.

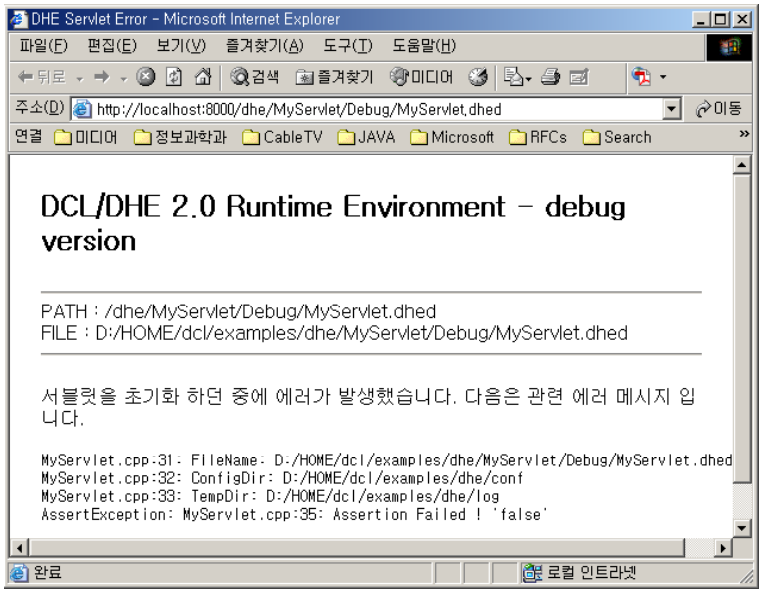
5.5.2 서블릿 컨테이너의 에러표시

DCL_HTTP_SERVLET 인터페이스의 메소드는 성공하면 TRUE를 그렇지 않으면 FALSE를 리턴하도록 되어 있다. 만약 실패할 경우 전달받은 에러 스트림(hReportError)핸들을 사용하여 에러 메시지를 되돌릴 수 있는데 pfnInitialize와 pfnHttpService 메소드에서 되돌린 메시지는 서블릿 컨테이너에 의하여 User-Agent로 전달된다. 만약 User-Agent가 웹 브라우저라면 사용자는 이에 대한 메시지를 볼 수 있게 된다.

HttpServlet 클래스는 이를 이용하고 있는데 디버그 버전의 서블릿은 예외(exception)가 발생했을 때 예외객체의 메시지와 더불어 멤버 함수들의 실행과정에서 사용한 __DCL_TRACE의 내용을 서블릿 컨테이너에 전달한다.

```
23:  bool MyServlet::onInitialize() __DCL_THROWS1(Exception*)
24:  {
25:  #ifdef __DCL_DEBUG
26:      // HttpServletEx 의 디버깅 정보 출력을 불가능하게 한다.
27:      // 기본값은 true 이다.
28:      // m_bEnableDebugOut = false;
29:  #endif
30:      __DCL_TRACE1("FileName: %s\n", m_strFileName.cstr());
31:      __DCL_TRACE1("ConfigDir: %s\n", m_strConfigDir.cstr());
32:      __DCL_TRACE1("TempDir: %s\n", m_strTempDir.cstr());
33:
34:  //      __DCL_ASSERT(false);
35:
36:      return true;
37:  }
```

<그림 5-7> MyServlet.cpp



<그림 5-8> 서블릿 컨테이너에 의한 에러표시

<그림 5-7>의 소스 프로그램은 디버그 버전에서 34번 라인의 주석을 제거하면 AssertException이 발생하게 되고 초기화는 실패하게 된다. <그림 5-8>는 이에 대

한 결과를 보여 주고 있다. 만약 34번 라인을 주석처리 하고 36번 라인에서 false를 리턴하게 되면 그림에서 예외가 발생한 부분만 제외된다.

서블릿 컨테이너에 의한 결과는 엄격히 따지면 디버깅을 위한 목적은 아니다. 따라서 이것은 서블릿이 디버그 버전이든 릴리즈 버전이든 상관이 없을 뿐만 아니라 DCL의 사용 여부와도 상관이 없고 서블릿의 개발에 사용된 프로그래밍 언어와도 상관이 없다.

5.5.3 HttpServletEx의 디버깅 정보 표시

HttpServletEx 클래스를 상속하여 서블릿을 구현하면 서블릿의 실행결과뿐만 아니라 __DCL_TRACE가 출력한 내용과 반환되지 않은 동적 메모리의 할당내용이 함께 출력된다. 이것은 서블릿이 생성한 콘텐츠의 MIME 타입이 “text/*”인 상태에서 HttpServletContextEx가 제공하는 스트림 객체를 사용했을 때 가능하다.

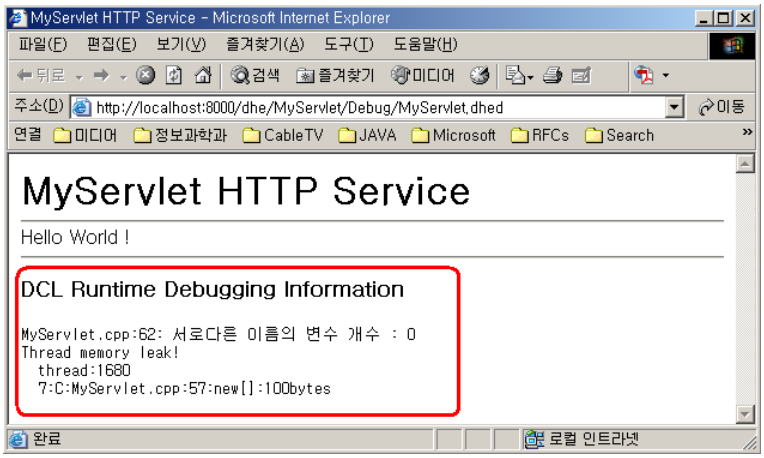
HttpServletContextEx의 스트림 객체를 사용하는 방법은 두 가지 이다. 첫 번째는 이 클래스의 getOutputStream 메소드를 호출하여 디폴트 파라미터(default parameter)를 사용하여 객체를 구성하는 것인데 HttpServletContextEx 내부에 스트림이 구성되어 있으면 그것의 참조를 리턴하고 그렇지 않으면 디폴트 파라미터를 사용하여 HtmlOutputStream를 생성한 후 이것의 참조를 리턴한다. 만약 스트림 객체의 구성을 조정하려면 createOutputStream 메소드를 사용한다.

<그림 5-10>은 <그림 5-9>코드를 포함하고 있는 서블릿을 실행한 결과이다. 57 라인은 new[] 연산자를 사용하여 메모리 블록을 할당 했는데 이를 반환하지 않았다는 내용과 62 라인의 __DCL_TRACE의 내용을 표시하고 있다.

HttpServletEx는 onService에서 발생한 예외에 대해서는 표시해 주지 않는다. 예외에 대한 처리는 HttpServlet의 기본 구현에 위임을 하게 되는데 이것은 예외의 발생이 엄격한 의미에서 서블릿의 실행이 실패한 것으로 볼 수 있기 때문이다.

```
40:  void MyServlet::onService(HttpServletContextEx& ctx)
41:                              __DCL_THROWS1(Exception*)
42:  {
43:      ctx.setContentType("text/html", "euc-kr");
44:
45:      // 디폴드 HtmlOutputStream 의 디폴트 파라미터를 사용하여
46:      // 스트림 객체를 구성한다.
47:      HtmlOutputStream& out = ctx.getOutputStream();
48:
49:      out << "<html>\n"
50:          "<head><title>MyServlet HTTP Service</title></head>\n"
51:          "<body>\n"
52:          "<h1>MyServlet HTTP Service</h1>\n"
53:          "<hr/>\n"
54:          "<p>Hello World !</p>\n";
55:
56:      // 해제하지 않은 메모리 블록
57:      char* p = new char[100];
58:
59:      __DCL_TRACE1(
60:          "서로다른 이름의 변수 개수 : %d\n",
61:          ctx.m_mapQuery.count()
62:          );
63:  }
```

<그림 5-9> MyServlet.cpp



<그림 5-10> HttpServletEx의 디버깅 정보 표시

제6장 성능평가

6.1 하드웨어 구성

성능평가를 위한 컴퓨터는 클라이언트를 위하여 Microsoft Windows 2000 Server가 설치된 컴퓨터 1대와 서버 역할을 위한 Microsoft Windows 2000 Server와 GNU/Linux가 각각 설치된 2대의 컴퓨터를 사용하였고 이들은 100Mbps Fast Ethernet으로 구성된 스위칭 허브로 연결하였다

<표 6-1> 성능평가를 위한 하드웨어 구성

구분	항목	내용
Client	OS	Microsoft Windows 2000 Server Service Pack 4
	CPU	Intel Pentium-4 1400Mhz
	RAM	512Mbytes
	HDD	Seagate ST340823A 40G 5400RPM
	NIC	Realtek RTL8139, 100Mbps
Server 1 (Windows)	OS	Microsoft Windows 2000 Server Service Pack 4
	CPU	Intel Pentium-4 1500Mhz
	RAM	512Mbytes
	HDD	Seagate ST3200822A 200G 7200RPM
	NIC	3Com 3C905B, 100Mbps
Server 2 (GNU/Linux)	OS	RedHat 7.2 linux-kernel 2.4.18
	CPU	Intel Pentium-III 850Mhz, 256 KB Cache
	RAM	512Mbytes
	HDD	Seagate ST340016A 40G 7200RPM
	NIC	3Com 3C905B, 100Mbps
Network	-	100Mbps Fast Ethernet Switching-Hub

6.2 소프트웨어 구성

6.2.1 클라이언트 소프트웨어

웹 서버의 성능평가를 위한 클라이언트 소프트웨어는 WebStone[6], httperf[7], 그리고 ApacheBench와 같이 여러 종류가 있으나, 본 성능평가에서는 Apache 서버에 포함된 ApacheBench를 사용하였다. 이 소프트웨어를 선택하게 된 가장 큰 이유는 성능평가를 위한 클라이언트 운영체제가 Windows 뿐만 아니라 Linux도 사용되었고 이들 운영체제의 바이너리 배포파일을 쉽게 얻을 수 있기 때문이다.

ApacheBench는 HTTP 요청의 개수, 동시 요청의 개수, Keep-Alive의 지정 등 여러 종류의 옵션을 주어 웹 서버의 성능 측정을 할 수 있다.

6.2.2 시스템 모니터링 소프트웨어

서버쪽 시스템 모니터링에 있어서 가장 중요한 요소는 서버 소프트웨어가 사용하는 메모리의 크기에 관한 부분이다. Microsoft Windows 2000 Server의 경우 MMC(Microsoft Management Console)의 성능모니터(perfmon.msc) 스냅인을 사용하여 측정하였고, GNU/Linux의 경우 ps(process status)명령을 사용했다.

GNU/Linux의 ps는 /proc 파일 시스템으로부터 프로세스 정보를 얻어 오는데 예전 버전의 경우 다중 스레드 어플리케이션에 대하여 커널이 각각의 스레드에 별도로 할당한 pid를 사용하여 정보를 수집하기 때문에 프로세스별 메모리 사용량에 대하여 정확하게 알 수 없다. 최근 버전은 이러한 문제가 해결되었으며 본 성능 평가에서는 2.0.11 버전의 ps를 사용하였고 1초를 주기로 프로세스의 상태를 확인하도록 쉘-스크립트를 작성하였다.

---

[6] http://www.mindcraft.com/webstone/
[7] http://www.hpl.hp.com/research/linux/httperf/

6.2.3 HTTP 서버의 설치와 구성

■ Apache

Apache 웹 서버는 Windows와 Linux 운영체제 모두에 설치하였다. Windows의 경우는 2.0.53버전의 바이너리 배포 패키지를 설치하였고 Linux에서는 RedHat 7.2 Linux의 배포 패키지에 포함된 1.3.23버전과 2.0.52 소스 패키지를 컴파일하여 worker MPM을 사용하여 설정하였다

■ IIS 5.0와 ASP, ASP.NET

Windows 2000 Server에서 IIS 5.0은 초기설치의 설정을 그대로 사용하였는데 성능 조정과 관련한 예상되는 하루 방문 수의 초기 값은 100,000미만이다. 성능 측정을 위한 응용프로그램 보호 설정은 낮음으로 하여 IIS 프로세스 내에서 실행되도록 하였다.

ASP의 실행환경은 IIS 5.0이 설치될 때 함께 설치된다. ASP.NET을 위한 .NET Framework는 별도로 설치했는데 그 버전은 1.1.4322이다.

■ Tomcat 5.5.9

자바 서블릿/JSP 컨테이너는 Apache Software Foundation의 Tomcat을 사용하였으며 여기에 사용된 JRE(Java Runtime Environment)는 1.4.02 버전이다.

■ PHP

PHP는 Linux의 Apache 2.0.52 버전을 제외하고 Windows와 Linux의 Apache 서버에 모두 설정하였다. UNIX에서 PHP는 다중 스레드에 안전하지 않기 때문에 다중 스레드 방식으로 동작하는 Apache 2.x의 worker MPM에 사용될 수 없다. Windows 2000 Server에서 사용된 PHP 버전은 5.1.0b2이고 Linux에서는 4.1.2 이다.

■ DHE

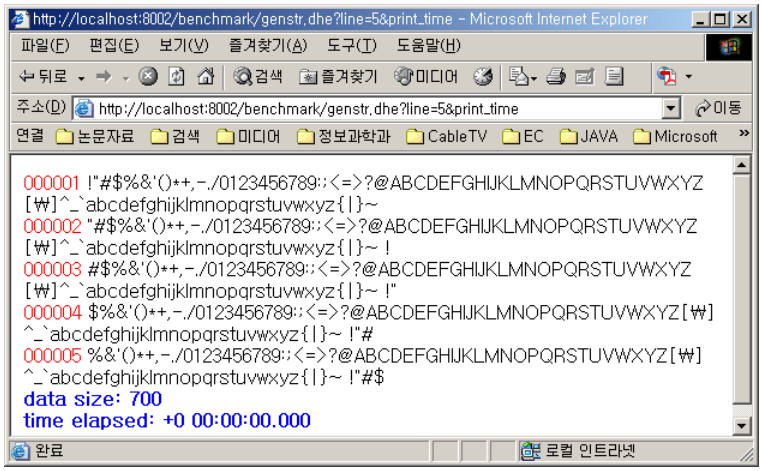
DHE의 설정은 Windows와 Linux에서 IIS 5.0, Apache 1,3.x, Apache 2.x 모두에 릴리즈 빌드를 사용 하였다.

<표 6-2> HTTP 서버의 설치와 구성

운영체제	서버 소프트웨어	설정내용	
Windows 2000 Server	apache 2.0.53	ServerType # WinNT MPM ThreadsPerChild MaxRequestsPerChild PHP 5	standalone 150 0
	Tomcat 5.5.9	maxThreads minSpareThreads maxSpareThreads	150 25 75
	IIS 5.0	예상되는 하루 방문 수 응용프로그램 보호 .NET Framework 1.1	100,000미만 낮음(IIS 프로세스)
GNU/Linux	apache 1.3.23	ServerType MinSpareServers MaxSpareServers StartServers MaxClients MaxRequestsPerChild PHP 4	standalone 5 20 8 150 1000
	apache 2.0.52	ServerType # worker MPM StartServers MaxClients MinSpareThreads MaxSpareThreads ThreadsPerChild MaxRequestsPerChild	standalone 2 150 25 75 25 0

6.2.4 서버 어플리케이션

서버 어플리케이션 기술들 간의 비교를 위하여 PHP, ASP, ASP.NET, JSP 스크립트와 DHE 서블릿을 작성 하였다[부록-2]. 각각은 그것을 작성하는 언어적 특성이 서로 다르기 때문에 동일하게 작성할 수는 없으나 문자열 연산을 중심으로 같은 알고리즘이 되도록 하였다. Microsoft의 ASP와 ASP.NET은 다양한 언어를 사용할 수 있는데 본 실험에서는 VBScript와 C#을 각각 사용했다.



<그림 6-1> 서버 어플리케이션의 실행결과

모든 서버 어플리케이션은 HTTP GET 메소드를 사용하여 호출 되며 QUERY_STRING을 통하여 공통적으로 line, print_time, no_line 파라미터를 지정할 수 있다. line 값은 어플리케이션이 동적으로 생성하는 문자열의 개수를 의미하고 각 문자열은 라인번호를 포함하는 ASCII 32에서 126까지의 문자로 이루어진다. print_time 파라미터를 지정하면 생성된 콘텐츠의 마지막에 소요된 시간이 첨부된다. ASP의 경우 시간을 얻기 위한 함수의 최대 정밀도는 초(seconds)단위 이기 때문에 별도의 ActiveX를 개발하여 ms(milliseconds)까지의 값을 얻었다.

no_line 파라미터는 생성된 문자열을 출력하지 않는다. 이것은 서버 어플리케이션의 언어적 특성에 관한 평가를 위해 사용된다. 일반적으로 동적인 콘텐츠를 생성하

는 기술들의 출력함수는 생성된 콘텐츠에 대하여 버퍼링을 하지만 생성된 콘텐츠의 크기가 주어진 버퍼보다 크게 되면 실행 중간에 클라이언트로 전송할 수도 있기 때문에 네트워크 트래픽에 관한 시간이 포함될 수 있다. 따라서 순수한 언어적 특성을 정확히 측정하기 위해서는 어플리케이션의 실행 중간에 출력함수를 사용하지 않아야 한다.

동일한 line 값을 지정하면 PHP, ASP, ASP.NET, JSP, DHE가 생성하는 HTML의 크기는 동일하다. 다만, HTTP 서버마다 HTTP Response-Header를 생성하는 것은 임의로 제어할 수 없기 때문에 전송되는 전체 바이트의 개수는 서버마다 다를 수 있다.

6.3 실험과 결과분석

6.3.1 성능측정의 초기상태

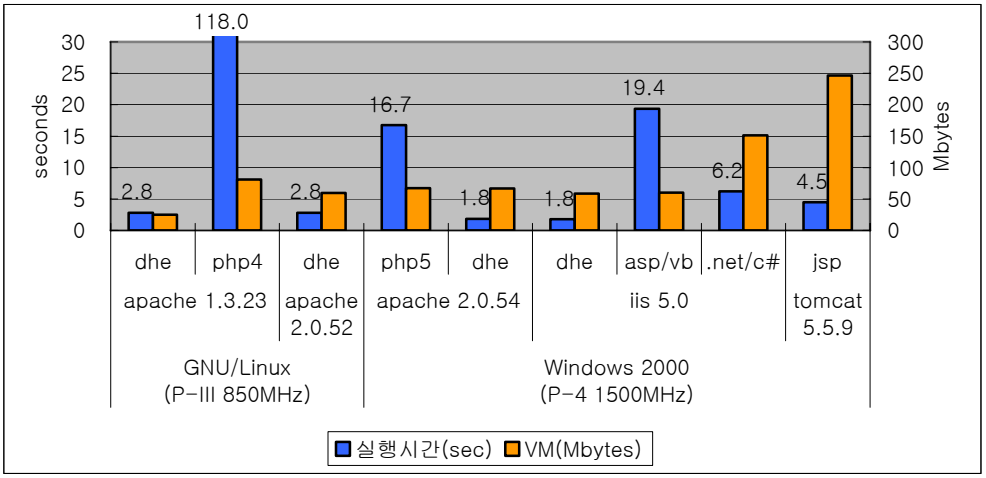
평가를 위한 성능측정은 웹 서버 프로세스를 시작한 후 서버와 클라이언트 컴퓨터의 CPU 사용이 0(zero)에 근접하여 안정된 상태에서 시작 하였다. 메모리 사용에 관한 측정을 위해서는 동일한 웹 서버를 사용할 경우 서로 다른 동적 콘텐츠 생성을 위한 플러그인 모듈은 제거하였다. 예를 들어 Apache 서버에서 PHP의 성능을 측정할 때에는 DHE 서블릿 컨테이너를 제거 하였고 DHE를 측정할 때에는 PHP 플러그인 모듈을 제거 하였다.

ASP.NET과 JSP의 경우 스크립트가 변경된 후 최초의 호출에는 스크립트를 컴파일하는 과정이 추가되기 때문에 웹 브라우저를 통해 최초 호출을 하여 성공된 상태를 확인하였다.

6.3.2 단일 호출에서의 시스템 자원사용

단일 호출에서의 시스템 자원 사용에 대한 측정은 서버 어플리케이션의 언어적 특성을 반영한다. 동일한 알고리즘에서의 실행시간은 같은 결과를 얻기 위한 CPU 자

원의 사용량으로서 간주될 수 있다.



<그림 6-2> 10만 라인 생성에 소요된 시간과 가상메모리

이것에 위한 측정은 네트워크 트래픽이나 웹 서버의 HTTP 프로토콜의 처리와 같은 부가적인 오버헤드를 제거하기 위해 서버 어플리케이션의 호출에서 no_line과 print_time 파라미터를 주어 결과를 측정하였다. 클라이언트 User-Agent는 웹 브라우저를 사용했으며 웹 브라우저에 표시되는 내용은 10만 라인에 해당하는 데이터의 크기와 소요된 시간만 표시된다.

실험에서 VM은 요청을 처리한 프로세스의 가상메모리(VM: Virtual Memory)의 크기로 Apache 서버와 Tomcat 5.5에서의 부모 프로세스, GNU/Linux에서 Apache 1.3의 유휴(idle) 프로세스가 사용한 가상메모리는 포함하고 있지 않으며, ASP.NET의 경우는 inetinfo.exe를 제외한 aspnet_wp.exe이 사용한 가상메모리의 크기이고, DHE와 ASP는 IIS의 응용프로그램 보호를 "낮음"으로 했기 때문에 inetinfo.exe의 가상메모리의 크기이다. 프로세스가 사용한 가상메모리의 측정에 대하여는 단위시간당 처리된 요청의 개수의 측정에서도 동일하다.

<그림 6-2>는 서버 어플리케이션 각각을 3번 실행하여 얻은 수치의 평균을 표시한 것이다. 같은 CPU를 사용하는 시스템에서의 DHE의 결과는 비슷한 결과를 보이고 있고, DHE외에 다음으로 가장 빠른 것은 Tomcat 5.5.9의 JSP이나 DHE와 비교

하여 2.4배 이상의 시간이 소요 되었다.

6.3.3 단위시간당 처리된 요청의 개수

웹 서버의 실질적인 성능은 단위 시간당 얼마나 많은 요청을 처리할 수 있는가에 달려 있다[20]. 이를 위하여 사용된 성능측정 도구로 ApacheBench를 사용하였다. ApacheBench의 –k 옵션은 서버와의 연결에서 HTTP Keep-Alive 사용하도록 할 수 있는데 본 측정에서는 제외하고 전체 요청의 개수와 동시연결의 개수만을 지정하였고, 서버 어플리케이션이 생성할 콘텐츠의 크기를 위하여 line 파라미터를 지정하였다.

<표 6-3> ApacheBench의 사용

요청의 개수	동시요청의 개수	line 파라미터	바이트수/1요청
640	64	20	2,964
640	64	200	28,525
640	64	2,000	284,126

DHE 서블릿은 서블릿 컨테이너로 하여금 강제로 캐시되도록 한 것과 그렇지 않은 것을 별도로 설정하여 실험을 하였으며 실험결과에서 “dhe(c)”로 표기되어 있는 것은 강제로 캐시된 것을 의미한다.

<표 6-4>는 서버의 프로세스가 사용한 가상메모리를 측정한 것이다. Windows 2000에서 각각의 웹 서버는 다중 프로세스 구조라 할지라도 실제 서비스는 단일 프로세스/다중 스레드 방식으로 서비스 하도록 하여 실험을 했기 때문에 서비스를 한 프로세스의 가상메모리만을 측정한 것이지만, GNU/Linux에서 Apache 서버의 경우는 다중 프로세스로 서비스를 하기 때문에 서비스에 관여된 모든 작업 프로세스가 사용한 가상메모리의 합이다.

<표 6-4>에서 보면 DHE를 사용할 때가 가장 적은 가상메모리를 사용하고 있는 것을 알 수 있다. 표에서 보면 서버 프로세스가 사용하고 있는 가상메모리의 크기는

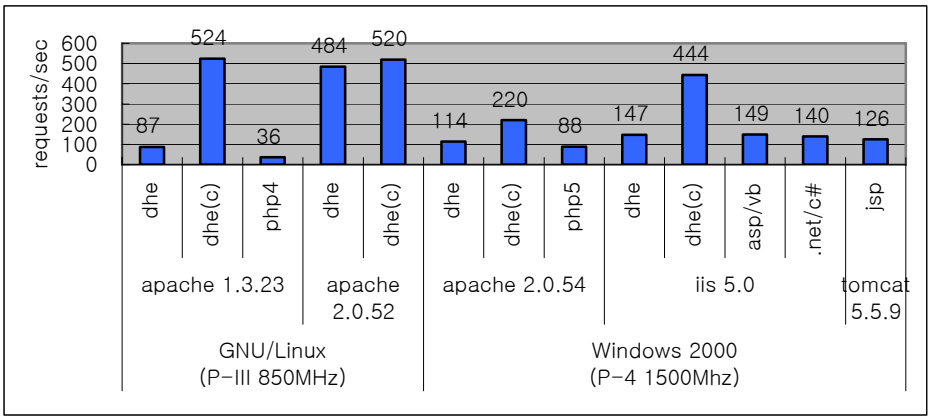
서버의 다중 스레드 방식의 사용여부에 큰 영향을 받고 있다. 그러나 Windows 2000에서의 경우는 실험을 한 모든 서버가 다중 스레드 방식임에도 불구하고 Tomcat 5.5.9의 가상메모리 사용이 가장 많았다. 이것은 JVM을 사용하는 기술이 여전히 많은 메모리 자원을 필요로 한다는 것을 보여주는 결과이다.

GNU/Linux에서 Apache 2.0.52 버전은 다중 스레드 방식을 적용했기 때문에 Apache 1.3.23보다 적은 가상메모리를 사용하고 있다. 그러나 <그림 6-3>, <그림 6-4>, <그림 6-5>의 단위 시간당 처리된 요청의 개수를 보게 되면 강제 캐시설정을 한 dhe(c)에서 다중 프로세스 방식만을 사용하고 있는 Apache 1.3.23의 성능이 더 나은 결과를 얻었다. 이것은 가상메모리의 크기만을 단순히 비교하여 다중 스레드 방식이 가용성이 높다고 판단할 수 없음을 보여준다.

<표 6-4> ApacheBench 실험에서 서버의 가상메모리 사용

운영체제	웹 서버	어플리케이션 기술	가상메모리(단위: Mbytes)		
			20라인	200라인	2,000라인
GNU/Linux (P-III 850MHz)	Apache 1.3.23	dhe	736	1,060	1,074
		dhe(c)	736	1,060	1,074
		php4	6,137	7,025	7,671
	Apache 2.0.52	dhe	186	214	230
		dhe(c)	186	214	230
Windows 2000 (P-4 1500Mhz)	Apache 2.0.54	dhe	55	55	55
		dhe(c)	55	55	55
		php5	72	72	72
	IIS 5.0	dhe	44	57	63
		dhe(c)	44	57	63
		asp/vb	65	65	73
		.net/c#	152	152	155
	Tomcat 5.5.9	jsp	296	296	296

<그림 6-3>, <그림 6-4>, <그림 6-5>는 단위 시간당 처리된 HTTP 요청의 개수를 표시하고 있다. 이들 세 그림에서 강제 캐시설정을 한 DHE 서블릿의 실험은 PHP, ASP, ASP.NET, JSP에 비하여 월등히 우수한 결과를 보이고 있고, 기본 캐시설정 상태에서는 20라인을 생성하는 실험을 제외한 모든 결과가 다른 기술들 보다 나은 결과를 보여준다.



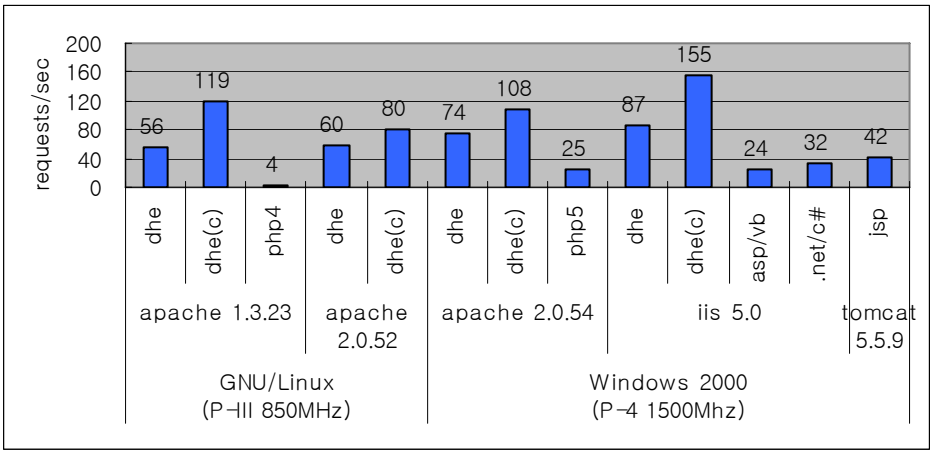
<그림 6-3> ApacheBench 실험결과: 20라인(2,964Bytes)

20 라인을 생성하는 <그림 6-3>의 결과에서 기본 캐시설정의 DHE는 다른 서버 어플리케이션에 비하여 두드러진 성능을 보이고 있지 않고 오히려 IIS 5.0에서는 ASP보다 떨어지는 것을 볼 수 있는데, 이것은 각 요청에 있어서 서블릿의 실제 실행시간보다 서블릿을 적재하고 제거하는 시간이 많이 포함되어 있기 때문이다. 이러한 결과는 C#으로 작성되어 DSO로 컴파일되는 ASP.NET의 경우도 마찬가지이다.

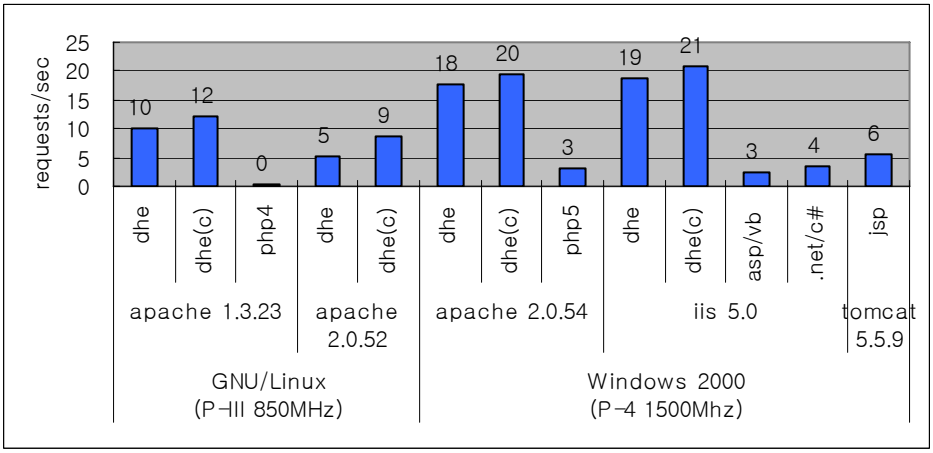
기본 캐시설정에서 최악의 경우는 GNU/Linux에서 다중 프로세스 프로세싱 모델을 사용하고 있는 Apache 1.3.23버전으로 여기에서 모든 DHE 서블릿은 HTTP 요청마다 적재되고 제거된다. 반면에 혼합형 다중 프로세스 다중 스레드 방식인 worker MPM을 사용하고 있는 Apache 2.0.52버전은 동시 요청에 대한 서블릿의 재사용율이 높기 때문에 캐시설정에 대하여 큰 영향을 받고 있지 않는 것을 알 수 있다.

DHE 서블릿은 강제 캐시설정을 한 상태에서 20라인 생성의 실험결과는 매우 인

상적이다. Intel의 Pentium-Ⅲ 850Mhz CPU를 사용하는 GNU/Linux가 Pentium-4 1500Mhz를 사용하는 Windows 2000보다 더 나은 성능을 보이고 있기 때문이다. 이 실험에서 각 요청의 처리에 관여한 시간은 서블릿의 실행시간에 비하여 웹 서버 자체와 운영체제의 커널시간이 더 많이 포함되어 있다고 볼 수 있다.



<그림 6-4> ApacheBench 실험결과: 200라인(28,525Bytes)



<그림 6-5> ApacheBench 실험결과: 2,000라인(284,126Bytes)

국내의 대표적인 검색엔진과 인터넷 쇼핑몰이 생성한 HTML의 크기가 대부분 50Kbytes 이상이고 300Kbytes에 이르는 것이 있는 것을 감안하면 200라인과

2,000라인을 생성하는 실험의 결과인 <그림 6-4>와 <그림 6-5>가 유효하다. 이것을 통하여 서버 어플리케이션의 연산시간이 길어질수록 DHE와 다른 서버 어플리케이션 기술들과의 차이는 더욱 벌어진다는 것을 알 수 있다.

연산시간이 길어질수록 DHE 서블릿의 경우 강제 캐시설정을 한 것과 기본 캐시설정을 한 것과의 격차가 줄어들고 있다. 다중 스레드 방식이 적용된 웹 서버에서 이러한 결과는 서블릿의 재사용율이 증가하는 것처럼 보일 수도 있다. 그러나 다중 스레드 방식이 적용되지 않는 GNU/Linux의 Apache 1.3.23에서도 성능격차가 줄어든 것은 마찬가지이고 다중 스레드 방식이 적용된 서블릿 컨테이너의 로그(log)에 서블릿의 참조카운터를 출력하여 확인해 본 결과 반드시 재사용된 것은 아닌 것으로 확인되었다. 이러한 결과가 나타난 것은 각각의 요청에 있어서 서블릿의 연산시간에 비하여 서블릿의 적재와 관련한 런타임 링킹시간이 상대적으로 짧기 때문이다.

Windows 2000의 IIS 5.0에서 강제 캐시설정을 한 DHE는 DHE외에 가장 나은 성능을 보이고 있는 Tomcat 5.5.9의 JSP에 비하여 200라인(27.8Kbytes)을 생성하는 실험에서는 1/5 미만의 가상메모리를 사용하고 있으면서 3.5배 이상의 성능을 보이고 있고 2,000라인(277.4Kbytes)을 생성하는 실험에서는 1/4 미만의 가상메모리를 사용함에도 3.5배의 성능을 보이고 있는데 이는 DHE가 매우 유효하다는 것을 증명하는 결과이다.

제7장 결 론

본 논문은 동적인 콘텐츠를 생성하도록 하는 웹 서버의 어플리케이션을 위하여 네이티브 바이너리 코드로 컴파일된 동적공유객체를 사용하는 DHE를 제안하였다. DHE는 웹 서버의 플러그인 모듈과 유사한 성능을 얻을 수 있지만 특정 서버의 API에 종속되지 않도록 설계되었기 때문에 서버 소프트웨어간의 이식성을 가진다.

DHE의 기본 구성요소는 서블릿과 서블릿 컨테이너가 있다. 서블릿은 DHE 환경에서 HTTP 요청에 대한 실질적인 서비스를 담당하는 동적공유객체로 서블릿 컨테이너에 의해 실행된다. 서블릿 컨테이너는 웹 서버마다 각기 다른 API 차이점을 흡수하여 서블릿에게 특정 웹 서버로부터 독립된 API 환경을 제공한다. 따라서 같은 운영체제하에서는 서로 다른 웹 서버라 할지라도 같은 서블릿을 사용할 수 있다. 서블릿 컨테이너는 기존에 널리 알려진 웹 서버의 플러그인 형태로 구현되어 동작하며 Windows와 UNIX에서 IIS 5.0과 Apache 1,3.x, 2.x 버전 및 명령행 컨테이너를 개발하였다.

DHE 인터페이스 모델은 서블릿의 구현에 있어서 별도의 라이브러리 코드를 요구하지 않고 있기 때문에 서블릿은 표준 C언어 함수호출규약을 지원하는 모든 프로그래밍 언어를 사용해서 개발할 수 있다. DCL의 DCLNet 라이브러리 패키지에는 서블릿 개발에 필요한 HTTP, HTML관련 클래스 및 서블릿 클래스가 포함되어 있으며 디버그 버전으로 컴파일된 서블릿은 DCL의 실행시간 디버깅에 사용되는 ASSERT와 TRACE를 가능하게 하고 잘못 사용된 동적 메모리 객체에 대한 정보를 표시한다.

기존의 널리 사용되고 있는 동적 콘텐츠 기술과의 성능비교를 위한 실험에서 DHE는 가장 적은 가상메모리를 사용하고 있었고 200라인(27.8Kbytes)이상의 문자열을 생성하는 실험에서 단위 시간당 처리할 수 있는 HTTP 요청의 개수가 인터프리터 기술을 사용하는 PHP, ASP와 비교하여 6배 이상, 컴파일러 기술을 사용하는 JSP와는 3배 이상의 결과를 얻었다. 이러한 결과는 DHE를 사용하여 웹 서비스를

하게 될 경우 기존 기술을 사용하는 3~6개 이상의 시스템을 단 한 개의 시스템으로 대체하는 것이 가능함을 보여준다.

향후 연구과제는 크게 두 가지가 있다. 첫 번째는 서블릿의 캐시설정과 관련한 부분으로 서블릿 컨테이너에 의해 사용되는 DHE.INI와 관련이 있다. 현재는 이 파일에서 서블릿에 대한 캐시설정을 결정하는 cacheable 속성의 값을 강제 캐시설정을 의미하는 true와 기본 캐시설정의 false로 두 가지만을 사용하고 있다. HTTP 요청에 의하여 적재된 서블릿은 cacheable이 true일 경우 별도의 조치를 하지 않으면 서버 프로세스가 종료될 때까지 캐시된 상태를 유지하게 된다. 그리고 cacheable이 false일 때에는 다중의 스레드에 의해 사용되는 경우를 제외하고는 단 한번만 사용이 된 후 제거되도록 하고 있다. 서블릿을 캐시하고 있으면 HTTP 요청에서 서블릿의 반복적인 동적링킹과 초기화 과정이 제거되기 때문에 더 나은 성능을 얻을 수 있지만, 자주 사용되지 않은 서블릿을 캐시하게 되면 불필요한 메모리의 낭비가 발생한다. 따라서 이를 좀더 발전시키기 위해 서블릿의 캐시설정을 위한 방법을 보완하여 강제 캐시설정을 하지 않더라도 자주 사용되는 서블릿을 위하여 통계적인 방법을 적용한 캐시알고리즘을 개발하는 것이다.

두 번째는 서블릿의 배치방법을 유연하게 하는 것이다. 현재의 방법은 URI의 경로와 연관된 파일시스템의 디렉토리에 서블릿을 위치시키고 있는데 이것은 특정 서블릿에 대하여 다양한 경로로 접근되는 웹 서비스의 구성을 어렵게 하는 문제점이 있다. 이를 해결하기 위해 URI 경로에 있는 텍스트 파일에 실제 서블릿의 경로를 포함시켜서 서블릿을 간접적으로 연결하는 방법이 있는데 현재의 방법에 비하여 부가적인 오버헤드가 발생하기 때문에 실험을 통한 최적의 방법을 찾아야 한다.

참고문헌

[1] Chamas Enterprises Inc, “Apache Hello World Benchmarks”, 2003.04, http://www.chamas.com/bench/index.html

[2] Apache Software Foundation, “The Apache HTTP Server Project”, http://httpd.apache.org/

[3] Ibrahim Haddad, “Apache 2.0: The Internals of the New, Improved”, 2001.08, http://www.linuxjournal.com/article/4559

[4] Microsoft Corporation, “IIS 6.0 Architecture”, http://www.microsoft.com/technet/prodtechnol/WindowsServer2003/Library/IIS/843df643-1dbb-4fb6-910d-ec1965fa9e43.mspx

[5] Microsoft Corporation, “Internet Information Services 5.0 설명서”

[6] Microsoft Corporation, “IIS Web Development SDK”, http://msdn.microsoft.com/library/en-us/iissdk/html/e8eff418-8e4f-4db8-ad70-01473bf10741.asp

[7] The PHP Group, “PHP Hypertext Preprocessor”, http://www.php.net/

[8] Zend Technologies Ltd., “Zend Engin II”, http://www.zend.com/php5/zend-engine2.php

[9] Microsoft Corporation, “Overview of ASP.NET Processes Isolation”, http://www.microsoft.com/technet/prodtechnol/WindowsServer2003/Library/IIS/e409289d-2786-4a34-bb7e-9c546602c2c8.mspx

[10] Sun Microsystems, “JavaServer Pages Technology”, http://java.sun.com/products/jsp/

[11] “The Common Gateway Interface”, http://hoohoo.ncsa.uiuc.edu/cgi/

[12] Open Market Inc., “FastCGI Home”, http://www.fastcgi.com/

[13] Sun Microsystems, “Linker and Libraries Guide”, 2002, http://docs.sun.com/app/docs/doc/816-1386

[14] Microsoft Corporation, “Dynamic-Link Libraries”, http://msdn.microsoft.com/library/en-us/dllproc/base/dynamic_link_libraries.asp

[15] 김대중, “DCL Project”, http://www.sysdeveloper.net/dcl/

[16] 김대중, “DCL Runtime Debugging Support”, 2005.06, http://www.sysdeveloper.net/dcl/

[17] 김대중, “DCL HTTP Server Extension Version 2”, 2005.06, http://www.sysdeveloper.net/dcl/

[18] 김대중, “DCL Database Connectivity 2”, 2005.04, http://www.sysdeveloper.net/dcl/

[19] 김대중, “DCL HTML Template”, 2005.06, http://www.sysdeveloper.net/dcl/

[20] 이원영, “웹 기반 시스템 하에서의 성능에 관한 이론적 고찰”, 2001.12, http://www-903.ibm.com/kr/techinfo/websphere/download/PerformanceConcept.ppt

[21] 김대중, 곽덕훈, “동적공유객체 서블릿을 사용한 웹 서버 어플리케이션 기술”, *제32회 한국정보과학회 추계학술발표회 논문집*, 제32권 제2호(I), pp. 838-840, 2005년 11월

ABSTRACT

Performance Upgrade of Web Services using DCL-based Servlet

by

Dae Jung Kim

Department of Computer Science

Graduate School

Korea National Open University

Supervised by Professor : Duk Hoon Kwak

In a web environment, there are various techniques for creating dynamic contents. These technologies are interpreter-based technology such as ASP or PHP and compiler-based technology such as JSP or ASP.NET. Due to the shortcomings included in the characteristics in each of them, these techniques are limited to raise the efficiency of system resources.

The best way to use the system resources is to make the application be compiled into native binary code. This study is on the DHE (DCL HTTP Server Extension) technology, which uses DSO (Dynamic Shared Object) of the operating system for

the application creating dynamic contents.

DHE is made up of DHE Servlet and DHE Servlet container. Under the DHE environment, Servlet is developed into DSO, performing detailed procedure in response to the request of HTTP performed by the Servlet container. Servlet container is developed as a plug-in of IIS or Apache, which is widely used as web server software. Servlet container absorbs the discrepancy of API (Application Programming Interface) between the web servers, provides the independent environment in which Servlets can be ported into various web servers.

DHE Servlets can be developed with any programming language, which provide standard C language function calling convention and can produce DSO. In case of using the C++ class library which is developed in the DCL (Daejung's Class Library) project, it has some advantages in porting DHE Servlets on various operating systems and provides enhanced productivity with the object-oriented methods.

To verify the validity of DHE, efficiency comparison test of PHP, ASP, ASP.NET, and JSP server application using the same algorithm has been performed.

As the results, it was found that DHE environment on the DCL-based Servlet used the minimum virtual memories. In an experiment creating more than 200 lines (27.8Kbytes), the number of HTTP request processed by DCL-based Servlet is 3 times higher then JSP and 6 times higher than ASP or PHP.

부 록

[부록-1] HttpServerExtension.h

```
1:  #ifndef __DCL_HTTP_SERVER_EXTENSION_H__
2:  #define __DCL_HTTP_SERVER_EXTENSION_H__      20050509
3:
4:  #ifndef __DCL_CONFIG_H__
5:  #include <dcl/_Config.h>
6:  #endif
7:
8:  /* RFC 2616/5.1.1 */
9:  enum HTTP_REQUEST_METHOD {
10:     HTTP_METHOD_UNKNOWN             = 0,
11:     HTTP_METHOD_OPTIONS             = 1,
12:     HTTP_METHOD_GET                 = 3,
13:     HTTP_METHOD_HEAD                = 4,
14:     HTTP_METHOD_POST                = 5,
15:     HTTP_METHOD_PUT                 = 6,
16:     HTTP_METHOD_DELETE              = 7,
17:     HTTP_METHOD_TRACE               = 8,
18:     HTTP_METHOD_CONNECT             = 9
19: };
20:
21: enum HTTP_RESPONSE_STATUS_CODE {
22:     HTTP_STATUS_CONTINUE                        = 100,
23:     HTTP_STATUS_SWITCHING_PROTOCOLS             = 101,
24:     HTTP_STATUS_OK                              = 200,
25:     HTTP_STATUS_CREATED                         = 201,
26:     HTTP_STATUS_ACCEPTED                        = 202,
27:     HTTP_STATUS_NON_AUTHORITIVE                 = 203,
28:     HTTP_STATUS_NO_CONTENT                      = 204,
29:     HTTP_STATUS_RESET_CONTENT                   = 205,
30:     HTTP_STATUS_PARTIAL_CONTENT                 = 206,
31:     HTTP_STATUS_MULTIPLE_CHOICES                = 300,
32:     HTTP_STATUS_MOVED_PERMANENTLY               = 301,
33:     HTTP_STATUS_FOUND                           = 302,
34:     HTTP_STATUS_SEE_OTHER                       = 303,
35:     HTTP_STATUS_NOT_MODIFIED                    = 304,
36:     HTTP_STATUS_USE_PROXY                       = 305,
37:     HTTP_STATUS_TEMPORARY_REDIRECT              = 307,
38:     HTTP_STATUS_BAD_REQUEST                     = 400,
```

```
39:     HTTP_STATUS_UNAUTHORIZED                                = 401,
40:     HTTP_STATUS_PAYMENT_REQUIRED                            = 402,
41:     HTTP_STATUS_FORBIDDEN                                   = 403,
42:     HTTP_STATUS_NOT_FOUND                                   = 404,
43:     HTTP_STATUS_METHOD_NOT_ALLOWED                          = 405,
44:     HTTP_STATUS_NOT_ACCEPTABLE                              = 406,
45:     HTTP_STATUS_PROXY_AUTHENTICATION_REQUIRED               = 407,
46:     HTTP_STATUS_REQUEST_TIME_OUT                            = 408,
47:     HTTP_STATUS_CONFLICT                                    = 409,
48:     HTTP_STATUS_GONE                                        = 410,
49:     HTTP_STATUS_LENGTH_REQUIRED                             = 411,
50:     HTTP_STATUS_PRECONDITION_FAILED                         = 412,
51:     HTTP_STATUS_REQUEST_ENTITY_TOO_LARGE                    = 413,
52:     HTTP_STATUS_REQUEST_URI_TOO_LARGE                       = 414,
53:     HTTP_STATUS_UNSUPPORTED_MEDIA_TYPE                      = 415,
54:     HTTP_STATUS_REQUESTED_RANGE_NOT_SATISFIABLE             = 416,
55:     HTTP_STATUS_EXPECTATION_FAILED                          = 417,
56:     HTTP_STATUS_INTERNAL_SERVER_ERROR                       = 500,
57:     HTTP_STATUS_NOT_IMPLEMENTED                             = 501,
58:     HTTP_STATUS_BAD_GATEWAY                                 = 502,
59:     HTTP_STATUS_SERVICE_UNAVAILABLE                         = 503,
60:     HTTP_STATUS_GATEWAY_TIME_OUT                            = 504,
61:     HTTP_STATUS_VERSION_NOT_SUPPORTED                       = 505
62: };
63:
64: typedef struct _DCL_HTTP_CONN               DCL_HTTP_CONN;
65: typedef DCL_HTTP_CONN*                      DCL_HTTP_HCONN;
66: typedef struct _DCL_HTTP_SERVER_API         DCL_HTTP_SERVER_API;
67: typedef struct _DCL_HTTP_SERVLET_CONTEXT    DCL_HTTP_SERVLET_CONTEXT;
68: typedef struct _DCL_HTTP_SERVLET_CONFIG     DCL_HTTP_SERVLET_CONFIG;
69: typedef struct _DCL_HTTP_SERVLET            DCL_HTTP_SERVLET;
70:
71: /* callback functions */
72: /* for DCL_HTTP_SERVER_API */
73: typedef BOOL (* DCLHttpWriteClient)(
74:     DCL_HTTP_HCONN          hConn,          /* (IN) */
75:     const void*             pvBuffer,       /* (IN) */
76:     UINT32*                 pnLength        /* (IN, OUT) */
77:     );
78: typedef BOOL (* DCLHttpReadClient)(
79:     DCL_HTTP_HCONN          hConn,          /* (IN) */
80:     void*                   pvBuffer,       /* (OUT) */
81:     UINT32*                 pnLength        /* (IN, OUT) */
82:     );
83: typedef BOOL (* DCLHttpSendResponseHeader)(
```

```
84:     DCL_HTTP_HCONN          hConn,          /* (IN) */
85:     UINT32                  uStatusCode,    /* (IN) */
86:     const char*             pszHeader,      /* (IN) */
87:     UINT32                  uLength         /* (IN) number of bytes in header */
88:     );
89: typedef BOOL (* DCLHttpGetRequestHeader)(
90:     DCL_HTTP_HCONN          hConn,          /* (IN) */
91:     const char*             pszName,        /* (IN) NULL => all */
92:     char*                   pchBuffer,      /* (OUT) */
93:     UINT32*                 pnLength        /* (IN, OUT) */
94:     );
95: typedef BOOL (* DCLHttpGetCgiVariable)(
96:     DCL_HTTP_HCONN          hConn,          /* (IN) */
97:     const char*             pszVarName,     /* (IN) NULL => all vars */
98:     void*                   pvBuffer,       /* (OUT) */
99:     UINT32*                 pnLength        /* (IN, OUT) */
100:    );
101: typedef void (* DCLHttpWriteStream)(
102:    void*                   hStream,        /* (IN) */
103:    const void*             pvData,         /* (IN) */
104:    UINT32                  uLength         /* (IN) */
105:    );
106: typedef void (* DCLHttpServerControl)(
107:    DCL_HTTP_HCONN          hConn,          /* (IN) */
108:    const char*             psControl,
109:    UINT32                  uLength,
110:    DCLHttpWriteStreampfnWriteResult,
111:    void*                   hResultOutputStream
112:    );
113:
114: /* for DCL_HTTP_SERVLET */
115: typedef BOOL (* DCLHttpServletInitialize)(
116:    const DCL_HTTP_SERVLET_CONFIG* pConfig,
117:    void*                   hReportError
118:    );
119: typedef BOOL (* DCLHttpServletCleanup)(
120:    void*                   hReportError
121:    );
122: typedef BOOL (* DCLHttpServletService)(
123:    const DCL_HTTP_SERVLET_CONTEXT* pContext,
124:    void*                   hReportError
125:    );
126:
127: struct _DCL_HTTP_CONN {
128:    void*                   pv;
```

```
129: };
130:
131: struct _DCL_HTTP_SERVER_API {
132:     UINT32                  uSize;              /* (IN) size of this structure */
133:     UINT32                  uVersion;           /* (IN) version info of this spec */
134:     DCLHttpWriteClient  pfnWriteClient;
135:     DCLHttpReadClient   pfnReadClient;
136:     DCLHttpSendResponseHeader pfnSendResponseHeader;
137:     DCLHttpGetRequestHeader   pfnGetRequestHeader;
138:     DCLHttpGetCgiVariable     pfnGetCgiVariable;
139:     DCLHttpWriteStreampfnReportError;
140:     DCLHttpServerControl      pfnServerControl;
141: };
142:
143: struct _DCL_HTTP_SERVLET_CONTEXT {
144:     UINT32              uSize;                  /* (IN) size of this structure */
145:     UINT32              uVersion;               /* (IN) version info of this spec */
146:     DCL_HTTP_HCONN      hConn;                  /* (IN) */
147:     const char*          pszRemoteAddr;         /* (IN) REMOTE_ADDR */
148:     UINT32              uRemotePort;            /* (IN) REMOTE_PORT */
149:     UINT32              uRequestMethod;         /* (IN) RFC 2616/5.1.1
HTTP_REQUEST_METHOD */
150:     const char*         pszRequestMethod;       /* (IN) REQUEST_METHOD */
151:     const char*         pszPath;                /* (IN) URI's abs_path */
152:     const char*         pszQueryString;         /* (IN) QUERY_STRING */
153:     const char*         pszContentType;         /* (IN) CONTENT_TYPE */
154:     UINT32              uContentLength;         /* (IN) CONTENT_LENGTH */
155: };
156:
157: struct _DCL_HTTP_SERVLET_CONFIG {
158:     const char*         pszFileName;
159:     const char*         pszConfigDir;
160:     const char*         pszTempDir;
161:     const DCL_HTTP_SERVER_API* pSAPI;
162: };
163:
164: struct _DCL_HTTP_SERVLET {
165:     /* DCL common members      */
166:     UINT32              uSize;          /* (OUT) size of this structure    */
167:     UINT32              uDCLVersion;    /* (OUT) DCL_VERSION, non DCL(0)*/
168:     const char*         pszBuildTimeStamp;  /* (OUT) __TIMESTAMP__          */
169:     UINT32              uBuildFlag;     /* (OUT) release(0), debug(1)          */
170:     UINT32              uModuleType;    /* (OUT) DCL_HTTP_SERVLET_MODULE */
171:     const char*         pszDescription; /* (OUT) module description                */
172:
```

```
173:      /* private members              */
174:      UINT32              uVersion;           /* (OUT) version info of this spec      */
175:                                              /* DCL_HTTP_SERVER_VERSION*/
176:
177:      DCLHttpServletInitialize      pfnInitialize;
178:      DCLHttpServletCleanup         pfnCleanup;
179:      DCLHttpServletService         pfnHttpService;
180: };
181:
182: #endif      /* __DCL_HTTP_SERVER_EXTENSION_H__      */
```

[부록-2] 성능평가를 위한 서버 어플리케이션

genstr.cpp

```
1:  /*
2:       USAGE: genstr.dhe?line=#####&no_str&print_time
3:            line : 생성할 라인의 수 140/one-line
4:            no_str : String 클래스이 사용여부
5:            print_time : 생성한 시간의 출력여부
6:  */
7:  #include <stdio.h>       // snprintf
8:  #include <string.h>      // strncpy
9:
10: #include <dcl/core/Numeric.h>
11: #include <dcl/core/DateTime.h>
12: #include <dcl/net/HttpServletEx.h>
13:
14: #ifdef __DCL_DEBUG
15: #undef __THIS_FILE__
16: static const char* __THIS_FILE__ = "genstr.cpp";
17: #endif
18:
19: __DCL_USING_NAMESPACE
20:
21: class GenStrServlet : public HttpServletEx
22: {
23:      DECLARE_CLASSINFO(GenStrServlet)
24: protected:
25:      virtual void onService(HttpServletContextEx& ctx)
26:                          __DCL_THROWS1(Exception*);
27:
28:      String getLine(int nLine);
```

```
29:         int getLine(char* pBuf, int nLine);
30:     };
31:
32:     HTTP_SERVLET_INSTANCE(GenStrServlet, "ASCII Text Generator")
33:
34:     IMPLEMENT_CLASSINFO(GenStrServlet, HttpServletEx)
35:
36:     String GenStrServlet::getLine(int nLine)
37:     {
38:         String strLine;
39:         for(int i = 32; i < 127; i++)
40:             strLine += (char)i;
41:
42:         int nLen = strLine.length();
43:         int nRotate = nLine % nLen;
44:         if (nRotate > 0)
45:             strLine = strLine.right(nLen - nRotate)
46:                     + strLine.left(nRotate);
47:
48:         String strLineNo;
49:         strLineNo.sprintf("<span style=\"color: red;\">%06d </span>", nLine);
50:         strLine = strLineNo + strLine + "<br/>";
51:
52:         return strLine;
53:     }
54:
55:     int GenStrServlet::getLine(char* pBuf, int nLine)
56:     {
57:         char aSrcBuf[128];
58:
59:         for(int i = 32; i < 127; i++)
60:             aSrcBuf[i - 32] = (char)i;
61:
62:         int n = sprintf(pBuf, "<span style=\"color: red;\">%06d </span>", nLine);
63:
64:         int nLen = 127 - 32;
65:         int nRotate = nLine % nLen;
66:         if (nRotate > 0)
67:         {
68:             strncpy(pBuf + n, aSrcBuf + nRotate, nLen - nRotate);
69:             strncpy(pBuf + n + nLen - nRotate, aSrcBuf, nRotate);
70:         }
71:         else
72:             strncpy(pBuf + n, aSrcBuf, nLen);
73:
```

```
74:         n += nLen;
75:
76:         strcpy(pBuf + n, "<br/>");
77:         n += 5;
78:
79:         return n;
80:     }
81:
82:     void GenStrServlet::onService(HttpServletContextEx& ctx)
83:     //                                  __DCL_THROWS1(Exception*)
84:     {
85:         bool bNoString = ctx.m_params.find("no_str") != ctx.m_params.end();
86:         bool bPrintTime = ctx.m_params.find("print_time") != ctx.m_params.end();
87:         bool bNoLine = ctx.m_params.find("no_line") != ctx.m_params.end();
88:
89:         int nMaxLine = 20;
90:         StringVector& v = ctx.m_params["line"];
91:         if (v.count() > 0)
92:         {
93:             try
94:             {
95:                 nMaxLine = Integer::parse(v[0]);
96:                 if (nMaxLine == 0)
97:                     nMaxLine = 20;
98:                 else if (nMaxLine > 100000)
99:                     nMaxLine = 100000;
100:            }
101:            catch(Exception* e)
102:            {
103:                e->destroy();
104:            }
105:        }
106:
107:        DateTime startTime = DateTime::getCurrentLocalTime();
108:
109:        OutputStream& out = ctx.createOutputStream(nMaxLine * 150);
110:
111:        ctx.setContentType("text/html", "euc-kr");
112:        out << "\n\n\n"
113:            "<html>\n<body>\n<div>\n";
114:
115:        int nLen = 0;
116:        if (bNoString)
117:        {
118:            for(int i = 1; i <= nMaxLine; i++)
```

```
119:                {
120:                    char aBuf[256];
121:                    int n = getLine(aBuf, i);
122:                    nLen += n;
123:                    if (!bNoLine)
124:                    {
125:                        out.write(aBuf, n);
126:                        out.write("\n", 1);
127:                    }
128:                }
129:            }
130:            else
131:            {
132:                for(int i = 1; i <= nMaxLine; i++)
133:                {
134:                    String strLine = getLine(i);
135:                    nLen += strLine.length();
136:                    if (!bNoLine)
137:                        out.println(strLine);
138:                }
139:            }
140:
141:            DateTime endTime = DateTime::getCurrentLocalTime();
142:            Interval elapsedTime = endTime - startTime;
143:
144:            out << "<div style=\"color: blue; font-weight: bold;\">\n"
145:                "data size: " << nLen << "<br/>\n";
146:            if (bPrintTime)
147:                out << "time elapsed: " << elapsedTime.toString() << "\n";
148:
149:            out << "</div>\n";
150:            // 다음문장의 html 은 HttpServletEx 가 붙인다.
151:    //      out << "</body>\n</html>\n";
152:    }
```

genstr.asp

```
1:  <%@ language=VBScript %>
2:  <%
3:  Function getLine(nLine)
4:      Dim ch, strLine, strLineNo          ' String
5:      Dim nLen, nRotate                   ' Long
6:
```

```
        For ch = 32 To 126
            strLine = strLine & Chr(ch)
        Next

        nLen = Len(strLine)
        nRotate = nLine Mod nLen
        If nRotate > 0 Then
            strLine = Right(strLine, nLen - nRotate) & Left(strLine, nRotate)
        End If

        strLineNo = CStr(nLine)
        If Len(strLineNo) < 6 Then
            nLead = 6 - Len(strLineNo)
            strLineNo = Left("000000", nLead) & strLineNo
        End If
        getLine = "<span style=" & Chr(34) & "color: red;" & Chr(34) & ">"
        getLine = getLine & strLineNo & " </span>" & strLine & "<br/>"
End Function

Dim chQuot, strLine                     ' String
Dim bPrintTime, bNoLine                 ' Boolean
Dim nMaxLine, I                         ' Long
Dim startTime, endTime, elapsedTime     ' DateTime

chQuot = Chr(34)
bPrintTime = False
bNoLine = False
nMaxLine = 20       'default

If Request.QueryString("print_time").Count > 0 Then
    bPrintTime = True
End If

If Request.QueryString("no_line").Count > 0 Then
    bNoLine = True
End If

If Request.QueryString("line").Count > 0 Then
    nMaxLine = CLng(Request.QueryString("line"))
    If nMaxLine = 0 Then
        nMaxLine = 20
    Else
        If nMaxLine > 100000 Then
        nMaxLine = 100000
        End If
```

```
52:         End If
53:     End If
54:
55:     startTime = Now
56:
57:     Response.ContentType = "text/html;"
58:     Response.Charset="euc-kr"
59:     Response.Write(vbCrLf & vbCrLf & vbCrLf)
60:     Response.Write("<html>" & vbCrLf & "<body>" & vbCrLf & "<div>" & vbCrLf)
61:     nLen = 0
62:     For I = 1 To nMaxLine
63:         strLine = getLine(I)
64:         nLen = nLen + Len(strLine)
65:         If Not bNoLine Then
66:             Response.Write(strLine & vbCrLf)
67:         End If
68:     Next
69:
70:     endTime = Now
71:     elapsedTime = endTime - startTime
72:
73:     Response.Write("<div style=" & chQuot)
74:     Response.Write("color: blue; font-weight: bold;" & chQuot & ">" & vbCrLf)
75:     Response.Write("data size: " & CStr(nLen) & "<br/>" & vbCrLf)
76:     If bPrintTime = True Then
77:         Response.Write("time elapsed: " & CStr(elapsedTime) & vbCrLf)
78:     End If
79:     Response.Write("</div>" & vbCrLf & "</body>" & vbCrLf & "</html>" & vbCrLf)
80:     %>
```

genstr2.asp

```
1:  <%@ language=VBScript %>
2:  <%
3:  Function getLine(nLine)
4:      Dim ch, strLine, strLineNo          ' String
5:      Dim nLen, nRotate                   ' Long
6:
7:      For ch = 32 To 126
8:          strLine = strLine & Chr(ch)
9:      Next
10:
11:     nLen = Len(strLine)
```

```
12:         nRotate = nLine Mod nLen
13:         If nRotate > 0 Then
14:             strLine = Right(strLine, nLen - nRotate) & Left(strLine, nRotate)
15:         End If
16:
17:         strLineNo = CStr(nLine)
18:         If Len(strLineNo) < 6 Then
19:             nLead = 6 - Len(strLineNo)
20:             strLineNo = Left("000000", nLead) & strLineNo
21:         End If
22:         getLine = "<span style=" & Chr(34) & "color: red;" & Chr(34) & ">"
23:         getLine = getLine & strLineNo & " </span>" & strLine & "<br/>"
24:     End Function
25:
26:     Dim chQuot, strLine                         ' String
27:     Dim bPrintTime, bNoLine                     ' Boolean
28:     Dim nMaxLine, I                             ' Long
29:     Dim startTime, endTime, elapsedTime         ' DateTime
30:
31:     chQuot = Chr(34)
32:     bPrintTime = False
33:     bNoLine = False
34:     nMaxLine = 20       'default
35:
36:     If Request.QueryString("print_time").Count > 0 Then
37:         bPrintTime = True
38:     End If
39:
40:     If Request.QueryString("no_line").Count > 0 Then
41:         bNoLine = True
42:     End If
43:
44:     If Request.QueryString("line").Count > 0 Then
45:         strMaxLine = Request.QueryString("line")
46:         nMaxLine = CLng(strMaxLine)
47:         If nMaxLine = 0 Then
48:             nMaxLine = 20
49:         Else
50:             If nMaxLine > 100000 Then
51:             nMaxLine = 100000
52:             End If
53:         End If
54:     End If
55:
56:     Set TimeEx = Server.CreateObject("TimeEx.SystemTime")
```

```
57: 'startTime = Now
58: startTime = TimeEx.CurrentTime
59:
60: Response.ContentType = "text/html;"
61: Response.Charset="euc-kr"
62: Response.Write(vbCrLf & vbCrLf & vbCrLf)
63: Response.Write("<html>" & vbCrLf & "<body>" & vbCrLf & "<div>" & vbCrLf)
64: nLen = 0
65: For I = 1 To nMaxLine
66:     strLine = getLine(I)
67:     nLen = nLen + Len(strLine)
68:     If Not bNoLine Then
69:         Response.Write(strLine & vbCrLf)
70:     End If
71: Next
72:
73: 'endTime = Now
74: endTime = TimeEx.CurrentTime
75: elapsedTime = endTime - startTime
76:
77: Response.Write("<div style=" & chQuot)
78: Response.Write("color: blue; font-weight: bold;" & chQuot & ">" & vbCrLf)
79: Response.Write("data size: " & CStr(nLen) & "<br/>" & vbCrLf)
80: 'Response.Write("time elapsed: " & CStr(Second(endTime) - Second(startTime)) & vbCrLf)
81: If bPrintTime = True Then
82:     Response.Write("time elapsed: " & FormatNumber(elapsedTime, 3) & vbCrLf)
83: End If
84: Response.Write("</div>" & vbCrLf & "</body>" & vbCrLf & "</html>" & vbCrLf)
85: %>
```

genstr.aspx

```
1: <%@ Page Language="C#"    %>
2:
3: <script runat="server" language="C#">
4:
5: String GetLine(int nLine)
6: {
7:     String str = "";
8:     for(int ch = 32; ch < 127; ch++)
9:         str += ((char)ch).ToString();
10:
11:     int nLen = str.Length;
```

```
12:         int nRotate = nLine % nLen;
13:
14:         if (nRotate > 0)
15:             str = str.Substring(nRotate, nLen - nRotate)
16:                 + str.Substring(0, nRotate);
17:
18:         String strLineNo = "<span style=\"color: red;\">";
19:         strLineNo += String.Format("{0:000000}", nLine) + " </span>";
20:
21:         str = strLineNo + str + "<br/>";
22:         return str;
23:     }
24:
25:     </script>
26:     <%
27:
28:     bool bPrintTime = Request.QueryString["print_time"] != null;
29:     bool bNoLine = Request.QueryString["no_line"] != null;
30:     int nMaxLine = 20;
31:     String[] aLines = Request.QueryString.GetValues("line");
32:     if (aLines != null)
33:     {
34:         try
35:         {
36:             nMaxLine = Int32.Parse(aLines[0]);
37:             if (nMaxLine <= 0)
38:                 nMaxLine = 20;
39:             else if (nMaxLine > 100000)
40:                 nMaxLine = 100000;
41:         }
42:         catch(Exception e)
43:         {
44:         }
45:     }
46:     DateTime startTime = DateTime.Now;
47:
48:     Response.ContentType = "text/html; charset=euc-kr";
49:     //Response.BufferOutput = true;
50:     Response.Write("\r\n\r\n\r\n<html>\r\n<body>\r\n<div>\r\n");
51:     int nLen = 0;
52:     for(int n = 0; n < nMaxLine; n++)
53:     {
54:         String strLine = GetLine(n);
55:         nLen += strLine.Length;
56:         if (!bNoLine)
```

```
57:             Response.Write(GetLine(n) + "\r\n");
58: }
59: DateTime endTime = DateTime.Now;
60: TimeSpan elapsedTime = endTime - startTime;
61: Response.Write("<div style=\"color: blue; font-weight: bold;\">\r\n"
62:             + "data size: " + nLen.ToString() + "<br/>\r\n");
63: if (bPrintTime)
64:     Response.Write("time elapsed: " + elapsedTime.ToString() + "\r\n");
65:
66: Response.Write("</div>\r\n</body>\r\n</html>\r\n");
67: %>
```

genstr.php

```
1:  <?php
2:  function getLine($nLine)
3:  {
4:      $strLine = "";
5:      for($ch = 32; $ch < 127; $ch++)
6:          $strLine .= chr($ch);
7:
8:      $nLen = strlen($strLine);
9:      $nRotate = $nLine % $nLen;
10:     if ($nRotate > 0)
11:         $strLine = substr($strLine, $nRotate)
12:             . substr($strLine, 0, $nRotate);
13:
14:     $strLine = sprintf("<span style=\"color: red;\">%06d </span>", $nLine)
15:                 . $strLine . "<br/>";
16:
17:     return $strLine;
18: }
19:
20: /*
21:     genstring.php?line=X
22: */
23:
24: $bPrintTime = false;
25: $bNoLine = false;
26: $nMaxLine = 20;
27:
28: $bNoLine = isset($_GET["no_line"]);
29: $bPrintTime = isset($_GET["print_time"]);
```

```
30:
31:  if (isset($_GET["line"]))
32:  {
33:      $nMaxLine = $_GET["line"];
34:      if ($nMaxLine == 0)
35:          $nMaxLine = 20;
36:
37:      if ($nMaxLine > 100000)
38:          $nMaxLine = 100000;
39:  }
40:
41:  $startTime = microtime(true);
42:
43:  header("Content-Type: text/html; charset=euc-kr");
44:  print("\r\n\r\n\r\n");
45:  print("<html>\r\n<body>\r\n<div>\r\n");
46:
47:  $nLen = 0;
48:  for($i = 1; $i <= $nMaxLine; $i++)
49:  {
50:      $strLine = getLine($i);
51:      $nLen += strlen($strLine);
52:      if (!$bNoLine)
53:          print($strLine . "\r\n");
54:  }
55:
56:  $endTime = microtime(true);
57:  $elapsedTime = $endTime - $startTime;
58:  $elapsedTime *= 1000;
59:  settype($elapsedTime, "integer");
60:  settype($elapsedTime, "float");
61:  $elapsedTime /= 1000.;
62:
63:  print("<div style=\"color: blue; font-weight: bold;\">\r\n");
64:  print("data size: " . $nLen . "<br/>\r\n");
65:  if ($bPrintTime)
66:      print("time elapsed: " . $elapsedTime . "\r\n");
67:
68:  print("</div>\r\n");
69:  print("</body>\r\n</html>\r\n");
70:  ?>
```

genstr.php4

```
1:  <?php
2:  function getLine($nLine)
3:  {
4:      $strLine = "";
5:      for($ch = 32; $ch < 127; $ch++)
6:          $strLine .= chr($ch);
7:
8:      $nLen = strlen($strLine);
9:      $nRotate = $nLine % $nLen;
10:     if ($nRotate > 0)
11:         $strLine = substr($strLine, $nRotate)
12:             . substr($strLine, 0, $nRotate);
13:
14:     $strLine = sprintf("<span style=\"color: red;\">%06d </span>", $nLine)
15:                 . $strLine . "<br/>";
16:
17:     return $strLine;
18: }
19:
20: /*
21:     genstring.php?line=X
22: */
23:
24: $bPrintTime = false;
25: $bNoLine = false;
26: $nMaxLine = 20;
27:
28: /*
29: $bNoLine = isset($_GET["no_line"]);
30: $bPrintTime = isset($_GET["print_time"]);
31:
32: if (isset($_GET["line"]))
33: {
34:     $nMaxLine = $_GET["line"];
35:     if ($nMaxLine == 0)
36:         $nMaxLine = 20;
37:
38:     if ($nMaxLine > 100000)
39:         $nMaxLine = 100000;
40: }
41: */
42: if (isset($print_time))
43:     $bPrintTime = true;
```

```php
if (isset($no_line))
	$bNoLine = true;
if (isset($line))
{
	$nMaxLine = $line;
	if ($nMaxLine == 0)
		$nMaxLine = 20;
	else if ($nMaxLine > 100000)
		$nMaxLine = 100000;
}

$startTime = time() + microtime(true);

header("Content-Type: text/html; charset=euc-kr");
print("\r\n\r\n\r\n");
print("<html>\r\n<body>\r\n<div>\r\n");

$nLen = 0;
for($i = 1; $i <= $nMaxLine; $i++)
{
	$strLine = getLine($i);
	$nLen += strlen($strLine);
	if (!$bNoLine)
		print($strLine . "\r\n");
}

$endTime = time() + microtime(true);
$elapsedTime = $endTime - $startTime;
//$elapsedTime *= 1000;
//settype($elapsedTime, "integer");
//settype($elapsedTime, "double");
//$elapsedTime /= 1000.;

print("<div style=\"color: blue; font-weight: bold;\">\r\n");
print("data size: " . $nLen . "<br/>\r\n");
if ($bPrintTime)
	print("time elapsed: " . $elapsedTime . "\r\n");

print("</div>\r\n");
print("</body>\r\n</html>\r\n");
?>
```

genstr.jsp

```
1:  <%@page contentType = "text/html; charset=euc-kr" %>
2:  <%@page import = "java.util.*" %>
3:  <%@page import = "java.text.*" %>
4:  <%
5:  class Foo
6:  {
7:      public String getLine(int nLine)
8:      {
9:          StringBuffer strLineBuffer = new StringBuffer();
10:         for(int ch = 32; ch < 127; ch++)
11:             strLineBuffer.append((char)ch);
12:
13:         String strLine = strLineBuffer.toString();
14:         int nLen = strLine.length();
15:         int nRotate = nLine % nLen;
16:         if (nRotate > 0)
17:             strLine = strLine.substring(nRotate, nLen)
18:                 + strLine.substring(0, nRotate);
19:
20:         DecimalFormat df = new DecimalFormat();
21:         df.applyPattern("000000");
22:
23:         String strLineNo = "<span style=\"color: red;\">"
24:                 + df.format(nLine) + " </span>";
25:
26:         strLine = strLineNo + strLine + "<br/>";
27:
28:         return strLine;
29:     }
30: }
31:
32: boolean bPrintTime = false;
33: boolean bNoLine = false;
34: int nMaxLine = 20;       // default
35: if (request.getParameter("print_time") != null)
36:     bPrintTime = true;
37: if (request.getParameter("no_line") != null)
38:     bNoLine = true;
39:
40: String strMaxLine = request.getParameter("line");
41: if (strMaxLine != null)
42: {
43:     try
```

```
44:         {
45:             nMaxLine = Integer.parseInt(strMaxLine);
46:             if (nMaxLine == 0)
47:                 nMaxLine = 20;
48:             else if (nMaxLine > 100000)
49:                 nMaxLine = 100000;
50:         }
51:         catch(Exception e)
52:         {
53:         }
54: }
55:
56: long startTime = Calendar.getInstance().getTimeInMillis();
57:
58: out.println("<html>\r\n<body>\r\n<div>");
59: int nLen = 0;
60: Foo f = new Foo();
61: for(int i = 1; i <= nMaxLine; i++)
62: {
63:     String strLine = f.getLine(i);
64:     nLen += strLine.length();
65:     if (!bNoLine)
66:         out.println(strLine);
67: }
68:
69: long endTime = Calendar.getInstance().getTimeInMillis();
70: long elapsedTime = endTime - startTime;
71:
72: out.println("<div style=\"color: blue; font-weight: bold;\">");
73: out.print("data size: "); out.print(nLen); out.println("<br/>");
74: if (bPrintTime)
75: {
76:     out.print("time elapsed: ");
77:     out.println(((double)(endTime - startTime)) / 1000.);
78: }
79: out.println("</div>");
80: out.println("</body>");
81: out.println("</html>");
82: %>
```

myps.bash

```
1:  #!/bin/bash
2:  if [ -z "$1"] || [ -z "$2"]; then
3:          echo usage: $0 command count
4:          exit
5:  fi
6:  count=0
7:  while [ $count -lt $2 ]; do
8:          echo $count
9:          ps -C $1 -o command,user,ppid,pid,%cpu,vsz,%mem,rss
10:         let count+=1
11:         #usleep 1000000
12:         sleep 1
13: done
```