Chapter 09 The Web, Untangled 뒷부분

- 발표일: 2017.09.15
- 발표자: 이우진

Flask

Bottle은 훌륭한 초기 웹 프레임 워크입니다. 몇가지 cowbell 및 whistles가 필요하면 플라스크를 사용해 보십시오. 그것은 April Fools의 농담으로 2010 년에 시작되었지만 열렬한 반응으로 저자 인 Armin Ronacher는 그것을 진정한 툴로 만들었습니다.

플라스크는 Bottle만큼 사용하기 쉽지만 Facebook 인증 및 데이터베이스 통합과 같은 전문 웹 개발에 유용한 많은 확장 기능을 지원합니다. 플라스크는 파이썬 웹 프레임 워크중 제가 제일 선호합니다. 사용 편의성과 풍부한 기능이 균형을 이루기 때문입니다.

Flask 패키지에는 werkzeug WSGI 라이브러리와 jinja2 템플릿 라이브러리가 포함되어 있습니다. 터미널에서 설치할 수 있습니다.

In [ ]:
```
pip install flask
```

플라스크에서 최종 bottle 예제 코드를 복제 해 봅시다. 먼저, 몇 가지 사항을 변경해야합니다.

• Flask의 기본 디렉토리 홈(static file을 위한)은 `static`이며 파일의 URL도 / static으로 시작합니다. 폴더를 '.'로 변경합니다. (현재 디렉토리) 및 URL 접두어를 ''(비어 있음)로 설정하여 URL /이 index.html 파일에 매핑되도록 합니다.

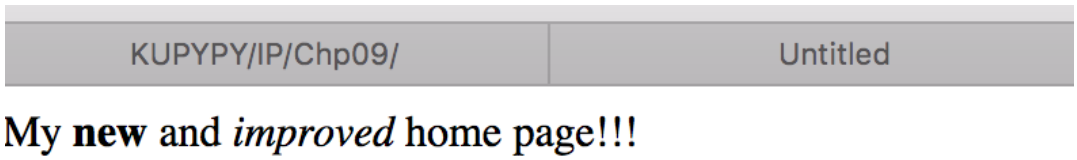
• run() 함수에서 debug = True로 설정하면 automatic reloader도 활성화됩니다. bottle은 디버깅 및 다시로드를 위해 별도의 인수를 사용했었습니다.

이 파일을 flask1.py에 저장하십시오.

In [ ]:
```
from flask import Flask

app = Flask(__name__, static_folder='.', static_url_path='')

@app.route('/')
def home():
    return app.send_static_file('index.html')

@app.route('/echo/<thing>')
def echo(thing):
    return "Say hello to my little friend: %s" % thing

app.run(port=9999, debug=True)
```

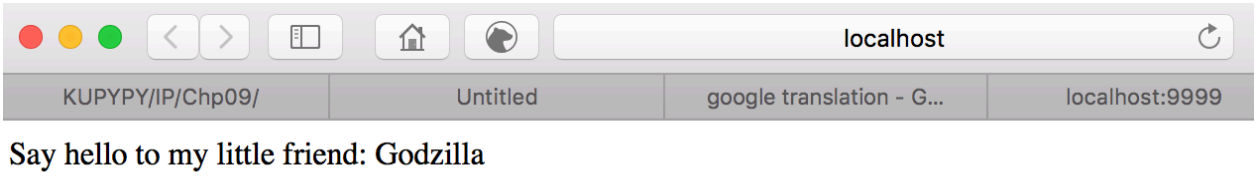
그다음, 커멘드에 다음을 치고 http://localhost:9999/ (http://localhost:9999/) 에 접속하면

In [ ]:
```
python flask1.py
```



다음에 접속하면

http://localhost:9999/echo/Godzilla (http://localhost:9999/echo/Godzilla)



run을 호출 할 때 debug를 True로 설정하는 또 다른 이점이 있습니다. 서버 코드에서 예외가 발생하면 Flask는 잘못된 내용과 위치에 대한 유용한 정보가 있는 특수 형식의 페이지를 반환합니다. 더 좋게는, 서버 프로그램의 변수 값을보기 위해 몇 가지 명령을 입력 할 수 있습니다.

> 프로덕션 웹 서버에서 `debug = True`를 설정하지 마십시오. 잠재적 인 침입자에게 서버에 대한 정보를 너무 많이 노출시킵니다.

지금까지 Flask 예제는 `bottle`로 수행 한 작업을 따라했습니다. 하지만 플라스크에는 더 광범위한 템플릿 시스템인 jinja2가 포함되어 있습니다.

`templates`라는 디렉토리와 *flask2.html*이라는 파일을 만듭니다.

In [ ]:
```
<html>
<head>
<title>Flask2 Example</title>
</head>
<body>
Say hello to my little friend: {{ thing }}
</body>
</html>
```

그런 다음 이 템플릿을 가져오고 thing 전달할 값을 채우고 HTML로 렌더링하는 서버 코드를 작성합니다 (공간을 절약하기 위해 여기에 home() 함수를 삭제할 예정입니다). 이것을 *flask2.py*로 저장하십시오.

In [ ]:
```
from flask import Flask, render_template
app = Flask(__name__)

@app.route('/echo/<thing>')
def echo(thing):
    return render_template('flask2.html', thing=thing)
app.run(port=9999, debug=True)
```

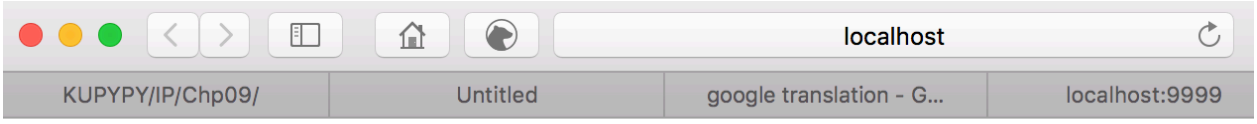
thing = thing argument는 thing 변수를 template에 thing이라는 문자열의 값으로 전달함을 의미합니다.

*flask1.py*를 정지하고, *flask2.py*를 시작합니다.

In [ ]:
```
python flask2.py
```

그다음

http://localhost:9999/echo/Gamera (http://localhost:9999/echo/Gamera) 를 키면 다음과 같이 나옵니다.



Say hello to my little friend: Godzilla

템플릿을 바꾸고, 이를 *flask3.html*로 저장합니다.

In [ ]:
```
<html>
<head>
<title>Flask3 Example</title>
</head>
<body> Say hello to my little friend: {{ thing }}.
Alas, it just destroyed {{ place }}!
</body>
</html>
```

이 두번째 argument를 echo URL 로 전달할수 있습니다.

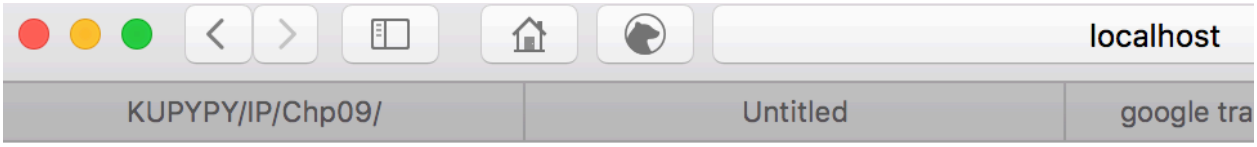
Pass an argument as part of the URL path

이 방법을 사용하면 단순히 URL 자체를 확장 할 수 있습니다 (flask3a.py로 저장).

In [ ]:
```
from flask import Flask, render_template

app = Flask(__name__)

@app.route('/echo/<thing>/<place>')
def echo(thing, place):
    return render_template('flask3.html', thing=thing, place=place)

app.run(port=9999, debug=True)
```

$ python flask3a.py를 키고

http://localhost:9999/echo/Rodan/McKeesport (http://localhost:9999/echo/Rodan/McKeesport) 로 접속하면



Say hello to my little friend: Rodan. Alas, it just destroyed McKeesport!

또는 인수를 GET 매개 변수로 제공 할 수 있습니다 (flask3b.py로 저장).

In [ ]:
```
from flask import Flask, render_template, request

app = Flask(__name__)

@app.route('/echo/')
def echo():
    thing = request.args.get('thing')
    place = request.args.get('place')
    return render_template('flask3.html', thing=thing, place=place)

app.run(port=9999, debug=True)
```

이제 http://localhost:9999/echo?thing=Gorgo&place=Wilmerding (http://localhost:9999/echo?thing=Gorgo&place=Wilmerding) 이를 실행하면

Say hello to my little friend: Gorgo. Alas, it just destroyed Wilmerding!

URL에 GET 명령을 사용하면 모든 인수는 & key1 = val1 & key2 = val2 & ... 형식으로 전달됩니다.

사전 ** 연산자를 사용하여 단일 사전에서 여러 인수를 템플릿에 전달할 수 있습니다 (flask3c.py라고 부름).

In [ ]:
```
from flask import Flask, render_template, request

app = Flask(__name__)

@app.route('/echo/')

def echo():
    kwargs = {}
    kwargs['thing'] = request.args.get('thing')
    kwargs['place'] = request.args.get('place')
    return render_template('flask3.html', **kwargs)

app.run(port=9999, debug=True)
```

** kwargs는 thing=thing, place=place와 같은 역할을 합니다. 입력 인수가 많으면 입력을 절약 할 수 있습니다.

jinja2 템플릿 언어는 이보다 훨씬 더 많은 작업을 수행합니다. PHP로 프로그래밍했다면 많은 유사점을 볼 수 있습니다.

Non-Python Web Servers

지금까지 우리가 사용한 웹 서버는 표준 라이브러리의 http.serv 또는 Bottle and Flask의 디버깅 서버입니다. 프로덕션 환경에서는 더 빠른 웹 서버로 Python을 실행하고 싶을 것입니다. 일반적인 선택은 다음과 같습니다.

• mod_wsgi 모듈을 가진 apache

• nginx와 uWSGI 응용 프로그램 서버

둘 다 잘 작동합니다. 아파치가 아마 가장 인기가 있고, nginx는 안정성과 낮은 메모리 사용에 대한 명성을 가지고 있습니다.

Apache

아파치 웹 서버의 가장 좋은 WSGI 모듈은 `mod_wsgi`입니다. 이것은 Apache 프로세스 내에서 또는 Apache와 통신하는 별도의 프로세스에서 Python 코드를 실행할 수 있습니다.

시스템이 Linux 또는 OS X 인 경우 이미 아파치가 있어야합니다. Windows의 경우 apache를 설치해야합니다.

마지막으로 원하는 WSGI 기반 Python 웹 프레임 워크를 설치하십시오. 여기서 bottle을 시험해 봅시다.

이 테스트 파일을 만들고 /var/www/test/home.wsgi로 저장하십시오.

In [ ]:
```
import bottle

application = bottle.default_app()

@bottle.route('/')

def home():
    return "apache and wsgi, sitting in a tree"
```

이번에는 내장 파이썬 웹 서버를 시작하기 때문에 run ()을 호출하지 마십시오. 우리는 변수를 응용 프로그램에 할당해야합니다. 왜냐하면 그것이 mod_wsgi가 웹 서버와 파이썬 코드를 결합하기 위해 찾는 것이기 때문입니다.

아파치와 mod_wsgi 모듈이 올바르게 작동한다면, 파이썬 스크립트에 연결해야합니다. 이 아파치 서버의 기본 웹 사이트를 정의하는 파일에 한 줄을 추가하고 싶지만 그 파일을 찾는 것은 그 자체의 작업입니다. / etc / apache2 / httpd.conf 또는 / etc / apache2 / sites-available / default 또는 누군가의 애완 동물 도롱뇽의 라틴어 이름 일 수 있습니다.

지금 당장 아파치를 이해하고 그 파일을 발견했다고 가정 해 봅시다. 기본 웹 사이트를 관리하는 섹션에 이 줄을 추가하십시오.

In [ ]:
```
WSGIScriptAlias / /var/www/test/home.wsgi
```

In [ ]:
```
<VirtualHost *:80>
    DocumentRoot /var/www

    WSGIScriptAlias / /var/www/test/home.wsgi

    <Directory /var/www/test>
    Order allow,deny
    Allow from all
    </Directory>
</VirtualHost>
```

아파치를 시작하거나 이 새로운 구성을 사용하도록 실행중인 경우 다시 시작하십시오. 그런 다음 http : // localhost /로 이동하면 다음과 같이 표시됩니다.

In [ ]:
```
apache and wsgi, sitting in a tree
```

이것은 mod_wsgi를 아파치 자체의 일부로 임베디드 모드에서 실행합니다.

데몬 모드로 실행할 수도 있습니다 : 하나 이상의 프로세스로서, 아파치와 분리됩니다. 이렇게하려면 아파치 설정 파일에 두 개의 새로운 커멘드를을 추가하십시오 :

$ WSGIDaemonProcess domain-name user=user-name group=group-name threads=25
WSGIProcessGroup domain-name

앞의 예에서 user-name 및 group-name은 운영 체제 사용자 및 그룹 이름이고 domain-name은 인터넷 도메인 이름입니다. 최소 아파치 설정은 다음과 같습니다 :

In [ ]:
```
<VirtualHost *:80>
    DocumentRoot /var/www
    WSGIScriptAlias / /var/www/test/home.wsgi

    WSGIDaemonProcess mydomain.com user=myuser group=mygroup threads=2
    WSGIProcessGroup mydomain.com

    <Directory /var/www/test>
    Order allow,deny
    Allow from all
    </Directory>
</VirtualHost>
```

The nginx Web Server

nginx 웹 서버에는 파이썬 모듈이 내장되어 있지 않습니다. 대신 uWSGI와 같은 별도의 WSGI 서버를 사용하여 통신합니다. 함께 그들은 Python 웹 개발을위한 매우 빠르고 구성 가능한 플랫폼을 만듭니다.

웹에서 nginx를 설치할 수 있습니다. 또한 uWSGI를 설치해야합니다. uWSGI는 조정할 수있는 많은 레버와 knob가 있는 대형 시스템입니다.

Other Frameworks

웹 사이트와 데이터베이스는 땅콩 버터와 젤리 같습니다. 함께 자주 볼 수 있기 때문입니다. bottle 및 플라스크와 같은 작은 프레임 워크는 데이터베이스에 대한 직접 지원을 포함하지 않지만 일부는 추가 기능을 제공합니다.

데이터베이스 기반 웹 사이트를 만들어야하고 데이터베이스 디자인이 자주 변경되지 않는다면 더 큰 파이썬 웹 프레임 워크 중 하나를 시도해 볼 가치가 있습니다. 현재 주요 경쟁자는 다음과 같습니다.

django

> 특히 대규모 사이트의 경우 가장 많이 사용됩니다. Python 작업 광고에서 django 경험을 자주 요청하는 이유는 여러 가지 이유로 가치가 있습니다. 장고는 ORM 코드(202 페이지에서 설명한 Object-Relational Mapper)를 포함합니다. ORM은 일반적인 데이터베이스 CRUD(create, replace, update, delete)함수에 대해 자동적으로 웹페이지를 생성합니다(194페이지 "SQL"에서 다뤘음). SQLAlchemy 나 직접적인 SQL 쿼리와 같은 또 다른 것을 선호한다면 django의 ORM을 사용할 필요가 없습니다.

web2py

> 이것은 다른 스타일로 장고와 거의 같은 것들을 다룹니다.

pyramid

> 이것은 이전 pylons 프로젝트에서 시작되었으며, django와 유사합니다.

turbogears

> 이 프레임 워크는 ORM, 많은 데이터베이스 및 여러 템플릿 언어를 지원합니다.

wheezy.web

> 이것은 성능을 위해 최적화 된 새로운 프레임 워크입니다. 최근 테스트에서 다른 툴들보다 빠릅니다.

관계형 데이터베이스로 된 웹사이트를 구축하려는 경우 반드시 이러한 대형 프레임 워크 중 하나가 필요하지는 않습니다. 관계형 데이터베이스 모듈에서 bottle, 플라스크 및 기타 항목을 직접 사용할 수도 있고 SQLAlchemy를 사용하여 해결할 수도 있습니다. 그런 다음 특정 ORM 코드 대신 일반 SQL을 작성합니다. 사실 더 많은 개발자가 특정 ORM 문법보다는 SQL을 더 잘 압니다.

또한, 데이터베이스가 관계형이어야만 하지는 않습니다. 데이터 스키마가 현저하게 다른 경우 (행간에 현저한 차이가있는 경우), 204 페이지 "NoSQL 데이터 저장소"에서 설명한 NoSQL 데이터베이스 중 하나와 같은 스키마없는 데이터베이스를 고려해 볼 가치가 있습니다. 저자의 경우 처음에는 NoSQL 데이터베이스에 데이터를 저장하고 관계형 데이터베이스로 전환 한 후 다른 관계형 데이터베이스로, 다른 NoSQL 데이터베이스로 전환 한 다음 관계형 데이터베이스 중 하나로 다시 돌아갔습니다.

Other Python Web Servers

다음은 동시 요청을 처리하기 위해 다중 프로세스 및 / 또는 쓰레드 (262 페이지의 "동시성"참조)를 사용하여 아파치 또는 nginx와 같이 작동하는 독립적 인 Python 기반 WSGI 서버 중 일부입니다.

• uwsgi

• cherrypy

• pylons

다음은 단일 프로세스를 사용하지만 단일 요청에서 블로킹하지 않는 이벤트 기반 서버입니다.

• tornado

• gevent

• gunicorn

concurrency(챕터11)에서 더 상세하게 다룹니다.

Web Services and Automation

우리는 HTML 페이지를 소비하고 생성하는 전통적인 웹 클라이언트 및 서버 응용 프로그램을 살펴 보았습니다. 그러나 HTML보다 다양한 형식으로 응용 프로그램 및 데이터를 붙이기위한 강력한 방법들이 있습니다.

The webbrowser Module

터미널 창에서 파이썬 세션을 시작하고 다음을 입력하십시오 :

```
In [3]: import antigravity
```

이것은 비밀리에 표준 라이브러리의 `webbrowser` 모듈을 호출하고 브라우저에 파이썬 링크를 연결합니다.

이 모듈을 직접 사용할 수 있습니다. 이 프로그램은 브라우저에 메인 파이썬 사이트의 페이지를 로드합니다 :

In [4]:
```
import webbrowser
url = 'http://www.python.org/'
webbrowser.open(url)
webbrowser.open_new(url) #새창 열기
webbrowser.open_new_tab('http://www.python.org/') #새탭 열기
```

Out[4]: True

Web APIs and Representational State Transfer

종종 데이터는 웹 페이지에서만 사용할 수 있습니다. 액세스하려면 웹 브라우저를 통해 페이지에 액세스하여 읽어야 합니다. 마지막으로 방문한 이후 웹 사이트의 저자가 변경 한 경우 데이터의 위치와 스타일이 변경되었을 수 있습니다.

웹 페이지를 퍼블리싱하는 대신 웹 응용 프로그램 프로그래밍 인터페이스 (API)를 통해 데이터를 제공 할 수 있습니다. 클라이언트는 URL에 요청하고 상태 및 데이터가 포함 된 응답을 반환하여 서비스에 액세스합니다. HTML 페이지 대신 데이터가 JSON 또는 XML과 같이 프로그램에서 사용하기 쉬운 형식으로되어 있습니다 (이러한 형식에 대한 자세한 내용은 8 장 참조).

REST (Representational State Transfer)는 Roy Fielding이 박사 학위 논문으로 정의했습니다. 많은 제품은 REST 인터페이스 또는 RESTful 인터페이스를 가지고 있다고 주장합니다. 실제로 이것은 종종 웹 인터페이스 (웹 서비스에 액세스하기 위한 URL의 정의)가 있음을 의미합니다.

RESTful 서비스는 여기에 설명 된 것처럼 특정 방식으로 HTTP 동사를 사용합니다.

- HEAD

리소스에 대한 정보는 가져 오지만 데이터는 가져 오지 않습니다.

- GET

그 이름에서 알 수 있듯이 GET은 리소스 데이터를 서버에서 검색합니다. 이것은 브라우저에서 사용하는 표준 방법입니다. ?표시가 있는 URL을 다음에 여러개의 argument가 있다면 GET 요청입니다. 데이터를 생성, 변경 또는 삭제하는 데 GET을 사용하면 안됩니다.

- POST

이 verb는 서버의 데이터를 업데이트합니다. HTML 양식 및 웹 API에서 자주 사용됩니다.

- PUT

이 동사는 새 리소스를 만듭니다.

- DELETE

삭제

RESTful 클라이언트는 HTTP 요청 헤더를 사용하여 서버에서 하나 이상의 콘텐츠 형식을 요청할 수도 있습니다. 예를 들어, REST 인터페이스를 가진 복잡한 서비스는 입력 및 출력으로 JSON을 선호합니다.

JSON

1 장에서는 인기있는 YouTube 동영상에 대한 정보를 얻기위한 두 개의 Python 코드 샘플을 보여주고 8 장에서는 JSON을 소개합니다. JSON은 웹 클라이언트 - 서버 데이터 교환에 특히 적합합니다. OpenStack과 같은 웹 기반 API에서 특히 많이 사용됩니다.

Crawl and Scrape

경우에 따라 영화 등급, 주가 또는 제품 가용성과 같은 약간의 정보가 필요할 수도 있지만, HTML 페이지에서만 이를 볼수 있고 여러 광고로 둘러싸여 있을수도 있습니다.

다음을 수행하여 수동으로 찾고있는 것을 추출할 수 있습니다.

1. 브라우저에 URL을 입력하십시오.
2. 원격 페이지가로드 될 때까지 기다립니다.
3. 표시된 페이지에서 원하는 정보를 찾습니다.
4. 어딘가에 적어놓고.
5. 관련 URL에 대한 프로세스를 반복 할 수 있습니다.

그러나 이러한 단계 중 일부 또는 전부를 자동화하는 것이 훨씬 더 만족 스럽습니다. 자동화된 웹 가져 오기 프로그램은 크롤러 또는 스파이더 (크롤러 또는 거미라고도 함)를 호출합니다. 내용이 원격 웹 서버에서 검색된 후 스크래퍼가 이를 파싱하게 됩니다.

산업 현장에서 결합 된 크롤러와 스크레이퍼가 필요한 경우, Scrapy는 다운로드 할 가치가 있습니다.

```
pip install scrapy
```

Scrapy는 BeautifulSoup와 같은 모듈이 아닌 프레임 워크입니다.

BeautifulSoup로 HTML 스크랩하기

이미 웹 사이트에서 HTML 데이터를 가져 와서 데이터를 추출하려는 경우 BeautifulSoup를 선택하는 것이 좋습니다. HTML 파싱은 생각보다 어렵습니다. 공개된 웹 페이지의 HTML은 대부분 복잡한 코드들(태그 안 닫힘, nesting 잘못됨 등)을 가지고 있기 때문입니다. 7 장에서 논의한 정규 표현식을 사용하여 자신의 HTML 구문 분석기를 작성하려고하면 곧 이러한 혼란이 발생할 것입니다.

BeautifulSoup를 설치하려면 다음 커멘드를 씁니다.

```
pip install beautifulsoup4
```

이제 웹 페이지에서 모든 링크를 가져 오는 데 사용합시다. HTML에서 a 요소는 링크를 나타내고 href는 링크 대상을 나타내는 속성입니다. 다음 예제에서 우리는 grunt 작업을 수행하는 get_links () 함수와 하나 이상의 URL을 argument로 가져 오는 기본 프로그램을 정의합니다.

In [ ]:
```
def get_links(url):
    import requests from bs4
    import BeautifulSoup as soup
    result = requests.get(url)
    page = result.text
    doc = soup(page)
    links = [element.get('href') for element in doc.find_all('a')]
    return links

if __name__ == '__main__':
    import sys
    for url in sys.argv[1:]:
        print('Links in', url)
        for num, link in enumerate(get_links(url), start=1):
            print(num, link)
        print()
```

이걸 *_links.py_*로 저장시키고 이를 실행하면..

`$ python links.py http://boingboing.net`

```
Links in http://boingboing.net/
1 http://boingboing.net/suggest.html
2 http://boingboing.net/category/feature/
3 http://boingboing.net/category/review/
4 http://boingboing.net/category/podcasts
5 http://boingboing.net/category/video/
6 http://bbs.boingboing.net/
7 javascript:void(0)
8 http://shop.boingboing.net/
9 http://boingboing.net/about
10 http://boingboing.net/contact
```