장고

Django로 배우는
쉽고 빠른 웹 개발

김석훈 지음

파이썬 웹 프로그래밍

한빛미디어
Hanbit Media, Inc.

지은이의 말

요즘은 웹 프로그래밍의 전성시대라고 할 만큼 수많은 웹 사이트가 운영되고 있고, 개발 프로젝트 설계 시에도 웹 프로그래밍 방식으로 개발하는 것을 우선시하고 있습니다. 현재의 웹 프로그래밍은 톰캣, 스프링 프레임워크, 마이바티스 등 Java 계열이 주류를 이루고 있으나 파이썬 언어 역시 그 입지를 빠르게 넓혀가고 있습니다.

이미 파이썬 언어를 사용해 본 독자라면 쉽고 간결함, 문법의 일관성, 빠르게 개발할 수 있는 생산성에 만족해하고 있으리라 생각합니다. 다양한 기능을 제공해주는 라이브러리, C 언어와의 접착성, 콜백 함수, 람다 함수, 이터레이터, 제너레이터 등을 경험해보면 절로 감탄하게 될 것입니다.

물론 파이썬에 장점만 있는 것은 아닙니다. 스크립트가 갖는 태생적 한계, 인터프리터당 1개의 스레드만 허용하는 GIL(Global Interpreter Lock)과 같은 특징으로 인해 속도가 우려되는 것도 사실입니다. 그러나 필자가 대기업의 개발 프로젝트에서 파이썬 토네이도(tornado) 프레임워크를 사용하여 웹 서버에 5개의 인스턴스를 띄우고 동시 접속으로 10만 개 이상의 요청을 처리해 내는 것을 경험한 후에는 위와 같은 우려가 기우였음을 깨닫게 되었습니다.

이후 파이썬의 웹 프로그래밍 라이브러리와 파이썬으로 된 프레임워크에 관심을 갖게 되었고, 범용성이 있고 장점이 많은 장고(Django) 프레임워크에 매력을 느껴 이 책을 집필하게 되었습니다.

파이썬에서 제공하는 표준 라이브러리를 사용하면 우리가 필요한 웹 클라이언트를 직접 개발할 수 있으며, 간단한 테스트용 웹 서버도 즉시 만들 수 있습니다. 물론, 상용 목적의 웹 서버를 개발하는 것이라면 다양한 웹 서버 프레임워크 중에서 자신의 목적에 맞는 프레임워크를 선택해서 사용하면 됩니다.

파이썬 웹 프레임워크는 매우 다양합니다. 이 중에서 장고는 파이썬 철학과도 맞닿아 있어 쉽고 빠르게 개발을 진행할 수 있도록 다양한 기능을 구비하고 있습니다. 웹 프로그래밍에 필요한 기능은 모두 있다고 할 수 있을 정도로 기능이 풍부하고, 이러한 기능을 간단히 몇 줄로 코딩할 수 있도록 장고 자체 기능으로 제공하고 있습니다.

중급자라면 프레임워크들의 장단점을 잘 따져서 자신의 프로젝트에 적합한 프레임워크를 선택해서 사용할 수도 있겠지만, 선택이 쉽지 않다거나 처음 파이썬 웹 프로그래밍을 접하는 독자들은 장

고 프레임워크로 시작하기를 권장합니다. 개발을 바로 시작할 수 있도록 프로그래밍 뼈대를 만들어주고, 어렵게 생각되는 데이터베이스 연동, 어드민 관리 기능이 쉽게 처리되기 때문입니다. 또한, 장고에는 웹 프로그래밍에 필요한 개념들이 잘 정립되어 있어서 장고를 어느 정도 사용해 본 후에는 다른 프레임워크로도 쉽게 웹 프로그램 개발이 가능할 것입니다.

필자가 생각하는 이 책의 독자층은 파이썬 문법을 공부한 후에 웹 프로그래밍을 배우고자 하거나, 다른 언어로 웹 프로그래밍을 하다가 파이썬으로 전환하고자 하는 개발자입니다. 해당 독자가 파이썬 웹 프로그래밍을 해보려고 한다면 이 책이 길잡이가 될 것입니다. 이 책의 설명과 예제를 통해 다음과 같은 기술을 습득할 수 있습니다.

- 웹 프로그래밍에 필요한 파이썬 표준 라이브러리를 활용할 수 있다.
- 웹 서버 뿐 아니라 자신의 목적에 맞는 웹 클라이언트를 개발할 수 있다.
- 파이썬 프레임워크 중 범용성이 가장 높은 장고를 사용해서 웹 서버를 개발할 수 있다.
- Model, Template, View에 따른 설명과 실습을 통해 Django에서의 애플리케이션 개발 방식과 그 원리에 대해 이해할 수 있다.

이 책은 파이썬 웹 프로그래밍을 시작하려는 독자를 대상으로, 필수적으로 알아야 하는 기본 기능 위주로 설명하였습니다. 실제 상용화 프로젝트를 개발코자 하는 독자는 이 책에서 다룬 주제에 대한 심화 학습 이외에도 장고 고급 기능에 대한 공부가 필요합니다.

독자 여러분이 이 책을 통해서 파이썬 웹 프로그래밍의 기본 기술과 핵심 원리를 이해하고, 장고 웹 프레임워크를 여러분의 프로젝트에 자유자재로 사용할 수 있기를 기대합니다.

끝으로, 파이썬 웹 프로그래밍 출간을 흔쾌히 허락해주시고, 정성들여 편집해주신 한빛미디어 송성근 팀장님, 이미향 과장님께 감사하다는 말을 전합니다.

또한 사랑하는 가족에게도 제 마음을 전하고 싶습니다. "항상 가족에게 헌신적인 혜정, 자신의 길을 당당히 걸어가고 있는 영림이, 뭔가 해낼 것 같은 학림아, 고마워..."

저자_ 김석훈

일러두기

필자의 실습 환경

이 책의 본문은 장고의 예제를 실습하면서 파이썬 코드를 입력하고 그 결과를 확인하도록 구성되어 있습니다. 파이썬은 리눅스/유닉스, Mac OS X, 윈도우와 같은 운영체제가 달라져도 애플리케이션 레벨에서 변경할 것은 없습니다. 따라서 파이썬 위에서 실행되는 장고 역시 운영체제와 무관하게 실행이 가능하므로, 본문의 예제를 실습하는 데 어려움은 없습니다.

다만, 독자 여러분이 실습하는 과정에서 원하는대로 진행되지 않을 경우, 이를 해결하는 과정에서 필자의 실습 환경을 알고 있는 것이 도움이 될 것입니다. 이 책을 집필하면서 사용한 필자의 실습 환경은 다음과 같습니다.

Django	버전 1.7.4
Python	버전 2.7.5
Linux	CentOS 7.0
VirtualBox	버전 4.3
Windows	Windows 8 (intel PC)

필자는 VirtualBox라는 가상 머신 위에 리눅스 OS를 탑재하여 진행하였으므로, 다음 사항을 유의하기 바랍니다.

❶ 본문에 나와 있는 프롬프트, 디렉토리 체계, 커멘드 등은 모두 리눅스에서 실행한 모습입니다.

- **리눅스 예 :** [shkim@localhost pyBook] $ python -c "import sys; print sys.path"
- **윈도우 예 :** C:₩shkim₩pyBook〉 python -c "import sys; print sys.path"

❷ 예제 코드를 입력하기 위한 편집기는 리눅스의 기본 편집기인 vi를 사용하였습니다.

- 리눅스의 emacs 편집기나 윈도우의 EditPlus, Mac OS X의 텍스트 편집기 등 일반 텍스트 편집기는 모두 가능합니다.

- 윈도우 환경이라면 파이썬 기본 편집기인 IDLE, 파이썬 개발 툴인 PyCharm, 통합 개발 툴인 Eclipse(PyDev 플러그인 필요) 등도 많이 사용하는 편입니다.

❸ 가상머신을 사용했으므로, 웹 서버의 IP 주소를 '192.168.56.101'로 사용하였습니다.

- 가상머신을 사용하지 않는 경우는 로컬호스트 IP 주소인 '127.0.0.1'을 사용하면 됩니다.
- 별도의 서버 박스에서 웹 서버를 실행하는 경우는 그 서버의 IP 주소를 사용하면 됩니다. (서버의 IP 주소는 ifconfig(리눅스), ipconfig(윈도우) 명령으로 알 수 있습니다.)

❹ 이 책의 예제는 Python 2.7.X 버전을 사용하였으므로 2.7.X 이상의 버전을 설치할 것을 권장합니다.

- Python 3.x 버전을 사용하더라도 변경사항이 많지는 않습니다. 1장과 2장은 print 문장 및 라이브러리명을 수정하는 정도이고, 3장 이후의 장고 예제는 수정사항이 없습니다.
  버전 변환 툴인 '2to3'(윈도우는 '2to3.py')을 사용하면 본문의 예제를 3.x 버전으로 변경할 수 있습니다.
- virtualenv 툴을 설치하면 파이썬 2.x와 3.x를 같이 사용할 수 있습니다.

❺ 이 책의 예제는 Django 1.7.4 버전을 사용하였으므로, 1.7.4 버전부터의 설치가 필요합니다. 현재 최신 버전인 Django 1.9.x 버전을 사용하더라도 본문의 예제는 정상적으로 동작합니다.

장고의 온라인 도큐먼트 활용

요즘은 오픈 소스 한두 가지 정도는 기본으로 사용하여 개발 프로젝트를 진행합니다. 이렇게 오픈 소스를 사용하여 개발하다가 막히거나 의문이 생기면 인터넷에서 자료를 찾고 관련 서적을 뒤적이게 됩니다. 그러나 많은 사람들은 가장 좋은 자료는 해당 오픈 소스의 온라인 도큐먼트라고 이야기합니다.

장고 역시 오픈 소스이고 이 책을 공부하다가 또는 장고를 사용하여 프로젝트를 진행하다가 자료를 찾고 싶으면 장고의 공식 홈페이지인 아래 주소를 참고하기 바랍니다.

- https://docs.djangoproject.com/en/1.7/

장고의 공식 온라인 도큐먼트를 자주 접하고 익숙해질수록 장고에 대한 이해도가 높아지고 여러분의 실력도 향상될 것입니다.

또한, 이 책에서 사용한 실습 예제는 장고의 온라인 도큐먼트에서 발췌하여 사용하였음을 알려드립니다. 그 이유는 이 책에서 다룬 내용을 좀 더 깊이 이해하고 싶어 온라인 도큐먼트를 찾는 경우에 동일한 예제를 공부했던 경험이 도큐먼트를 이해하는 데 많은 도움을 줄 수 있을 것이라 판단했기 때문입니다.

부디 독자 여러분이 이 책을 통해서 장고의 온라인 도큐먼트에 익숙해지고 더욱 더 장고를 잘 활용할 수 있기를 기대합니다.

소스 제공 리스트

예제 디렉토리	예제 파일명	본문의 관련 부분
pyBook/ch1/	example.py	**1.2.4 직접 만든 클라이언트로 요청**에서 실습하는 예제
pyBook/ch2/	parse_image.py	예제 2-7 urllib2 모듈 예제 - parse_image.py
pyBook/ch2/	download_image.py	예제 2-12 httplib 모듈 예제 - download_image.py
pyBook/ch2/	DOWNLOAD/*	download_image.py 실행 결과
pyBook/ch2/	myhttpserver.py	예제 2-13 간단한 웹 서버 - Hello World
pyBook/ch2/	cgi_client.py	예제 2-14 CGIHTTPServer 시험용 클라이언트 - cgi_client.py
pyBook/ch2/	cgi-bin/script.py	예제 2-15 CGIHTTPServer 시험용 CGI 스크립트 - cgi-bin/script.py
pyBook/ch2/	mywsgiserver.py	예제 2-16 WSGI 서버
pyBook/ch3-4/	*.py, *.html	**3.4 프로젝트 뼈대 만들기**에서 진행한 프로젝트 및 애플리케이션 파일들로, **3.4.4 지금까지 작업 확인하기**에서 실습 시 사용하는 예제

pyBook/ch3-6/	*.py, *.html	**3.6 애플리케이션 개발하기 – Model 코딩**에서 진행한 프로젝트 및 애플리케이션 파일들로, **3.6.5 지금까지 작업 확인하기**에서 실습 시 사용하는 예제
pyBook/ch3-7/	*.py, *.html	**3.7 애플리케이션 개발하기 – View 및 Template 코딩**에서 진행한 프로젝트 및 애플리케이션 파일들로, **3.7.6 지금까지 작업 확인하기**에서 실습 시 사용하는 예제
pyBook/ch4/	*.py, *.html	**4.1 Admin 사이트 꾸미기**에서 진행한 프로젝트 및 애플리케이션 파일들로, **4.1.11 polls/admin.py 변경 내역 정리**에서 최종 정리한 예제와 **4.1.12 Admin 사이트 템플릿 수정**에서 설명하는 예제
pyBook/ch5-1/	*.py, *.html	**5.1 새로운 애플리케이션 만들기**에서 진행한 프로젝트 및 애플리케이션 파일들로, **5.1.8 지금까지 작업 확인하기**에서 실습시 사용하는 예제들임
pyBook/ch5-2/	*.py, *.html	**5.2 프로젝트 첫 페이지 만들기**에서 진행한 프로젝트 및 애플리케이션 파일들로, **5.2.5 지금까지 작업 확인하기**에서 실습 시 사용하는 예제
pyBook/ch5-3/	*.py, *.html	**5.3 polls 애플리케이션 – 클래스형 뷰로 변경하기**에서 진행한 프로젝트 및 애플리케이션 파일들로, **5.3.5 지금까지 작업 확인하기**에서 실습 시 사용하는 예제
pyBook/ch6-4/	httpd.conf-move TO-etc.httpd.conf	예제 6-1 아파치 설정 – 내장 모드로 실행하는 경우 아파치 설정 파일인 /etc/httpd/conf/httpd.conf 파일에서 수정이 필요한 사항을 기록한 파일
pyBook/ch6-4/	*.py, *.html	내장 모드로 실행 시 필요한 프로젝트 및 애플리케이션 파일들로, **6.4.2 지금까지 작업 확인하기**에서 실습 시 사용하는 예제
pyBook/ch6-5/	httpd.conf-move TO-etc.httpd.conf	예제 6-2 아파치 설정 – 데몬 모드로 실행하는 경우 아파치 설정 파일인 /etc/httpd/conf/httpd.conf 파일에서 수정이 필요한 사항을 기록한 파일
pyBook/ch6-5/	*.py, *.html	데몬 모드로 실행 시 필요한 프로젝트 및 애플리케이션 파일들로, **6.5.2 지금까지 작업 확인하기**에서 실습 시 사용하는 예제

1. *.py : 모델, 뷰 등에 사용되는 파이썬 파일
2. *.html : 템플릿에 사용되는 HTML 파일
3. 디렉토리명이 /ch장-절/인 경우는 /ch장/으로 변경 후 실습하세요(디렉토리명 변경 명령 예시 $ mv ch6-5 ch6).

베타 리뷰어의 말

제가 파이썬을 처음 접했던 것은 2002년도였습니다. 처음 파이썬을 접했을 때 뱀 모양 아이콘이 참 독특했던 기억이 납니다. 인터프리터 언어로 IDE 창에서 입력하면 그대로 결과가 반영되는 것이 신기하고 재미있기도 했습니다. 독특한 프로그램 언어가 새로 나온 것에 관심도 갔지만, 한편으로 누군가는 이렇게 재미난 언어를 만들어내는데 나는…? 하면서 자괴감에 빠지기도 했습니다.

파이썬의 들여쓰기 블록구분은 지금도 익숙하지 않은 부분입니다. 일반적인 블록구분에 익숙한 저에게 들여쓰기 블록구분은 여전히 낯설게 다가옵니다. 요즘 나오는 언어들의 프로그래밍을 보면 상당히 직관적인 표현으로 로직을 풀어내는 것을 볼 수 있습니다. 자바에 익숙한 저는 그런 언어들을 접할 때마다 너무 포장하여 풀어내는 방식에 익숙한 것이 아닌가 하는 생각도 듭니다. 파이썬은 이러한 직관적인 표현의 대표적인 언어 중 하나라고 생각합니다.

처음에는 이러한 특징들이 익숙지 않아 어색하기도 했지만 그럼에도 불구하고 한 번씩 파이썬에 대한 새로운 소식이 없는지 살펴보는 것은 파이썬이 그만큼 매력이 있는 언어이기 때문입니다. 이번에도 그렇게 파이썬을 살펴보다 장고 프레임워크에 대해서 알게 되었고, 좋은 기회가 되어 이렇게 베타 리뷰어로 참여하게 되었습니다.

개인적으로는 레일스 같은 자동화 프레임워크 같아 책의 내용에 더욱 흥미가 있었습니다. 여러 언어에서 자동화 프레임워크가 나오는 것으로 봐서 이제는 자동화가 흐름인가 하는 생각도 듭니다. 책을 읽으면서 장고 프레임워크만이 아닌 기본적인 HTTP 프로토콜에 대해서도 좀 더 많은 내용을 전달하려는 저자의 노력을 엿볼 수 있었습니다. 덕분에 장고 프레임워크 외에도 여러 내용을 배울 수 있어 좋았습니다.

장고 프레임워크를 이용하여 실제로 실습해보는 부분은 거창하게 프로젝트를 설계하고 구현해보지는 않지만, 프레임워크를 이용하여 개발하는 데 필요한 기본적인 부분에 중점을 두어 자세히 설명하고 있기 때문에 장고 프레임워크를 맛보고 싶은 독자들에게 좋은 가이드가 될 것입니다.

여러분도 파이썬의 장고 프레임워크의 매력에 빠져보길 바랍니다.

송재운, 프리랜서 개발자

CONTENTS

CONTENTS

CHAPTER 2 파이썬 웹 표준 라이브러리

CHAPTER 3 Django 웹 프레임워크

CONTENTS

CHAPTER 5 실습 예제 확장하기

CHAPTER 6 웹 서버(Apache)와 연동

APPENDIX A 장고의 데이터베이스 연동

CHAPTER 01

웹 프로그래밍의 이해

파이썬으로 웹 프로그래밍을 시작하기 전에 웹 프로그래밍의 기본 기술에 대해 이해할 필요가 있습니다. 이는 파이썬뿐만 아니라 다른 언어를 사용하여 웹 프로그래밍을 한다고 해도 반드시 이해하고 있어야 하는 필수 기술입니다.

웹 프로그램은 기본적으로 클라이언트-서버로 이루어집니다. 1장에서는 웹 프로그래밍의 기본 개념인 웹 클라이언트와 웹 서버뿐만 아니라, 웹 클라이언트와 웹 서버 간의 통신 규약인 HTTP 프로토콜, 주고받는 메시지 중에서 가장 중요한 URL, 그리고 혼동하기 쉬운 웹 서버와 웹 애플리케이션 서버 간의 차이점과 각각의 특징 등에 대해서 알아보겠습니다.

1.1 웹 프로그래밍이란?

웹 프로그래밍이란 무엇일까요? 간단히 말하면, HTTP 프로토콜로 통신하는 클라이언트와 서버를 개발하는 것입니다. 웹 클라이언트와 웹 서버를 같이 개발할 수도 있고, 웹 클라이언트 또는 웹 서버 하나만 개발할 수도 있습니다. 보통은 웹 서버를 개발하는 경우가 많아서 파이썬 웹 프로그래밍이라고 하면 먼저 장고Django와 같은 웹 프레임워크를 사용하여 웹 서버를 개발하는 것을 떠올리게 됩니다.

쉬운 이야기부터 시작하겠습니다. 브라우저를 실행하여 네이버에 접속하는 것도 웹 프로그램이 동

작하는 것이라 볼 수 있습니다. 이 경우에는 브라우저가 웹 클라이언트이고, 네이버 서버가 웹 서버가 됩니다. 즉, 웹 클라이언트가 요청하고 웹 서버가 응답하는 클라이언트-서버 프로그램이 동작하는 것입니다. Internet Explorer, Chrome, Firefox와 같은 브라우저는 이미 웹 클라이언트로서 개발되어 있기 때문에 웹 프레임워크를 활용해서 웹 서버를 개발하는 것을 마치 웹 프로그래밍의 전부인 것처럼 착각하기 쉽지만, 실제 프로젝트를 진행하다 보면 웹 클라이언트를 개발해야 되는 상황도 많이 발생합니다.

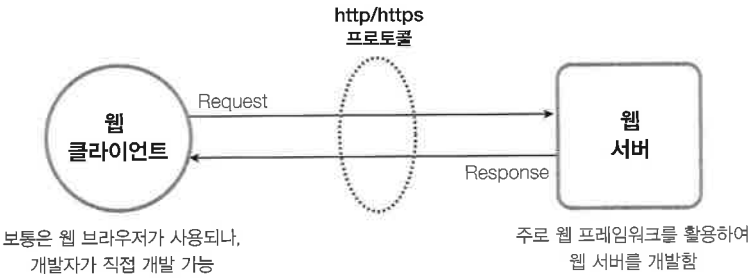


그림 1-1 웹 프로그래밍 – 클라이언트와 서버

브라우저 이외에도 웹 서버에 요청을 보내는 웹 클라이언트는 다양하게 만들 수 있는데, 대략 네 가지로 분류할 수 있습니다.

- 웹 브라우저를 사용하여 요청
- 리눅스 **curl** 명령을 사용하여 요청
- Telet을 사용하여 요청
- 직접 만든 클라이언트로 요청

이에 대해서는 **1.2 다양한 웹 클라이언트**에서 살펴보도록 하겠습니다.

1.2 다양한 웹 클라이언트

네이버와 같은 상용 웹 서버를 사용해도 되지만, 간단한 테스트이므로 여기서는 **www.example.com** 도메인에 있는 웹 서버를 대상으로 HTTP 요청Request을 보내고 응답Response을 확인해보겠습니다.

1.2.1 웹 브라우저를 사용하여 요청

다음 그림처럼 브라우저를 열고 주소창에 접속하고자 하는 웹 서버의 URL, **www.example.com**을 입력합니다.

그림 1-2 브라우저 주소 창에 URL 입력

브라우저는 주소창에 입력된 문장을 해석하여 웹 서버에게 HTTP 요청을 보내는 웹 클라이언트의 역할을 수행합니다. 요청을 받은 www.example.com 도메인의 웹 서버는 그 결과를 브라우저에게 전송해줍니다. 브라우저는 전송받은 결과를 사용자가 볼 수 있도록 화면에 보여줍니다.

그림 1-3 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 명령입니다. 쉘 프롬프트에서 다음과 같이 **curl** 명령을 입력합니다.

```
$ curl  www.example.com
```

curl 명령은 인자로 넘어온 URL로 HTTP 요청을 보내는 웹 클라이언트의 역할을 수행합니다. 이 요청을 받은 **www.example.com** 도메인의 웹 서버는 그 결과를 응답해줍니다.

```
[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

브라우저에서 보았던 문장이 동일하게 나오는 것을 확인할 수 있습니다. 즉, 어떤 방법을 사용하는지와 상관없이 웹 서버는 동일한 요청을 받을 경우 동일한 응답을 주고 있는 것입니다.

1.2.3 Telnet을 사용하여 요청

리눅스의 **telnet** 프로그램을 사용하여 HTTP 요청을 보낼 수도 있습니다. 쉘 프롬프트에서 다음과 같이 명령을 입력합니다. 마지막에는 〈Enter〉키를 두 번 입력합니다.

```
[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>키 입력
Host: www.example.com  - <Enter>키 입력
- <Enter>키 입력
```

telnet 명령은 터미널 창에서 입력하는 내용을 그대로 웹 서버에 전송합니다. 위에서 입력한 내용은 HTTP 프로토콜의 요청 메시지 규격에 정의된 규칙에 따라 HTTP 요청을 보내는 것으로, **telnet** 프로그램이 웹 클라이언트의 역할을 수행하고 있습니다. 이 요청을 받은 **www.example.com** 도메인의 웹 서버는 그 결과를 응답해줍니다. 응답 메시지는 바로 전에 입력한 두 번의 〈Enter〉키 바로 다음에 [그림 1-5]와 같이 출력됩니다. HTTP 프로토콜에 대한 자세한 설명은 이어지는 절을 참고하도록 합니다.

```
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>키 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 것에 유의해야 합니다. 이번에도 역시 동일한 응답을 확인할 수 있습니다. 다만 telnet 프로그램의 특성상 HTTP 응답 메시지의 헤더가 화면에 출력되었습니다. 이는 정상적으로 처리된 것이며 동일한 응답 결과입니다.

1.2.4 직접 만든 클라이언트로 요청

이번에는 파이썬 프로그램으로 간단한 웹 클라이언트를 만들어 보겠습니다. 다음과 같이 명령어를 통해 vi 에디터를 열고 두 개의 문장을 입력합니다.

```
$ vi example.py
import urllib2
print urllib2.urlopen("http://www.example.com").read()
```

여기서는 방금 작성한 **example.py** 프로그램이 웹 클라이언트가 됩니다. 파이썬 라이브러리를 이용해서 단 두 줄로 웹 서버에 HTTP 요청을 보내는 웹 클라이언트를 만들었습니다. 이제 웹 클라이언트를 실행하고 그 결과를 확인해보겠습니다.

```
[shkim@localhost 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>
[shkim@localhost ch1]$
```

그림 1-6 직접 만든 클라이언트 요청에 대한 웹 서버의 응답

앞에서 보았던 응답 결과와 동일한 것을 확인할 수 있습니다. 즉, 웹 클라이언트의 형태는 달라도 동일한 요청에 대해서 동일한 응답을 받는 것을 확인할 수 있습니다. 또한 반드시 웹 브라우저가 아니더라도 웹 클라이언트의 요청을 보낼 수 있다는 것을 알 수 있습니다.

참고로 파이썬에서는 한 문장으로도 HTTP 요청을 보내는 웹 클라이언트를 만들 수 있습니다. 단, 따옴표를 잘 구분해서 입력해야 합니다.

```
$ python -c "import urllib2; print urllib2.urlopen('http://www.example.com').read()"
```

> **NOTE_** 웹 클라이언트를 직접 만드는 방법은 **2.2 웹 클라이언트 라이브러리**에서 자세히 설명하고 있으니, 참고하기 바랍니다.

1.3 HTTP 프로토콜

HTTP Hypertext Transfer Protocol는 웹 서버와 웹 클라이언트 사이에서 데이터를 주고받기 위해 사용하는 통신 방식으로, TCP/IP 프로토콜 위에서 동작합니다. 즉, 우리가 웹을 이용하려면 웹 서버와 웹

클라이언트는 각각 TCP/IP 동작에 필수적인 IP 주소를 가져야 한다는 의미입니다.

HTTP란 이름대로라면 하이퍼텍스트Hypertext 전송용 프로토콜이지만, 실제로는 HTML이나 XML과 같은 하이퍼텍스트뿐만 아니라 이미지, 음성, 동영상, 자바스크립트, PDF와 각종 오피스 도큐먼트 파일 등 컴퓨터에서 다룰 수 있는 데이터라면 무엇이든 전송할 수 있습니다.

예를 들어, 우리가 웹 브라우저의 주소창에 **http://www.naver.com**을 입력하고 〈Enter〉키를 누르면 웹 클라이언트와 웹 서버 사이에 HTTP 연결이 맺어지고, 웹 클라이언트는 웹 서버에게 HTTP 요청request 메시지를 보내게 됩니다. 웹 서버는 요청에 따른 처리를 진행한 후에 그 결과를 웹 클라이언트에게 HTTP 응답response 메시지로 보냅니다. 이런 방식으로 요청 메시지와 응답 메시지가 반복적으로 오가면서 웹을 사용하게 되는 것입니다.

1.3.1 HTTP 메시지의 구조

HTTP 메시지는 클라이언트에서 서버로 보내는 요청 메시지와 서버에서 클라이언트로 보내는 응답 메시지 2가지가 있고, 그 구조는 다음 그림과 같습니다.

그림 1-7 HTTP 메시지 구조

[그림 1-7]에서 스타트라인은 요청 메시지인 경우는 요청라인request line이라고 하고, 응답 메시지인 경우는 상태라인status line이라고 합니다. 스타트라인에 이어 헤더는 각 행의 끝에 줄 바꿈 문자인 CRLF Carriage Return Line Feed가 있으며, 헤더와 바디는 빈 줄로 구분합니다. 헤더와 바디는 생략할 수 있고, 바디에는 텍스트뿐만 아니라 바이너리 데이터도 들어갈 수 있습니다.

다음은 바디가 없는 요청 메시지의 예시입니다.

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

첫 번째 줄은 요청라인으로, **요청 방식**[method], **요청 URI**, **프로토콜 버전**으로 구성됩니다. 두 번째 줄은 헤더로, **이름: 값** 형식으로 표현하며, 위 예시의 경우 헤더가 한 줄 뿐이지만 여러 줄도 가능합니다. 또한, Host 항목은 필수로 표시해줘야 하는데, 위 예시처럼 Host 헤더로 표시할 수도 있고, 아래 예시처럼 요청라인의 URI에 Host를 표시하면 Host 헤더는 생략 가능합니다. 만약 포트번호를 표시하고 싶다면 Host 항목에 같이 표시해줍니다.

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

다음은 응답 메시지의 예시입니다.

```
HTTP/1.1 200 OK
Content-Type: application/xhtml+xml; charset=utf-8

<html>
...
</html>
```

첫 번째 줄의 상태라인은 **프로토콜 버전, 상태 코드, 상태 텍스트**로 구성됩니다. 서버에서 처리 결과를 상태라인에 표시하는데, 위 예시에서는 **200 OK**이므로 정상적으로 처리되었음을 알 수 있습니다. 그 외의 상태 코드와 텍스트 구문은 [부록 B]를 참고하기 바랍니다.

두 번째 줄부터 헤더입니다. 위 예시는 헤더 항목이 하나뿐인 응답 메시지로, 이 메시지는 바디를 갖고 있기 때문에 헤더와 바디를 빈 줄로 구분하고 있습니다. 바디에는 HTML이 포함되어 있습니다.

> **NOTE_ URI란?**
> 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 무방합니다.

1.3.2 HTTP 처리 방식

HTTP 메소드method를 통해서 클라이언트가 원하는 처리 방식을 서버에게 알려줍니다. HTTP 메소드는 [표1-1]처럼 8가지로 정의되어 있는데, 이 중에서 많이 사용되는 메소드는 GET, POST, PUT, DELETE 4개의 메소드로서, 데이터 조작의 기본이 되는 CRUDCreate, Read, Update, Delete와 매핑되는 처리를 합니다.

표 1-1 HTTP 메소드 종류

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	

GET 방식은 지정한 URI의 정보를 가져오는 메소드로, 가장 많이 사용됩니다. 브라우저를 이용해서 서버로부터 웹 페이지, 이미지, 동영상 등을 가져온다고 할 때 수많은 GET 방식의 요청을 사용하게 됩니다.

POST의 대표적인 기능은 리소스를 생성하는 것으로, 블로그에 글을 등록하는 경우가 이에 해당됩니다. **PUT**은 리소스를 변경하는 데 사용됩니다. 예를 들어 블로그에서 글을 업로드한 작성자를 변경하는 경우가 이에 해당됩니다.

PUT 메소드도 리소스를 생성하는 데 사용할 수 있습니다. 이런 경우, 굳이 POST와 PUT의 용도를 구분해야 한다면 새롭게 생성한 리소스에 대한 URI 결정권이 서버 측에 있으면 POST를 사용하고, URI 결정권이 클라이언트에 있으면 PUT을 사용합니다. 예를 들어 트위터에 글을 포스팅하면 그 글에 대한 URI는 서버에서 결정하므로 POST를 사용하고, 위키처럼 클라이언트가 결정한 타이틀이 그대로 URI가 되는 경우는 PUT을 사용하는 것이 적합합니다. 그러나 이렇게 POST와 PUT의 용도를 구분하는 것이 엄밀하게 지켜지는 것은 아니고, 오히려 혼동을 줄 수도 있습니다. 그러므로 리소스의 생성은 POST, 리소스의 변경은 PUT으로 쉽게 이해하고 사용해도

무방합니다.

DELETE는 이름 그대로 리소스를 삭제하는 메소드입니다. 일반적으로 DELETE 요청에 대한 응답은 바디가 없습니다.

1.3.3 GET과 POST 메소드

앞에서 8가지의 HTTP 메소드를 소개하였지만, 현실적으로 가장 많이 사용하는 메소드는 GET과 POST 2가지입니다. 이것은 HTML의 폼에서 지정할 수 있는 메소드가 GET과 POST밖에 없기 때문이기도 합니다.

폼에서 사용자가 입력한 데이터들을 서버로 보낼 때, GET과 POST는 그 방식에 차이가 있습니다. GET은 아래의 예시처럼 URI 부분의 **?** 뒤에 **키=값** 쌍으로 이어붙여 보냅니다.

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

반면, POST에서는 GET에서 URI에 포함시켰던 파라미터들을 아래 예시처럼 요청 메시지의 바디에 넣습니다.

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

q=forms&release=1
```

이렇게 파라미터를 보내는 방식의 차이로 인하여 GET 방식을 이용하면 많은 양의 데이터를 보내기 어렵습니다. URI는 길이 제한이 있기 때문입니다. 또한 전달되는 사용자의 데이터가 브라우저의 주소창에 노출된다는 단점이 있어 보안 측면에서도 불리합니다.

따라서 폼을 사용하거나 추가적인 파라미터를 서버로 보내는 경우에는 GET보다 POST 방식을 많이 사용하게 되는데, 파이썬의 장고 프레임워크에서도 폼은 POST 방식만을 사용하고 있습니다. 위의 단점들이 아무런 영향을 끼치지 않을 경우에는 GET을 사용해도 무방합니다. 예를 들어, 네이버의 검색창에서 원하는 단어를 검색해보면 GET 방식을 사용하는 것을 확인할 수 있습니다.

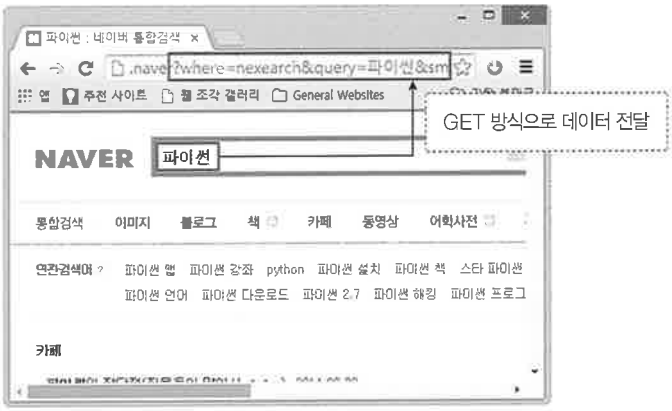


그림 1-8 네이버의 검색 창 – GET 방식 전달

1.3.4 상태 코드

서버에서의 처리 결과는 응답 메시지의 상태라인에 있는 상태 코드Status code를 보고 파악할 수 있습니다. 상태 코드는 세 자리 숫자로 되어 있는데, 첫 번째 숫자는 HTTP 응답의 종류를 구분하는 데 사용하며, 나머지 두 개의 숫자는 세부적인 응답 내용의 구분을 위한 번호입니다.

현재 100번~500번대 상태 코드가 정의되어 있는데, 첫 번째 자리 숫자를 다음 표와 같이 다섯 가지로 분류하여 사용하고 있습니다.

표 1-2 상태 코드 분류

메소드명	의미	CRUD와 매핑되는 역할
1 x x	Informational (정보 제공)	임시적인 응답으로, 현재 클라이언트의 요청까지 처리되었으니 계속 진행하라는 의미입니다. HTTP 1.1 버전부터 추가되었습니다.
2 x x	Success(성공)	클라이언트의 요청이 서버에서 성공적으로 처리되었다는 의미입니다.
3 x x	Redirection (리다이렉션)	완전한 처리를 위해서 추가적인 동작을 필요로 하는 경우입니다. 주로 서버의 주소 또는 요청한 URI의 웹 문서가 이동되었으니, 그 주소로 다시 시도해보라는 의미입니다.
4 x x	Client Error (클라이언트 에러)	없는 페이지를 요청하는 것처럼 클라이언트의 요청 메시지 내용이 잘못된 경우입니다.
5 x x	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.

다음은 자주 사용되는 상태 코드를 요약 정리한 것입니다. 첫 번째와 두 번째 컬럼인 상태 코드와 상태 텍스트가 응답 메시지의 첫 줄인 상태라인에 나타납니다. 더 자세한 사항은 [부록 B]를 참고하기 바랍니다.

표 1-3 자주 사용되는 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	작성됨	**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.
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.

1.4 URL 설계

웹 애플리케이션을 개발할 때, 고객의 요구사항이 정리되면 먼저 디자인 측면에서는 화면 UI를 설계하고, 프로그램 로직 측면에서는 URL을 설계하게 됩니다. 즉, URL의 설계는 웹 서버 로직 설계의 첫걸음이고, 사용자 또는 웹 클라이언트에게 웹 서버가 가지고 있는 기능을 명시해주는 중요한 단계입니다. 전체 프로그램 로직을 생각하면서 차후에 로직이 변경되더라도 URL 변경은 최소화할 수 있도록 유연하게 설계하는 것이 중요합니다.

URL은 보통 다음과 같이 구성됩니다.

그림 1-9 URL 구성 항목

- **URL 스킴 :** URL에 사용된 프로토콜을 의미합니다.
- **호스트명 :** 웹 서버의 호스트명으로, 도메인명 또는 IP 주소로 표현됩니다.
- **포트번호 :** 웹 서버 내의 서비스 포트번호입니다. 생략 시에는 디폴트 포트번호로, http는 80을, https는 443을 사용합니다.
- **경로 :** 파일이나 애플리케이션 경로를 의미합니다.
- **쿼리스트링 :** 질의 문자열로, 앰퍼샌드(&)로 구분된 **키=값** 쌍 형식으로 표현합니다.
- **프라그먼트 :** 문서 내의 앵커 등 조각을 지정합니다.

1.4.1 URL을 바라보는 측면

URL은 웹 클라이언트에서 호출한다는 시점에서 보면, 웹 서버에 존재하는 애플리케이션에 대한 API[Application Programming Interface]라고 할 수 있습니다. 웹 프로그래밍 기술의 발전 과정을 살펴보면, 이러한 API의 명명 규칙을 정하는 방법을 두 가지로 분류할 수 있습니다. 하나는 URL을 RPC[Remote Procedure Call]로서 바라보는 방식이고, 다른 하나는 REST[Representational State Transfer]로서 바라보는 방식입니다.

RPC란 클라이언트가 네트워크상에서 원격에 있는 서버가 제공하는 API 함수를 호출하는 방식입니다. 이 방식의 배경에는 URL 설계와 API 설계를 동일하게 고려하여 URL의 경로를 API 함수명으로, 쿼리 파라미터를 함수의 인자로 간주합니다. 그래서 웹 클라이언트에서 URL을 전송하는 것이 웹 서버의 API 함수를 호출한다고 인식하는 것입니다.

RPC 방식에서는 URL 경로의 대부분이 동사가 됩니다. 자바의 일반적인 함수나 메소드명이 동사인 것과 동일한 개념입니다. RPC 방식은 웹 개발 초기부터 사용된 방식으로, 다음에 설명하는 REST 방식이 나오면서 사용 빈도가 줄어드는 추세이긴 하지만, 여전히 많이 사용되고 있습니다. RPC 방식은 다음과 같은 형태로 사용합니다.

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

URL을 바라보는 또 한 가지 측면은 REST 방식으로 URL을 설계하는 것입니다. REST 방식이란 웹 서버에 존재하는 요소들을 모두 리소스라고 정의하고, URL을 통해 웹 서버의 특정 리소스를 표현한다는 개념입니다. 리소스는 시간이 지남에 따라 상태state가 변할 수 있기 때문에 클라이언트와 서버 간에 데이터의 교환을 리소스 상태의 교환Representational State Transfer으로 간주하고 있습니다. 그리고 리소스에 대한 조작을 GET, POST, PUT, DELETE 등의 HTTP 메소드로 매핑합니다.

이와 같은 REST 방식으로 바라본다면, 웹 클라이언트에서 URL을 전송하는 것이 웹 서버에 있는 리소스 상태에 대한 데이터를 주고받는 것으로 간주할 수 있습니다. REST 방식은 다음과 같은 형태로 사용합니다.

```
http://blog.example.com/search/test
```

1.4.2 간편 URL

최근에는 위에서 설명한 REST 방식의 URL 개념을 기반으로, 간단하면서도 사용자에게 친숙하게 URL을 표현하려는 노력이 진행되었습니다. 주로 검색 엔진 분야에서 이러한 노력이 많이 진전되었고, 그 결과 기존의 길고 복잡한 URL에서 특수 문자 등을 제거하고 간결하게 만드는 방식인 간편Clean URL이 탄생하게 되었습니다. 기존 URL 방식에서 사용되는 문자(?, =, &, #)들은 웹 프로그래밍 언어 입장에서는 연산자나 특수한 용도의 기호로 사용될 가능성이 높기 때문에 검색 엔진에서 이런 주소를 저장하고 표현하는 데 불편이 따르기도 했습니다.

정리하면, 간편 URL은 쿼리스트링 없이 경로만 가진 간단한 구조의 URL을 말합니다. 검색 엔진의 처리를 최적화하기 위해 생겨난 간편한 URL은 URL을 입력하거나 기억하기 쉽다는 부수적인 장점도 있어, 검색 엔진 친화적 URLsearch engine friendly url 또는 사용자 친화적 URLuser friendly url이라고 부르기도 합니다.

다음과 같이 기존의 URL과 간편 URL을 비교할 수 있습니다.

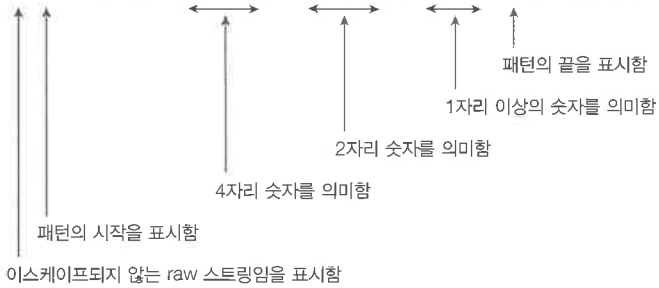
표 1-4 기존 URL과 간편 URL 간의 비교를 위한 대응 예시

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

1.4.3 파이썬의 우아한 URL

파이썬 프레임워크에서는 처음부터 간편 URL 체계를 도입하였습니다. 그 외에도 URL을 정의하기 위해 정규표현식Regular Expression을 추가적으로 사용하였는데, 이를 파이썬에서는 우아한Elegant URL이라고 부르며 다음과 같이 표현할 수 있습니다.

```
url(r'^articles/(₩d{4})/(₩d{2})/(₩d+)/$', views.article_detail),
```



만일 URL이 /articles/2015/03/1/이라면, 위 URL 패턴과 매칭되고, 아래와 같이 인자를 넘겨주어 뷰 함수를 호출함.
views.article_detail(request, '2015', '03', '1')

그림 1-10 우아한 URL – 패턴 매칭 예시

다음은 장고 프레임워크에서 사용하는 우아한 URL의 예시입니다. 이에 대해서는 **3.7.1 URLconf 코딩**에서 자세히 설명하도록 하겠습니다.

```
urlpatterns = patterns('',
    url(r'^articles/2003/$', views.special_case_2003),
    url(r'^articles/(\d{4})/$', views.year_archive),
    url(r'^articles/(\d{4})/(\d{2})/$', views.month_archive),
    url(r'^articles/(\d{4})/(\d{2})/(\d+)/$', views.article_detail),
)
```

1.5 웹 애플리케이션 서버

웹 클라이언트(보통은 웹 브라우저)의 요청을 받아서 처리하는 서버를 통칭하여 웹 서버라고 부르기도 하지만, 좀 더 세분화하면 다음 표와 같이 웹 서버와 웹 애플리케이션 서버로 분류할 수 있습니다.

표 1-5 웹 서버와 웹 애플리케이션 서버 구분

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 등

> **NOTE_** 웹 서버 및 웹 애플리케이션 서버라는 용어는 SW 측면의 서버 프로그램을 의미합니다. HW 측면의 용어는 **1.5.5 웹 서버와의 역할 구분**에서 설명하고 있습니다.

이 두 개의 서버가 어떻게 다른지, 또 웹 애플리케이션 서버가 무엇인지를 이해하기 위해서는 웹 서버 기술의 발전 과정을 알아볼 필요가 있습니다. [그림 1-11]에서 웹 서버 기술의 발전 과정을 요약하여 보여주고 있는데, 이 그림을 참고하여 동적 페이지를 생성하는 CGI 프로그램부터 살펴보겠습니다.

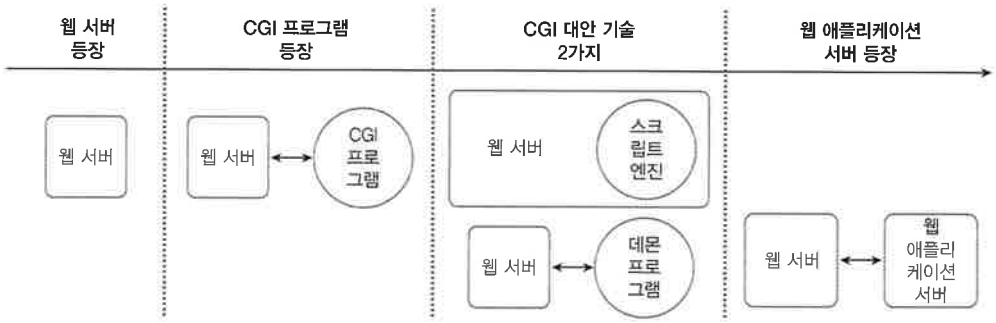


그림 1-11 기술의 발전에 따른 웹 서버 기술의 변화

1.5.1 정적 페이지 vs 동적 페이지

정적 페이지란 누가, 언제 요구하더라도 항상 같은 내용을 표시하는 웹 페이지를 말합니다. 정적 페이지들은 해당 웹 서비스의 제공자가 사전에 준비하여 서버 측에 배치한 것으로, 동일한 리소스URI의 요청에 대해서는 항상 동일한 내용의 페이지를 반환합니다. 주로 HTML, 자바스크립트, CSS, 이미지만으로 이루어진 페이지가 해당됩니다.

반면, 동적 페이지란 동일한 리소스의 요청이라도 누가, 언제, 어떻게 요구했는지에 따라 각각 다른 내용이 반환되는 페이지를 말합니다. 예를 들면 현재 시각을 보여주는 페이지나 온라인 쇼핑 사이트에서 사용자마다 다른 카트 내용을 보여주는 페이지 등이 있습니다.

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

초창기 웹이 출현했을 때는 논문 열람 사이트와 같이 정적인 웹 페이지들을 하이퍼링크로 연결하여 보여주는 것이 목적이었고, 웹 서버도 정적인 페이지를 보여주는 것이 주된 역할이었습니다. 그러나 점차 동적 페이지에 대한 요구사항이 생기고, 필요한 데이터를 저장하고 꺼내오는 등의 데

이터베이스 처리에 대한 요구가 많아짐에 따라 웹 서버와는 다른 별도의 프로그램이 필요하게 되었습니다. 이러한 별도의 프로그램과 웹 서버 사이에 정보를 주고받는 규칙을 정의한 것이 바로 CGI Common Gateway Interface 규격입니다.

1.5.2 CGI 방식의 단점

CGI 자체는 정식 프로그래밍 언어나 스크립트가 아니라, 웹 서버와 독립적인 프로그램(프로세스) 사이에 정보를 주고받는 규격을 의미하며, 이 규격을 준수하면 어떤 언어를 사용해도 CGI 프로그램을 개발할 수 있습니다. 전통적인 CGI 방식은 다음 그림과 같이 웹 서버가 C, C++, Perl, PHP 등으로 만들어진 CGI 프로그램을 직접 호출하여 개별 프로세스를 생성하는 방식입니다.

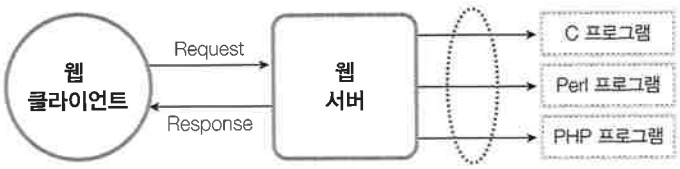


그림 1-12 전통적인 CGI 방식의 요청 처리

CGI 방식의 근본적인 문제점은 각각의 클라이언트 요청에 대하여 독립적인 별도의 프로세스가 생성된다는 것입니다. 요청이 많아질수록 프로세스가 많아지고, 프로세스가 많아질수록 비례적으로 프로세스가 점유하는 메모리 요구량도 커져서 시스템에 많은 부하를 주는 요인이 됩니다. 그래서 현재는 CGI 방식은 거의 사용되지 않고 있으며, 이러한 단점을 해결하기 위해 대안책으로 여러 가지 기술이 등장하였습니다.

1.5.3 CGI 방식의 대안 기술

CGI 방식의 대안 기술 중 하나는 별도의 애플리케이션(CGI 프로그램과 같은 역할을 하는 프로그램)을 Perl, PHP 등의 스크립트 언어로 작성하고, 스크립트를 처리하는 스크립트 엔진(인터프리터)을 웹 서버에 내장시켜서 CGI 방식의 단점이었던 별도의 프로세스를 기동시키는 오버헤드를 줄이는 방식입니다. 아파치 웹 서버에서 사용하는 mod_perl 혹은 mod_php 모듈이 Perl이나 PHP 스크립트 엔진을 웹 서버에 내장시켜 애플리케이션의 처리를 고속화하기 위해 개발된 기술

들입니다. 파이썬의 경우에는 예전의 mod_python 모듈은 더 이상 사용하지 않고 있으며, 현재는 mod_wsgi 모듈을 사용하고 있습니다.

또 다른 방식은 애플리케이션을 처리하는 프로세스를 미리 데몬으로 기동시켜 놓은 후, 웹 서버의 요청을 데몬에서 처리하는 것입니다. 이 또한 프로세스 생성 부하를 줄일 수 있는 방법입니다. 파이썬의 경우에는 데몬 방식에도 mod_wsgi 모듈을 사용합니다. mod_wsgi 모듈은 앞에서처럼 웹 서버 내장 방식으로도 실행이 가능하고, 별도의 데몬 방식으로도 실행이 가능합니다.

CGI 애플리케이션을 별도의 데몬으로 처리하는 방식은 기술이 점차 발전함에 따라, 스레드 처리가 보강되고 객체 지향 기술이 반영되면서 애플리케이션 전용 데몬인 애플리케이션 서버 방식으로 발전하였습니다. 현재 가장 많이 사용되고 있는 있는 JSP[Java Server Page], ASP[Active Server Page] 기술에서 애플리케이션 서버 방식을 사용하고 있습니다.

1.5.4 애플리케이션 서버 방식

애플리케이션 서버 방식은 웹 서버가 직접 프로그램을 호출하기보다는 다음 그림처럼 웹 애플리케이션 서버를 통해서 간접적으로 웹 애플리케이션 프로그램을 실행합니다. 웹 애플리케이션 서버는 애플리케이션 프로그램의 실행 결과를 웹 서버에 전달해주며, 웹 서버는 웹 애플리케이션 서버로부터 전달받은 응답 결과를 웹 클라이언트에 전송합니다.

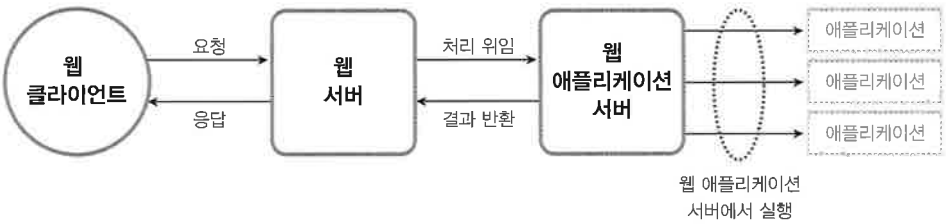


그림 1-13 애플리케이션 서버 방식의 요청 처리

애플리케이션 서버 방식을 사용하면 각 서버 간 구성도는 다음과 같은 모습이 될 것입니다.

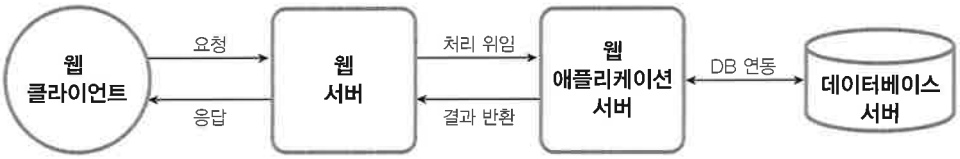


그림 1-14 애플리케이션 서버 방식에서의 서버 간 구성도

웹 서버와 웹 애플리케이션 서버가 분리됨에 따라, 서로의 역할도 구분하여 사용하는 것이 좋습니다. 왜냐하면 정적 페이지를 처리하는 경우에 비해서 동적 페이지를 처리하는 경우가 수 배에서 수십 배의 메모리를 소비하기 때문입니다. 즉, 정적 페이지 처리에 특화된 웹 서버는 정적 페이지만 처리하고, 웹 애플리케이션 서버는 동적 페이지만 처리하도록 역할을 분담하는 것이 웹 애플리케이션 서버에서 정적 페이지와 동적 페이지를 모두 처리하는 것보다 훨씬 더 많은 요청을 처리할 수 있습니다.

두 서버의 역할은 전문화되는 방향으로 계속해서 발전하고 있습니다. 웹 서버는 정적 페이지를 웹 클라이언트에게 제공하는 것이 주 역할이지만, 그 외에도 캐시 기능, 프록시 기능 등의 추가적인 기능을 제공합니다. 또한, 다수의 클라이언트로부터 동시에 요청을 받아 처리해야 하기 때문에 동시에 접속을 허가하는 클라이언트 수의 제한 및 처리 프로세스의 관리, 요청 및 응답에 관한 로그의 기록, 안정성 확보를 위한 인증 제어 및 암호화 처리 등 HTTP/HTTPS의 제어에 필요한 여러 가지 기능을 제공합니다.

웹 애플리케이션 서버는 웹 서버보다 기능이 더 추가되었고, 역할도 세분화되었습니다. 그 이유는 웹 애플리케이션 서버는 웹 서버와의 연동 규격을 잘 따르기만 하면, 임의의 언어 플랫폼을 사용해서 애플리케이션 프로그램을 작성하고 실행시킬 수 있기 때문입니다. 자바 계열의 Tomcat, Nginx 및 루비 계열의 Unicorn, 파이썬 계열의 uWSGI 애플리케이션 서버 등이 대표적인 예입니다. 대다수의 웹 애플리케이션 서버는 웹 클라이언트로부터 직접 요청을 받아 처리하는 웹 서버의 기능을 제공합니다. 그러나 이러한 웹 애플리케이션 서버 내의 웹 서버 기능들이 성능과 안정성 측면에서는 적합하지 않기 때문에 대규모의 사이트에서는 잘 사용되지 않습니다.

1.5.5 웹 서버와의 역할 구분

지금까지 웹 서버 및 웹 애플리케이션 서버라는 용어를 사용하였는데, 이는 SW Software 측면의 서버 프로그램을 의미합니다. 이들은 HW hardware 측면의 용어를 의미할 때도 있는데, 본 절에서는 좀 더 정확하게 구분하기 위해 HW 측면의 용어는 웹 서버 박스 및 웹 애플리케이션 서버 박스라는 용어를 사용하도록 하겠습니다.

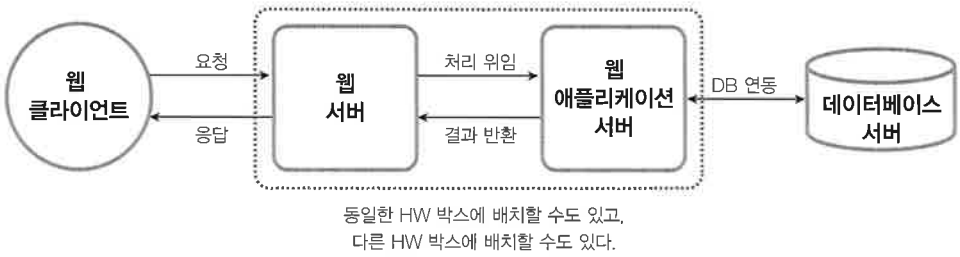


그림 1-15 웹 서버와 애플리케이션 서버의 역할과 HW 배치

앞 절에서도 설명했듯이, 거의 모든 웹 사이트가 정적 페이지와 동적 페이지를 같이 제공하는 환경에서 웹 서버 또는 웹 애플리케이션 서버 하나만으로 서비스하는 것은 매우 비효율적입니다. 정적 페이지를 처리할 때와 동적 페이지를 처리할 때의 서버 자원 소요량이 다르기 때문입니다.

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

이렇게 하나의 HW 박스에 웹 서버와 웹 애플리케이션 서버를 모두 탑재하는 것도 가능하지만, HW 박스를 분리하여 구성하면 메모리 효율을 더욱 더 높일 수 있습니다. 정적 페이지를 처리하는 웹 서버 박스와 동적 페이지를 처리하는 웹 애플리케이션 서버 박스 간의 메모리 사이즈 비율을 조절할 수 있기 때문입니다. 물론 이를 위해서는 해당 웹 사이트의 트래픽 중 정적 페이지와 동적 페이지 요청 건수 비율을 분석해야 합니다.

그래서 대형 웹 사이트에서는 HW 증설에 의해 웹 처리 용량을 높이는 작업이 용이하도록, 웹 서버를 탑재하는 HW 박스와 웹 애플리케이션 서버를 탑재하는 HW 박스를 분리하여 구성하는 것이 보통입니다. 이런 경우 HW 측면에서 L4 또는 L7 스위치 및 리버스 프록시 HW 박스 등의 도입을 고려하여 구성하게 됩니다.

CHAPTER 02

파이썬 웹 표준 라이브러리

파이썬을 설치하면 기본적으로 같이 설치되는 표준 라이브러리가 있는데, 크게는 웹 클라이언트 프로그래밍이냐 웹 서버 프로그래밍이냐에 따라 사용하는 라이브러리 모듈이 달라집니다. 또한, 파이썬 3.x 버전에서는 웹 개발 수요가 많아짐에 따라 라이브러리 모듈도 개선하고 재구성하였습니다.

2장에서는 파이썬 웹 라이브러리의 전체적인 구성 및 버전 2.x와 3.x를 비교하여 달라진 점에 대해 알아보겠습니다. 또한, 라이브러리 기능에 따라 웹 클라이언트 라이브러리와 웹 서버 라이브러리로 나누어 각각 중요한 모듈을 설명하고, 아파치와 같은 상용 웹 서버와의 연동에 필요한 WSGI 서버에 대해서도 설명하겠습니다.

2.1 웹 라이브러리 구성

파이썬의 웹 관련 라이브러리는 2.x 버전과 3.x 버전이 다르게 구성되어 있습니다. 함수와 클래스 등의 내용 자체는 거의 동일하지만, 패키지명과 모듈명이 재구성되었습니다. 이 책의 예제와 내용 설명은 모두 2.x 버전을 기준으로 하였지만, 여기서는 좀 더 개선된 구조인 3.x 버전을 기준으로 라이브러리를 설명하겠습니다.

다음 그림과 같이 3.x 버전에서는 관련된 모듈들을 모아서 패키지를 만들었고, 모듈명을 통해 서버쪽 라이브러리와 클라이언트쪽 라이브러리를 좀 더 확실히 구분하였습니다.

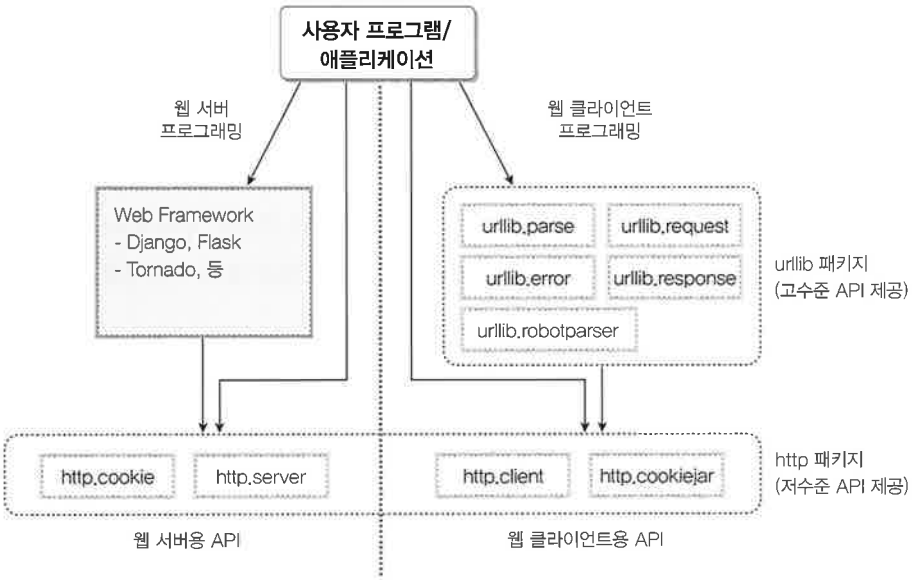


그림 2-1 웹 표준 라이브러리 구성

urllib 패키지에는 웹 클라이언트를 작성하는 데 사용되는 모듈들이 있으며, 가장 빈번하게 사용하는 모듈들입니다. http 패키지는 크게 서버용과 클라이언트용 라이브러리로 나누어 모듈을 담고 있으며, 쿠키 관련 라이브러리도 http 패키지 내에서 서버용과 클라이언트용으로 모듈이 구분되어 있습니다.

그런데 우리가 웹 서버를 개발할 때는 http.cookie 모듈이나 http.server 모듈은 거의 사용할 일이 없을 것입니다. 왜냐하면 보통 장고와 같은 웹 프레임워크를 사용하여 웹 서버를 개발하기 때문입니다.

웹 프레임워크는 사용자 프로그램과 저수준의 http.server 라이브러리 중간에 위치하여 웹 서버의 개발을 좀 더 편리하게 해주면서, 표준 라이브러리의 기능을 확장해주는 역할을 하고 있습니다. 물론 웹 프레임워크는 파이썬의 표준 라이브러리를 사용하여 개발되었습니다. 만일 여러분들이 웹 프레임워크에 관심이 있어 소스를 분석해보면 이를 확인할 수 있을 것입니다. 웹 프레임워크를 직접 개발하는 고급 프로그래머라면 http.server 모듈에서 제공하는 API를 다루겠지만, 보통의 경우라면 웹 프레임워크에서 제공하는 API를 이용하여 웹 서버를 개발해도 충분합니다.

웹 클라이언트를 개발하는 경우에는 주로 urllib 패키지를 사용합니다. http.client 모듈이 HTTP 프로토콜 처리와 관련된 저수준의 클라이언트 기능을 제공하는 반면, urllib 패키지의 모듈들은

HTTP 서버뿐만 아니라 FTP 서버 및 로컬 파일 등을 처리하는데, 클라이언트에서 공통적으로 필요한 함수와 클래스 등을 제공하고 있습니다. 주로 URL 처리와 서버 액세스 관련 API를 제공하고 있으며, HTTP 프로토콜과 관련해서는 http.client 모듈의 API를 한 번 더 추상화해서 좀 더 쉬운 고수준의 API를 제공하고 있습니다.

참고로, 파이썬 3.x의 웹 라이브러리 모듈은 크게 서버와 클라이언트 기능을 패키지로 구분하여, 파이썬 2.x의 모듈명 및 함수와 클래스들을 재구성하였습니다. 다음 표를 통해 변경사항을 확인해 보겠습니다.

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, 2.x urllib 모듈의 내용은 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 모아서 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		

2.2 웹 클라이언트 라이브러리

1장에서도 언급한 사항이지만, 우리가 가장 많이 사용하는 웹 브라우저는 단지 웹 클라이언트 중의 하나일 뿐입니다. 웹 브라우저 이외에도 웹 서버에 요청을 보내는 애플리케이션은 모두 웹 클라이언트라고 할 수 있습니다. 웹 브라우저 이외의 웹 클라이언트를 개발하고 사용하는 경우는 생각

보다 많습니다. 예를 들어, 날씨 정보를 제공하는 사이트에서 내가 사는 지역의 날씨 정보를 가져오는 프로그램이나, 트위터 사이트에서 대한민국이라는 단어가 포함된 트윗 메시지를 가져오는 프로그램 등을 직접 개발하는 경우가 이에 해당됩니다. 즉, 트위터나 구글 같은 인터넷 서비스를 제공하는 회사들은 외부의 프로그램들이 자신의 서비스를 사용할 수 있도록 OpenAPI를 제공하고 있는데, 이런 OpenAPI의 대부분이 http 프로토콜을 사용하고 있기 때문에 OpenAPI를 호출하는 프로그램도 웹 클라이언트라고 볼 수 있습니다. 또한, 시스템 간에 연동을 위하여 우리 시스템에서 상대 시스템에게 정보를 요청하고, 그 응답을 받는 연동 프로그램을 http 프로토콜을 사용하여 개발하는 경우도 이에 해당됩니다. 중요한 점은 웹 클라이언트와 서버 간에 http/https 프로토콜을 사용하여 통신한다는 것입니다.

파이썬은 이런 클라이언트를 프로그래밍할 수 있도록 여러 가지 라이브러리를 제공하고 있으며, 이러한 기능을 간단히 몇 줄로 코딩할 수 있습니다.

2.2.1 urlparse 모듈

이 모듈은 URL의 분해, 조립, 변경 등을 처리하는 함수를 제공합니다. 가장 기본적인 함수인 urlparse() 함수에 대한 예를 살펴보겠습니다.

```
>>> from urlparse 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')
```

urlparse() 함수는 URL을 파싱한 결과로 **ParseResult** 인스턴스를 반환합니다. ParseResult 클래스 속성의 의미는 다음과 같습니다.

- **scheme** : URL에 사용된 프로토콜을 의미합니다.
- **netloc** : 네트워크 위치. **user:password@host:port** 형식으로 표현되며, HTTP 프로토콜인 경우는 **host:port** 형식입니다.
- **path** : 파일이나 애플리케이션 경로를 의미합니다.
- **params** : 애플리케이션에 전달될 매개변수입니다.
- **query** : 질의 문자열로, 앰퍼샌드(&)로 구분된 **키=값** 쌍 형식으로 표시합니다.
- **fragment** : 문서 내의 앵커 등 조각을 지정합니다.

NOTE_ 그 외에도 urlsplit( ), urljoin( ), parse_qs( ) 함수 등이 있으며, 이에 대한 설명은 파이썬 홈페이지를 참고하기 바랍니다.

https://docs.python.org/2.7/library/urlparse.html#module-urlparse

2.2.2 urllib2 모듈

urllib2 모듈은 주어진 URL에서 데이터를 가져오는 기본 기능을 제공합니다. 가장 기본이 되는 urlopen() 함수의 형식은 다음과 같습니다.

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

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

NOTE_ 이외에도 URL을 처리하기 위해서는 urlretrieve( ), quote( ), unquote( ), urlencode( ) 등의 함수가 더 필요하며, 이들은 urllib 모듈에 정의되어 있습니다. 이에 대한 설명은 파이썬 홈페이지를 참고하기 바랍니다.

https://docs.python.org/2.7/library/urllib.html#module-urllib

여기서는 urlopen() 함수를 이용한 웹 클라이언트 프로그램 작성 방법 위주로 설명합니다. urlopen() 함수만 잘 다루어도 웬만한 웹 클라이언트는 대부분 작성할 수 있기 때문입니다. urlopen() 함수를 사용할 때는 다음과 같은 케이스별 작성 요령을 알아두면 편리합니다.

표 2-2 urlopen() 함수 사용 방법

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

간단한 GET, POST 방식의 요청에서부터 요청 헤더의 처리 및 인증, 쿠기의 처리와 같은 복잡한 기능 순서로 설명을 전개해 나가겠습니다.

다음은 example 사이트에 접속하는 가장 간단한 웹 클라이언트 프로그램입니다.

예제 2-1 urlopen() 함수 - GET 방식 요청

```
>>> from urllib2 import urlopen
>>> f = urlopen("http://www.example.com")
>>> print f.read(500)
```

위 프로그램은 아래처럼 쉘 프롬프트에서 바로 실행할 수도 있습니다.

```
$ python -c "import urllib2; print urllib2.urlopen('http://www.example.com').read(500)"
```

위 프로그램은 브라우저의 주소창에 **www.example.com**이라고 입력하는 것과 동일한 데이터를 웹 서버로부터 가져옵니다. 다만 브라우저는 HTML 형식의 데이터를 해석하여 화면에 보기 좋게 보여주는 반면, 위에서 살펴본 파이썬 프로그램(웹 클라이언트 프로그램)은 HTML 형식의 데이터를 해석하지 않고 그대로 보여준다는 차이가 있을 뿐입니다. 위 프로그램은 HTTP GET 방식을 디폴트로 사용하여 웹 서버에 요청을 보냅니다.

이번에는 POST 방식으로 웹 서버에 요청을 보내는 방법을 알아보겠습니다. 아래 예제처럼 urlopen() 함수를 호출 시 data 인자를 지정해주면, 함수는 자동으로 POST 방식으로 요청을 보냅니다. data 인자는 URL 인코딩된 문자열입니다.

아래 예제는 설명을 위한 코드이고, 실제 동작을 확인하려면 POST 요청을 처리할 수 있는 서버가 필요하므로, 이는 뒤에서 설명하고 있는 **2.3.4 CGIHTTPServer 모듈** 부분을 같이 참고하기 바랍니다.

예제 2-2 urlopen() 함수 - POST 방식 요청

```
>>> from urllib2 import urlopen
>>> data = "query=python"
```

```
>>> f = urlopen("http://www.example.com", data)
>>> print f.read(300)
```

만일 요청을 보낼 때 요청 헤더를 지정해서 보내고 싶은 경우에는 URL을 지정하는 방식을 변경하면 됩니다. url 인자에 문자열 대신에 Request 객체를 지정합니다. 즉, Request 객체를 생성하고, add_header()와 add_data() 메소드로 헤더와 데이터를 추가하여 웹 서버로 요청을 보내면 됩니다.

다음 예제는 설명을 위한 코드로서, GET 요청은 정상적으로 처리되지만, example 도메인의 웹 서버가 POST 처리를 구분하지 않기 때문에 GET 요청과 동일한 처리 결과가 나오므로 참고하기 바랍니다.

예제 2-3 urlopen() 함수 – Request 클래스로 요청 헤더 지정

```
>>> import urllib2
>>> req = urllib2.Request("http://www.example.com")
>>> req.add_header("Content-Type", "text/plain")
>>> req.add_data("query=python")    # POST method
>>> f = urllib2.urlopen(req)
>>> print f.read(300)
```

> **NOTE_** urllib.urlopen( ) 함수는 개선된 urllib2.urlopen( ) 함수가 있어서 2.6 버전부터 폐지 예고(deprecated)되었습니다. 3.x 버전에서는 제거되었으니 참고하기 바랍니다.

조금 더 복잡한 요청을 보내봅시다. 인증 데이터나 쿠키 데이터를 추가하여 요청을 보내거나, 프록시 서버로 요청을 보내는 등 HTTP의 고급 기능을 포함하여 요청을 보낼 수도 있습니다. 이를 위해서는 각 기능에 맞는 핸들러 객체를 정의하고, 그 핸들러를 bulid_opener() 함수를 사용하여 오프너로 등록합니다. 이 오프너를 urlopen() 함수로 열기 위해서는 install_opener() 함수를 사용하여 디폴트 오프너로 설정하면 됩니다.

다음 예제는 urllib2 모듈에 정의된 HTTPBasicAuthHandler 클래스를 사용하여 인증 데이터를 같이 보내는 프로그램입니다. 이 예제는 설명을 위한 코드로서, 정상적으로 동작하려면 인증 요청을 받는 서버의 인증 요청 API에 대한 URL을 정확히 입력해야 합니다.

예제 2-4 urlopen( ) 함수 – HTTPBasicAuthHandler 클래스로 인증 요청

```
import urllib2

# HTTP 기본 인증 요청을 위한 핸들러 생성
auth_handler = urllib2.HTTPBasicAuthHandler()
auth_handler.add_password(realm='PDQ Application',
                          uri='https://mahler:8092/site-updates.py',
                          user='klem',
                          passwd='kadidd!ehopper')
opener = urllib2.build_opener(auth_handler)

# 디폴트 오프너로 설정하면 urlopen() 함수로 요청 가능
urllib2.install_opener(opener)
u = urllib2.urlopen('http://www.example.com/login.html')
```

다음 예제는 urllib2 모듈에 정의된 HTTPCookieProcessor 클래스를 사용하여 쿠키 데이터를 같이 보내는 프로그램입니다. 이 예제는 설명을 위한 코드로, 동작을 확인하려면 쿠키를 처리하는 웹 서버에 접속해서 확인해야 합니다.

예제 2-5 urlopen( ) 함수 – HTTPCookieProcessor 클래스로 쿠키 데이터를 포함하여 요청

```
import urllib2

# 쿠키 핸들러 생성, 쿠키 데이터 처리는 디폴트로 CookieJar 객체를 사용함
cookie_handler = urllib2.HTTPCookieProcessor()

opener = urllib2.build_opener(cookie_handler)
urllib2.install_opener(opener)
# 쿠키 데이터와 함께 서버로 요청
u = urllib2.urlopen('http://www.example.com/login.html')
```

다음 예제는 urllib2 모듈에 정의된 ProxyHandler 및 ProxyBasicAuthHandler 클래스를 사용하여 프록시 서버를 통과해서 웹 서버로 요청을 보내는 프로그램입니다. 이 예제는 설명을 위한 코드로서, 정상적으로 동작하려면 프록시 서버에 대한 정보를 정확히 입력해야 합니다.

예제 2-6 urlopen( ) 함수 – ProxyHandler 및 ProxyBasicAuthHandler 클래스로 프록시 처리

```
import urllib2

proxy_handler = urllib2.ProxyHandler({'http': 'http://www.example.com:3128/'})
proxy_auth_handler = urllib2.ProxyBasicAuthHandler()
proxy_auth_handler.add_password('realm', 'host', 'username', 'password')

opener = urllib2.build_opener(proxy_handler, proxy_auth_handler)
# install_opener(), urlopen() 함수 대신에 직접 open() 함수를 사용할 수도 있음
u = opener.open('http://www.example.com/login.html')
```

2.2.3 urllib2 모듈 예제

위에서 설명한 내용을 응용하여 실제로 사용할 수 있는 웹 클라이언트를 작성해보겠습니다. 다음 예제는 특정 웹 사이트에서 이미지만을 검색하여 그 리스트를 보여주는 코드입니다. 앞에서 설명한 urllib2.urlopen() 기능을 주로 사용했으며, 추가적으로 HTMLParser 클래스를 사용하였습니다. HTMLParser 클래스는 표준 라이브러리 HTMLParser 모듈에 정의되어 있고, HTML 문서를 파싱하는 데 사용되는 클래스입니다.

예제 2-7 urllib2 모듈 예제 – parse_image.py 소스제공

```
$ cd /home/shkim/pyBook/ch2
$ vi parse_image.py

#!/usr/bin/env python                                          ❶
#
from urllib2 import urlopen                                    ❷
from HTMLParser import HTMLParser

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

```
def parseImage(data):                                                    ❻
    parser = ImageParser()
    parser.feed(data)                                                    ❼
    dataSet = set(x for x in parser.result)                              ❽
    print '\n'.join( sorted(dataSet) )                                   ❾

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

    f = urlopen(url)                                                     ⓫
    charset = f.info().getparam('charset')                               ⓬
    data = f.read().decode(charset)
    f.close()

    print "\n>>>>>>>>> Fetch Images from", url
    parseImage(data)                                                     ⓭

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

위 소스를 라인별로 설명하겠습니다.

❶ 이 프로그램이 파이썬 스크립트임을 표시합니다. 생략해도 무관합니다.

❷ 필요한 함수 및 클래스를 임포트합니다.

❸ HTMLParser 클래스를 사용할 때는 이렇게 상속받는 클래스를 정의하고 필요한 내용을 오버라이드합니다.

❹ 〈img〉 태그를 찾기 위하여 handle_starttag() 함수를 오버라이드합니다.

❺ 〈img src〉 속성을 찾으면 속성값을 self.result 리스트에 추가합니다.

❻ HTML 문장이 주어지면 ImageParser 클래스를 사용해서 이미지를 찾고, 그 리스트를 출력해주는 함수입니다.

❼ HTML 문장을 feed() 함수에 주면 바로 파싱하고 그 결과를 parser.result 리스트에 추가합니다.

❽ 파싱 결과를 set 타입의 dateSet으로 모아줍니다.

❾ dataSet으로 모은 파싱 결과를 정렬한 후에 출력합니다.

❿ 프로그램의 시작점인 메인 함수로, www.google.co.kr 사이트를 검색하여 이미지를 찾는 함수입니다.

⓫ urllib2 모듈의 urlopen() 함수를 사용하여 구글 사이트에 접속한 후 첫 페이지의 내용을 가져옵니다.

⓬ 사이트에서 가져오는 데이터는 인코딩된 데이터이므로, 인코딩 방식(charset)을 알아내어 그 방식으로 디코딩해줍니다.

⓭ 이미지를 찾기 위해 parseImage() 함수를 호출합니다.

⓮ 프로그램을 시작하기 위해 main() 함수를 호출합니다.

위 프로그램을 실행하기 위해서 다음 명령을 입력합니다.

```
$ python parse_image.py
```

정상적으로 실행이 되면 다음과 같은 메시지가 나타납니다. www.google.co.kr 사이트의 첫 페이지에는 2개의 이미지가 존재한다는 것을 확인할 수 있습니다.

```
[shkim@localhost ch2]$ python parse_image.py

>>>>>>>>> Fetch Images from http://www.google.co.kr
/images/icons/product/chrome-48.png
/textinputassistant/tia.png
[shkim@localhost ch2]$
```

그림 2-2 urllib2 모듈 예제 - parse_image.py 실행 결과

2.2.4 httplib 모듈

대부분의 웹 클라이언트 프로그램은 urllib2 모듈에 정의된 기능만으로도 작성이 가능합니다. 그러나 GET, POST 이외의 방식으로 요청을 보내거나, 요청 헤더와 바디 사이에 타이머를 두어 시간을 지연시키는 등 urllib2 모듈로는 쉽게 처리할 수 없는 경우 혹은 HTTP 프로토콜 요청에 대한 저수준의 더 세밀한 기능이 필요할 때는 httplib 모듈을 사용합니다. urllib2 모듈도 httplib 모듈에서 제공하는 API를 사용해서 만든 모듈이므로, urllib2 모듈로 작성한 로직은 httplib 모듈을 사용해도 동일하게 작성할 수 있습니다.

httplib 모듈을 사용하여 웹 클라이언트를 작성할 때는 아래와 같은 순서를 기준으로 삼고, 필요에 따라 변경하여 코딩하는 것을 권장합니다.

표 2-3 httplib 모듈 사용 시 코딩 순서

순번	코딩 순서	코딩 예시
1	연결 객체 생성	conn = httplib.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( )

httplib 모듈을 사용하여 GET, HEAD, POST, PUT 방식으로 요청을 보내는 방법을 살펴보겠습니다.

첫 번째는 httplib 모듈을 사용하여 GET 메소드로 요청을 보내는 예제입니다.

예제 2-8 httplib 모듈 사용 - GET 방식 요청

```
>>> import httplib
>>> conn = httplib.HTTPConnection("www.example.com")          ❶
>>> conn.request("GET", "/index.html")                        ❷
>>> r1 = conn.getresponse()
>>> print r1.status, r1.reason                                ❸
200 OK
>>> data1 = r1.read()                                         ❹
>>> conn.request("GET", "/parrot.spam")
>>> r2 = conn.getresponse()
>>> print r2.status, r2.reason
404 Not Found
>>> data2 = r2.read()
>>> conn.close()
```

위 소스를 라인별로 설명하겠습니다.

❶ HTTPConnection() 클래스 생성 시, 첫 번째 인자는 url이 아니라 host입니다. 그래서 **http://www.example.com**이라고 하면 에러가 발생합니다.

❷ GET 방식임을 명시적으로 표현합니다. **request(method, url, body, headers)** 형식이며, method, url 인자는 필수이고, body, headers 인자는 옵션입니다.

❸ r1.msg 속성에는 응답 헤더 정보가 들어 있습니다.

❹ 데이터를 모두 읽어야 다음 request()를 요청할 수 있습니다.

다음은 httplib 모듈을 사용하여 HEAD 메소드로 요청을 보내는 예제입니다.

예제 2-9 httplib 모듈 사용 - HEAD 방식 요청

```
>>> import httplib
>>> conn = httplib.HTTPConnection("www.example.com")          ❶
>>> conn.request("HEAD", "/index.html")                       ❷
```

```
>>> res = conn.getresponse()
>>> print res.status, res.reason                                    ❸
200 OK
>>> data = res.read()
>>> print len(data)                                                 ❹
0
>>> data == ''
True
```

위 소스를 라인별로 설명하겠습니다.

❶ HTTPConnection() 클래스 생성 시, 첫 번째 인자는 url이 아니라 host입니다. 그래서 **http://www.example.com**이라고 하면 에러가 발생합니다.

❷ HEAD 방식임을 명시적으로 표현합니다. **request(method, url, body, headers)** 형식이며, method, url 인자는 필수이고, body, headers 인자는 옵션입니다.

❸ res.msg 속성에는 응답 헤더 정보가 들어 있습니다.

❹ HEAD 요청에 대한 응답에 헤더는 있지만 바디가 없 으므로 data 길이는 0이 됩니다.

다음은 httplib 모듈을 사용하여 POST 메소드로 요청을 보내는 예제입니다.

예제 2-10 httplib 모듈 사용 - POST 방식 요청

```
>>> import httplib, urllib
>>> params = urllib.urlencode({'@number': 12524, '@type': 'issue', '@action': 'show'}) ❶
>>> headers = {"Content-type": "application/x-www-form-urlencoded",   ❷
...            "Accept": "text/plain"}
>>> conn = httplib.HTTPConnection("bugs.python.org")                  ❸
>>> conn.request("POST", "", params, headers)                         ❹
>>> response = conn.getresponse()
>>> print response.status, response.reason
302 Found
>>> data = response.read()
>>> data                                                              ❺
'Redirecting to <a href="http://bugs.python.org/issue12524">
http://bugs.python.org/issue12524</a>'
>>> conn.close()
```

위 소스를 라인별로 설명하겠습니다.

❶ POST 요청으로 보낼 파라미터에 대해 URL 인코딩을 합니다.

❷ POST 요청으로 보낼 헤더를 지정합니다.

❸ bugs.python.org 사이트에 접속을 준비합니다. HTTPConnection() 클래스 생성 시, 첫 번째 인자는 url이 아니라 host입니다. 그래서 **http://bugs.python.org**라고 하면 에러가 발생합니다.

❹ POST 방식임을 명시적으로 표현하고, 앞에서 지정한 파라미터와 헤더를 같이 보냅니다. **request(method, url, body, headers)** 형식이며, method, url 인자는 필수이고, body, headers 인자는 옵션입니다.

❺ 302 Found의 의미는 Redirection이 필요하다는 의미입니다. 만일 브라우저로 이 요청을 보냈다면, Redirection 응답에 따라 **http://bugs.python.org/issue12524**로 다시 요청을 보낼 것입니다. 상태 코드에 대한 설명은 [부록 B]를 참고하세요.

다음은 httplib 모듈을 사용하여 PUT 메소드로 요청을 보내는 예제입니다. 이 예제는 설명을 위한 코드로, 실제 동작을 확인하려면 PUT 요청을 처리할 수 있는 웹 서버가 준비되어야 합니다.

예제 2-11 httplib 모듈 사용 - PUT 방식 요청

```
>>> import httplib
>>> BODY = "***filecontents***"                                    ❶
>>> conn = httplib.HTTPConnection("localhost", 8888)               ❷
>>> conn.request("PUT", "/file", BODY)                             ❸
>>> response = conn.getresponse()
>>> print response.status, response.reason                          ❹
200 OK
```

위 소스를 라인별로 설명하겠습니다.

❶ 파일에 기록할 내용을 지정합니다.

❷ localhost에 8888번 포트로 접속을 준비합니다. HTTPConnection() 클래스 생성 시, 첫 번째 인자는 url이 아니라 host입니다. 그래서 **http://localhost**라고 하면 에러가 발생합니다.

❸ PUT 방식임을 명시적으로 표현하고, url과 body를 같이 보냅니다. **request(method, url, body, headers)** 형식이며, 앞의 method, url 인자는 필수이고, body, headers 인자는 옵션입니다.

❹ 200 OK는 성공했다는 의미로, 웹 서버에서 PUT 요청을 정상적으로 처리하여 **/file**이라는 파일을 만들고, 그 내용에 BODY에서 지정한 문자열을 기록하였다는 것입니다.

2.2.5 httplib 모듈 예제

위에서 설명한 내용을 응용하여 실제로 사용할 수 있는 웹 클라이언트를 작성해보겠습니다. 다음 예제는 특정 웹 사이트에서 이미지만을 검색하여 그 이미지들을 다운로드하는 코드입니다. 앞에서 설명한 [예제 2-7]과 유사하게 동작하지만, 이번에는 urllib2 모듈이 아니라 httplib 모듈을 사용하여 코딩하였습니다.

예제 2-12 httplib 모듈 예제 - download_image.py 소스제공

```
$ cd /home/shkim/pyBook/ch2
$ vi download_image.py

#!/usr/bin/env python                                          ❶
#
import httplib                                                 ┐
from urlparse import urljoin, urlunparse                       │
from urllib import urlretrieve                                 ├❷
from HTMLParser import HTMLParser                              │
import os                                                      ┘

class ImageParser(HTMLParser):                                 ❸
    def handle_starttag(self, tag, attrs):                     ❹
        if tag != 'img':                                       ┐
            return                                             │
        if not hasattr(self, 'result'):                       │
            self.result = []                                   ├❺
        for name, value in attrs:                              │
            if name == 'src':                                  │
                self.result.append(value)                      ┘

def downloadImage(srcUrl, data):                               ❻
    if not os.path.exists('DOWNLOAD') :                        ┐
        os.makedirs('DOWNLOAD')                                ├❼

    parser = ImageParser()                                     ┐
    parser.feed(data)                                          ├❽
    resultSet = set(x for x in parser.result)                  ❾

    for x in sorted(resultSet) :                               ┐
        src = urljoin(srcUrl, x)                               ❿
        basename = os.path.basename(src)                       ┘⓫
```

```
        targetFile = os.path.join('DOWNLOAD', basename)

        print "Downloading...", src
        urlretrieve(src, targetFile)                                    ⑫

def main():                                                             ⑬
    host = "www.google.co.kr"

    conn = httplib.HTTPConnection(host)
    conn.request("GET", '')                                             ⑭
    resp = conn.getresponse()

    charset = resp.msg.getparam('charset')
    data = resp.read().decode(charset)                                  ⑮
    conn.close()

    print "\n>>>>>>>>> Download Images from", host
    url = urlunparse(('http', host, '', '', '', ''))                    ⑯
    downloadImage(url, data)                                            ⑰

if __name__ == '__main__':                                              ⑱
    main()
```

위 소스를 라인별로 설명하겠습니다.

❶ 이 프로그램이 파이썬 스크립트임을 표시합니다. 생략해도 무관합니다.

❷ 필요한 함수 및 클래스를 임포트합니다.

❸ HTMLParser 클래스를 사용할 때는 이렇게 상속받는 클래스를 정의하고 필요한 내용을 오버라이드합니다.

❹ <img> 태그를 찾기 위하여 handle_starttag() 함수를 오버라이드합니다.

❺ <img src> 속성을 찾으면 속성값을 self.result 리스트에 추가합니다.

❻ HTML 문장이 주어지면 ImageParser 클래스를 사용해서 이미지를 찾고, 그 이미지들을 DOWNLOAD 디렉토리에 다운로드해주는 함수입니다.

❼ DOWNLOAD 디렉토리가 없으면 만들어 줍니다.

❽ HTML 문장을 feed() 함수에 주면, 바로 파싱하고 그 결과를 parser.result 리스트에 추가합니다.

❾ 파싱 결과를 set 타입의 resultSet으로 모아줍니다.

❿ resultSet으로 모은 파싱 결과를 정렬한 후에 하나씩 처리합니다.

⓫ 다운로드하기 위해 소스 URL과 타겟 파일명을 지정합니다. 소스 URL을 지정할 때 urljoin() 함수를 사용합니다. urljoin() 함수는 baseURL과 파일명을 합쳐서 완전한 URL을 리턴하는 함수입니다.

⑫ 이미지 파일을 다운로드하기 위해 urlretrieve() 함수를 사용합니다. urlretrieve() 함수는 src로부터 파일을 가져와서 targetFile 파일로 생성해줍니다.

⑬ 프로그램의 시작점인 메인 함수로, www.google.co.kr 사이트를 검색하여 이미지를 찾고 다운로드해주는 함수입니다.

⑭ httplib 모듈을 사용하여 구글 사이트에 접속한 후 첫 페이지 내용을 가져옵니다. HTTPConnection() 함수의 인자는 url이 아니라 **host:port** 형식임을 주의하기 바랍니다. port는 생략하면 디폴트로 80번 포트를 사용합니다.

⑮ 사이트에서 가져오는 데이터는 인코딩된 데이터이므로, 인코딩 방식(charset)을 알아내서 그 방식으로 디코딩해줍니다.

⑯ 다운로드 소스 URL을 지정하기 위하여 urlunparse() 함수를 사용합니다. urlunparse() 함수는 urlparse() 함수와 반대 기능을 하는데, URL 요소 6개를 튜플로 받아서 이를 조립하여 완성된 URL을 리턴하는 함수입니다.

⑰ 이미지를 다운로드하기 위해 downloadImage() 함수를 호출합니다.

⑱ 프로그램을 시작하기 위해 main() 함수를 호출합니다.

> **NOTE_ httplib 모듈**
> httplib 모듈에는 이 외에도 헤더를 정밀하게 제어할 수 있는 putheader( ), endheaders( ) 및 send( ) 등의 여러 가지 메소드가 있으며, 자세한 설명은 파이썬 홈페이지를 참고하기 바랍니다.
>
> https://docs.python.org/2.7/library/httplib.html#httpconnection-objects

위 프로그램을 실행하기 위해서 다음 명령을 입력합니다.

```
$ python download_image.py
```

정상적으로 실행되면, 아래와 같은 메시지가 나타납니다. www.google.co.kr 사이트의 첫 페이지에는 2개의 이미지가 있고, DOWNLOAD 디렉토리에 이미지가 다운로드된 것을 확인할 수 있습니다.

```
[shkim@localhost ch2]$ python download_image.py

>>>>>>>>> Download Images from www.google.co.kr
Downloading... http://www.google.co.kr/images/icons/product/chrome-48.png
Downloading... http://www.google.co.kr/textinputassistant/tia.png
[shkim@localhost ch2]$
[shkim@localhost ch2]$ ls -l DOWNLOAD/
합계 8
-rw-rw-r--. 1 shkim shkim 1834 12월  7 00:56 chrome-48.png
-rw-rw-r--. 1 shkim shkim  387 12월  7 00:56 tia.png
[shkim@localhost ch2]$
```

그림 2-3 httplib 모듈 예제 – download_image.py 실행 결과

2.3 웹 서버 라이브러리

웹 프로그래밍의 클라이언트쪽은 브라우저라는 강력한 프로그램을 사용하기 때문에, 웹 프로그래밍이라고 하면 보통은 서버쪽 프로그래밍으로 인식하는 경우가 많습니다. 이러한 웹 서버 프로그램을 작성할 때는 개발자가 직접 파이썬 라이브러리를 사용해서 웹 서버를 개발하기보다는 웹 프레임워크를 사용해서 개발하는 경우가 많습니다. 프레임워크는 개발자가 웹 서버 프로그램을 개발하기 쉽도록 저수준의 기능을 이미 만들어 놓은 기반 프로그램으로, 웹 서버 개발자는 프레임워크를 활용하여 응용 로직만 개발하면 되기 때문에 훨씬 효율적입니다.

그렇다고 해서 프레임워크만 공부하고 웹 서버쪽 표준 라이브러리를 소홀히 해서는 안 됩니다. 좀 더 경험이 쌓이고 중급, 고급 전문가가 되기 위해서는 웹 프레임워크가 어떻게 동작하는지, 파이썬의 웹 서버 라이브러리가 웹 프레임워크에 어떻게 사용되는지 등에 대한 기술을 파악할 필요가 있습니다. 여기서는 이런 측면에서 웹 서버 라이브러리의 기본 개념과 동작 원리 등을 설명하여, 차후에라도 웹 서버의 내부 동작을 분석 시 도움이 될 수 있도록 하였습니다. 만일 당장은 웹 서버 애플리케이션을 주로 개발할 예정이라면, 이 내용은 건너 뛰고 바로 장고 프레임워크를 설명하는 **3. Djang 웹 프레임워크**로 넘어가도 무관합니다.

2.3.1 간단한 웹 서버

웹 서버의 역할은 http 통신에서 클라이언트의 요청을 받고 이를 처리하여 그 결과를 되돌려주는

것입니다. 아래 예제는 웹 클라이언트로부터 요청을 받고 "Hello World"라는 문장을 되돌려주는 아주 간단한 웹 서버입니다.

예제 2-13 간단한 웹 서버 - Hello World 소스제공

```
$ cd /home/shkim/pyBook
$ vi myhttpserver.py

#!/usr/bin/env python

from BaseHTTPServer import HTTPServer, BaseHTTPRequestHandler ----------❶

class MyHandler(BaseHTTPRequestHandler):
    def do_GET(self):                                          ----------❷
        self.wfile.write("Hello World")

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

위 예제는 코딩은 간단하지만, 반드시 알아두어야 할 중요한 사항이 있습니다. 바로 웹 서버를 만드는 방법이 일정한 룰에 의해 작성되었다는 것입니다. 웹 서버를 만드는 가장 기본적인 방법을 아래에 정리하였습니다. 이를 기본으로 해서 HTTPServer 클래스나 적절한 핸들러 클래스를 상속받아 그 기능을 확장해 나가게 됩니다.

❶ BaseHTTPServer 모듈을 임포트합니다.

❷ BaseHTTPRequestHandler를 상속받아, 원하는 로직으로 핸들러 클래스를 정의합니다.

❸ 서버의 IP, PORT 및 핸들러 클래스를 인자로 하여 HTTPServer 객체를 생성합니다.

❹ HTTPServer 객체의 serve_forever() 메소드를 호출합니다.

파이썬에서는 웹 서버를 만드는 데 필요한 라이브러리를 3개의 모듈로 나누어 정의하고 있습니다. 이에 대하여 하나씩 살펴보겠습니다.

표 2-4 웹 서버용 파이썬 라이브러리 모듈

모듈명	모듈에서 정의하고 있는 내용	처리 기능
BaseHTTPServer	• 기반 서버 클래스용으로, HTTPServer 정의 • 핸들러 클래스용으로, BaseHTTPRequestHandler 정의 • 테스트용 웹 서버를 실행하는 함수, test( ) 정의	기반 클래스로, HTTP 프로토콜 처리
SimpleHTTPServer	• 기반 서버 클래스인 HTTPServer를 임포트하여 사용 • 핸들러 클래스용으로, SimpleHTTPRequestHandler 정의 • 테스트용 웹 서버를 실행하는 함수, test( ) 정의	GET과 HEAD 메소드 처리 가능
CGIHTTPServer	• 기반 서버 클래스인 HTTPServer를 임포트하여 사용 • 핸들러 클래스용으로, CGIHTTPRequestHandler 정의 • 테스트용 웹 서버를 실행하는 함수, test( ) 정의	POST와 CGI 처리 가능

2.3.2 BaseHTTPServer 모듈

앞 절에서 설명한 것처럼 우리가 원하는 웹 서버를 만들기 위해서는 기반 클래스를 임포트하거나 상속받아야 합니다. 이처럼 기반이 되는 클래스가 바로 **HTTPServer** 및 **BaseHTTPRequestHandler** 클래스이고, 이 클래스들은 BaseHTTPServer 모듈에 정의되어 있습니다. 기반 클래스에는 HTTP 프로토콜을 처리해주는 기능이 있어서, 기반 클래스를 상속받으면 따로 HTTP 프로토콜 관련된 로직을 코딩하지 않아도 되는 것입니다. 다음에 설명되는 SimpleHTTPServer, CGIHTTPServer 모듈에서도 이 기반 클래스를 상속받아 핸들러 클래스를 정의하고 있습니다.

다음 모듈 설명으로 넘어가기 전에 앞 절에서 코딩한 myhttpserver.py 파일이 정상적으로 동작하는지 확인해보겠습니다.

myhttpserver 웹 서버를 실행하기 위해서 다음 명령을 입력합니다.

```
$ python myhttpserver.py
```

정상적으로 실행이 되면, 아래와 같이 8888 포트로 요청을 기다리고 있다는 메시지가 나타납니다.

```
[shkim@localhost myBook]$ python myhttpserver.py
Started WebServer on port 8888...
Press ^C to quit WebServer
```

그림 2-4 웹 서버(myhttpserver.py) 실행 화면

위와 같이 웹 서버가 정상적으로 실행되었다면, 이제 웹 브라우저를 열고 주소창에 다음과 같이 입력합니다. IP 주소는 웹 서버가 동작하는 여러분 자신의 서버 IP 주소를 적어줍니다.

```
http://192.168.56.101:8888/
```

다음 화면처럼 "Hello World" 메시지가 나타나면 정상입니다.

그림 2-5 웹 서버(myhttpserver.py) 접속 결과

2.3.3 SimpleHTTPServer 모듈

앞 절에서는 웹 서버를 만들기 위해 우리가 직접 MyHandler라는 자신의 핸들러를 코딩하였습니다. 반면 SimpleHTTPServer 모듈에는 간단한 핸들러가 미리 구현되어 있어서, 필요할 때 즉시 웹 서버를 실행할 수 있습니다. 이 모듈에는 **SimpleHTTPRequestHandler** 클래스가 정의되어 있는데, 이 핸들러에는 do_GET() 및 do_HEAD() 메소드가 정의되어 있어서 GET 및 HEAD 방식을 처리할 수 있습니다. 그러나 POST 등 그 이외의 HTTP 메소드는 처리할 수 없습니다.

별도의 코딩 없이도 SimpleHTTPServer가 동작하는지 확인해보겠습니다. 다음 명령으로 SimpleHTTPServer 웹 서버를 실행하면 됩니다. 포트번호를 주지 않으면 디폴트로 8000번을 사용합니다.

```
$ python -m SimpleHTTPServer 8888
```

동일한 명령을 다음과 같이 사용할 수도 있습니다.

```
$ python -c 'import SimpleHTTPServer; SimpleHTTPServer.test()' 8888
```

정상적으로 실행이 되면, 아래와 같이 8888 포트로 요청을 기다리고 있다는 메시지가 나타납니다.

```
[shkim@localhost myBook]$ python -m SimpleHTTPServer 8888
Serving HTTP on 0.0.0.0 port 8888 ...
```

그림 2-6 웹 서버(SimpleHTTPServer.py) 실행 화면

위와 같이 웹 서버가 정상적으로 실행되었다면, 이제 웹 브라우저를 열고 주소창에 다음과 같이 입력합니다. IP 주소는 웹 서버의 IP 주소를 적어줍니다.

```
http://192.168.56.101:8888/
```

다음 화면처럼 서버를 실행한 디렉토리의 리스트가 나타나면 정상입니다. 디렉토리 리스트가 나오는 것은 SimpleHTTPRequestHandler의 do_GET() 메소드가 그렇게 구현되어 있기 때문입니다.

그림 2-7 웹 서버(SimpleHTTPServer.py) 접속 결과

2.3.4 CGIHTTPServer 모듈

SimpleHTTPServer 모듈과 마찬가지로, CGIHTTPServer 모듈에는 CGIHTTPRequest Handler 핸들러가 미리 구현되어 있어서 필요할 때 즉시 웹 서버를 실행할 수 있습니다. CGI HTTPRequestHandler 클래스에는 do_POST() 메소드가 정의되어 있어서 POST 방식을 처

리할 수 있습니다. 물론 SimpleHTTPRequestHandler 클래스를 상속받고 있어서, GET 및 HEAD 방식을 처리하는 것은 SimpleHTTPServer 모듈과 동일합니다. 다만 CGIHTTPServer 클래스의 do_POST() 메소드는 CGI 처리 기능만 구현되어 있어서 모든 POST 방식을 처리할 수 있는 것은 아닙니다.

이번에도 별도의 코딩 없이 CGIHTTPServer가 잘 동작하는지 확인해보겠습니다. 아래 명령으로 CGIHTTPServer 웹 서버를 실행하면 됩니다. 포트번호를 주지 않으면 디폴트로 8000번을 사용합니다.

```
$ python -m CGIHTTPServer 8888
```

또는, 동일한 명령을 다음과 같이 사용할 수도 있습니다.

```
$ python -c 'import CGIHTTPServer; CGIHTTPServer.test()' 8888
```

정상적으로 실행이 되면, 아래와 같이 8888 포트로 요청을 기다리고 있다는 메시지가 나타납니다.

```
[shkim@localhost ch2]$ cd /home/shkim/pyBook/ch2/
[shkim@localhost ch2]$ python -m CGIHTTPServer 8888
Serving HTTP on 0.0.0.0 port 8888 ...
```

그림 2-8 웹 서버(CGIHTTPServer.py) 실행 화면

이번에는 CGIHTTPServer 웹 서버가 CGI 스크립트를 정상적으로 처리하는지 확인하기 위해서 브라우저가 아니라 직접 웹 클라이언트를 만들어서 POST 방식으로 요청을 보내보겠습니다. 다음과 같이 웹 클라이언트를 코딩합니다.

예제 2-14 CGIHTTPServer 시험용 클라이언트 - cgi_client.py 소스제공

```
$ cd /home/shkim/pyBook/ch2
$ vi cgi_client.py

#!/usr/bin/env python
#
import urllib2
```

```
from urllib import urlencode

url = "http://localhost:8888/cgi-bin/script.py"
data = {
    'language' : 'python',
    'framework' : 'django',
    'email' : 'kim@naver.com'
}
encData = urlencode(data)

f = urllib2.urlopen(url, encData)       # POST
print f.read()
```

또한 CGI 스크립트도 코딩합니다. 중요한 사항은 클라이언트 요청에 담겨진 질의 문자열에 액세스하기 위해서 FieldStorage() 클래스의 인스턴스를 생성하고, 그 인스턴스의 getvalue() 메소드를 호출한다는 점입니다. 그리고 주의할 점이 있는데, CGI 스크립트 파일은 cgi-bin 디렉토리 하위에 위치해야 하고, 파일의 액세스 모드도 755로 변경해서 실행 가능한 파일로 변경되어야 한다는 것입니다.

예제 2-15 CGIHTTPServer 시험용 CGI 스크립트 - cgi-bin/script.py **소스제공**

```
$ cd /home/shkim/pyBook/ch2
$ mkdir cgi-bin
$ vi cgi-bin/script.py

#!/usr/bin/env python
#
import cgi

form = cgi.FieldStorage()
language = form.getvalue('language')
framework = form.getvalue('framework')
email = form.getvalue('email')

print "Content-type: text/plain"
print '\r'
print "Welcome, CGI Scripts"
```

```
print "language is", language
print "framework is", framework
print "email is", email

$ chmod 755 cgi-bin/script.py
```

웹 클라이언트를 직접 만들었으니, CGIHTTPServer 웹 서버로 요청을 보냅니다.

```
$ cd /home/shkim/pyBook/ch2
$ python cgi_client.py
```

아래 화면처럼 CGI 스크립트 처리 결과가 나타나면 정상입니다.

```
[shkim@localhost ch2]$ python cgi_client.py
Welcome, CGI Scripts
language is python
framework is django
email is kim@naver.com
```

그림 2-9 웹 서버(CGIHTTPServer.py)에서 CGI 스크립트 처리 결과

2.3.5 xxxHTTPServer 모듈 간의 관계

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

다음은 세 가지 모듈 간의 관계를 그림으로 표현한 것입니다.

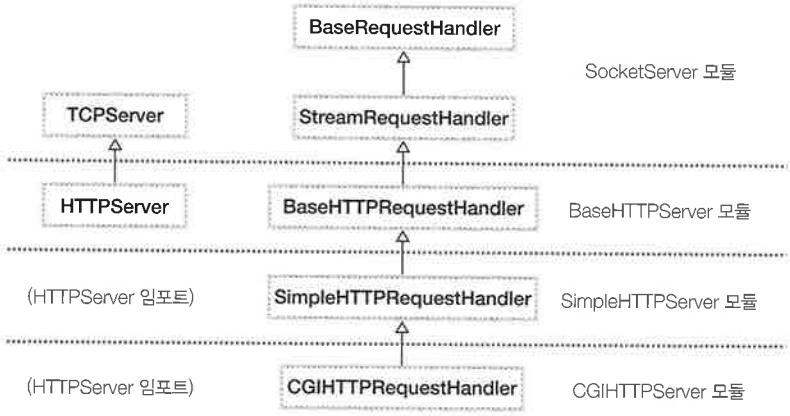


그림 2-10 xxxHTTPServer 모듈의 클래스 간 상속도

웹 서버를 정의하는 데 사용하는 HTTPServer 클래스는 BaseHTTPServer 모듈에만 정의되어 있으며, SimpleHTTPServer 및 CGIHTTPServer 모듈에서는 BaseHTTPServer 모듈의 HTTPServer 클래스를 임포트하여 웹 서버를 정의하고 있습니다. 그 의미는 다른 두 모듈에서도 기반 클래스인 HTTPServer 클래스를 변경 없이 그대로 재사용하고 있다는 것입니다.

핸들러 클래스는 각 모듈마다 정의되어 있으며, 각 핸들러 클래스는 상위 모듈의 핸들러 클래스를 상속받아 정의하고 있습니다. 그 의미는 다른 두 모듈에서는 기반 핸들러인 BaseHTTPRequestHandler 클래스의 기능을 확장하면서 정의하고 있다는 것입니다. 앞에서도 설명했듯이 HTTP 프로토콜 처리에서 GET/HEAD 메소드 처리로, 그리고 POST 메소드를 통한 CGI 처리로 그 기능을 확장하고 있습니다. 참고로, HTTP 프로토콜은 TCP 프로토콜 기반으로 동작하므로, 그림에 TCP 프로토콜을 처리하는 **SocketServer** 모듈의 클래스도 같이 표시하였습니다.

또한, 각 모듈에는 해당 HTTPServer를 별도의 코딩 없이 기동할 수 있도록 test() 함수를 정의하고 있습니다. 앞 절의 예제에서 해당 HTTPServer를 기동 시 사용했던 명령을 기억할 것입니다. 아래 명령과 같이 각 모듈의 test() 함수를 호출하여 웹 서버를 기동할 수가 있었습니다.

```
$ python -c 'import CGIHTTPServer; CGIHTTPServer.test()' 8888
```

이 또한 다음 그림처럼 각 모듈의 test() 함수 간에 상위 모듈로 호출하는 것이 체인처럼 엮여 있어서 가능한 일입니다.

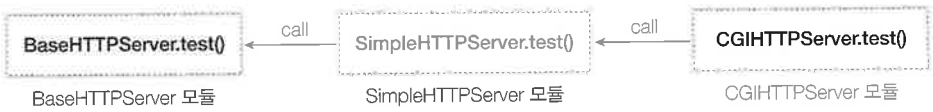


그림 2-11 웹 서버용 모듈의 test( ) 함수 간 호출 관계

CGIHTTPServer 모듈에서 test() 함수를 호출해도, 결국에는 BaseHTTPServer 모듈의 test() 함수를 호출하게 됩니다. BaseHTTPServer 모듈의 test() 함수에서는 HTTPServer 객체를 참조하는데, 그 HTTPServer 객체의 serve_forever() 메소드를 호출해서 해당 웹 서버를 기동하는 방식으로 동작하는 것입니다.

> **NOTE_** xxxHTTPServer 세 가지 모듈의 관계에 대해 좀 더 자세히 알고 싶은 분은 파이썬의 소스코드를 읽어보기 바랍니다. 자세한 기능까지는 아니더라도 모듈 간의 관계 정도는 어렵지 않게 파악할 수 있을 것입니다. 파이썬의 소스는 다음 디렉토리에서 확인할 수 있습니다.
>
> /usr/lib64/python2.7 또는 /usr/lib/python2.7

2.4 CGI/WSGI 라이브러리

파이썬에서는 WSGI[Web Server Gateway Interface] 규격을 정의하고, 파이썬 애플리케이션을 실행하고자 하는 웹 서버는 이 규격을 준수해야 합니다. WSGI는 웹 서버와 웹 애플리케이션을 연결해주는 규격으로, 장고[Django]와 같은 파이썬 웹 프레임워크를 개발하거나, 이런 웹 프레임워크를 아파치[Apache]와 같은 웹 서버와 연동할 때 사용합니다.

파이썬의 WSGI 규격은 1장에서 설명한 전통적인 웹 CGI 기술의 단점을 개선하고 파이썬 언어에 맞게 재구성한 것입니다. 이런 WSGI 규격을 구현하기 위해 파이썬 표준 라이브러리에는 cgi 모듈과 wsgiref 모듈이 같이 존재합니다. 주의할 점은 CGI 기술은 이미 사라진 기술이지만, cgi 모듈의 라이브러리는 WSGI 기술에 여전히 사용되고 있다는 것입니다. cgi 모듈을 먼저 살펴보고 wsgiref 모듈에 대해 설명하겠습니다.

2.4.1 CGI 관련 모듈

1장에서 설명한 CGI 기술을 리뷰한다는 의미로, 이번에는 웹 서버와 애플리케이션이라는 용어로 변경해서 다시 설명해보겠습니다.

사용자의 요청은 웹 서버에 있는 파일을 있는 그대로 요청하는 정적요청Static Request과 현재의 시간을 요청하는 것처럼 동일한 요청이라도 시점에 따라 응답 내용이 달라지는 동적요청Dynamic Request으로 구분됩니다. 동적요청은 웹 서버에서 처리하는 것이 아니라 별도의 애플리케이션에서 처리하는 것이 보통입니다. 그래서 웹 서버에는 동적요청을 애플리케이션에게 넘겨주고 그 결과를 받는 기능이 필요합니다.

이와 같이 웹 서버가 사용자의 요청을 애플리케이션에 전달하고 애플리케이션의 처리 결과를 애플리케이션으로부터 되돌려받기 위한, 즉 웹 서버와 애플리케이션 간에 데이터를 주고받기 위한 규격을 CGICommon Gateway Interface라고 합니다.

파이썬 표준 라이브러리에서는 이러한 CGI 처리를 할 수 있도록 CGIHTTPServer 모듈과 cgi 모듈을 제공하고 있습니다. **2.3.4 CGIHTTPServer 모듈**에서 설명한 것처럼 CGIHTTPServer 모듈은 웹 서버를 작성하는 데 사용하며, cgi 모듈은 요청에 포함된 파라미터를 처리하기 위한 FieldStorage 클래스를 정의하고 있습니다. 또한 CGI 애플리케이션(스크립트)이 실행 중 에러가 발생하는 경우, 에러에 대한 상세 정보를 브라우저에 표시해주는 cgitb 모듈도 있습니다.

파이썬의 경우에는 다음 절에서 설명하고 있는 WSGI 기술을 사용하여 CGI 처리를 합니다. CGI 모듈에 대한 설명은 이 정도로 간략히 마치고, WSGI 관련 모듈에 대해서 설명을 이어가겠습니다.

2.4.2 WSGI 개요

앞 절에서 설명한 CGI 방식은 요청이 들어올 때마다 처리를 위한 프로세스를 생성하는 방식이라서, 짧은 시간에 수천, 수만 건과 같이 다량의 요청을 받는 경우 서버의 부하가 높아져서 프로세스가 멈추거나 다운될 수도 있습니다. 이러한 CGI의 단점을 해결하기 위한 방법으로 Fast CGI, 쓰레드 처리 방식, 외부 데몬 프로세스 방식 등 여러 가지가 사용되고 있습니다. 이때 고려해야 할 사항이 웹 애플리케이션을 작성하는 개발자에게 이러한 방식을 선택하고 연동 방식을 맞추라고 한다면, 웹 애플리케이션 개발이 너무 어려워진다는 것입니다. 그래서 웹 서버와 웹 애플리케이션 중간에서 이런 까다로운 처리를 해주는 것이 장고와 같은 웹 프레임워크입니다.

또한 웹 애플리케이션을 한 번만 작성하면 다양한 웹 서버에서 동작할 수 있도록, 즉 웹 서버에 독립적인 웹 애플리케이션을 작성할 수 있도록 웹 서버와 웹 애플리케이션 간에 연동규격을 정의한 것이 WSGI 규격입니다.

따라서 웹 서버와의 연동 기능을 필수적으로 제공해야 하는 파이썬의 웹 프레임워크는 대부분이 WSGI 규격을 준수하고 있습니다. WSGI 스펙은 2003년에 PEP Python Enhancement Proposals 333으로 만들어졌고, 파이썬 3.x용으로 만들면서 PEP 333을 업그레이드하여 2010년에 PEP 3333 규격이 제정되었습니다.

2.4.3 WSGI 서버의 애플리케이션 처리 과정

WSGI 규격을 준수하는 WSGI 서버에서 실행되는 애플리케이션을 작성하는 것은 편리한 점이 많습니다. 그래서 대부분의 파이썬 웹 프레임워크는 WSGI 서버를 제공하며, 애플리케이션 개발자는 WSGI 서버에 대한 API 규격만 맞추면, 웹 서버와는 독립적으로 애플리케이션을 작성할 수 있어 생산성이 높아집니다.

WSGI 규격에 따라 애플리케이션을 작성하고, 해당 애플리케이션에서 웹 클라이언트의 요청을 처리하는 과정을 다음 그림과 같이 요약해 보았습니다. 웹 서버에서 클라이언트의 요청을 받아 WSGI 서버로 처리를 위임하고, WSGI 서버는 애플리케이션을 실행하여 그 결과를 웹 서버에게 되돌려주면, 웹 서버는 클라이언트에게 응답하는 순서입니다. 그림에 있는 번호를 순서대로 따라가면서 이해하면 되므로 처리 순서에 대한 자세한 설명은 생략하겠습니다. 이번 절에서는 이와 같은 처리 순서를 생각하면서 애플리케이션 개발 측면에서 준수해야 할 WSGI 규격에 대하여 설명하겠습니다.

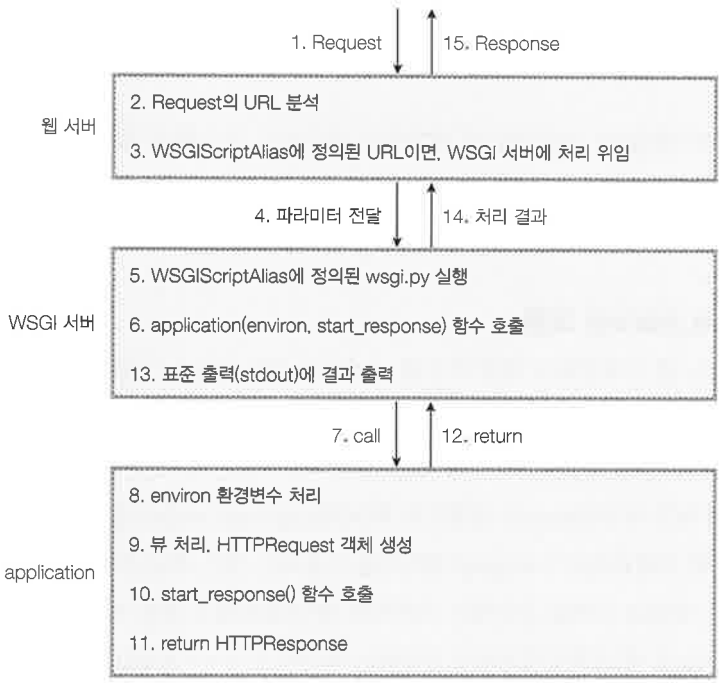


그림 2-12 WSGI 애플리케이션의 처리 순서

WSGI 규격에 따라 애플리케이션을 개발할 때 중요한 사항은 아래 세 가지입니다.

첫 번째, 개발이 필요한 애플리케이션을 함수 또는 클래스의 메소드로 정의하고, 애플리케이션 함수의 인자는 다음과 같이 정의합니다.

```
def application_name(environ, start_response) :
```

- **environ** : 웹 프레임워크에 이미 정의되어 있으며, HTTP_HOST, HTTP_USER_AGENT, SERVER_PROTOCOL과 같은 HTTP 환경변수를 포함합니다.
- **start_response** : 애플리케이션 내에서 응답을 시작하기 위해 반드시 호출해야 하는 함수입니다.

두 번째, start_response 함수의 인자 역시 다음과 같이 정해져 있으므로 그대로 따릅니다.

```
start_response(status, headers)
```

- **status** : 응답 코드 및 응답 메세지를 지정합니다(**200 OK, 404 Not Found** 등).
- **headers** : 응답 헤더를 포함합니다.

세 번째, 애플리케이션 함수의 리턴값은 응답 바디에 해당하는 내용으로, 리스트나 제너레이터와 같은 iterable 타입이어야 합니다.

2.4.4 wsgiref.simple_server 모듈

파이썬 표준 라이브러리에서는 웹 프레임워크 개발자가 웹 서버와의 연동 기능을 개발할 수 있도록 wsgiref 패키지의 하위 모듈로 wsgiref.simple_server 모듈을 제공하고 있습니다. 이 모듈은 WSGI 스펙을 준수하는 웹 서버(일명, WSGI 서버)에 대한 참조reference 서버, 즉 개발자에게 참고가 될 수 있도록 미리 만들어 놓은 WSGIServer 클래스와 WSGIRequestHandler 클래스를 정의하고 있습니다. 다만 모든 웹 프레임워크가 wsgiref 패키지를 사용하는 것은 아닙니다. wsgiref 패키지를 사용하지 않더라도, WSGI 스펙을 준수하는 자신만의 웹 프레임워크 또는 WSGI 서버를 만들면 되기 때문입니다. Flask 웹 프레임워크에서 사용하는 벡자이크Werkzeug WSGI 서버가 한 예로, wsgiref 패키지를 사용하지 않는 WSGI 웹 서버입니다.

wsgiref.simple_server 모듈을 사용해서 WSGI 서버를 만들어 보겠습니다. 다음과 같이 입력합니다.

예제 2-16 WSGI 서버 소스제공

```
$ cd /home/shkim/pyBook/ch2
$ vi mywsgiserver.py

#!/usr/bin/env python

def my_app( environ, start_response) :

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

    return ["This is a sample WSGI Application."]

if __name__ == '__main__' :
```

```
from wsgiref.simple_server import make_server

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

위 예제에서 WSGI 서버를 만들 때 중요한 사항은 다음과 같습니다.

- wsgiref.simple_server 모듈은 WSGI 규격을 준수하여 WSGI 서버를 작성할 수 있도록 API를 제공하고 있으며, make_server() 및 serve_forever() 메소드가 그런 API의 일부입니다.
- my_app()과 같은 애플리케이션 로직을 호출 가능한(callable) 함수나 메소드로 정의하여, 이 함수를 make_server() 인자로 넘겨주어 WSGI 웹 서버를 만듭니다. 이는 애플리케이션 프로그램과 웹 서버 프로그램을 독립적으로 작성할 수 있게 해주는 WSGI 규격의 중요한 원칙에 해당됩니다.
- my_app() 함수가 WSGI 규격을 준수하는 애플리케이션 코드입니다. 이 애플리케이션 함수에서 응답을 위한 헤더 및 바디를 구성해서 반환해줍니다. 헤더 및 바디를 구성하는 방법은 앞 절의 설명을 참고하기 바랍니다.

아마 눈치 빠른 독자는 알아차렸겠지만, WSGI 서버도 웹 서버이므로 WSGI 서버를 만드는 방식도 **2.3.1 간단한 웹 서버**에서 설명한 웹 서버를 만드는 방식을 그대로 따르고 있습니다. 즉, 서버의 IP 및 PORT, 그리고 핸들러 클래스를 정의한 후에 이들을 인자로 하여 HTTPServer 객체를 생성합니다. 그런 다음 HTTPServer 객체의 serve_forever() 메소드를 호출하는 방식입니다.

다만 예제에서 만든 WSGI 서버에서 다른 점은 핸들러 클래스가 아니라 애플리케이션 로직을 작성하는 함수를 사용한다는 점과 애플리케이션 함수를 작성하는 방법입니다. 또한, 다음 그림처럼 HTTPServer 객체를 상속받은 WSGIServer 객체를 생성하여 serve_forever() 메소드를 호출한다는 점도 다릅니다. 물론 핸들러 클래스인 WSGIRequestHandler를 사용하여 WSGI 서버를 만들 수도 있습니다.

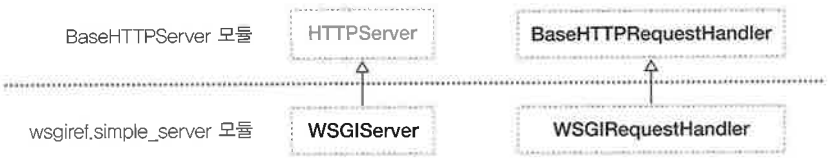


그림 2-13 wsgiref.simple_server 모듈에 정의된 WSGIServer 클래스와 핸들러

2.4.5 WSGI 서버 동작 확인

앞 절에서 만든 WSGI 서버를 실행해서 정상적으로 동작하는지 확인해보겠습니다. WSGI 서버를 실행하기 위해서 다음 명령을 입력합니다.

```
$ python mywsgiserver.py
```

정상적으로 실행이 되면, 아래와 같이 8888 포트로 요청을 기다리고 있다는 메시지가 나타납니다.

```
[shkim@localhost myBook]$
[shkim@localhost myBook]$ python mywsgiserver.py
Started WSGI Server on port 8888...
```

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

위와 같이 정상적으로 WSGI 서버가 실행되었다면, 이제 웹 브라우저를 열고 주소창에 다음과 같이 입력합니다.

```
http://192.168.56.101:8888/
```

다음 화면처럼 my_app() 함수에서 정의한 메시지가 나타나면 정상입니다.

그림 2-15 WSGI 서버에서 my_app() 애플리케이션 실행 결과