<웹 프로그래밍>

장고 공식 홈페이지: <https://docs.djangoproject.com/>

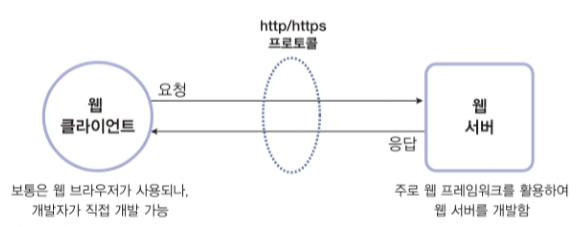
예제 소스 다운로드: [www.hanbit.co.kr/src/10104](http://www.hanbit.co.kr/src/10104)

**1. 웹 프로그래밍의 이해**

웹 프로그램: 기본적으로 클라이언트-서버로 이루어짐

HTTP(S) 프로토콜로 통신하는 클라이언트와 서버를 개발하는 것

웹 클라이언트(웹 브라우저)와 웹 서버(네이버 서버) 중 보통 웹 서버를 개발



다양한 웹 클라이언트

1)웹 브라우저를 사용하여 요청

웹 브라우저의 주소창에 접속하고자 하는 웹 서버의 URL 입력([www.example.com](http://www.example.com))

웹 브라우저는 주소창에 입력된 문장을 해석하여 웹 서버에게 HTTP 요청을 보내는 웹 클라이언트 역할을 수행. 요청을 받은 URL 도메인의 웹 서버는 그 결과를 웹 브라우저로 전송. 웹 브라우저는 전송 받은 결과를 사용자가 볼 수 있도록 HTML 텍스트를 해석하여 화면에 보여줌.

2) 리눅스 curl 명령을 사용하여 요청

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

$ curl <http://www.example.com>

curl 명령: 인자로 넘어온 URL로 HTTP 요청을 보내는 웹 클라이언트의 역할을 수행함.



웹 브라우저에서 보았던 문장과 동일함. (HTML 형식)

즉 어떤 방법을 사용하던 웹 서버는 동일한 요청을 받을 경우 동일한 응답을 주고 있음.

3) Telnet을 사용하여 요청

리눅스의 telnet 프로그램 사용하여 HTTP 요청 보냄.

[shkim@localhost ch1]$ telnet [www.example.com](http://www.example.com) 80

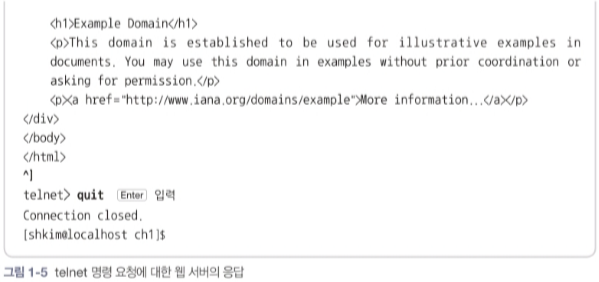
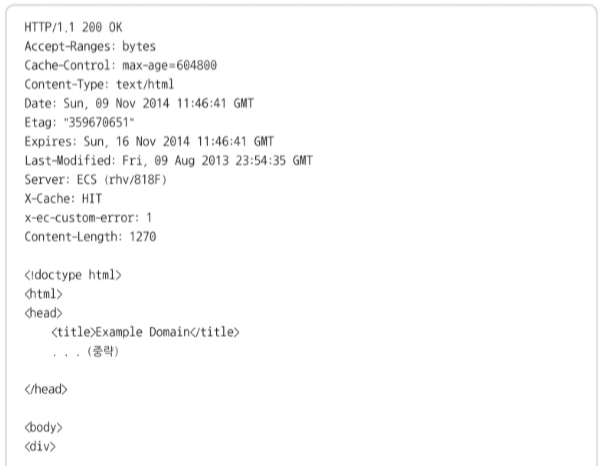
…

GET / HTTP/1.1

Host:www.example.com

telnet 명령은 터미널 창에서 입력하는 내용을 그대로 웹 서버에 전송함.

위 입력 내용은 HTTP 프로토콜의 요청 메시지 규격에 정의된 규칙에 따라 HTTP 요청을 보내는 것으로 telnet 프로그램이 웹 클라이언트의 역할을 수행. 도메인의 웹 서버는 결과 응답.



역시 동일한 출력. (telnet 프로그램 특성상 HTTP 응답 메시지의 헤더가 화면에 출력)

4) 직접 만든 클라이언트로 요청

-notepad 텍스트 에디터

C:\RedBook\ch1>notepad example.py

import urllib.request

print(urllib.request.urlopen(“<http://www.example.com>”).read().decode(‘utf-8’))



HTTP 프로토콜(Hypertext Transfer Protocol)

HTTP는 웹 서버와 웹 클라이언트 사이에서 데이터를 주고받기 위해 사용하는 통신 방식

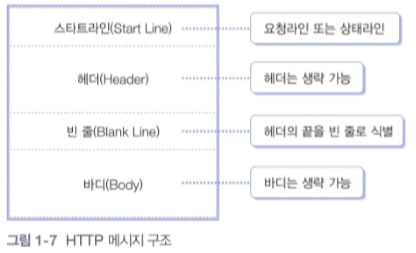
TCP/IP 프로토콜 위에서 동작. 즉, 우리가 웹을 이용하려면 웹 서버와 웹 클라이언트는 각각 TCP/IP 동작에 필수적인 IP 주소를 가져야 함.

HTTP: HTML이나 XML 같은 하이퍼텍스트 뿐 아니라 이미지, 음성, 동영상, 자바스크립트, PDF와 각종 오피스 도큐먼트 파일 등 모든 데이터 전송 가능

우리가 주소창에 주소를 입력하고 엔터를 입력하면 웹 클라이언트와 웹 서버 사이에 HTTP 연결이 맺어지고 서로 요청 메시지와 응답 메시지가 반복적으로 오가면서 웹 사용.

1) HTTP 메시지의 구조

HTTP 메시지: 요청 메시지(클라이언트->서버), 응답 메시지(서버->클라이언트)



스타트라인: 요청라인(요청 메시지일 때), 상태라인(응답 메시지일 때)

헤더: 각 행의 끝에 줄 바꿈 문자인 CRLF가 있음

빈 줄: 헤더와 바디의 구분

헤더와 바디: 생략 가능

바디: 텍스트 뿐 아니라 바이너리 데이터도 들어갈 수 있음

<바디가 없는 요청 메시지의 예>

GET /book/Shakespeare HTTP/1.1

Host: [www.example.com:8080](http://www.example.com:8080)

첫번째 줄: 요청라인

요청 방식, 요청 URL, 프로토콜 버전으로 구성

두번째 줄: 헤더

이름: 값 형식

\*\*Host 항목은 필수로 표시해줘야 함

(요청라인의 URL에 Host를 표시하면 Host 헤더 생략 가능)

GET [http://www.example.com:8080/book/shakespeare HTTP/1.1](http://www.example.com:8080/book/shakespeare%20HTTP/1.1)

<응답 메시지의 예>

HTTP/1.1 200 OK

Content-Type: application/xhtml+xml; charset=utf-8

<html>

…

</html>

첫번째 줄: 상태라인

프로토콜 버전, 상태코드, 상태 텍스트로 구성.

(위 예시에서 200 OK->정상적으로 처리되었음.)

두번째 줄: 헤더

빈 줄

바디: HTML 텍스트 포함

2) HTTP 처리 방식

HTTP 메소드를 통해서 클라이언트가 원하는 처리 방식을 서버에 알려줌.

<HTTP 메소드>

|  |  |  |
| --- | --- | --- |
| 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 취득 |  |
| OPTIONS | 리소스가 서포트하는 메소드 취득 |  |
| TRACE | 루프백 시험에 사용 |  |
| CONNECT | 프록시 동작의 터널 접속으로 변경 |  |

GET 방식: 지정한 URL의 정보를 가져오는 메소드(가장 많이 사용). 웹 브라우저를 이용하여 서버로부터 웹 페이지, 이미지, 동영상 등을 가려오려고 할 때 GET 방식의 요청 사용.

POST 방식: POST의 대표적인 기능은 리소스 생성. 블로그에 글을 등록하는 경우.

PUT 방식: 리소스를 변경하는 데 사용.

예) 블로그에서 글을 업로드한 작성자를 변경하거나 글의 내용을 업데이트

\*리소스를 생성하는 데도 사용 가능.

POST와의 차이점: -POST: 새롭게 생성한 리소스에 대한 URL 결정권이 서버에 있을 때

-PUT: 결정권이 클라이언트에 있을 때

예) 트위터에 글을 포스팅하면 그 글에 대한 URL: 서버에서 결정-POST

위키: 클라이언트가 결정한 타이틀이 그대로 URL이 됨: PUT

BUT 그냥 리소스의 생성은 POST, 리소스의 변경은 PUT

DELETE 방식: 리소스 삭제 메소드

3) GET과 POST 메소드

HTML의 폼에서 지정할 수 있는 메소드가 GET과 POST 밖에 없음

폼에서 사용자가 입력한 데이터들을 서버로 보낼 때

GET: URL 부분의 ? 뒤에 이름=값 쌍으로 이어 붙여 보냄.

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

POST: GET에서 URL에 포함시켰던 파라미터들을 아래 예시처럼 요청 메시지의 바디에 넣음

POST <http://docs.djangoproject.com/search/> HTTP/1.1

Content-Type: application/x-www-form-urlencoded

q=forms&release=1

\*\*URL의 길이 제한이 있기 때문에 GET 방식을 이용하면 많은 양의 데이터를 보내기가 어려움

또한 전달되는 사용자의 데이터가 웹 브라우저의 주소창에 노출되므로 보안 상 단점

(네이버-GET 방식 사용됨을 확인할 수 있음)

4) 상태 코드

서버에서의 처리 결과는 응답 메시지의 상태라인에 있는 상태 코드를 보고 파악 가능함.

상태 코드는 세 자리 숫자로 되어 있음.

첫 번째 숫자는 HTTP 응답의 종류를 구분하는 데 사용, 나머지 두 개의 숫자는 세부적인 응답 내용의 구분을 위한 번호.



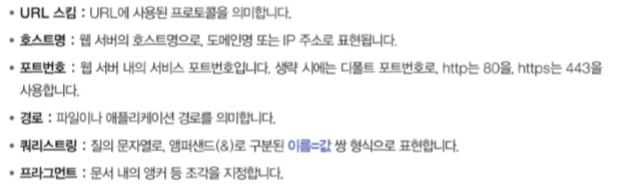
<URL 설계>

디자인: 화면 UI 설계

프로그램 로직: URL 설계

차후에 로직이 변경되더라도 URL 변경은 최소화할 수 있도록 설계!

URL의 구성 



<URL을 바라보는 측면>

URL: 웹 클라이언트에서 호출하므로 애플리케이션에 대한 API라고 할 수 있음.

-RPC/REST로 바라보는 방식

-RPC: 클라이언트가 네트워크상에서 원격에 있는 서버가 제공하는 API 함수를 호출하는 방식

URL 설계와 API 설계를 동일하게 고려하여 URL 경로를 API 함수명으로, 쿼리 파라미터를 함수의 인자로 간주

웹 클라이언트에서 URL을 전송하는 것=웹 서버의 API 함수를 호출한다고 인식

\*URL 경로의 대부분이 동사가 됨

ex) <http://blog.example.com/search?q=test&debug=true>

-REST: 웹 서버에 존재하는 요소들을 모두 리소스라고 정의하고 URL을 통해 웹 서버의 특정

리소스를 표현한다는 개념

리소스는 시간이 지남에 따라 상태가 변할 수 있으므로 서버 간 데이터 교환을 리소스

상태의 교환으로 간주

\*리소스에 대한 조작을 GET, POST, PUT, DELETE 등의 HTTP 메소드로 구분

웹 클라이언트에서 URL을 전송하는 것=웹 서버에 있는 리소스 상태에 대한 데이터를 주고받는 것

ex) <http://blog.example.com/search/test> #GET 메소드 사용

<간편 URL>

쿼리스트링 없이 경로만 가진 간단한 구조의 URL

검색 엔진의 처리를 최적화하기 위해 생겨난 간편한 URL은 URL 입력과 기억이 쉬움



<파이썬의 우아한 URL>

장고에서 사용하는 URL 표현 방식

urlpatterns=[

path(‘articles/2003/’, views.special\_case\_2003),

path(‘articles/<int:year>/’, views.year\_archieve),

path(‘articles/<int:year>/<int:month>/’, views.month\_archieve),

path(‘articles/<int:year>/<int:month>/<slug:slug>/’, views.article\_detail),

]

\*위 표현을 정규표현식을 사용하여 표현할 수도 있음

<웹 애플리케이션 서버>

웹 서버: 웹 클라이언트(웹 브라우저)의 요청을 받아서 처리하는 서버

-웹 서버: 웹 클라이언트의 요청을 받아 요청을 처리하고 그 결과를 웹 클라이언트에게 응답

정적 페이지인 HTML, 이미지, CSS, 자바스크립트 파일을 웹 클라이언트에 제공할 때 웹

서버 사용.

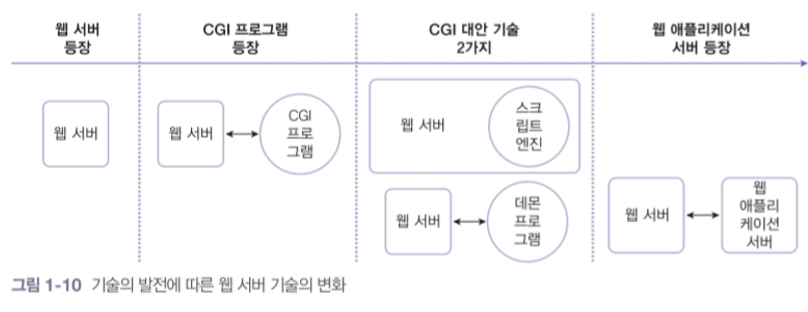
동적 페이지 처리가 필요하다면 웹 애플리케이션 서버에 처리를 넘김.

-프로그램: Apache httpd, Nginx, lighttpd, IIS 등

-웹 애플리케이션 서버: 웹 서버로부터 동적 페이지를 요청 받아서 요청을 처리하고 그 결과를 서버로 반환

동적 페이지 생성을 위한 프로그램 실행과 데이터베이스 연동 기능을 처리

-프로그램: Apache Tomcat, JBoss, WebLogic, WebSphere, Jetty, Jeus, mod\_wsgi, Uwsgi, Gunicorn 등



<정적 페이지 vs 동적 페이지>

정적 페이지: 누가, 언제 요구하더라도 항상 같은 내용을 표시하는 웹 페이지

해당 웹 서비스의 제공자가 사전에 준비하여 서버 측에 배치한 것

동일한 리소스(URL)의 요청-항상 동일한 내용의 페이지 반환

주로 HTML, 자바스크립트, CSS, 이미지만으로 이루어진 페이지 해당

동적 페이지: 동일한 리소스의 요청이라도 누가, 언제, 어떻게 요구했는지에 따라 각각 다른 내용

이 반환되는 페이지

예를 들어 현재 시각을 보여주는 페이지, 온라인 쇼핑 사이트-장바구니 페이지

동적 페이지에는 프로그래밍 코드가 포함되어 있어 페이지 요청 시점에 HTML 문장 을 만들어 냄

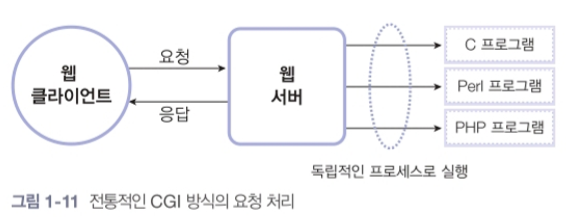
동적 페이지에 대한 요구사항이 생김에 따라 필요한 데이터를 저장하고 꺼내 오는 등의 데이터베이스 처리에 대한 요구가 많아져 웹 서버와는 다른 별도의 프로그램 필요해짐

별도의 프로그램과 웹 서버 사이에 정보를 주고받는 규칙을 정의한 것: CGI 규격

<CGI 방식의 단점>

\*CGI 자체는 정식 프로그래밍 언어나 스크립트가 아님. 정보를 주고받는 규격!

이 규격을 준수하면 어떤 언어를 사용해도 CGI 프로그램 개발 가능



CGI 방식의 근본적인 문제점: 각각의 클라이언트 요청에 대하여 독립적인 별도의 프로세스가 생 성. 요청이 많아질수록 프로세스가 많아지고, 메모리 요구량이 커져

시스템에 많은 부하를 줌.

<CGI 방식의 대안 기술>

1. 별도의 애플리케이션(CGI 프로그램과 같은 역할을 하는 프로그램)을 Perl, PHP 등의 스크립트 언어로 작성, 스크립트를 처리하는 스크립트 엔진을 웹 서버에 내장시켜서 CGI 방식의 단점이었던 별도의 프로세스를 가동시키는 오버헤드를 줄이는 방식.

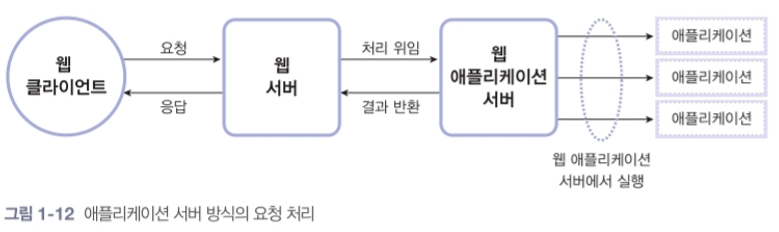
2. 애플리케이션을 처리하는 프로세스를 미리 데몬으로 가동시켜 놓고 웹 서버의 요청을 데몬에서 처리.

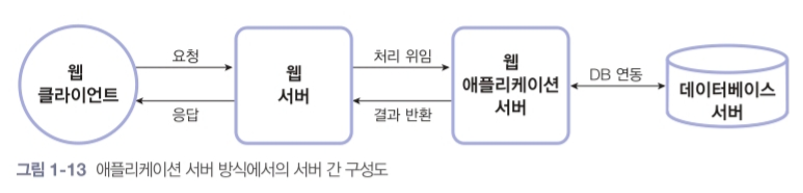
\*파이썬-mod\_wsgi 모듈: 웹 서버 내장 방식, 별도의 데몬 방식으로 실행 가능

현재 가장 많이 사용되고 있는 JSP, ASP 기술에서 애플리케이션 서버 방식을 사용하고 있음

<애플리케이션 서버 방식>

웹 서버가 직접 프로그램을 호출하기보다 웹 애플리케이션 서버를 통해 간접적으로 웹 애플리케이션 프로그램 실행.





웹 서버와 웹 애플리케이션 서버가 분리됨에 따라 서로의 역할도 구분하여 사용하는 것이 좋음

웹 서버는 정적 페이지만 처리하고 웹 애플리케이션 서버는 동적 페이지만 처리하도록.

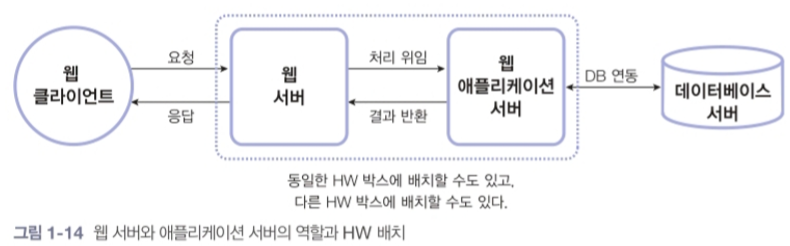
웹 서버: 캐시 기능, 프록시 기능 등의 추가적인 기능 제공

다수의 클라이언트로부터 동시에 요청 받아 처리해야 하므로 동시에 접속을 허가하는 클라이언트 수의 제한 및 처리 프로세스 관리, 요청 및 응답에 관한 로그의 기록, 안정성 확보를 위한 인증 제어 및 암호화 처리 등 다양한 기능

웹 애플리케이션 서버: 웹 서버와의 연동 규격을 잘 따르기만 하면 임의의 언어 플랫폼을 사용해서 애플리케이션 프로그램을 작성하고 실행시킬 수 있기 때문에 기능 더 추가됨

<웹 서버와의 역할 구분>

하드웨어 측면의 용어: 웹 서버 박스, 웹 애플리케이션 서버 박스



HW 박스를 분리하여 구성하면 메모리 효율을 더 높일 수 있음.

웹 서버 박스와 웹 애플리케이션 서버 박스 간의 메모리 사이즈 비율을 조절.(요청 건수 분석)