DCL HTTP Server Extension

Version 2

2005.06.17

김대중<daejung@sysdeveloper.net>
http://www.sysdeveloper.net/daejung

요약

DHE(DCL HTTP Server Extension)은 Apache나 IIS와 같은 기존의 웹 서버 환경에서 동작하며 운영제제의 DSO(Dynamic Shared Object)차원에서 서버의 기능을 확장하도록 하는 환경을 제공한다. DHE 환경에서 서버 어플리케이션은 DHE 서블릿으로 불리는 DSO이며 이것은 C/C++와 같은 컴파일러 언어를 사용하여 구현된다.

DHE 서블릿은 운영체제의 DSO이기 때문에 기존의 ASP, PHP, JSP 및 자바 서블릿과 같은 웹 서버 어플리케이션과는 비교될 수 없을 정도의 빠른 성능을 가진다. 따라서 복잡한 연산을 하거나 뛰어난 응답 속도록 요구하는 서버 어플리케이션의 개발에 적합하다고 할 수 있다.

본 문서는 DHE의 전체적인 내용을 다루고 있다. 1장은 DHE의 연구 배경 및 목적, 구성요소, 인터페이스 모델에 대하여 소개한다. 2장은 DHE를 이루는 인터페이스에 대하여 자세하게 다루고 있고 3장은 Apache, IIS의 서블릿 매니저와 서버의 실행시간에 서블릿 매니저를 제어에 대한 내용을 다룬다. 4장은 서블릿을 개발을 위해 DCL이 제공하는 C++ 클래스 라이브러리와 디버깅 환경에 대하여 설명하고 마지막으로 5장에서 예제를 통하여 서블릿 모듈의 컴파일과정과 설치에 관한 내용을 설명한다.

- 목 차 -

제 1 장 개요 3
1.1 연구의 배경과 역사 3
1.2 연구의 목적 3
1.3 구성요소 5
1.4 인터페이스 모델 7
제 2 장 인터페이스 레퍼런스 7
2.1 Callback Functions 7
2.2 Data Structure 10
제 3 장 서블릿 매니저 14
3.1 DHE.INI 14
3.2 DHEAdmin 15
3.3 Apache1, Apache2 16
3.4 Microsoft IIS 17
3.5 명령행 18
제 4 장 서블릿 개발환경 19
4.1 서블릿 객체 관련 클래스 19
4.2 HTTP 프로토콜 관련 클래스 21
4.3 HTML 관련 클래스 24
4.4 기타 관련 클래스 24
4.5 서블릿의 DCL Runtime Debugging Support 25
제 5 장 서블릿 예제와 설치 28
5.1 소스파일의 구성 28
5.2 Microsoft Visual C++ 프로젝트 35
5.3 UNIX 버전을 위한 GCC Makefile 36
5.4 서블릿 모듈의 설치 39

제 1 장 개요

1.1 연구의 배경과 역사

DHE(DCL HTTP Server Extension)에 관한 연구의 시작은 2001년 가을에 최초로 시작되었다. 당시에 PHP를 사용하여 웹 메일 시스템을 개발 하던 중 인터프리터 환경의 PHP로 개발된 웹 서버 어플리케이션의 성능의 한계를 인식하게 되었고 대부분의 웹 서버의 서비스들이 특정 시점에 부하가 많이 걸릴 경우 기존의 방법으로 이러한 문제점을 극복하는 데에는 한계가 있음을 인식하고 이에 대한 해결책을 찾게 되었다.

문제 해결을 위한 최초의 결정은 웹 서버 어플리케이션을 아파치 웹 서버의 모듈과 같은 형태로 개발하기로 한 것이었다. 그러나 아파치 서버의 모듈을 개발 하는 것은 결코 쉬운 일이 아니었다. 아파치 서버의 기본 API(Application Programming Interface)는 표준 C 언어로 되어 있었고 이들에 대한 자세한 참조 매뉴얼이 없었기 때문에 모듈 개발을 위해 참고 하기 위해 기존의 모듈과 아파치의 소스 프로그램을 분석할 수 밖에 없었다. 더더욱 문제점이 되었던 것은 아파치 모듈을 개발하는 과정에는 컴파일된 모듈을 테스트 하기 위해 매번 서버를 재시작 해야 하는 번거로움이 있었다. 결국엔 이러한 환경에서 어플리케이션 개발자의 생산성에 관한 제고는 생각할 수 조차 없다는 결론이 내려 졌다.

결국, Apache1, Apache2, IIS와 같은 기존의 웹 서버의 어플리케이션으로 동작을 하지만 웹 서버 어플리케이션 개발을 위하여 웹 서버의 API와는 별도의 인터페이스를 개발하기로 했다. 2002년 4월에 ISAPI와 Apache의 모듈 소스들을 분석하여 최초의 인터페이스를 정의 하였고 이를 사용하여 Apache1, Apache2 모듈로 프로토타입이 개발 되었다.

DHE 2.0에 관한 연구는 2004년 7월에 재개하였다. 기존의 인터페이스는 프로토타입 수준으로 상업적인 어플리케이션을 개발하는 데에는 안정성과 개발환경에 관한 유연성이 부족하다고 판단 되었기 때문에 이들에 대한 보완을 중심으로 새로운 인터페이스를 설계하였다.

본 연구는 2005년 5월까지 지속되었는데 연구기간이 이렇게 오랫동안 지속된 원인은 DHE 자체의 인터페이스에 관한 개발보다도 DHE의 개발 환경이 DCL 프로젝트의 나머지 구성 요소들로부터 많은 부분이 영향을 받기 때문이다. 특히, 이번 버전에서 추가된 가장 중요한 부분은 DCL의 디버그 버전에서 지원하는 실행시간 디버깅 정보를 웹 브라우저를 통해서 확인 할 수 있도록 했다는 것이다.

DHE 2.0부터 DHE 어플리케이션을 위해 “서블릿(Servlet)”이란 용어를 사용한다. 자바 어플리케이션 서버에서 사용되기 시작한 Servlet이란 용어는 “Server”와 조각의 의미를 가진 접미사 “-let”의 합성어로 “서버를 구성하는 작은 어플리케이션”을 의미한다.

1.2 연구의 목적

■ 웹 서버의 가용성 및 성능의 극대화

웹 서버의 가용성과 성능을 극대화 하는 가장 이상적인 방법은 웹 서버의 어플리케이션 개발을 웹 서

버의 API를 사용하여 웹 서버와 동일한 프로세스 환경 내에서 동작하도록 하는 것이다. 그러나, 이러한 방법은 어플리케이션에 치명적인 버그가 존재할 경우 서버 자체의 안정적 동작을 위협할 수 있다는 단점이 있다.

DHE 인터페이스를 만족하는 어플리케이션은 C/C++와 같은 네이티브(native) 바이너리 코드를 생성하는 컴파일러 언어를 사용하여 개발된 동적 공유 객체(DSO: Dynamic Shared Object, DLL: Dynamic-Link Library, so: Shared Object)로 서버의 프로세스 환경 내에 있으면서 서버의 API를 사용하는 것과 동일한 성능을 발휘하도록 함과 동시에 어플리케이션의 예외(exception)들에 대하여 안정적인 동작을 유지하도록 한다.

■ **소스레벨 이식성과 바이너리의 사용**

DCL 프로젝트의 기본 목적은 운영체제 간의 소스레벨 이식성에 있다. 여기에는 DHE도 역시 예외가 아니다.

이식성과 관련하여 DHE의 달성해야 하는 중요한 목표는 동일한 운영체제 하에서는 웹 서버가 다르다 할지라도 같은 바이너리가 사용될 수 있다. 가령 동일한 플랫폼의 Win32 운영체제에서는 서버가 IIS이든 Apache1, 또는 Apache2이든 같은 바이너리가 사용된다.

■ **유연한 운영환경의 제공**

DHE 서블릿의 실행은 HTTP 요청마다 적재(load), 서비스, 제거(unload)를 반복하는 방식과 한번 적재되면 웹 서버 프로세스 내부에 캐시(cache)되어 서비스를 반복하는 방식이 있다.

첫 번째 방식은 서블릿의 개발 단계, 또는 자주 요청 되지 않는 서블릿에 대하여 적용될 수 있다. 서블릿의 개발 단계에서는 서블릿이 서버에 적재된 이 후에는 서블릿 모듈이 변경될 수 없기 때문에 이 방식이 필요하다. 두 번째는 서블릿이 자주 호출되는 정상적인 서비스 방식이라 할 수 있다.

유연한 운영 환경이라 함은 위의 실행 방식과 정책에 대한 변경을 웹 서버를 재구동 하지 않고 실행 시간에 제어하도록 하게 하는 것이다.

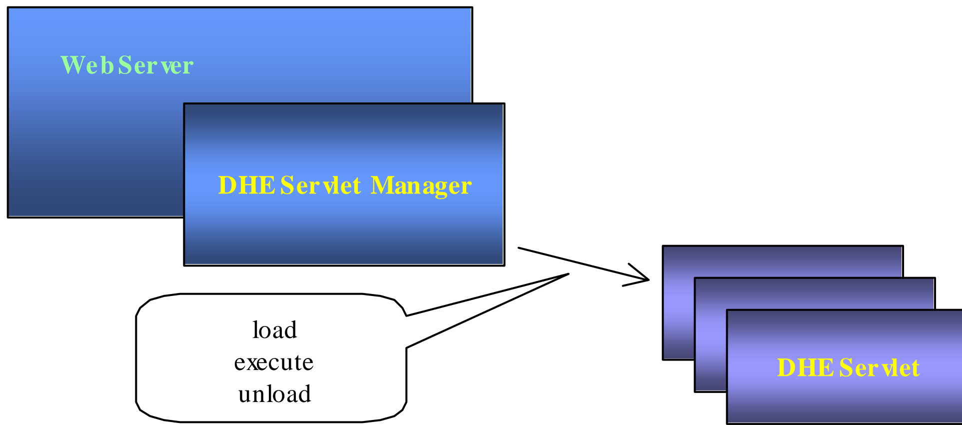
■ **개발단계에서 생산성 향상**

웹 어플리케이션 개발에 있어서 PHP, ASP와 같은 스크립트 언어의 사용은 작성된 코드의 실행 결과를 즉시 확인해 볼 수 있는 장점이 있다. 그러나, 복잡한 웹 어플리케이션의 경우 스크립트간의 의존관계는 실행시켜 보지 않고는 버그를 발견할 수 없는 단점이 있다. 반면에, 컴파일러 언어를 통한 어플리케이션의 개발은 컴파일러를 통하여 모듈이나 함수들 간의 의존관계를 확인 할 수 있다.

표준 C언어를 사용하여 명세화 한 DHE 인터페이스는 서블릿 개발에 다양한 언어를 사용할 수 있도록 하게 함으로써 기존에 개발된 많은 라이브러리들을 그대로 사용할 수 있도록 하여 서블릿 개발에 관한 생산성을 제고할 수 있다.

1.3 구성요소

DHE이 실행환경을 이루는 구성요소는 웹 서버, DHE 서블릿 매니저 그리고 DHE 서블릿의 세 요소로 구성되어 있다.



<그림 1-1> DHE 구성요소

■ 웹 서버

Apache1, Apache2, IIS와 같은 웹 서버를 말한다. DHE는 독립적인 서버로 동작하지 않고 기존의 웹 서버들의 API를 확장하는 형태를 취한다. 이렇게 하는 이유는 기존의 웹 서버들이 널리 사용되고 있고 이들을 사용해서 개발된 웹 어플리케이션을 그대로 사용할 수 있도록 함과 동시에 또 다른 개발 환경을 제공하고자 위함이다.

■ 서블릿 매니저

서블릿 매니저는 웹 서버가 제공하는 API의 프로세스 환경 내에서 동작하는 웹 서버의 어플리케이션으로 IIS에서는 ISAPI Extension의 형태를 취하고 Apache 서버에서는 모듈로 구현된다.

서블릿 매니저의 첫번째 역할은 서블릿 모듈을 적재(load)하여 실행시키고 제거(unload)하는 것이다.

서블릿 모듈을 적재하는 과정에는 서블릿 모듈의 인터페이스 버전과 적법하게 작성되었는지를 검사한다. 서블릿 모듈은 운영체제의 동적 공유객체(DSO)이기 때문에 운영체제의 DSO 로드 함수 호출 중에 바인딩 에러와 같은 문제가 발생할 수 있다. 만약 서블릿 모듈의 로드와 검사에서 에러가 발생하면 서블릿 매니저는 클라이언트인 User-Agent에 이에 관한 내용을 되돌린다.

서블릿 모듈의 적재가 성공적으로 완료되면 서블릿 모듈의 초기화 Callback 함수를 호출하여 서블릿 객체를 초기화 한다. 초기화 까지 성공적으로 마친 서블릿 모듈은 서블릿 매니저가 관리하는 서블릿 풀(pool)에 이를 삽입하고 서블릿 모듈의 HTTP 서비스 콜백 함수 통하여 HTTP 요청을 전달한다.

서블릿의 초기화 콜백과 HTTP 서비스 콜백은 그 함수 내에서 에러가 발생할 수 있고, 이 에러의 원인은 서블릿 매니저에 전달되어 User-Agent에 되돌린다.

서블릿의 적재와 실행 및 제거의 과정에는 DHE.INI에 설정된 서블릿의 실행에 관한 설정이 사용된다. 한번 초기화된 서블릿은 서블릿 풀에 캐시되어 DSO의 로드와 관련된 오버헤드가 제거된 상태에서 HTTP 서비스를 수행할 수 있다. 만일 DHE.INI의 설정이 서블릿의 서비스를 금지시키면 서블릿 매니저는 파일 시스템에 서블릿 모듈이 존재하더라도 User-Agent에 서비스를 거절할 수 도 있다.

서블릿 매니저의 또 다른 중요한 역할은 웹 서버마다 각기 다른 API 차이점을 흡수하여 서블릿 모듈에게 특정 웹 서버로부터 독립적인 API 환경을 제공하는 것이다. 2장에서 언급하고 있는 DHE 인터페이스들 중 DCL_HTTP_SERVER_API가 여기에 해당된다. 서블릿 매니저는 이 인터페이스를 구현하여 서블릿으로 하여금 HTTP 요청을 완료하기 위해 웹 서버의 API에 접근 하도록 한다.

마지막으로, 서블릿 매니저는 실행 시간에 서블릿 매니저의 설정에 관한 변경이 가능 하도록 하는 제어(control) 인터페이스를 제공한다. 이 인터페이스는 단일 프로세스/다중 스레드 웹 서버 뿐만 아니라 다중 프로세스/다중 스레드 웹 서버에서도 동일한 형태의 제어가 가능 하도록 한다.

■ **서블릿**

서블릿은 HTTP 요청을 분석하고 이에 대한 실질적인 서비스를 담당하는 부분으로 운영체제의 동적 공유 객체(DSO: Dynamic Shared Object)로 구현되며 모듈의 파일 확장자는 “.dhe”(디버그 버전에서는 “.dhed”)를 갖는다.

서블릿 매니저로부터 전달 받은 HTTP 요청은 CGI 인터페이스와 관련한 변수와 서버의 환경변수를 제외하고는 가공되지 않은 상태이다. 이 때문에 서블릿은 QUERY_STRING, Cookie 뿐만 아니라 POST 메소드에 동반한 폼데이터(multipart/form-data)를 디코드 해야만 한다.

하나의 서블릿 모듈은 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 그것이 제거 되기 직전에 서블릿 매니저에 의하여 크린업(cleanup)함수가 호출된다.

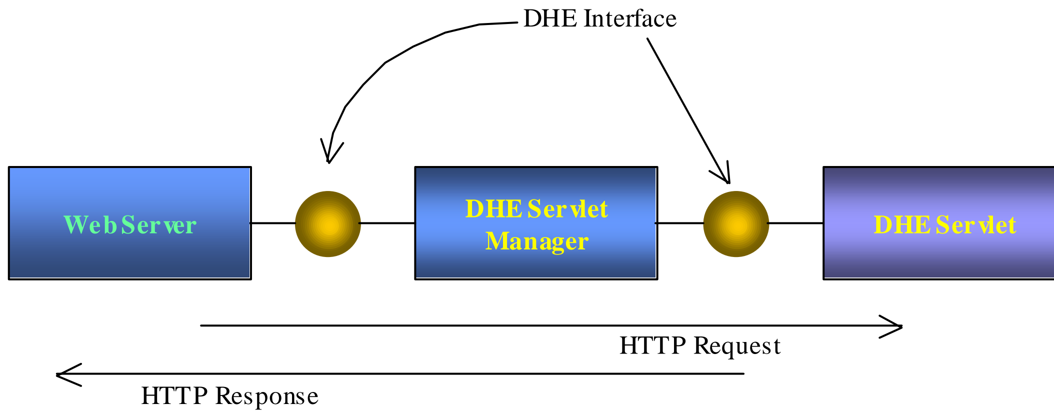
서블릿은 서블릿 매니저로부터 제공받은 서버 API 때문에 다양한 웹 서버로부터의 API 독립성을 보장 받는다. 따라서 동일한 운영체제하에서는 서로 다른 웹 서버라 할지라도 재컴파일을 하지 않고 실행될 수 있다.

하나의 웹 서버 프로세스는 동일한 서블릿 모듈에 대하여 중복되어 적재하지 않는다. 서블릿 객체의 생성 책임은 서블릿 모듈에 있는데 하나의 서블릿 모듈에는 서블릿 객체가 정적(static)이든 동적(dynamic)이든 한 개만 존재해야 한다. 이 때문에 다중 스레드로 서비스 하는 웹 서버에서 하나의 서블릿 객체는 둘 이상의 HTTP 요청에 관한 스레드 상태에 놓일 수 있다.

1.4 인터페이스 모델

■ 역할

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



<그림 1-2> 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 서블릿 모듈의 DCL_HTTP_SERVLET 인터페이스의 DSO 진입점(entry point)은 반드시 표준 C언어 네임 맹글링(name-mangling)을 사용하여야 한다.

■ 특징

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

제 2 장 인터페이스 레퍼런스

2.1 Callback Functions

■ 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. 리턴값이 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.

리턴값이 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 매니저는 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 리턴값이 FALSE이고 *pnLength가 0보다 크면 버퍼의 크기가 작음을 의미한다. 이 경우 *pnLength는 필요한 버퍼의 크기를 의미한다. 리턴값이 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) */
            );
```

CGI(Common Gateway Interface)에서 지정한 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 리턴값이 FALSE이고 *pnLength가 0보다 크면 버퍼의 크기가 작음을 의미한다. 이 경우 *pnLength는 필요한 버퍼의 크기를 의미한다. 리턴값이 FALSE이고 *pnLength가 0이면 pszVarName이 없거나 값이 없는 경우이다.

- DCLHttpWriteStream

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

서블릿과 서블릿 매니저 사이에 스트림 데이터를 주고 받을 때 사용하는 함수 이다.

- DCLHttpServerControl

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

```
DCLHttpWriteStream          pfnWriteResult,
void*                       hResultOutputStream
);
```

실행시간에 서블릿 매니저의 설정을 변경하는데 사용된다.

- DCLHttpServletInitialize

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

DCL_HTTP_SERVLET의 멤버 함수로 서블릿 모듈에서 구현되어 있어야 한다. 서블릿 매니저에 의해 서블릿이 성공적으로 로드되면 최초에 단 한번 불려 진다. 서블릿의 초기화 중에 에러가 발생하면 DCL_HTTP_SERVER_API::pfnReportError에 hReportError을 사용하여 서블릿에 에러 메시지를 전달 할 수 있으며 FALSE를 리턴해야 한다.

- DCLHttpServletCleanup

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

DCL_HTTP_SERVLET의 멤버 함수로 서블릿 모듈에서 구현되어 있어야 한다. 서블릿이 닫혀지기 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.

2.2 Data Structure

- DCL_HTTP_SERVER_API

서블릿 매니저가 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 절대 경로명을 포함하는 서블릿 모듈의 파일명이다. 서블릿은 웹 서버와 동일한 프로세스 환경에 있기 때문에 프로세스의 실행 디렉토리를 변경하는 chdir과 같은 함수를 사용해서는 안되며 pszFileName 으로부터 실행 디렉토리를 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 매니저에 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 DHE의 버전을 나타낸다. 이 값은 dcl/.Config.h에 DCL_HTTP_SERVER_VERSION로 정의되어 있다.

hConn DCL_HTTP_SERVER::pfnReportError를 제외한 모든 DCL_HTTP_SERVER의 멤버를 호출할 때에는 이 값을 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. 이 값은 다음의 값들 중의 하나이다.

```
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
};
```

HTTP 프로토콜이 확장되어 요청된 메소드가 메소드가 OPTIONS, GET, HEAD, POST, PUT, TRACE, CONNECT 중의 하나가 아니면 이 멤버는 HTTP_METHOD_UNKNOWN의 값을 가진다. 이 경우 서블릿은 pszRequestMethod의 값을 검사하여야 한다.

pszRequestMethod HTTP Request-Method에 해당하는 문자열이다.

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

pszQueryString URI의 QUERY_STRING을 나타낸다.

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

uContentLength POST 데이터의 크기이다.

■ DCL_HTTP_SERVLET

서블릿 모듈은 단 하나의 DCL_HTTP_SERVLET 엔트리 포인트를 가져야 하며 서블릿 매니저에 의해 서블릿 모듈이 적재 된 직후 각 멤버의 값들을 검사 한다.

```
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 사이즈를 바이트 단위로 나타낸다.

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

pszBuildTimeStamp 서블릿 모듈이 컴파일된 __TIMESTAMP__ 값이다.

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

uModuleType 모듈의 타입을 나타내며 dcl/.Config.h에서 정의한 DCL_HTTP_SERVLET_MODULE 이어야 한다.

pszDescription 서블릿 모듈에 대한 간략한 설명을 포함하는 문자열이다.

uVersion DHE의 버전을 나타낸다. 이 값은 dcl/.Config.h에 DCL_HTTP_SERVER_VERSION로 정의되어 있다.

pfnInitialize 서블릿 매니저가 서블릿을 적재한 후 서블릿에 초기화 기회를 주기위해 최초 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, 서블릿 모듈에서 전역적으로 사용되는 객체들에 대한 동기화에 대하여 주의하여야 한다.

제 3 장 서블릿 매니저

3.1 DHE.INI

서블릿 매니저는 서블릿 모듈을 적재하고 HTTP 요청을 서블릿 모듈에 전달하는 역할을 수행 하는데 있어서 사용하는 설정 파일이 DHE.INI 이다. 이 파일은 서블릿 매니저 자신에 대한 설정과 서블릿 서비스에 관한 정책을 저장하고 있는 파일로 기본적으로 "dhe.ini" 라는 파일명을 갖는다(디버그 버전의 서블릿 매니저는 "dhed.ini"라는 파일명을 사용한다).

```
 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 3-1> dhe.ini

<그림 3-1> 에서 보이고 있는 DHE.INI 파일은 텍스트 파일로 만들어 지며. ';'나 '#'으로 시작되는 라인이나 비어 있는 라인은 무시된다.

4라인의 CONTROL_ALLOWS은 3.2절의 DHEAdmin을 사용해서 서블릿 매니저의 설정을 변경하게 될 경우 허용되는 클라이언트의 IP 값으로 ','로 구분되는 둘 이상의 값이 될 수 있다.

5라인의 CONTROL_PASSWORD는 DHEAdmin이 서블릿 매니저에 접속할 때 사용되는 패스워드로 패스워드가 설정되면 패스워드의 MD5값을 갖고 그렇지 않으면 비어 있다.

7라인의 LOG_TYPES은 서블릿 매니저가 dhe.log에 저장하는 로그의 타입을 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 abs_path에 해당한다.

filename 서버의 파일시스템에 위치한 서블릿 모듈의 실제 파일명이다.

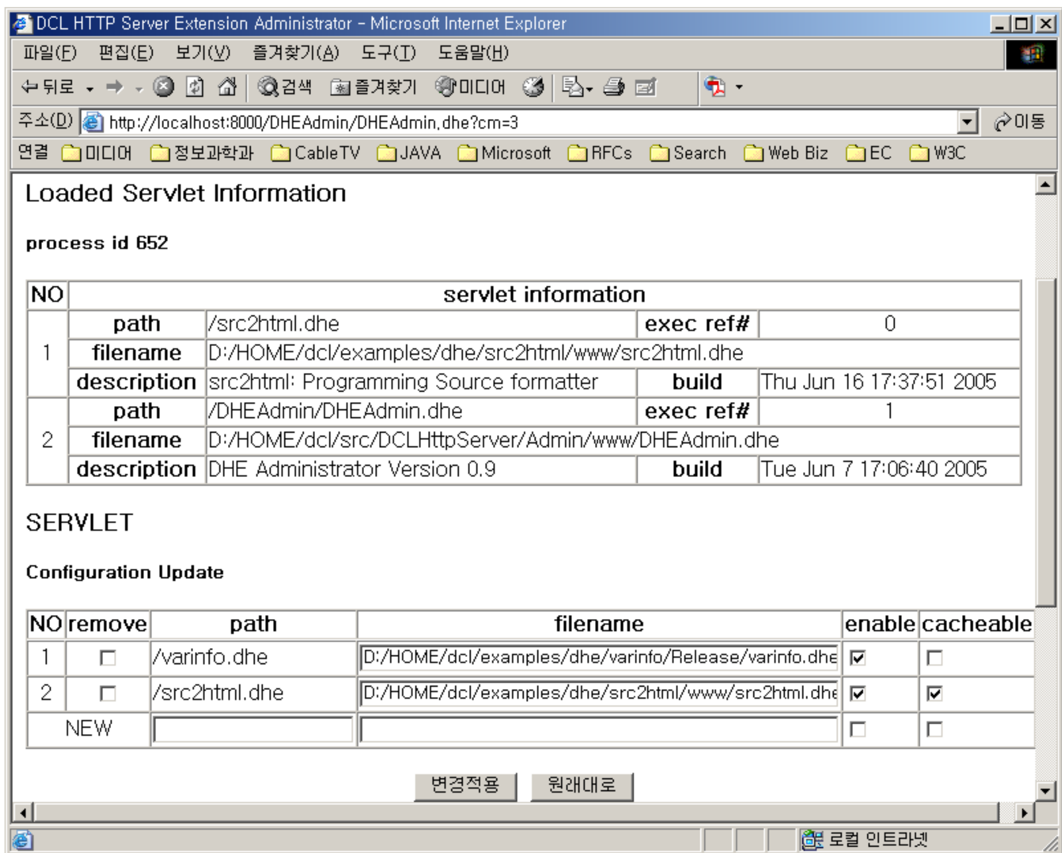
enable 서블릿 서비스의 가능(enable)을 나타낸다. User-Agent가 이 값이 false인 서블릿의 서비스를 요청하면 서블릿 매니저는 오류를 되돌린다.

cacheable 서블릿 모듈의 cache여부를 나타낸다. 이 값이 true이면 서블릿은 한번 초기화 되어 서블릿 매니저의 서블릿 풀에 보관된다.

3.2 DHEAdmin

DHEAdmin은 서블릿 매니저로 하여금 DHE.INI를 실시간으로 수정하도록 하기 위한 서블릿 모듈로 2장에서 설명한 DCL_HTTP_SERVER_API::pfnServerControl을 사용하고 있다.

DHEAdmin을 사용하면 DHE.INI의 모든 설정들에 대한 변경이 가능하지만 이것을 사용하는 가장 중요한 목적은 DHE.INI의 [SERVLET] 섹션과 관련이 있다



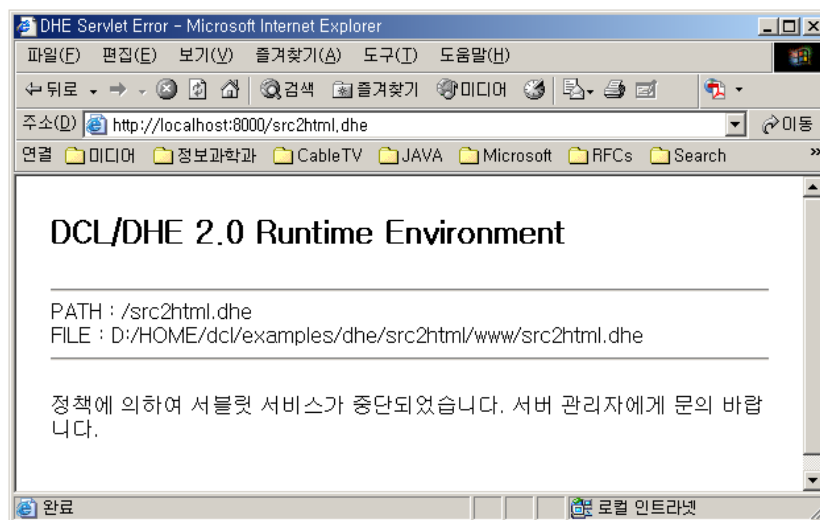
<그림 3-2> DHEAdmin

<그림 3-1>의 16라인을 보면 src2html.dhe는 cacheable 부분이 true로 되어 있다. 이렇게 되어 있는 경우 User-Agent에 의하여 이 서블릿이 요청된 경우 서블릿 모듈에 대하여 다른 조치를 취하지 않는 한 웹 서버 프로세스가 종료될 때까지 서블릿 메니저에 의하여 캐시되어 있게 된다. 서블릿 매니저를 실시간으로 제어할 수 있는 방법이 없는 상황이라면 src2html.dhe 모듈을 변경해야 할 경우 웹 서버를 종료하지 않고는 서블릿 모듈을 교체할 수 없게 된다.

따라서, 웹 서버의 실행 시간에 서블릿 매니저를 제어할 방법이 반드시 필요한 것이다. <그림 3-2>는 src2html.dhe를 호출 한 후 DHEAdmin을 통해 적재된 서블릿의 상태를 보여주고 있다. 이것을 실행하고 있는 서버는 Apache2/Win32인데 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 매니저의 풀에서 src2html.dhe를 제거하려면 cacheable의 체크박스의 체크를 제거하고 변경사항을 적용하면 된다. 서블릿 매니저는 DHEAdmin의 이 요청을 수행하는데 있어서 src2html.dhe의 실행참조카운터를 확인하여 참조카운터가 0이면 즉시 서블릿 풀에서 제거한다.

<그림 3-3>은 src2html.dhe의 enable과 cacheable을 모두 false로 적용한 후 src2html.dhe 요청에 대하여 서블릿 매니저의 서비스 거부 메시지 이다.



<그림 3-3> 서블릿 메니저의 서비스 거부

3.3 Apache1, Apache2

Apache 웹 서버에서 DHE 서블릿 매니저의 실제 구현은 Microsoft의 윈도우와 GNU/Linux와 같은 UNIX계열의 운영체제에서 많은 차이점이 있다. 윈도우에서 Apache1과 Apache2는 단일 프로세스 다중 스레드 상태에서 서비스를 하고, UNIX에서는 다중 프로세스 상태에서 소켓을 공유하며 서비스를 하게 된다. 이러한 차이점 때문에 서블릿 매니저의 서블릿 제어와 관련된 부분은 운영체제에 직접적으로 의존되어 구현되게 된다.

아파치 서버의 버전과 운영체제별 구현의 차이점에도 불구하고 설치 방법은 윈도우와 UNIX 모두 동일하다.

아파치 서버의 서블릿 매니저는 아파치 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.

아파치 서버에서 서블릿 매니저 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 3 -4> httpd.conf

<그림 3-4>는 DHE 서블릿 매니저를 위한 httpd.conf의 추가 내용이다.

DHE_INI와 DHE_LOG 지시자가 생략될 경우에는 각각 “conf/dhe.ini”, “logs/dhe.log”가 기본값으로 설정된다. AddHandler 지시자는 파일의 확장자가 .dhe일 경우 HTTP 요청을 DHE 서블릿 매니저에 전달해 준다.

DCL 프로젝트의 구성요소와 마찬가지로 DHE 서블릿 매니저도 디버그 버전과 릴리즈 버전이 존재하는데 이는 디버그 버전으로 빌드된 서블릿 모듈은 디버그 버전의 서블릿 매니저에 의해서만 서비스 되도록 했기 때문이다. <그림 3-4>는 디버그 버전의 서블릿 매니저를 위한 설정 내용을 포함하고 있는데 이 내용은 윈도우 버전의 Apache1,2에서 그대로 적용될 수 있으나 GNU/Linux에서는 둘 중 하나만 설정해야 한다. GNU/Linux에서 아파치 서블릿 매니저는 공유 라이브러리의 심볼 충돌에 관한 문제 때문에 같은 프로세스 내에서 동시에 사용될 수 없다.

아파치 서버에서 서블릿 매니저의 초기화와 종료에 관한 내용은 logs/error에 기록된다.

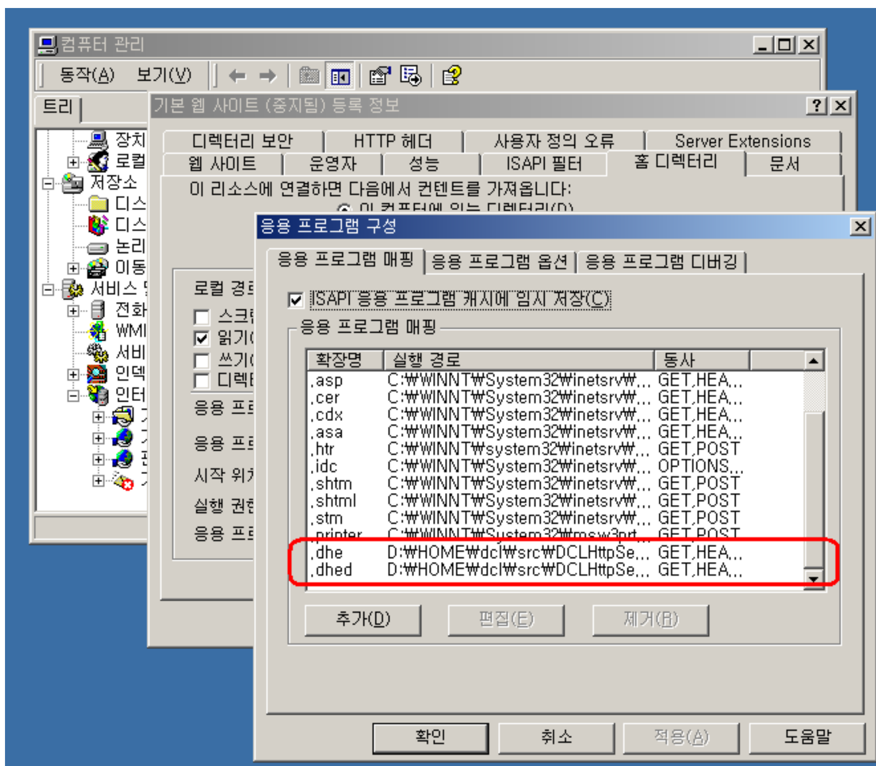
3.4 Microsoft IIS

IIS에서 사용되는 DHE 서블릿 매니저는 “dhe.dll”(디버그 버전은 “dhed.dll”)이다. 서블릿 매니저의 설치는 서블릿 매니저를 위한 레지스트리 내용을 <그림 3-5>의 예와 같이 설정해 주고 <그림 3-6>와 같이 IIS에 응용프로그램을 등록하면 DHE 서블릿을 사용할 수 있다.

IIS에서 서블릿 매니저의 초기화와 종료에 관한 정보는 윈도우의 이벤트 로그의 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 3-5> IIS에서의 서블릿 매니저의 레지스트리 내용



<그림 3-6> IIS 응용프로그램 구성

3.5 명령행

명령행 서블릿 매니저는 윈도우의 콘솔이나 UNIX의 쉘 상태에서 서블릿을 실행하도록 하는 유틸리티 성격의 프로그램이다. 이것은 Microsoft Visual C++의 디버거와 같은 소스 프로그램의 실행 상태를 추적하면서 디버깅을 하도록 하는 환경에서 유용하게 사용될 수 있다.

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

```
명령 프롬프트
D:₩HOME₩dcl₩src₩DCLHttpServer₩Cmd₩Release>dhe
USAGE: dhe dhe_servlet [QUERY_STRING]
        ex) dhe foo.dhe aa=1&bb=2

D:₩HOME₩dcl₩src₩DCLHttpServer₩Cmd₩Release>
D:₩HOME₩dcl₩src₩DCLHttpServer₩Cmd₩Release>dhe "/HOME/dcl/examples/dhe/MyServlet/
Release/MyServlet.dhe"
Info:  Startup success. Using "", ""
200 OK
Content-Length: 148
Content-Type: text/html; charset=euc-kr

<html>
<head><title>MyServlet HTTP Service</title></head>
<body>
<h1>MyServlet HTTP Service</h1>
<hr/>
<p>Hello World !</p>
</body>
</html>
Info:  Shutdown success

D:₩HOME₩dcl₩src₩DCLHttpServer₩Cmd₩Release>
```

<그림 3-7> 명령행 서블릿 매니저

제 4 장 서블릿 개발환경

서블릿을 개발하는데 있어서 다양한 언어를 사용할 수 있지만 서블릿 매니저가 HTTP 프로토콜에 관한 모든 부분을 처리해 주지 않기 때문에 상당 부분의 HTTP 프로토콜 관련 코드를 작성해야 한다.

DCL 프로젝트의 DCLNet 라이브러리는 서블릿을 구현하는데 필요한 관련 HTTP 프로토콜 클래스, HTML 클래스 그리고 서블릿 클래스를 사용한 프레임 워크를 제공한다.

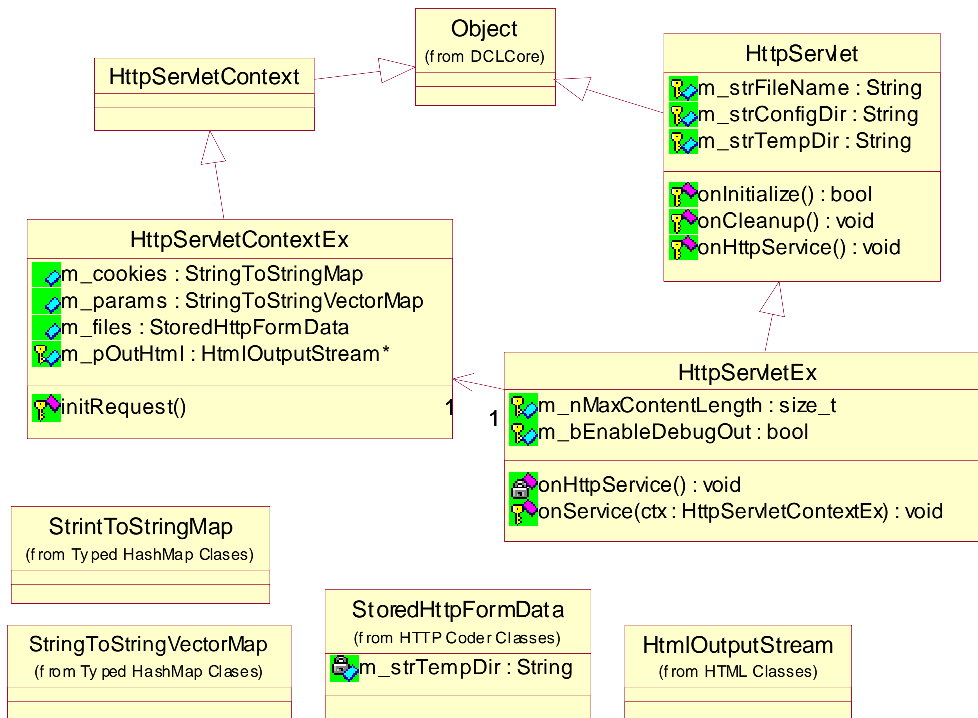
4.1 서블릿 객체 관련 클래스

■ class HttpServlet

DCLNet을 사용하여 서블릿 모듈을 개발할 때에는 HttpServlet 클래스를 상속하여 구현한 단 한 개의 클래스가 있어야 하고 실행시간에는 그 클래스의 인스턴스가 단 한 개 있어야 한다. HttpServlet 클래스는 모든 서블릿 클래스의 추상클래스로 서블릿 모듈의 프레임워크를 구성 하도록 한다.

멤버변수 m_strFileName, m_strConfigDir, m_strTempDir은 onInitialize가 호출되기 전에 프레임워크에 의하여 초기화 되어진다.

이 클래스는 세 개의 가상(virtual) 메소드를 가지고 있는데 서블릿을 개발할 때 이들을 오버라이드 하여 구현하게 된다. 이들 중 onInitialize는 서블릿 객체가 생성된 직후 이를 초기화 하기위해 호출되는데 성공하면 true를 그렇지 않으면 false를 리턴해야 한다. 만약 false를 리턴하게 되면 서블릿의 서비스는 사용되지 않는다.



<그림 4-1> Servlet and ServletContext

HTTP_SERVLET_INSTANCE(ServletClass, Description) 매크로는 DCL_HTTP_SERVLET 인터페이스와 서블릿 클래스를 연결해 줌과 동시에 서블릿 모듈의 DSO 엔트리 포인트를 만들어 준다.

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 서블릿 모듈을 개발하고 할 때에는 이 클래스를 사용한다. 이 클래스의 주요 목적은 디버그 버전(__DCL_DEBUG가 정의된)의 서블릿이 생성한 HTML 스트림에 __DCL_TRACE의 출력을 추가하는 기능을 한다. 이것은 디버그 버전에서 m_bEnableDebugOut이 true이고 HttpServletContextEx의 HtmlOutputStream 객체를 사용했을 때 가능하다.

이 클래스를 상속받은 클래스는 onService 순수가상함수(pure virtual method)를 오버라이드 해야 하며 HttpServlet::onHttpService는 사용할 수 없다. 이 클래스의 onHttpService 내부에서는 onService를 호출 하기전에 HttpServletContextEx::initRequest를 호출하여 HTTP 요청에 관한 파라미터들을 초기화 한다.

m_nMaxContentLength는 POST 데이터의 최대 크기를 나타내며 0 일 경우 제한을 두지 않는다. POST 데이터가 m_nMaxContentLength보다 크게 되면 POST 데이터는 디코드 되지 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 사용되며 HTTP 요청과 함께 동반한 Cookie, QUERY_STRING, POST 메소드에 의한 데이터 등은 HttpServletEx::onService가 호출되기 전에 초기화되어 진다. m_cookies, m_params, m_files 멤버에 이들에 대한 데이터를 보관하게 되는데 m_params은 QUERY_STRING뿐만 아니라 POST 데이터에서 파일을 제외한 모든 값들이 함께 보관된다.

4.2 HTTP 프로토콜 관련 클래스

■ class URLEncoder, class URLDecoder

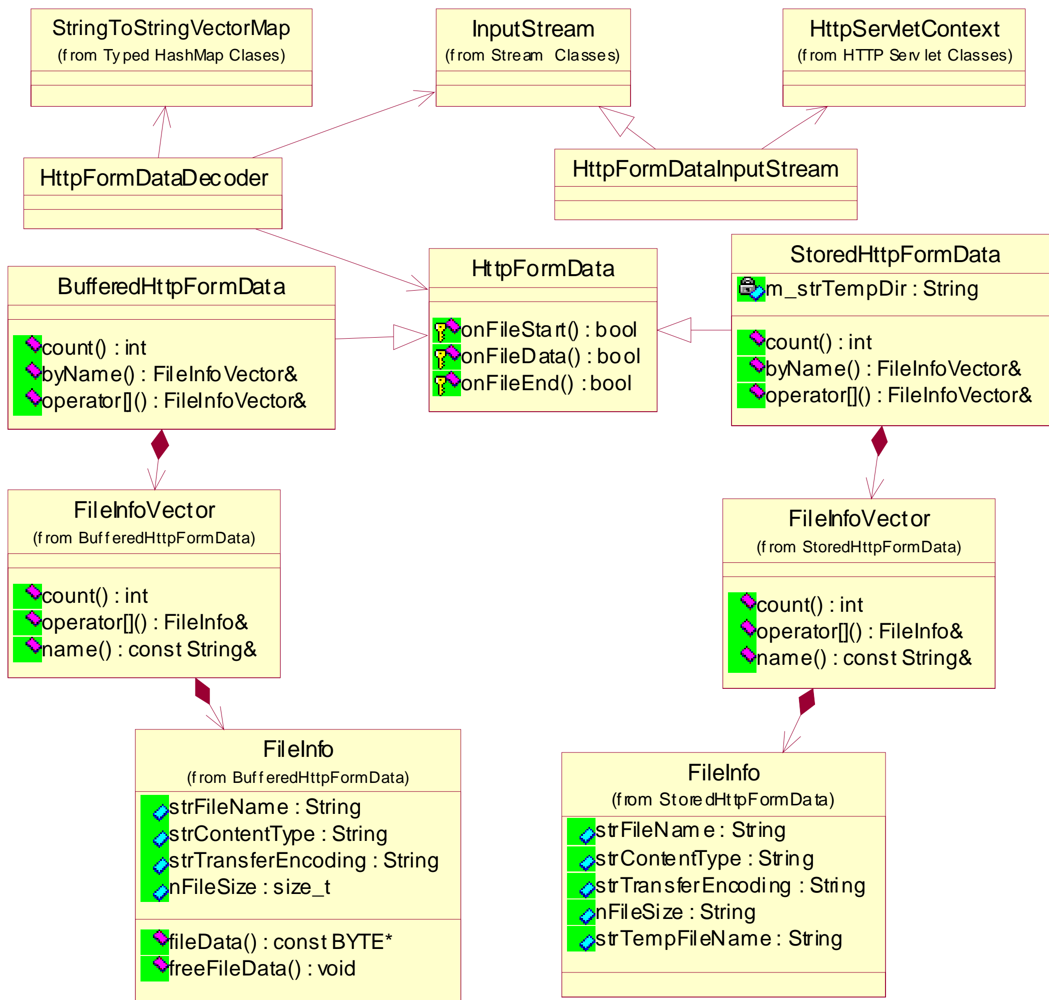
URI(RFC2396)에 포함될 수 없는 문자에 대하여 문자열 변환을 한다.

■ class HttpCookieDecoder

User-Agent로부터 받은 HTTP Header-Field 중 Cookie를 디코드 하여 StringToStringMap에 보관한다.

■ class HttpQueryStringDecoder

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



<그림 4 -2> HTTP POST Data Decode

■ class HttpFormData

"multipart/form-data"의 MIME 타입으로 된 POST 데이터에 포함된 파일데이터의 디코드 결과를 보관하기 위한 콜렉션 클래스의 추상 클래스 이다. 이 클래스의 객체를 적용하여 데이터를 디코드 하게 되면 파일 데이터는 버려진다. 파일 데이터를 사용하고자 하면 이 클래스로부터 파생된 StoredHttpFormData나 BufferedHttpFormData 클래스를 사용하여야 한다.

■ class StoredHttpFormData

"multipart/form-data"의 파일데이터를 주어진 디렉토리의 임시파일에 저장한다. 객체가 파괴될 때 이들 임시파일이 존재하면 파일을 삭제하므로 파일을 계속 사용하고자 하면 파일을 다른 곳으로 옮겨 놓아야 한다.

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 이를 해결하기

위해 멤버로 구현된 FileInfoVector와 FileInfo 클래스를 두고 있다. 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_files.isEmpty())
{
	String toDir = File::getDirName(m_strFileName);
	for(int i = 0; i < ctx.m_files.count(); i++)
	{
		StoredHttpFormData::FileInfoVector& v = ctx.m_files[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 보관하고 있는다. 데이터를 위해 할당된 버퍼를 해제(free)하려면 BufferedHttpFormData::FileInfo::freeFileData를 호출하면 된다.

■ class HttpFormDataDecoder

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

디코드 과정에서 IOException* 예외가 던져질(throw) 수도 있으며 그 외의 데이터 오류와 같은 경우 이에 대한 메시지는 HttpFormDataDecoder::warnings 을 통해 확인할 수 있다.

■ class HttpFormDataInputStream

HttpFormDataDecoder를 위하여 HttpServletContext에 대하여 InputStream 인터페이스를 구현한다.

■ class HttpHeader, class HttpSetCookie

HTTP 응답헤더(Response-Header)를 표현하는 문자열을 생성한다.

4.3 HTML 관련 클래스

■ class HtmlEntityEncoder

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

■ class HtmlOutputStream

DCLCore의 ByteBufferOutputStream으로부터 상속되어 있고 출력 문자열에 대하여 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 버퍼링하여 스트림을 유지한다.

■ class HtmlTemplate

이 클래스는 HTML 페이지를 표현하는데 있어서 디자인 부분과 프로그램에 의하여 변화하는 데이터의 분리를 가능하게 해주는 클래스 이다. 이 클래스에 대한 자세한 내용은 별도의 문서로 정리되어 있다.

4.4 기타 관련 클래스

■ class StringToStringMap

DCLCore에 포함되어있는 클래스로 해시함수를 사용하여 문자열을 연관 짓는다. 이 클래스는 User-Agent로부터 전송 받은 HTTP Header-Field의 Cookie의 디코드 결과를 저장하는데 사용된다.

■ class StringToStringVectorMap

하나의 이름에 하나 이상의 문자열을 포함하는 StringVector를 연관 짓는다. 이 클래스는 QUERY_STRING과 FORM_DATA의 디코드 결과를 저장하는데 사용된다.

다음은 QUERY_STRING의 디코드 결과를 보여주는 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
```

4.5 서블릿의 DCL Runtime Debugging Support

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 이루어 지고 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 해프닝이 일어나기도 한다.

DCL은 소스코드 차원에서 디버깅을 위한 코드 삽입을 자동화 하여 소프트웨어 개발 단계 및 테스트 단계에서 필요한 정보를 생성하도록 하는 런타임 환경을 제공하는데 이것을 사용하면 별도의 디버거를 사용하지 않아도 프로그램의 실행에 관련된 변수들의 값을 쉽게 확인할 수 있을 뿐만 아니라 동일한 코드를 릴리즈로 빌드하게 되면 디버깅을 위해 소스에 삽입되었던 코드들은 제거되는 강력한 기능이다.

- AssertException

DCL의 __DCL_ASSERT의 동작은 두 가지 형태의 결과를 얻는다. 첫 번째는 표준 C 런타임 라이브러리에 포함되어 있는 assert처럼 표현식에 대한 문자열을 출력하고 abort 시스템콜을 호출하여 프로세스를 종료하는 것이다. 그러나 이 형태는 서블릿에 적용될 수 없다. 왜냐하면 서블릿은 이미 웹 서버 프로세스의 일부분 이기 때문에 서블릿에서 abort 해 버리면 웹 서버의 서비스가 중단되기 때문이다

두 번째는 AssertException 예외를 던지는 형태이다. 서블릿에서 모든 ASSERT 실패는 abort 하지 않고 AssertException 예외를 발생한다. 이렇게 함으로써 서블릿의 실행 중에 예외가 발생하더라도 웹

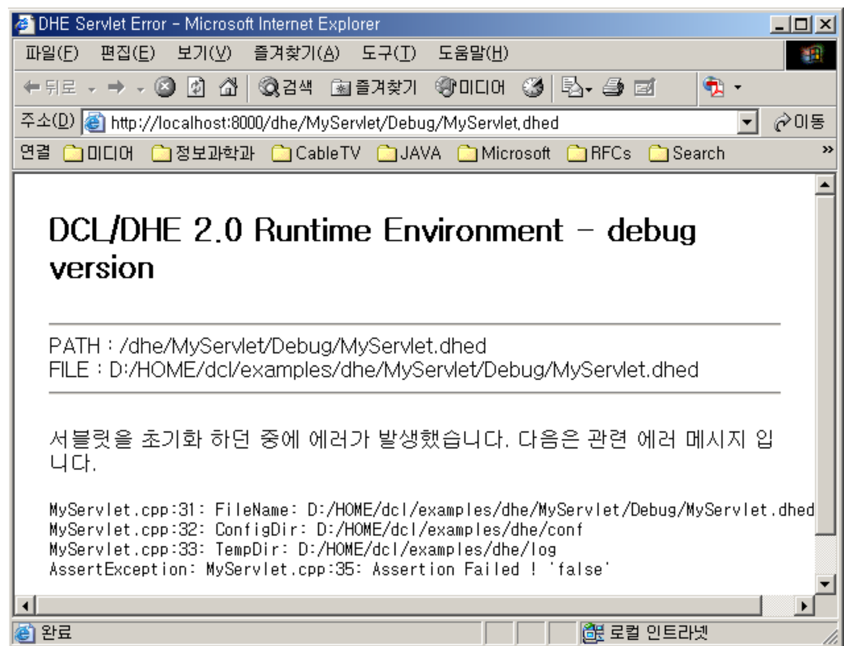
서버의 프로세스 실행에는 영향을 주지 않는다.

■ 서블릿 매니저의 에러표시

DCL_HTTP_SERVLET 인터페이스의 메소드는 성공하면 TRUE를 그렇지 않으면 FALSE를 리턴하도록 되어 있다. 만약 실패할 경우 전달 받은 에러 스트림(hReportError)핸들을 사용하여 에러 메시지를 되돌릴 수 있는데 pfnInitialize와 pfnHttpService 메소드에서 되돌린 메시지는 서블릿 매니저는 이를 User-Agent로 되돌린다. 만약 User-Agent가 웹 브라우저라면 사용자는 이에 대한 메시지를 볼 수 있게 되는 것이다.

```
23:  bool MySerlvet::onInitialize()
24:                                     __DCL_THROWS1(Exception*)
25:  {
26:  #ifdef __DCL_DEBUG
27:         // HttpServletEx 의 디버깅 정보 출력을 불가능하게 한다.
28:         // 기본값은 true 이다.
29:         // m_bEnableDebugOut = false;
30:  #endif
31:         __DCL_TRACE1("FileName: %s\n", m_strFileName.cstr());
32:         __DCL_TRACE1("ConfigDir: %s\n", m_strConfigDir.cstr());
33:         __DCL_TRACE1("TempDir: %s\n", m_strTempDir.cstr());
34:
35:  //         __DCL_ASSERT(false);
36:
37:         return true;
38:  }
```

<그림 4-3> MySerlvet.cpp



<그림 4-4> 서블릿 매니저의 에러표시

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

<그림 4-3>의 소스 프로그램은 디버그 버전에서 35번 라인의 주석을 제거하면 AssertException이 발생하게 되고 초기화는 실패하게 된다. <그림 4-4>는 이에 대한 결과를 보여 주고 있다. 만약 35번 라인을 주석처리 하고 37번 라인에서 false를 리턴하게 되면 다음 그림에서 예외가 발생한 부분만 제외된다.

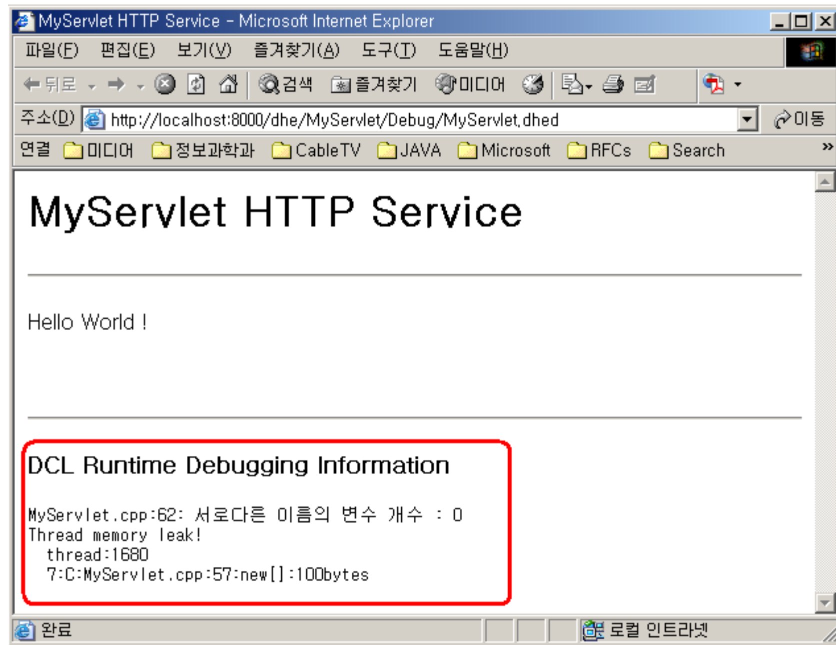
■ HttpServletEx의 디버깅 정보 표시

HttpServletEx 클래스를 상속하여 서블릿을 구현하면 서블릿 실행 결과 뿐만 아니라 서블릿 실행 과정의 __DCL_TRACE의 내용과 해제하지 않은 동적 메모리의 할당 내용이 함께 출력 된다. 이것의 기능은 서블릿 객체가 생성한 CONTENT_TYPE이 “text/*”인 상태에서 HttpServletContextEx가 제공하는 스트림 객체를 사용했을 때 가능하다.

```
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_params.count()
62:              );
63:  }
```

<그림 4 -5> MyServlet.cpp

HttpServletContextEx의 스트림 객체를 사용하는 방법은 두 가지 이다. 첫 번째는 이 클래스의 getHtmlOutputStream() 메소드를 호출하여 디폴트 파라미터를 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 조정하고자 하면 createOutputStream 메소드를 사용한다.



<그림 4-6> HttpServletEx의 디버깅 정보 표시

<그림 4-6>은 <그림 4-5>코드를 포함하고 있는 서블릿을 실행한 결과이다. 57 라인은 new[] 연산자를 사용하여 메모리 블록을 할당 했는데 이를 해제하지 않았다는 내용과 62 라인의 __DCL_TRACE의 내용을 표시하고 있다.

HttpServletEx는 onService에서 발생한 예외(exception)에 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.

제 5 장 서블릿 예제와 설치

5.1 소스파일의 구성

■ namespace

C++ 프로그램을 작성하는데 있어서 간단한 프로그램은 단일 파일로 작성할 수도 있지만 프로그램이 커지고 복잡해지면 클래스의 선언(declare)을 기술하는 헤더파일(.h)과 이것을 구현(implement)하는 파일을 별도로 분리할 수 있다. 지극히 당연한 내용을 굳이 언급하는 이유는 DCL이 C++의 namespace를 사용하고 있기 때문이다. DCL의 모든 라이브러리들은 __DCL_HAVE_NAMESPACE를 사용하여 컴파일타임에 namespace의 사용 여부를 결정하는데 기본적으로 namespace를 사용하도록 되어 있다.

다음은 dcl/.Config.h에 정의된 namespace를 위한 매크로이다.

```
#if __DCL_HAVE_NAMESPACE
        namespace dcl { }              /* dummy declaration */
        #define __DCL_NAMESPACE                     dcl::
        #define __DCL_USING_NAMESPACE           using namespace dcl;
        #define __DCL_BEGIN_NAMESPACE           namespace dcl {
        #define __DCL_END_NAMESPACE   }
#else
        #define __DCL_NAMESPACE
        #define __DCL_USING_NAMESPACE
        #define __DCL_BEGIN_NAMESPACE
        #define __DCL_END_NAMESPACE
#endif
```

■ 단일파일 구성 : MySerlvet

다음은 4.5에서 부분적으로 표시했던 것으로 그것의 전체이다.

```
 1:  // MyServlet.cpp
 2:
 3:  #include <dcl/net/HttpServletEx.h>
 4:
 5:  #if __DCL_HAVE_THIS_FILE__
 6:  #undef __THIS_FILE__
 7:  static const char* __THIS_FILE__ = "MyServlet.cpp";
 8:  #endif
 9:
10:  __DCL_USING_NAMESPACE
11:
12:  class MyServlet : public HttpServletEx
13:  {
14:  protected:
15:        virtual bool onInitialize()
16:                                    __DCL_THROWS1(Exception*);
17:        virtual void onService(HttpServletContextEx& ctx)
18:                                    __DCL_THROWS1(Exception*);
19:  };
20:
21:  HTTP_SERVLET_INSTANCE(MyServlet, "MyServlet HTTP Service")
22:
23:  bool MyServlet::onInitialize()
24:                                    __DCL_THROWS1(Exception*)
25:  {
26:  #ifdef __DCL_DEBUG
27:        // HttpServletEx 의 디버깅 정보 출력을 불가능하게 한다.
28:        // 기본값은 true 이다.
29:        // m_bEnableDebugOut = false;
30:  #endif
31:        __DCL_TRACE1("FileName: %s\n", m_strFileName.cstr());
32:        __DCL_TRACE1("ConfigDir: %s\n", m_strConfigDir.cstr());
33:        __DCL_TRACE1("TempDir: %s\n", m_strTempDir.cstr());
34:
35:  //        __DCL_ASSERT(false);
36:
37:        return true;
```

```
38:  }
39:
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_params.count()
62:                );
63:  }
```

<그림 5-1> MyServlet.cpp

<그림 5-1> MyServlet.cpp의 3라인에 포함된 HttpServletEx.h은 HttpServletEx 클래스를 사용하기 위해서 이다. HttpServletEx.h에는 HttpServletEx 클래스와 HttpServletContextEx 클래스가 선언되어 있으며 이 파일 내부에는 관련된 헤더 파일들이 포함되어 있다.

5-8라인은 __DCL_ASSERT, __DCL_TRACE, __DCL_THROW에서 사용되어질 __THIS_FILE__을 정의하고 있다. 이것을 정의하지 않으면 __THIS_FILE__은 __FILE__로 대치되기 때문에 디버그 버전에서 표시되는 __DCL_TRACE의 파일명은 절대경로로 표시된다.

10라인은 라이브러리가 namespace을 사용하여 컴파일 되었을 경우 “using namespace dcl;”로 대치된다.

12-19라인은 MyServlet 클래스를 선언하고 있다. 오버라이드(override)된 두 멤버 함수의 프로토타입 마지막에 __DCL_THROWS1(Exception*) 이 추가되어 있는데 이것은 이 함수들이 Exception 클래스로부터 파생된 예외 객체를 던질(throw) 수 있으며 이 함수를 호출하는 쪽에서 이들에 대한 처리(handling)를 하고 있다는 의미이다.

21라인은 MyServlet의 인스턴스를 정의하고 있다. HTTP_SERVLET_INSTANCE 매크로는 서블릿 객체를 만들고 서블릿 매니저로부터 접근하도록 하는 인터페이스를 만든다. 단일 서블릿 모듈에서 HTTP_SERVLET_INSTANCE 매크로의 사용은 단 한번만 있어야 한다. 만약 하나 이상이 있을 경우 링크에러가 발생한다.

■ 다중파일 구성 : varinfo

헤더파일과 구현파일로 분리하여 다중 파일로 된 프로젝트를 구성할 때에는 단일 파일로 구성할 때와 커다란 차이점은 없으며 namespace와 관련한 부분만 해결해 주면 된다. 이는 namespace의 지시 부분을 잘못 하면 컴파일러가 식별자를 찾을 수 없다는 오류를 보고하기 때문이다.

다음은 다중파일 구성을 보여주는 예제이다.

```
 1:  // varinfo.h
 2:
 3:  #ifndef __DCL_HTTP_VARINFO_H__
 4:  #define __DCL_HTTP_VARINFO_H__
 5:
 6:  #include <dcl/net/HttpServletEx.h>
 7:
 8:  __DCL_BEGIN_NAMESPACE
 9:
10:  class VarInfoServlet : public HttpServletEx
11:  {
12:          DECLARE_CLASSINFO(VarInfoServlet)
13:  protected:
14:          virtual void onService(HttpServletContextEx& ctx)
15:                                  __DCL_THROWS1(Exception*);
16:  };
17:
18:  __DCL_END_NAMESPACE
19:
20:  #endif      // __DCL_HTTP_VARINFO_H__
```

<그림 5-2> varinfo.h

<그림 5-2>의 varinfo.h는 HttpServletEx 클래스를 상속받아 VarInfoServlet 클래스를 선언하고 있다 이 프로그램은 단순히 HttpServletContextEx의 멤버들의 내용을 보여주기 때문에 HttpServletEx클래스의 onService 가상함수만 오버라이드 하고 있다.

8라인과 18라인은 namespace의 범위를 지정하고 있다. 만약에 이것을 지정하지 않으면 <그림 5-3>의 varinfo.cpp의 15라인 직전에 __DCL_USING_NAMESPACE를 삽입하고 22라인과 143라인을 삭제해야 한다. 그 외에는 <그림 5-1>의 MyServlet.cpp와 동일하다.

```
 1:  // varinfo.cpp
 2:
 3:  #include <time.h>
 4:  #ifdef _WINDOWS
 5:  #include <process.h>
 6:  #else
 7:  #include <unistd.h>
 8:  #endif
 9:
10:  #include <dcl/core/Numeric.h>
```

```
11:  #include <dcl/core/File.h>
12:  #include <dcl/core/FileStream.h>
13:  #include <dcl/core/MD5.h>
14:
15:  #include "varinfo.h"
16:
17:  #if __DCL_HAVE_THIS_FILE__
18:  #undef __THIS_FILE__
19:  static const char* __THIS_FILE__ = "varinfo.cpp";
20:  #endif
21:
22:  __DCL_BEGIN_NAMESPACE
23:
24:  HTTP_SERVLET_INSTANCE(VarInfoServlet, "Show HTTP Variable Information")
25:
26:  IMPLEMENT_CLASSINFO(VarInfoServlet, HttpServletEx)
27:
28:  void VarInfoServlet::onService(
29:                      HttpServletContextEx& ctx
30:                      ) __DCL_THROWS1(Exception*)
31:  {
32:      ctx.setContentType("text/plain", "euc-kr");
33:
34:      String strSessionID = ctx.m_cookies["session_id"];
35:      if (strSessionID.isEmpty())
36:      {
37:          String str = ctx.remoteAddr();
38:          str += UInt32::toString(ctx.remotePort());
39:          str += UInt32::toString((UINT32)time(NULL));
40:          strSessionID = MD5::final(str);
41:
42:          HttpSetCookie cookie(
43:                      "session_id",
44:                      strSessionID,
45:                      0,      // time(NULL) + 3600,
46:                      "/"
47:                      );
48:          ctx.addHeader(cookie);
49:      }
50:
51:      HtmlOutputStream& out = ctx.getOutputStream();
52:
53:      out << "PID : " << getpid() << "\n";
54:
55:      out << "BEGIN SERVER CONFIG\n";
56:      out << "fileName : " << m_strFileName << "\n";
57:      out << "configDir : " << m_strConfigDir << "\n";
58:      out << "END SERVER CONFIG\n\n";
59:
60:      out << "BEGIN CONTEXT INFO\n";
61:      if (ctx.remoteAddr())
62:          out << "remoteAddr : " << ctx.remoteAddr() << "\n";
63:      out << "remotePort : " << (int)ctx.remotePort() << "\n";
64:      if (ctx.method())
65:          out << "method : " << ctx.method() << "\n";
66:      out << "remoteMethodNo : " << (int)ctx.methodNo() << "\n";
67:      if (ctx.path())
68:          out << "path : " << ctx.path() << "\n";
69:      if (ctx.queryString())
```

```
70:             out << "queryString : " << ctx.queryString() << "\n";
71:         if (ctx.contentType())
72:             out << "contentType : " << ctx.contentType() << "\n";
73:         out << "contentLength: " << ctx.contentLength() << "\n";
74:         out << "END CONTEXT INFO\n\n";
75:
76:         out << "BEGIN CGI VARIABLES\n"
77:             << ctx.getCgiVariable(NULL)
78:             << "END CGI VARIABLES\n\n";
79:
80:         out << "BEGIN REQUEST HTTP HEADERS\n"
81:             << ctx.getHttpHeader(NULL)
82:             << "END REQUEST HTTP HEADERS\n\n";
83:
84:         out << "BEGIN REQUEST PARAMS - QUERY_STRING and FORM_DATA\n";
85:         if (!ctx.m_params.isEmpty())
86:         {
87:             StringToStringVectorMap::Iterator it = ctx.m_params.begin();
88:             for( ; it != ctx.m_params.end(); it++)
89:             {
90:                 out << (*it).key << ":";
91:                 StringVector& v = (*it).value;
92:                 for(int i = 0; i < v.count(); i++)
93:                 {
94:                     if (i > 0)
95:                         out << ", ";
96:                     out << v[i];
97:                 }
98:                 out << "\n";
99:             }
100:        }
101:        out << "END REQUEST PARAMS\n\n";
102:
103:
104:        out << "BEGIN COOKIES\n";
105:        if (!ctx.m_cookies.isEmpty())
106:        {
107:            StringToStringMap::Iterator it = ctx.m_cookies.begin();
108:            for(; it != ctx.m_cookies.end(); it++)
109:            {
110:                out << (*it).key << ":" << (*it).value << "\n";
111:            }
112:        }
113:        out << "END COOKIES\n\n";
114:
115:        out << "BEGIN FILES\n";
116:        if (!ctx.m_files.isEmpty())
117:        {
118:            String toDir = File::getDirName(m_strFileName);
119:            for(int i = 0; i < ctx.m_files.count(); i++)
120:            {
121:                StoredHttpFormData::FileInfoVector& v = ctx.m_files[i];
122:                out << v.name() << ":";
123:                for(int j = 0; j < v.count(); j++)
124:                {
125:                    StoredHttpFormData::FileInfo& info = v[j];
126:                    out << "\n\tfilename: " << info.strFileName
127:                        << "\n\tfilesize: " << info.nFileSize
128:                        << "\n\tContent-Type: " << info.strContentType
```

```
129:                                 << "\n\tContent-Transfer-Encoding: " << info.strTransferEncoding
130:                                 << "\n\ttemp filename: " << info.strTempFileName
131:                                 << "\n";
132:
133:                         File::rename(
134:                                 info.strTempFileName,
135:                                 toDir + info.strFileName
136:                                 );
137:                 }
138:         }
139:     }
140:     out << "END FILES\n\n";
141: }
142:
143: __DCL_END_NAMESPACE
```

<그림 5 -3> varinfo.cpp

32라인은 이 서블릿이 생성할 CONTENT_TYPE를 지정하고 있다. 이것은 나중에 HTTP Response-Header에 추가된다.

34-49라인은 세션 식별을 위하여 HTTP Cookie를 사용하고 있는데 이 페이지 접근의 두 번째 이후부터는 104-112라인에서 쿠키 이름과 값을 표시하게 된다.

51라인은 서블릿의 서비스 결과를 표시하기 위해 HttpServletContextEx로부터 스트림 객체의 참조를 얻는다. 만약 이 라인을 HttpOutputStream out = ctx.getOutputStream()과 같이 "&"를 빠뜨리고 코딩을 하면 ctx의 스트림 객체를 사용하는 것이 아니라 스택에 새로운 객체를 만드는 결과가 되기 때문에 웹 브라우저에는 어떠한 것도 표시되지 않으므로 주의하여야 한다.

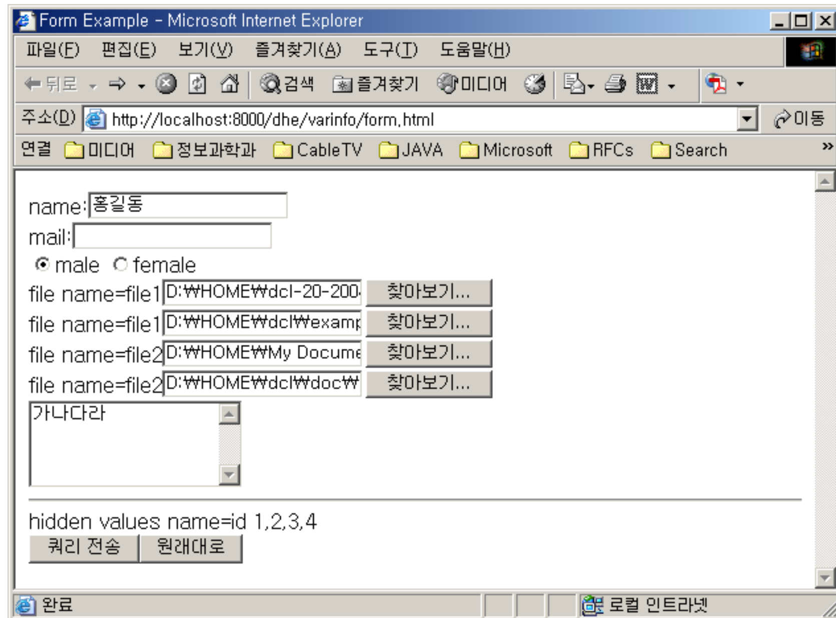
55-58라인은 서블릿이 초기화 될 때 설정된 값이고 60-74라인은 기본 연결 정보이다.

76-78라인은 웹 서버가 제공하고 있는 모든 CGI 환경 변수를 표시하고 있고 80-82라인은 HTTP 요청에 동반한 모든 HTTP Header-Field를 표시한다.

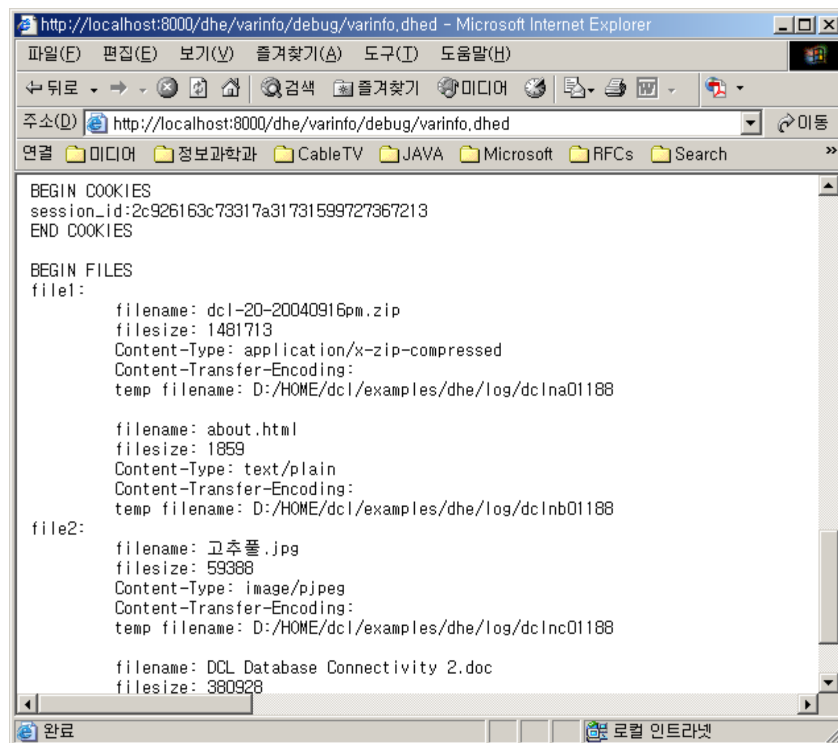
84-101라인은 QUERY_STRING과 FORM_DATA의 "file"을 제외한 내용들을 표시하는 부분이다. HTTP에서 이들은 같은 이름을 가질 수 있기 때문에 91-97라인은 이들을 ","로 분리하여 모두 표시한다.

115-140라인은 "multipart/form-data"의 "file" 부분에 대한 내용을 표시하고 있다. <그림 5-4>의 폼 데이터를 사용한 결과에서 파일에 대한 부분만 표시한 것이 <그림 5-5>이다. "file1"에 대하여 2개가 나타난 것은 HTML의 input 요소의 name 속성값 "file1"을 두 번 사용했기 때문이다.

<그림 5-5>의 Content-Transfer-Encoding 부분에는 아무런 것도 표시되지 않고 있다. RFC1867에서는 이에 대하여 언급하고 있지만 Microsoft IE와 Netscape Navigator는 "file" 데이터에 대하여 다른 엔코딩 방식을 사용하지 않고 이진데이터를 그대로 전송한다.



<그림 5-4> varinfo.dhe를 위한 폼데이터



<그림 5-5> varinfo의 결과

5.2 Microsoft Visual C++ 프로젝트

윈도우에서 DHE 서블릿은 DLL(Dynamic Link Library)로 만들어지기 때문에 Microsoft Visual C++ 6.0에서 프로젝트의 구성은 DLL 프로젝트에 준한다.

5.1에서 설명한 varinfo.cpp를 위한 프로젝트 구성에 관한 기본적인 사항은 다음과 같다.

항목	릴리즈 버전	디버그 버전
프로젝트	Win32 Dynamic-Link Library	
Preprocessor Definitions	NDEBUG, WIN32,_USRDLL	__DCL_DEBUG, NDEBUG, WIN32,_USRDLL
C-Runtime	MSVCRT.DLL	
DCL Runtime-Library	DCLCore.dll, DCLNet.dll	DCLCored.dll, DCLNetd.dll
DCL Import-Library	DCLCore.lib, DCLNet.lib	DCLCored.lib, DCLNetd.lib
Output file name	varinfo.dhe	varinfo.dhed

새 프로젝트를 생성할 때 Projects 탭에서 "Win32 Dynamic-Link Library"를 선택하고 OK을 누른 다음 "An empty DLL project"를 선택한다 프로젝트가 만들어지고 난 다음 Project/Settings 메뉴를 선택해서 Projects Setting 다이얼로그에서 의 내용을 확인한다.

디버그 버전의 경우 C/C++ 탭의 General Category의 Proprocessor Definitions에 _DEBUG를 제거하고 **__DCL_DEBUG, NDEBUG**를 추가한다. _DEBUG 매크로는 MFC와 MS의 LIBC에서 디버그 버전으로 컴파일 하기 위해 사용되는 것으로 DCL은 이들의 런타임 디버깅 환경을 사용하지 않기 때문에 제거하는 것이다.

윈도우에서 DCL은 디버그 버전과 릴리즈 버전 모두 MSVCRT.DLL을 사용하고 있기 때문에 DHE 서블릿 프로젝트도 MSVCRT.DLL을 사용하도록 해야 한다. C-Runtime이 MSVCRT.DLL을 사용하게 하려면 C/C++ 탭의 Code Generation Category의 Use run-time library 부분을 **Multithread DLL**을 선택한다.

프리프로세스 및 컴파일 설정을 하고 나면 Link 탭의 Object/library module에 **DCL Import-Library**를 추가한다.

마지막으로 Link 탭의 Output file name을 변경한다. 디버그 버전은 **varinfo.dhed**이고 릴리즈 버전은 **varinfo.dhe**이다.

프로젝트 설정이 완료되면 위의 varinfo.h와 varinfo.cpp 두 파일을 프로젝트에 추가하여 빌드하면 된다.

5.3 UNIX 버전을 위한 GCC Makefile

UNIX에서 DHE 서블릿의 빌드는 GNU/Linux에서의 예를 들어 설명하도록 한다.

GNU/Linux에서 서블릿은 so(Shared Object)로 빌드 되며 다음의 표는 g++, gmake를 사용하여 빌드하기 위한 기본적인 내용을 요약한다.

항목	릴리즈 버전	디버그 버전
프로젝트	Win32 Dynamic-Link Library	

Preprocessor Definitions	NDEBUG, _LINUX,	__DCL_DEBUG, NDEBUG, _LINUX,
Compile Options	-pthread	
Link Options	-shared	
C-Runtime	libc	
DCL Runtime-Library	libDCLCore.so, libDCLNet.so	libDCLCored.so, libDCLNetd.so
Output file name	varinfo.dhe	varinfo.dhed

리눅스에서 같은 소스를 가지고 서로 다른 서블릿 모듈을 빌드 하려면 각각의 Makefile이 있어야 하기 때문에 번거로울 수 있다. 이를 해결하기 위해 gmake의 VPATH 라는 변수를 사용하면 소스 프로그램의 디렉토리와 빌드를 위한 디렉토리를 다른 위치에 둘 수 있다.

다음은 이를 이용하여 Makefile을 세 개의 파일로 분할하여 소스, 디버그 빌드 그리고 릴리즈 빌드 디렉토리를 서로 다른 곳에 위치하도록 하는 방법이다.

```
 1:  DCL_HOME = $(HOME)/dcl
 2:  INCLUDE_PATH       = $(DCL_HOME)/include
 3:  #LIB_PATH          = $(DCL_HOME)/lib
 4:  LIB_PATH           = $(DCL_LOCAL_ROOT)/lib
 5:
 6:  #CFLAGS = -Wall -D_UNIX $(INCLUDE_PATH)
 7:
 8:  #DEBUG_BUILD =
 9:
10:  CFLAGS = -DNDEBUG -Wall -O3 -pthread \
11:        -D_LINUX -D_GNU_SOURCE -D_LIBC_REENTRANT -D_REENTRANT \
12:        -I$(INCLUDE_PATH)
13:
14:  LINK_DCLCORE = -lDCLCore
15:  LINK_DCLNET = -lDCLNet
16:
17:  ifdef DEBUG_BUILD
18:  CFLAGS = -D__DCL_DEBUG -DNDEBUG -Wall -pthread \
19:        -D_LINUX -D_GNU_SOURCE -D_LIBC_REENTRANT -D_REENTRANT \
20:        -I$(INCLUDE_PATH)
21:
22:  LINK_DCLCORE = -lDCLCored
23:  LINK_DCLNET = -lDCLNetd
24:
25:  endif
26:  LDFLAGS = -L$(LIB_PATH) -pthread $(LINK_DCLCORE)
27:
28:  CC          = gcc
29:  CXX         = g++
30:  LD          = g++
31:  RM          = /bin/rm
32:  COPY        = /bin/cp -f
33:  AR          = ar
34:  MAKE        = make
35:
36:  .SUFFIXES: .cpp .cxx .cc .c
```

```
37:  .cpp.o:
38:        $(CXX) -c $(CFLAGS) -o $@ $<
39:  .cxx.o:
40:        $(CXX) -c $(CFLAGS) -o $@ $<
41:  .cc.o:
42:        $(CXX) -c $(CFLAGS) -o $@ $<
43:  .c.o:
44:        $(CC) -c $(CFLAGS) -o $@ $<
```

<그림 5-6> rules_common.mk

<그림 5-6>의 rules_common.mk는 DCL 자체를 빌드할 때 뿐만 아니라 DCL을 이용하게 될 모든 어플리케이션의 빌드 때에도 사용될 수 있다.

1-4번 라인은 DCL의 헤더파일 위치와 라이브러리 파일의 위치를 지정한다.

```
 1:
 2:  OBJS = varinfo.o
 3:
 4:  SONAME = varinfo.dhe
 5:  ifdef DEBUG_BUILD
 6:  SONAME = varinfo.dhed
 7:  endif
 8:
 9:  TARGET = $(SONAME)
10:
11:  override CFLAGS := -fPIC $(CFLAGS)
12:  override LDFLAGS := $(LDFLAGS) $(LINK_DCLNET)
13:
14:  all: $(TARGET)
15:
16:  $(TARGET): $(OBJS)
17:        $(CXX) -shared -Wl,-soname,$(SONAME) -o $(SONAME) $(LDFLAGS) $(OBJS)
18:
19:  varinfo.o: varinfo.cpp varinfo.h
20:
21:  clean:
22:        $(RM) -rf $(OBJS) $(TARGET) debug
23:
24:  debug:
25:        test -f $@/Makefile || (mkdir $@; echo "DEBUG_BUILD=1" > $@/Makefile ;
cat ./Makefile >> $@/Makefile)
26:        $(MAKE) -C $@
27:
28:  install:
29:        cp -f $(TARGET) $(INSTALL_TO)/
```

<그림 5-7> Makefile.in

<그림 5-7>의 Makefile.in은 varinfo.h, varinfo.cpp와 같은 디렉토리에 둔다.

```
 1:  ifndef $(MAKE)
```

```
2:  include $(DCL_SOURCE_ROOT)/build/rules_common.mk
3:  endif
4:
5:  SOURCE_DIR = $(DCL_SOURCE_ROOT)/examples/dhe/varinfo
6:  VPATH = $(SOURCE_DIR)
7:  INSTALL_TO      = $(HOME)/www
8:
9:  include $(SOURCE_DIR)/Makefile.in
```

<그림 5-8> Makefile

<그림 5-8>의 Makefile은 rules_common.mk와 Makefile.in을 포함시켜 전체 Makefile을 완성한다.

2라인과 4라인의 DCL_SOURCE_ROOT는 쉘 환경 변수로 설정한다.

```
djkim@earth:~/dcl-local/examples/dhe/varinfo
[djkim@earth varinfo]$ echo $DCL_SOURCE_ROOT
/home/djkim/dcl
[djkim@earth varinfo]$ make
g++ -c -fPIC -DNDEBUG -Wall -O3 -pthread -D_LINUX -D_GNU_SOURCE -D_LIBC_REENTRAN
T -D_REENTRANT -I/home/djkim/dcl/include  -o varinfo.o /home/djkim/dcl/examples/
dhe/varinfo/varinfo.cpp
g++ -shared -Wl,-soname,varinfo.dhe -o varinfo.dhe -L/home/djkim/dcl-local/lib -
pthread -lDCLCore -lDCLNet varinfo.o
[djkim@earth varinfo]$ make debug
test -f debug/Makefile || (mkdir debug; echo "DEBUG_BUILD=1" > debug/Makefile ;
cat ./Makefile >> debug/Makefile)
make -C debug
make[1]: 들어감 '/home/djkim/dcl-local/examples/dhe/varinfo/debug' 디렉토리
g++ -c -fPIC -D__DCL_DEBUG -DNDEBUG -Wall -pthread -D_LINUX -D_GNU_SOURCE -D_LIB
C_REENTRANT -D_REENTRANT -I/home/djkim/dcl/include  -o varinfo.o /home/djkim/dcl
/examples/dhe/varinfo/varinfo.cpp
g++ -shared -Wl,-soname,varinfo.dhed -o varinfo.dhed -L/home/djkim/dcl-local/lib
 -pthread -lDCLCored -lDCLNetd varinfo.o
make[1]: 나감 '/home/djkim/dcl-local/examples/dhe/varinfo/debug' 디렉토리
[djkim@earth varinfo]$

[영어][완성][두벌식]
```

<그림 5-9> 리눅스에서 서블릿의 빌드

5.4 서블릿 모듈의 설치

서블릿 모듈의 설치 방법은 두 가지를 사용할 수 있다. 첫 번째는 웹 서버의 서비스 경로에 모듈의 파일을 복사하는 것으로 간단히 설치된다. 또 다른 방법은 서블릿 매니저의 설정을 사용하는 것으로 서블릿 모듈이 웹 서버의 서비스 경로에는 없지만 서블릿 매니저가 관리하는 DHE.INI 파일에 서비스 경로에 서블릿 모듈의 파일이름을 연관시키는 것이다. 이에 대한 자세한 내용은 3장을 참고한다.