CHAPTER 01 웹 프로그래밍의 이해

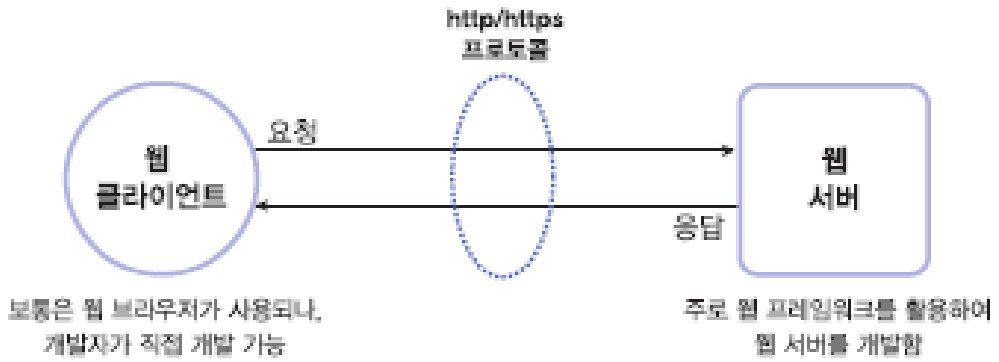
웹 프로그래밍의 기본 기술에 대해 이해하기

CHAPTER 01 웹 프로그래밍의 이해

HTTP(S) 프로토콜로 통신하는, 클라이언트와 서버를 개발

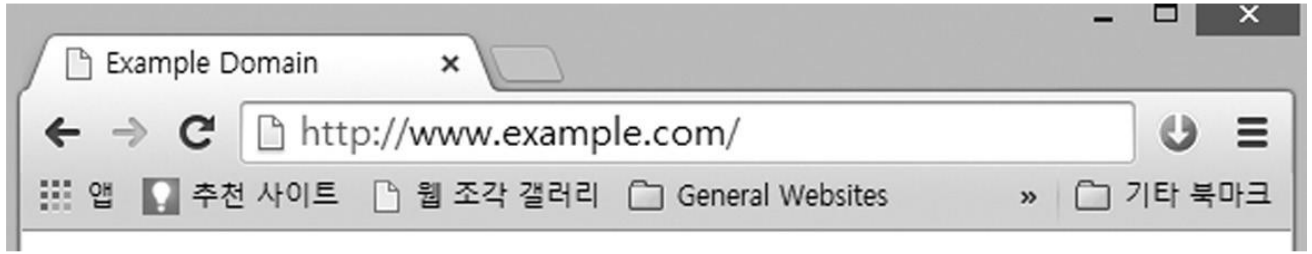
웹 클라이언트와 웹 서버를 같이 개발 및 웹 클라이언트 또는 웹 서버 하나만 개발

웹 서버를 개발하는 경우가 많아서 파이썬 웹 프로그래밍이라고 하면 우선적으로 장고Django와 같은 웹 프레임워크를 사용하여 웹 서버를 개발



SECTION 01_2 다양한 웹클라이언트

1.2.1 웹 브라우저를 사용하여 요청



브라우저는 주소창에 입력된 문장을 해석하여 웹 서버에게 HTTP 요청을 보내는 웹 클라이언 트의 역할을 수행합

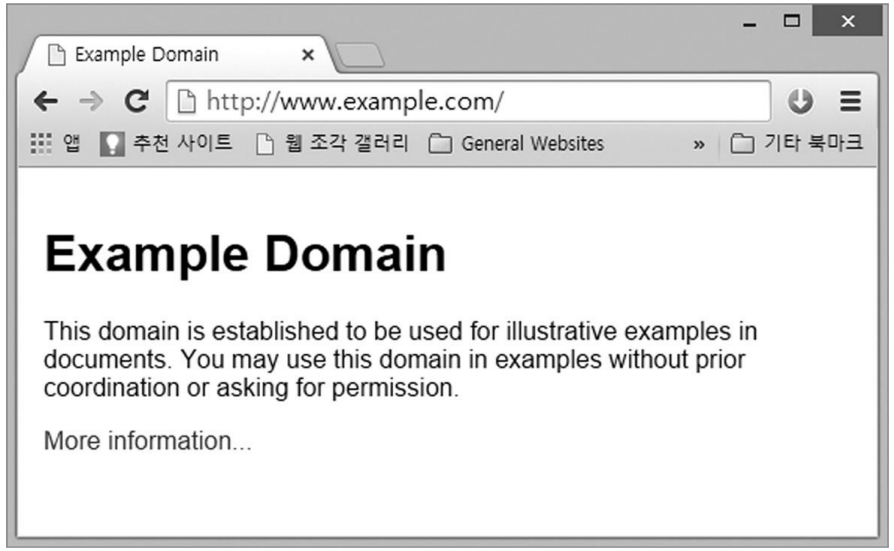


그림 **1-3** 웹 브라우저 요청에 대한 웹 서버의 응답

1.2.2 리눅스 curl 명령을 사용하여 요청

리눅스 **curl** 명령은 HTTP/HTTPS/FTP 등 여러 가지의 프로토콜을 사용하여 데이터를 송수신할 수 있는 명령

```
$ curl  http://www.example.com
```

```
[shkim@localhost ch1]$ curl http://www.example.com
<!doctype html>
<html>
<head>
    <title>Example Domain</title>
     . . . (중략)

</head>

<body>
<div>
    <h1>Example Domain</h1>
    <p>This domain is established to be used for illustrative examples in
    documents. You may use this domain in examples without prior coordination or
    asking for permission.</p>
    <p><a href="http://www.iana.org/domains/example">More information...</a></p>
</div>
</body>
</html>
[shkim@localhost ch1]$
```

그림 **1-4** curl 명령 요청에 대한 웹 서버의 응답

[그림 1-3]과 비교해서 HTML 태그들과 <head> 부분이 더 보이지만 이는 동일한 응답

1.2.3 Telnet을 사용하여 요청

리눅스의 **telnet** 프로그램을 사용하여 HTTP 요청을 보낼 수도 있음

```
[shkim@localhost ch1]$ telnet www.example.com 80
Trying 93.184.216.119...
Connected to www.example.com.
Escape character is '^]'.
GET / HTTP/1.1  Enter 입력
Host: www.example.com  Enter 입력
Enter 입력
```

telnet 명령은 터미널 창에서 입력하는 내용을 그대로 웹 서버에 전송

telnet 프로그램이 웹 클라이언트의 역할을 수행

```
HTTP/1.1 200 OK
Accept-Ranges: bytes
Cache-Control: max-age=604800
Content-Type: text/html
Date: Sun, 09 Nov 2014 11:46:41 GMT
Etag: "359670651"
Expires: Sun, 16 Nov 2014 11:46:41 GMT
Last-Modified: Fri, 09 Aug 2013 23:54:35 GMT
Server: ECS (rhv/818F)
X-Cache: HIT
x-ec-custom-error: 1
Content-Length: 1270

<!doctype html>
<html>
<head>
    <title>Example Domain</title>
    . . . (중략)

</head>

<body>
<div>

    <h1>Example Domain</h1>
    <p>This domain is established to be used for illustrative examples in
    documents. You may use this domain in examples without prior coordination or
    asking for permission.</p>
    <p><a href="http://www.iana.org/domains/example">More information...</a></p>
</div>
</body>
</html>
^]
telnet> quit  Enter 입력
Connection closed.
[shkim@localhost ch1]$
```

그림 **1-5** telnet 명령 요청에 대한 웹 서버의 응답

응답 메시지를 수신한 후, **telnet** 프로그램을 종료하기 위해 마지막 두 라인을 입력한 것에유의.
동일한 응답을 확인할 수 있음

SECTION 01_2 다양한 웹클라이언트

1.2.4 직접 만든 클라이언트로 요청

```
C:\RedBook\ch1>notepad example.py

import urllib.request
print(urllib.request.urlopen("http://www.example.com").read().decode('utf-8'))
```

```
C:\RedBook\ch1> python example.py
<!doctype html>
<html>
<head>
    <title>Example Domain</title>
     . . . (중략)
</head>

<body>
<div>
    <h1>Example Domain</h1>
    <p>This domain is established to be used for illustrative examples in
    documents. You may use this domain in examples without prior coordination or
    asking for permission.</p>
    <p><a href="http://www.iana.org/domains/example">More information...</a></p>
</div>
</body>
</html>

C:\RedBook\ch1>
```

그림 **1-6** 직접 만든 클라이언트 요청에 대한 웹 서버의 응답

앞에서 보았던 응답 결과와 동일한 것을 확인할 수 있음.
즉, 웹 클라이언트의 형태는 달라도 동일한 요청에 대해서 동일한 응답을 받는 것을확인

```
C:\RedBook\ch1>python -c "import urllib.request; print(urllib.request.urlopen('http://
www.example.com').read().decode('utf-8'))"
```

웹 클라이언트를 직접 만드는 방법은 **2.2** 웹 클라이언트 라이브러리에서 자세히 설명하고 있으니, 참 고 바랍니다

SECTION 01_3 HTTP프로토콜

1.3.1 HTTP 메시지의 구조

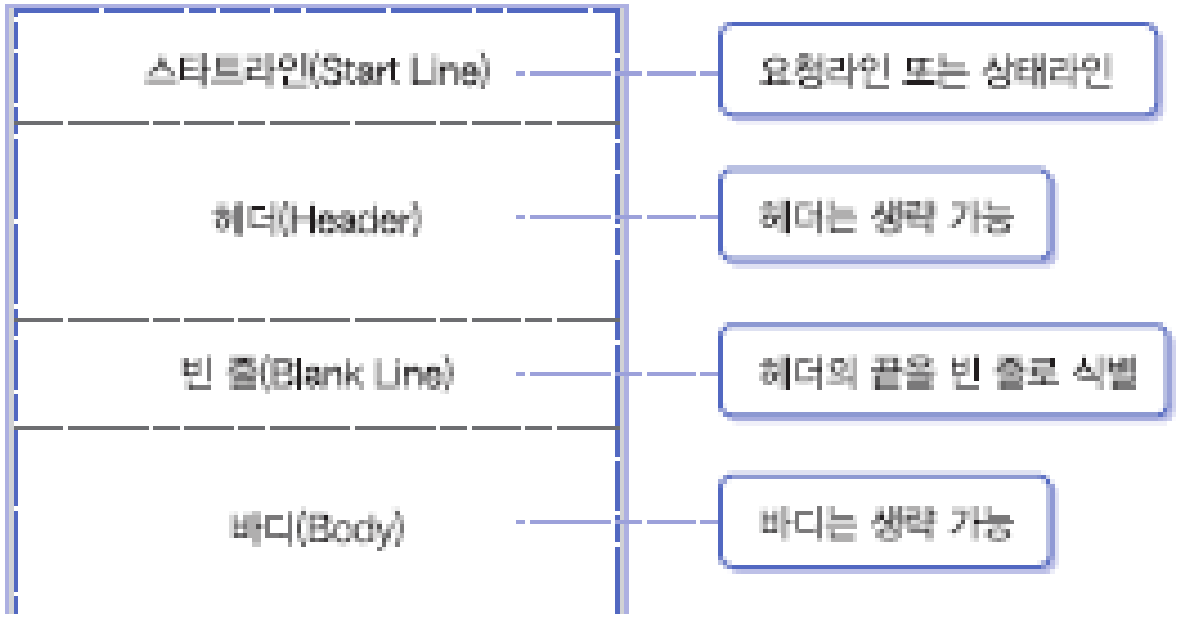


그림 **1-7** HTTP 메시지 구조

[그림 1-7]에서 스타트라인은 요청 메시지일 때 요청라인request line이라고 하고, 응답 메시지일 때 상태라인status line이라고함

```
GET /book/shakespeare HTTP/1.1
Host: www.example.com:8080
```

첫 번째 줄은 요청라인으로, 요청 방식**method**,
요청 **URL**, 프로토콜 버전으로 구성

NOte
URI vs URL란?URI는 Uniform Resource Indentifier의 약자로
URL(Uniform Resource Locator)과 URN(Uniform Resource Name)을 포함하는 좀 더 넓은 의미의 표현이지만, 웹 프로그래밍에서는
URI와 URL을 동일한 의미로 사용해도 무방

SECTION 01_3 HTTP프로토콜

1.3.2 HTTP 처리 방식

메소드명	의미	CRUD와 매핑되는 역할
GET	리소스 취득	Read(조회)
POST	리소스 생성, 리소스 데이터 추가	Create(생성)
PUT	리소스 변경	Update(변경)
DELETE	리소스 삭제	Delete(삭제)
HEAD	리소스의 헤더(메타데이터) 취득	
OPTIONS	리소스가 서포트하는 메소드 취득	
TRACE	루프백 시험에 사용	
CONNECT	프록시 동작의 터널 접속으로 변경	

표 **1-1** HTTP 메소드 종류

GET 방식은 지정한 URL의 정보를 가져오는 메소드로, 가장 많이 사용

POST의 대표적인 기능은 리소스를 생성하는 것으로, 블로그에 글을 등록하는 경우가 이에 해당.
PUT은 리소스를 변경하는 데 사용

SECTION 01_3 HTTP프로토콜

1.3.3 GET과 POST 메소드

앞에서 8가지의 HTTP 메소드를 소개하였지만,
현실적으로 가장 많이 사용하는 메소드는 GET과 POST 2가지

```
GET  http://docs.djangoproject.com/search/?q=forms&release=1  HTTP/1.1
```

```
POST  http://docs.djangoproject.com/search/ HTTP/1.1
Content-Type: application/x-www-form-urlencoded

q=forms&release=1
```

파이썬의 장고 프레임워크에서도 폼의 데이터는
POST 방식만을 사용하고 있음

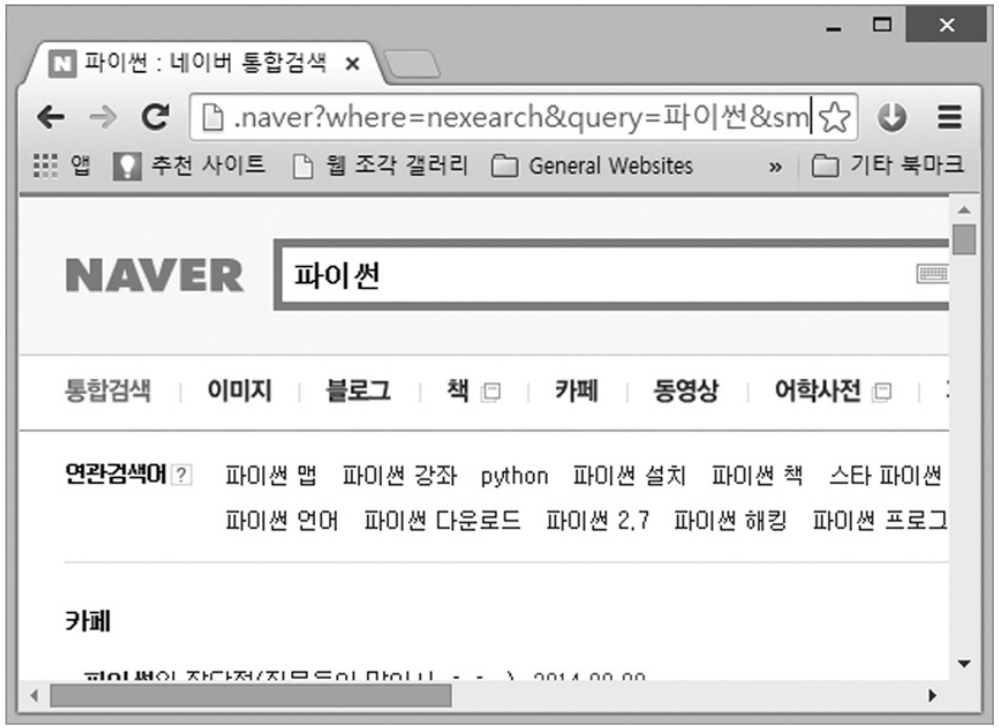


그림 **1-8** 네이버의 검색 창 – GET 방식 전달

SECTION 01_3 HTTP프로토콜

1.3.4 상태 코드

서버에서의 처리 결과는 응답 메시지의 상태라인에 있는
상태 코드Status code를 보고 파악할 수 있음

메소드명	의미	CRUD와 매핑되는 역할
1xx	Informational (정보 제공)	임시적인 응답으로, 현재 클라이언트의 요청까지 처리되었으니 계속 진행하라는 의미입니다. HTTP 1.1 버전부터 추가되었습니다.
2xx	Success(성공)	클라이언트의 요청이 서버에서 성공적으로 처리되었다는 의미입니다.
3xx	Redirection (리다이렉션)	완전한 처리를 위해서 추가적인 동작을 필요로 하는 경우입니다. 주로 서버의 주소 또는 요청한 URI의 웹 문서가 이동되었으니, 그 주소로 다시 시도해보라는 의미입니다.
4xx	Client Error (클라이언트 에러)	없는 페이지를 요청하는 것처럼 클라이언트의 요청 메시지 내용이 잘못된 경우입니다.
5xx	Server Error (서버 에러)	서버 측 사정에 의해서 메시지 처리에 문제가 발생한 경우입니다. 서버의 부하, DB 처리 과정 오류, 서버에서 익셉션이 발생하는 경우가 이에 해당합니다.

표 **1-2** 상태 코드 분류

SECTION 01_3 HTTP프로토콜

1.3.4 상태 코드

상태 코드	상태 텍스트	응답 문구	서버 측면에서의 의미
2xx	Success	성공	**클라이언트가 요청한 동작을 수신하여 이해했고, 승낙했으며 성공적으로 처리했다.**
200	OK	성공	**서버가 요청을 성공적으로 처리했다.**
201	Created	생성됨	**요청이 처리되어서 새로운 리소스가 생성되었다.** 응답 헤더 Location에 새로운 리소스의 절대 URI를 기록합니다.
202	Accepted	허용됨	**요청은 접수했지만 처리가 완료되지 않았다.** 클라이언트는 응답 헤더의 Location, Retry-After를 참고하여 다시 요청을 보냅니다.
3xx	Redirection	리다이렉션	**클라이언트는 요청을 마치기 위해 추가적인 동작을 취해야 한다.**
301	Moved Permanently	영구 이동	**지정한 리소스가 새로운 URI로 이동했다.** 이동할 곳의 새로운 URI는 응답 헤더 Location에 기록합니다.

표 **1-3** 자주 사용되는 상태 코드

SECTION 01_3 HTTP프로토콜

1.3.4 상태 코드

상태 코드	상태 텍스트	응답 문구	서버 측면에서의 의미
303	See Other	다른 위치 보기	**다른 위치로 요청하라.** 요청에 대한 처리 결과를 응답 헤더 Location에 표시된 URI에서 GET으로 취득할 수 있습니다. 브라우저의 폼 요청을 POST로 처리하고 그 결과 화면으로 리다이렉트시킬 때, 자주 사용하는 응답 코드입니다.
307	Temporary Redirect	임시 리다이렉션	**임시로 리다이렉션 요청이 필요하다.** 요청한 URI가 없으므로, 클라이언트는 메소드를 그대로 유지한 채 응답 헤더 Location에 표시된 다른 URI로 요청을 재송신할 필요가 있습니다. 클라이언트는 향후 요청 시 원래 위치를 계속 사용해야 합니다. 302의 의미를 정확하게 재정의해서 HTTP/1.1의 307 응답으로 추가되었습니다.
4xx	Client Error	클라이언트 에러	**클라이언트의 요청에 오류가 있다.**
400	Bad Request	잘못된 요청	**요청의 구문이 잘못되었다.** 클라이언트가 모르는 4xx 계열의 응답 코드가 반환된 경우에도 클라이언트는 400과 동일하게 처리하도록 규정하고 있습니다.
401	Unauthorized	권한 없음	**지정한 리소스에 대한 액세스 권한이 없다.** 응답 헤더 WWW-Authenticate에 필요한 인증 방식을 지정합니다.
403	Forbidden	금지됨	**지정한 리소스에 대한 액세스가 금지되었다.** 401 인증 처리 이외의 사유로 리소스에 대한 액세스가 금지되었음을 의미합니다. 리소스의 존재 자체를 은폐하고 싶은 경우는 404 응답 코드를 사용할 수 있습니다.
404	Not Found	찾을 수 없음	**지정한 리소스를 찾을 수 없다.**
5xx	Server Error	서버 에러	**클라이언트의 요청은 유효한데, 서버가 처리에 실패했다.**
500	Internal Server Error	내부 서버 오류	**서버쪽에서 에러가 발생했다.** 클라이언트가 모르는 5xx 계열의 응답 코드가 반환된 경우에도 클라이언트는 500과 동일하게 처리하도록 규정하고 있습니다.
502	Bad Gateway	불량 게이트웨이	**게이트웨이 또는 프록시 역할을 하는 서버가 그 뒷단의 서버로부터 잘못된 응답을 받았다.**
503	Service Unavailable	서비스 제공불가	**현재 서버에서 서비스를 제공할 수 없다.** 보통은 서버의 과부하나 서비스 점검 등 일시적인 상태입니다.

SECTION 01_4 URL 설계

그림 **1-9** URL 구성 항목

URL 스킴 : URL에 사용된 프로토콜을 의미.

- 호스트명 : 웹 서버의 호스트명으로, 도메인명 또는 IP 주소로 표현.
- 포트번호 : 웹 서버 내의 서비스 포트번호. 생략 시에는 디폴트 포트번호로, http는 80을, https는 443을 사용
- 경로 : 파일이나 애플리케이션 경로를 의미
- 쿼리스트링 : 질의 문자열로, 앰퍼샌드(&)로 구분된 이름=값 쌍 형식으로 표현
- 프라그먼트 : 문서 내의 앵커 등 조각을 지정

SECTION 01_4 URL 설계

1.4.1 URL을 바라보는 측면

URL은 웹 클라이언트에서 호출한다는 시점에서 보면, 웹 서버에 존재하는 애플리케이션에 대 한 API Application Programming Interface라고 할 수 있음

```
http://blog.example.com/search?q=test&debug=true
```

URL을 바라보는 또 한 가지 측면은 REST 방식으로 URL을 설계하는 것.
REST 방식이란 웹 서버에 존재하는 요소들을 모두 리소스라고 정의하고, URL을 통해 웹 서버의 특정 리소스를 표현한다는 개념.

SECTION 01_4 URL 설계

1.4.2 간편 URL

간편 URL은 쿼리스트링 없이 경로만 가진 간단한 구조의 URL을 말함.
검색 엔진 의 처리를 최적화하기 위해 생겨난 간편한 URL은 URL을 입력하거나 기억하기 쉽다는 부수적인 장점도 있어, 검색 엔진 친화적 URL(search engine friendly url) 또는 사용자 친화적 URL(user friendly url)이라고 부르기도 함

기존 URL	간편 URL
http://example.com/index.php?page=foo	http://example.com/foo
http://example.com/index.php?page=consulting/marketing	http://example.com/consulting/marketing
http://example.com/products?category=2&pid=25	http://example.com/products/2/25
http://example.com/cgi-bin/feed.cgi?feed=news&frm=rss	http://example.com/news.rss
http://example.com/services/index.jsp?category=legal&id=patents	http://example.com/services/legal/patents
http://example.com/index.asp?mod=profiles&id=193	http://example.com/user/john-doe
http://example.com/app/dashboard/dsptchr_c80.dll?page=38661&mod1=bnr_ant&UID=4511681&SESSID=4fd8b561ac867195fba2cc5679&...	http://example.com/app/dashboard/reports#monthly

표 **1-4** 기존 URL과 간편 URL 간의 비교를 위한 대응 예시

SECTION 01_4 URL 설계

1.4.3 파이썬의 우아한 URL

URL을 정의하 기 위해 정규표현식Regular Expression을 추가적으로 사용할 수 있음.

```
urlpatterns = [
    path('articles/2003/', views.special_case_2003),
    path('articles/<int:year>/', views.year_archive),
    path('articles/<int:year>/<int:month>/', views.month_archive),
    path('articles/<int:year>/<int:month>/<slug:slug>/', views.article_detail),
]
```

```
urlpatterns = [
    path('articles/2003/', views.special_case_2003),
    re_path(r'^articles/(?P<year>[0-9]{4})/$', views.year_archive),
    re_path(r'^articles/(?P<year>[0-9]{4})/(?P<month>[0-9]{2})/$', views.month_archive),
    re_path(r'^articles/(?P<year>[0-9]{4})/(?P<month>[0-9]{2})/(?P<slug>[\w-]+)/$', views.
article_detail),
]
```

SECTION 01_5 웹 애플리케이션 서버

구분	역할	프로그램 명
웹 서버	웹 클라이언트의 요청을 받아서 요청을 처리하고, 그 결과를 웹 클라이언트에게 응답합니다. 주로 정적 페이지인 HTML, 이미지, CSS, 자바스크립트 파일을 웹 클라이언트에 제공할 때 웹 서버를 사용합니다. 만약 동적 페이지 처리가 필요하다면 웹 애플리케이션 서버에 처리를 넘깁니다.	Apache httpd, Nginx, lighttpd, IIS 등
웹 애플리케이션 서버	웹 서버로부터 동적 페이지 요청을 받아서 요청을 처리하고, 그 결과를 웹 서버로 반환합니다. 주로 동적 페이지 생성을 위한 프로그램 실행과 데이터베이스 연동 기능을 처리합니다.	Apache Tomcat, JBoss, WebLogic, WebSphere, Jetty, Jeus, mod_wsgi, uWSGI, Gunicorn 등

웹 서버 및 웹 애플리케이션 서버라는 용어는 SW 측면의 서버 프로그램을 의미합니다.
HW 측면의 용어는 **1.5.5** 웹 서버와의 역할 구분에서 설명

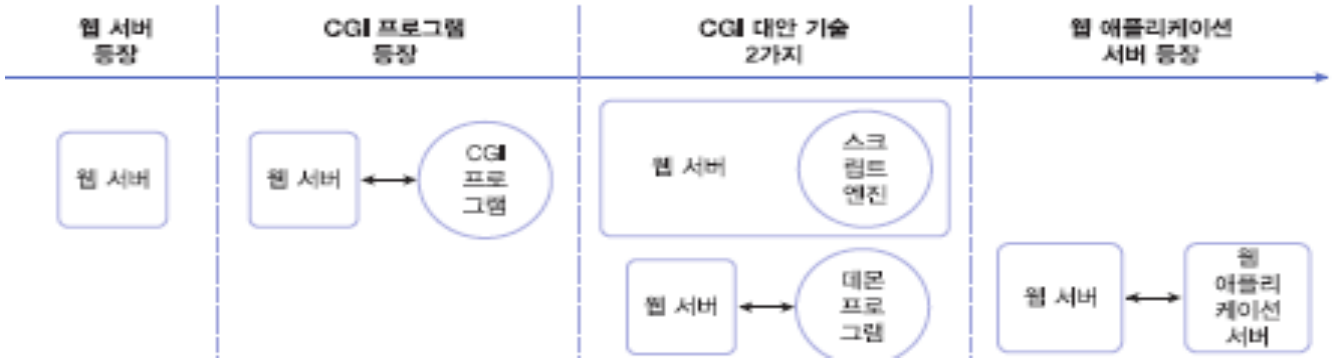


그림 **1-10** 기술의 발전에 따른 웹 서버 기술의 변화

SECTION 01_5 웹 애플리케이션 서버

1.5.1 정적 페이지 vs 동적 페이지

정적static, 동적dynamic이란 용어는 사용자가 페이지를 요청하는 시점에 페이지의 내용이 유지되는가 또는 변경되는가를 구분해주는 용어.
동적 페이지에는 프로그래밍 코드가 포함되어 있어서 페이지 요청 시점에 HTML 문장을 만들어내는 것

SECTION 01_5 웹 애플리케이션 서버

1.5.2 CGI 방식의 단점

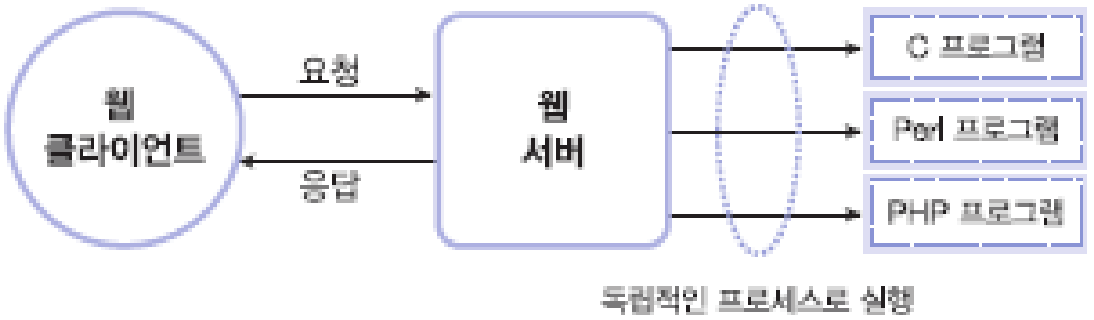


그림 **1-11** 전통적인 CGI 방식의 요청 처리

CGI 방식의 근본적인 문제점은 각각의 클라이언트 요청에 대하여 독립적인 별도의 프로세스가 생성

요청이 많아질수록 프로세스가 많아지고, 프로세스가 많아질수록 비례적으로 프로세스가 점유하는 메모리 요구량도 커져서 시스템에 많은 부하를 주는 요인

SECTION 01_5 웹 애플리케이션 서버

1.5.3 CGI 방식의 대안 기술

CGI 방식의 대안 기술 중 하나는 별도의 애플리케이션(CGI 프로그램과 같은 역할을 하는 프로그 램)을 Perl, PHP 등의 스크립트 언어로 작성하고, 스크립트를 처리하는 스크립트 엔진(인터프리 터)을 웹 서버에 내장시켜서 CGI 방식의 단점이었던 별도의 프로세스를 기동시키는 오버헤드를 줄이는 방식

SECTION 01_5 웹 애플리케이션 서버

1.5.4 애플리케이션 서버 방식

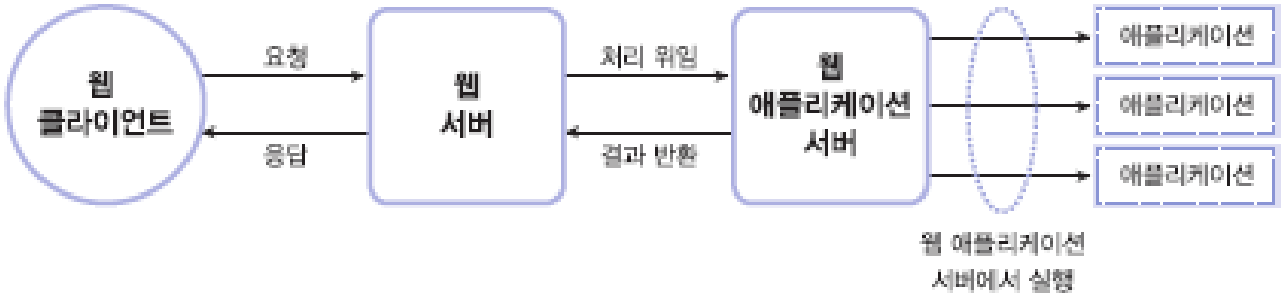


그림 **1-12** 애플리케이션 서버 방식의 요청 처리

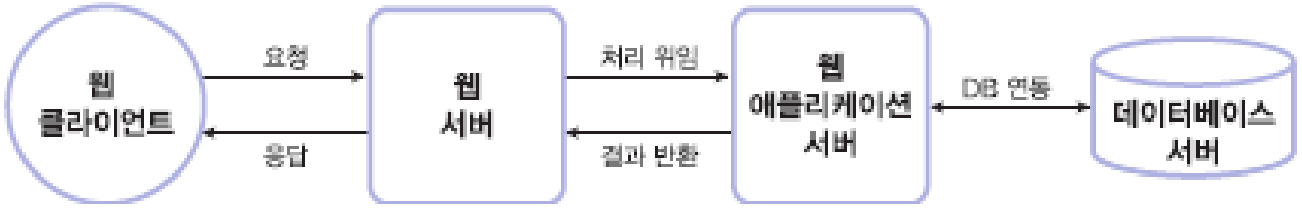


그림 **1-13** 애플리케이션 서버 방식에서의 서버 간 구성도

웹 서버와 웹 애플리케이션 서버가 분리됨에 따라, 서로의 역할도 구분하여 사용하는 것이 좋음. 왜냐하면 정적 페이지를 처리하는 경우에 비해서 동적 페이지를 처리하는 경우가 수 배에서 수십 배의 메모리를 소비하기 때문.

SECTION 01_5 웹 애플리케이션 서버

1.5.5 웹 서버와의 역할 구분

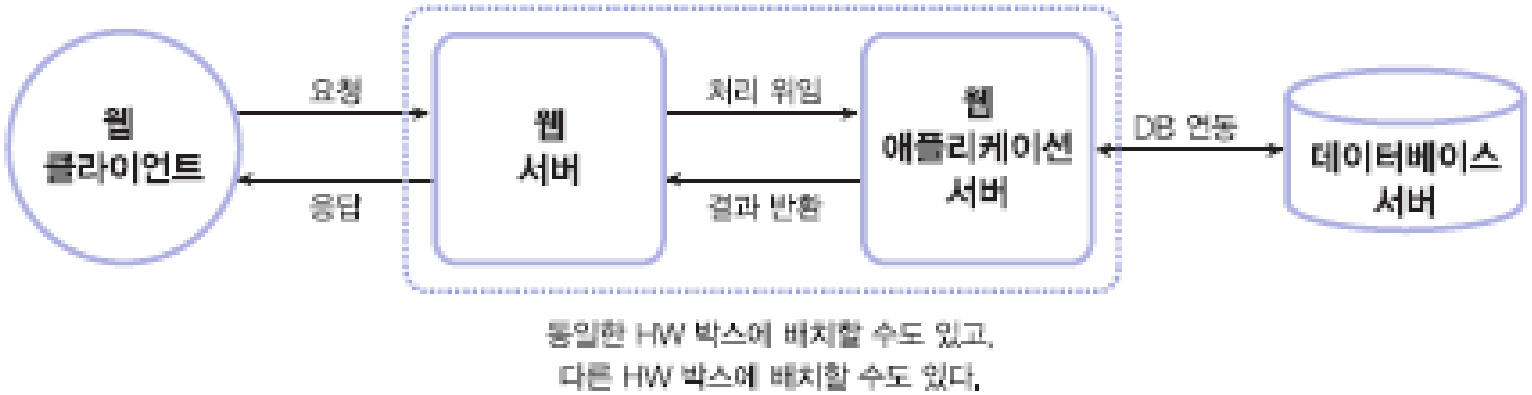


그림 **1-14** 웹 서버와 애플리케이션 서버의 역할과 HW 배치

지금까지 웹 서버 및 웹 애플리케이션 서버라는 용어를 사용하였는데, 이는 SW$_{Software}$ 측면의 서버 프로그램을 의미

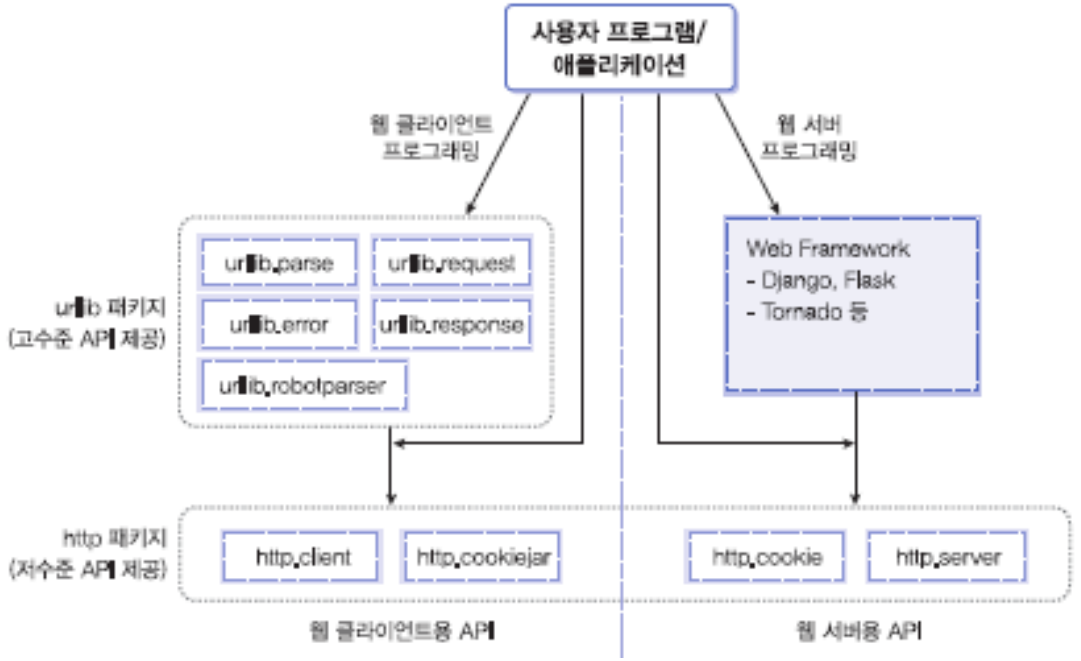
웹 서버와 웹 애플리케이션 서버 프로그램이 함께 필요하며, 이 두 개의 서버를 동일한 HW 박스에서 기동시키는 것도 충분히 가능한 구성.
서비스 운용 관리 측면에서 하나의 HW 박스에 구성하는 것이 좀 더 간편한 방식이기 때문

CHAPTER 02 파이썬 웹 표준 라이브러리

파이썬 웹 라이브러리의 전체 구성 및 버전 2.x와 3.x의 달라진 점

SECTION 02 웹 라이브러리 구성

2.1 웹 라이브러리 구성



urllib 패키지에는 웹 클라이언트를 작성하는 데 사용되는 모듈들이 있으며, 가장 빈번하게 사용하는 모듈.

http 패키지는 크게 서버용과 클라이언트용 라이브러리로 나누어 모듈을 담고 있으며, 쿠키 관련 라이브러리도 http 패키지 내에서 서버용과 클라이언트용으로 모듈이 구분

SECTION 02 웹 라이브러리 구성

표 **2-1** 파이썬 3.x에서 표준 라이브러리의 모듈 구성 변경사항

파이썬 3.x 모듈명	파이썬 2.x 모듈명		파이썬 3.x에서의 변화
urllib.parse	urlparse	urllib 일부	하나의 urllib 패키지로 모아서 모듈을 기능별로 나눴고, urllib 모듈은 기능에 따라 여러 모듈로 흩어졌습니다.
urllib.request	urllib2 대부분	urllib 일부	
urllib.error	urllib2 에러 부분	urllib 일부	
urllib.response		urllib 일부	
urllib.robotparser	robotparser		
http.server	BaseHTTPServer		하나의 http 패키지로 모아 server와 client 모듈로 구분했습니다.
http.server	CGIHTTPServer		
http.server	SimpleHTTPServer		
http.client	httplib		
http.cookies	Cookie		하나의 http 패키지로 모았습니다.
http.cookiejar	cookielib		
html.parser	HTMLParser		하나의 html 패키지로 모았습니다.
html.entities	htmlentitydefs		

SECTION 02_2웹 클라이언트 라이브러리

사용웹 클라이언트를 위한 파이썬 표준 라이브러리가 있지만, 실제 프로젝트에서는 외부 라이브러리인 requests, beautifulsoup4 등을 더 많이 사용하는 편입니다. 좀 더 간편하고 이해하기 쉬운 문법을 제공하기 때문

Python Shell 실행 파이썬 언어는 2가지 실행 방법을 제공합니다. 하나는 python 명령어로 파이썬 스크립트 파일 즉, *.py 파일을 실행하는 방법(> **python example.py**). 또 다른 방법은 파이썬 쉘 모드에서 라인 단위로 실행하는 것입 니다. 파이썬 쉘 모드로 진입하려면 python 명령어만 입력(>**python**

2.2.1 urllib.parse 모듈

```
C:₩RedBook₩ch2>python
Python 3.6.1 (v3.6.1:69c0db5, Mar 21 2017, 18:41:36) [MSC v.1900 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
>>> from urllib.parse import urlparse
>>> result = urlparse("http://www.python.org:80/guido/python.html;philosophy?overall=3#n10")
>>> result
ParseResult(scheme='http', netloc='www.python.org:80',
path='/guido/python.html', params='philosophy', query='overall=3', fragment='n10')
```

scheme : URL에 사용된 프로토콜을 의미합니다.

- netloc : 네트워크 위치. **user :password@host :port** 형식으로 표현되며, HTTP 프로토콜인 경우는 **host:port** 형식.
- path : 파일이나 애플리케이션 경로를 의미.
- params : 애플리케이션에 전달될 매개변수입니다. 현재는 사용하지 않음.
- query : 질의 문자열 또는 매개변수로, 앰퍼샌드(&)로 구분된 이름=값 쌍 형식으로 표현.
- fragment : 문서 내의 앵커 등 조각을 지정.

SECTION 02_2웹 클라이언트 라이브러리

```
urlopen(url, data=None, [timeout])
```

- url 인자로 지정한 URL로 연결하고, 유사 파일 객체를 반환.
  url 인자는 문자열이거나, **Request** 클래스의 인스턴스가 올 수 있음.
- url에 **file** 스킴을 지정하면 로컬 파일을 열 수 있음.
- 디폴트 요청 방식은 GET이고, 웹 서버에 전달할 파라미터가 있으면 질의 문자열을 url 인자에 포함해서 보냄.
- 요청 방식을 POST로 보내고 싶으면 data 인자에 질의 문자열을 지정해주면 됨.
- 옵션인 timeout은 응답을 기다리는 타임아웃 시간을 초로 표시

사용 케이스	사용 방법
URL로 GET/POST 방식의 간단한 요청 처리	urlopen( ) 함수만으로 가능
PUT, HEAD 메소드 등, 헤더 조작이 필요한 경우	Request 클래스를 같이 사용
인증, 쿠키, 프록시 등 복잡한 요청 처리	인증/쿠키/프록시 해당 핸들러 클래스를 같이 사용

표 **2-2** urlopen() 함수 사용 방법

2.2.3 urllib.request 모듈 예제

```
from urllib.request import urlopen
from html.parser import HTMLParser

class ImageParser(HTMLParser):
    def handle_starttag(self, tag, attrs):
        if tag != 'img':
            return
        if not hasattr(self, 'result'):
            self.result = []
        for name, value in attrs:
            if name == 'src':
                self.result.append(value)
```

1
2
3
4

```
def parse_image(data):                                          ⑤
    parser = ImageParser()
    parser.feed(data)                                           ⑥
    dataSet = set(x for x in parser.result)                     ⑦
    return dataSet                                              ⑧

def main():                                                     ⑨
    url = "http://www.google.co.kr"

    with urlopen(url) as f:                                     ⑩
        charset = f.info().get_param('charset')                 ⑪
        data = f.read().decode(charset)

    dataSet = parse_image(data)                                 ⑫

    print("\n>>>>>>>>> Fetch Images from", url)
    print('\n'.join(sorted(dataSet)))                           ⑬

if __name__ == '__main__':                                      ⑭
    main()
```

2.2.4 http.client 모듈

순번	코딩 순서	코딩 예시
1	연결 객체 생성	conn = http.client.HTTPConnection("www.python.org")
2	요청을 보냄	conn.request("GET", "/index.html")
3	응답 객체 생성	response = conn.getresponse( )
4	응답 데이터를 읽음	data = response.read( )
5	연결을 닫음	conn.close( )

표 **2-3** http.client 모듈 사용 시 코딩 순서

GET, POST 이외의 방식으로 요청을 보내거나, 요청 헤더와 바디 사이에 타이머 를 두어 시간을 지연시키는 등 urllib.request 모듈로는 쉽게 처리할 수 없는 경우 혹은 HTTP 프로토콜 요청에 대한 저수준의 더 세밀한 기능이 필요할 때는 http.client 모듈을 사용함

SECTION 02_2웹 클라이언트 라이브러리

2.2.5 http.client 모듈 예제

예제 2-12 http.client 모듈 예제 — 소스제공: ch2₩download_image.py

```
C:₩Users₩shkim>cd C:₩RedBook₩ch2
C:₩RedBook₩ch2>notepad download_image.py

import os                                                              ①
from http.client import HTTPConnection
from urllib.parse import urljoin, urlunparse
from urllib.request import urlretrieve
from html.parser import HTMLParser

class ImageParser(HTMLParser):                                         ②
    def handle_starttag(self, tag, attrs):                             ③
        if tag != 'img':                                               ④
            return
        if not hasattr(self, 'result'):
            self.result = []
        for name, value in attrs:
            if name == 'src':
                self.result.append(value)

def download_image(url, data):                                         ⑤
    if not os.path.exists('DOWNLOAD'):                                 ⑥
        os.makedirs('DOWNLOAD')

    parser = ImageParser()                                             ⑦
    parser.feed(data)
    dataSet = set(x for x in parser.result)                            ⑧
    for x in sorted(dataSet) :                                         ⑨
        imageUrl = urljoin(url, x)                                     ⑩
        basename = os.path.basename(imageUrl)
        targetFile = os.path.join('DOWNLOAD', basename)

        print("Downloading...", imageUrl)
        urlretrieve(imageUrl, targetFile)                              ⑪

def main():                                                            ⑫
    host = "www.google.co.kr"

    conn = HTTPConnection(host)                                        ⑬
    conn.request("GET", '')
    resp = conn.getresponse()

    charset = resp.msg.get_param('charset')                            ⑭
    data = resp.read().decode(charset)
    conn.close()

    print("₩n>>>>>>>>> Download Images from", host)
    url = urlunparse(('http', host, '', '', '', ''))                   ⑮
    download_image(url, data)                                          ⑯

if __name__ == '__main__':                                             ⑰
    main()
```

SECTION 02_3웹 서버라이브러리

2.3.1 간단한 웹 서버

예제 2-13 간단한 웹 서버 — 소스제공: ch2₩my_httpserver.py

```
C:₩Users₩shkim>cd C:₩RedBook₩ch2
C:₩RedBook₩ch2>notepad my_httpserver.py

from http.server import HTTPServer, BaseHTTPRequestHandler ------ 1

class MyHandler(BaseHTTPRequestHandler):
    def do_GET(self):
        self.send_response_only(200, 'OK')
        self.send_header('Content-Type', 'text/plain')
        self.end_headers()
        self.wfile.write(b"Hello World")

if __name__ == '__main__':
    server = HTTPServer(('', 8888), MyHandler) ------ 3
    print("Started WebServer on port 8888...")
    print("Press ^C to quit WebServer.")
    server.serve_forever() ------ 4
```

(2)

클래스명	주요 기능
HTTPServer	• 웹 서버를 만들기 위한 클래스로, 서버 IP와 PORT를 바인딩함 • HTTPServer 객체 생성 시, 핸들러 클래스가 반드시 필요함
BaseHTTPRequestHandler	• 핸들러를 만들기 위한 기반 클래스로, HTTP 프로토콜 처리 로직이 들어있음 • 이 클래스를 상속받아, 자신의 로직 처리를 담당하는 핸들러 클래스를 만듦
SimpleHTTPRequestHandler	• BaseHTTPRequestHandler 클래스를 상속받아 만든 클래스 • GET과 HEAD 메소드 처리가 가능한 핸들러 클래스
CGIHTTPRequestHandler	• SimpleHTTPRequestHandler 클래스를 상속받아 만든 클래스 • 추가적으로 POST 메소드와 CGI 처리가 가능한 핸들러 클래스

2.3.2 HTTPServer 및 BaseHTTPRequestHandler 클래스

앞 절에서 설명한 것처럼 우리가 원하는 웹 서버를 만들기 위해서는 기반 클래스를 임포트하거나 상 속받아야 함.
이처럼 기반이 되는 클래스가 바로 **HTTPServer** 및 **BaseHTTPRequestHandler** 클래스

```
C:₩RedBook₩ch2>python my_httpserver.py
```

```
C:₩RedBook₩ch2>python my_httpserver.py
Started WebServer on port 8888...
Press ^C to quit WebServer.
```

그림 **2-4** 웹 서버(my_httpserver.py) 실행 화면

```
http://127.0.0.1:8888/
```

그림 **2-5** 웹 서버(my_httpserver.py) 접속 결과

2.3.3 SimpleHTTPRequestHandler

```
C:₩RedBook₩ch2>python -m http.server 8888
```

```
C:₩RedBook₩ch2>python -m http.server 8888
Serving HTTP on 0.0.0.0 port 8888 (http://0.0.0.0:8888/) ...
```

그림 **2-6** Simple 웹 서버 실행 화면

2.3.4 CGIHTTPRequestHandler 클래스

SimpleHTTPRequestHandler 클래스와 유사하게 CGIHTTPRequestHandler 클래스가 미리 구현되어 있어서 필요할 때 즉시 웹 서버를 실행할 수 있습니다. CGIHTTPRequestHandler 클래스에는 do_POST() 메소드가 정의되어 있어서 POST 방식을 처리할 수 있습니다. 물론 SimpleHTTPRequestHandler 클래스를 상속받고 있어서, GET 및 HEAD 방식도 처리합니다.

```
C:\Users\shkim>cd C:\RedBook\ch2\cgi-server
C:\RedBook\ch2\cgi-server>python  -m http.server 8888 --cgi
```

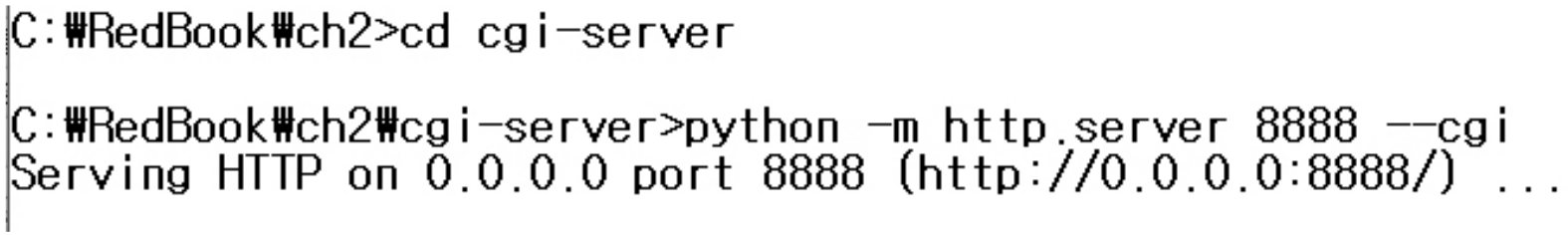


그림 **2-8** CGI 웹 서버 실행 화면

2.3.5 xxxHTTPServer 모듈 간의 관계(파이썬 2.× 버전만 해당)

모든 HTTP 웹 서버는 BaseHTTPServer 모듈의 HTTPServer 클래스를 사용하여 작성하고, 웹 서버에 사용되는 핸들러는 BaseHTTPServer 모듈의 BaseHTTPRequestHandler를 상속받아 작성. BaseHTTPServer 모듈이 기본이 되고, 이를 확장한 모듈이 SimpleHTTPServer 모듈이며, CGIHTTPServer 모듈은 SimpleHTTPServer 모듈을 확장하여 작성

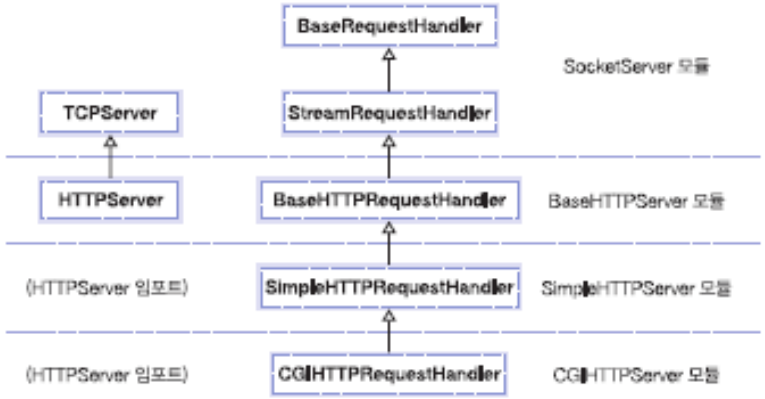


그림 **2-10** xxxHTTPServer 모듈의 클래스 간 상속도

그림 **2-11** 웹 서버용 모듈의 test( ) 함수 간 호출 관계

SECTION 2.4 CGI/WSGI 라이브러리

2.4.1 CGI 관련 모듈

웹 서버가 사용자의 요청을 애플리케이션에 전달하고 애플리케이션의 처리 결과를 애플 리케이션으로부터 되돌려받기 위한, 즉 웹 서버와 애플리케이션 간에 데이터를 주고받기 위한 규 격을 CGICommon Gateway Interface라고 함

SECTION 2.4 CGI/WSGI 라이브러리

2.4.2 WSGI 개요

앞 절에서 설명한 CGI 방식은 요청이 들어올 때마다 처리를 위한 프로세스가 생성되는 방식이라 서, 짧은 시간에 수천, 수만의 다량 요청을 받으면 서버의 부하가 높아져서 프로세스가 멈추거나 다운될 수도 있습니다. 이러한 CGI의 단점을 해결하고, 파이썬 언어로 애플리케이션을 좀 더 쉽게 작성할 수 있도록 웹 서버와 웹 애플리케이션 간에 연동 규격을 정의한 것이W SGI 규격

2.4.3 WSGI 서버의 애플리케이션 처리 과정

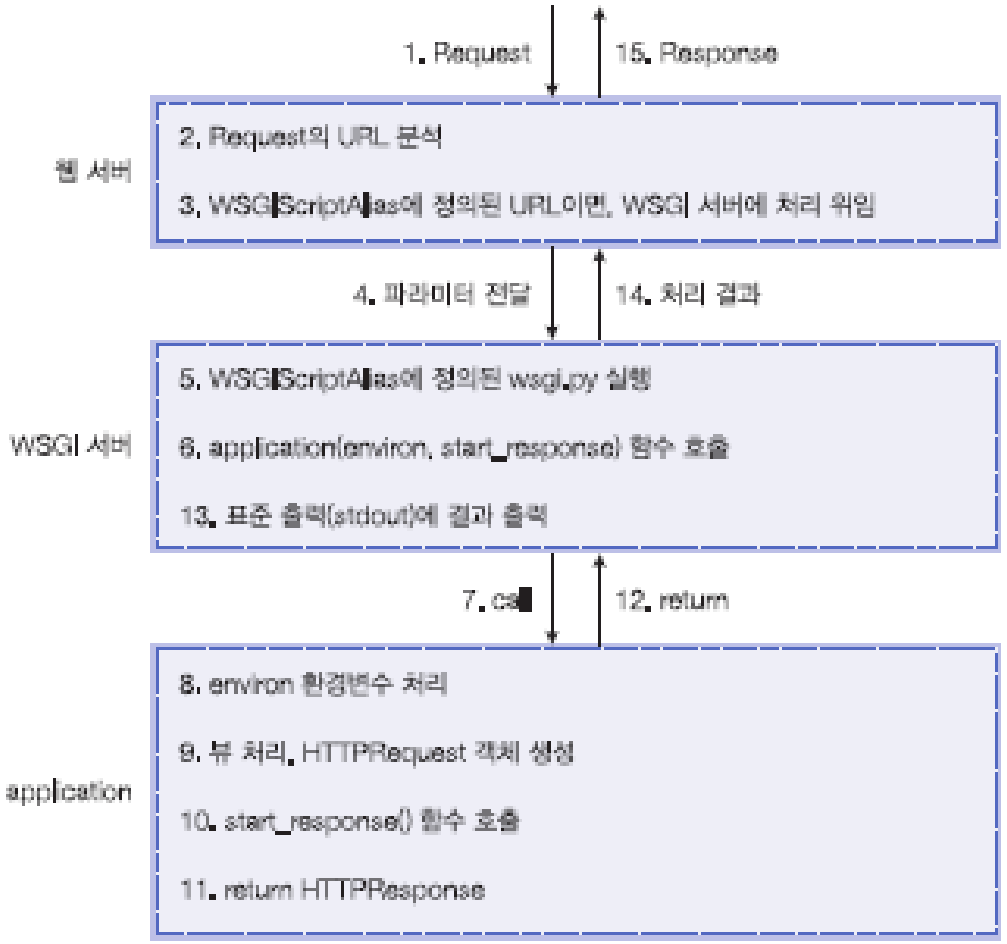


그림 **2-12** WSGI 애플리케이션의 처리 순서

SECTION 2.4 CGI/WSGI 라이브러리

2.4.4 wsgiref.simple_server 모듈

이 모듈은 WSGI 스펙을 준수하는 웹 서버(일명, WSGI 서버)에 대한 참조reference 서버, 즉 개발자에게 참고 가 될 수 있도록 미리 만들어 놓은 WSGIServer 클래스와 WSGIRequestHandler 클래스를 정의.
장고의 runserver도 이들 클래스를 사용하여 만든 테스트용 웹 서버

예제 2-16 WSGI 서버 — 소스제공: ch2₩wsgi-server₩my_wsgiserver.py

```
C:₩Users₩shkim>cd C:₩RedBook₩ch2₩wsgi-server
C:₩RedBook₩ch2₩wsgi-server>notepad my_wsgiserver.py

from wsgiref.simple_server import make_server

def my_app(environ, start_response):

    status = '200 OK'
    headers = [('Content-Type', 'text/plain')]
    start_response(status, headers)

    response = [b"This is a sample WSGI Application."]

    return response

if __name__ == '__main__':
    print("Started WSGI Server on port 8888...")
    server = make_server('', 8888, my_app)
    server.serve_forever()
```

2.4.5 WSGI 서버 동작 확인

```
C:\Users\wshkim>cd C:\RedBook\ch2\wsgi-server
C:\RedBook\ch2\wsgi-server>python my_wsgiserver.py
```

```
C:₩RedBook₩ch2₩wsgi-server>python my_wsgiserver.py
Started WSGI Server on port 8888...
```

그림 **2-14** WSGI 서버(my_wsgiserver.py) 실행 화면

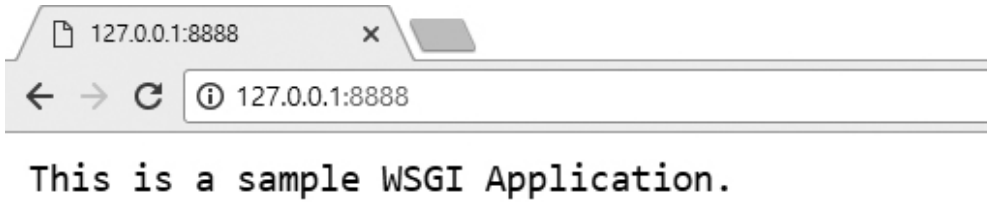


그림 **2-15** WSGI 서버에서 my_app( ) Application 실행 결과