Ask Company

크롤링 차근차근 시작하기

Requests 라이브러리

여러분의 파이썬/장고 페이스메이커가 되겠습니다.

requests: HTTP for Human

파이썬에서는 기본 라이브러리로 urllib이 제공되지만, 이보다 간결한 코드로 다양한 HTTP요청을 할 수 있는 최고의 라이브러리

JavaScript 처리가 필요한 경우에는 selenium을 고려할 순 있겠습니다. 하지만 이 경우에도 requests 적용이 가능할 수도 있습니다. 크롤링할 페이지에 대해 다각도로 검토가 필요합니다

크롤링 시에 웹요청에서
requests를 쓸 수만 있다면,
가장 효율적인 처리

손쉬운 HTTP 요청 (GET)

단순 GET 요청

```
>>> import requests
>>> response = requests.get('http://news.naver.com/main/home.nhn')

>>> response.status_code  # 응답 상태코드 (int)
200

>>> response.ok           # status_code가 200이상 400미만의 값인지 여부 (bool)
True

>>> response.content      # 응답 데이터 (bytes 타입)
...

>>> response.text         # 응답 문자열 (str타입) → .content 디코딩 결과
...
```

요청 시에 QueryString 인자 지정

IT/과학 탭

https://news.naver.com/main/main.nhn?mode=LSD&mid=shm&sid1=105

```
# 이와 같이 url 문자열에 모두 구성하여도 되나,
url = 'https://news.naver.com/main/main.nhn?mode=LSD&mid=shm&sid1=105'
response = requests.get(url)

# 다음과 같이 params인자로 구분하면 좀 더 편리 (dict/list/tuple 모두 지원)
params = {'mode': 'LSD', 'mid': 'shm', 'sid1': '105'}
# key 중복을 허용하기 위한 방법 → 본디 HTTP에서는 Parameter 이름 중복을 지원
# params = [('mode', 'LSD'), ('mid', 'shm'), ('sid1', '105')]
response = requests.get('http://news.naver.com/main/main.nhn',
                        params=params)
```

응답 헤더

dict 타입이 아니라 requests.structures.CaseInsensitiveDict 타입

Key문자열의 대소문자를 가리지 않습니다.

각 헤더의 값은 헤더이름을 Key로 접근하여 획득

```
>>> response.headers
>>> response.headers['Content-Type']  # Key문자열 대소문자에 상관없이 접근
'text/html; charset=UTF-8'
>>> response.headers['content-type']
'text/html; charset=UTF-8'
>>> response.encoding
'UTF-8'
```

response.encoding값은 Content-Type헤더의 charset값으로 획득.
Content-Type헤더에 charset값이 없을 경우 iso-8859-1로 처리될 수 있습니다.
→ 대개 "에이~ 한글 깨졌네~"라고 생각하는 포인트

응답 body

```
bytes_data = response.content  # 응답 Raw 데이터 (bytes)
str_data = response.text       # response.encoding으로 디코딩하여 유니코드 변환
```

```
# 이미지 데이터일 경우에는 .content만 사용
with open('flower.jpg', 'wb') as f:
    image_data = response.content
    f.write(image_data)
```

```
# 문자열 데이터일 경우에는 .text를 사용하면 직접 디코딩하지 않아도 되니 편리
html_string = response.text
html_string = response.content.decode('utf8')  # 혹은 .content 필드를 직접 디코딩
```

```
# json 포맷의 문자열 응답일 경우,
# .json()을 활용하면 직접 Deserialize를 하지 않아도 되니 편리
python_obj = response.json()
python_obj = json.loads(response.text)         # 혹은 직접 수행. "import json" 필요
```

한글이 깨진 것처럼 보여질 경우

```
>>> url = 'http://www.puka.co.kr/'
>>> response = requests.get(url)
>>> response.text[90:125]
':::  Ç Ñ ± ¹ À ¯ Ä ¡ ¿ ø Ã Ñ ¿ ¬ Ç Õ È ¸  º Î » ê Á ö È ¸  :::'   # 흔히 한글이 깨졌다라고 표현하는 상황

    # 인코딩을 확인해봅시다.
>>> response.headers['Content-Type']    # Content-Type 헤더에 charset이 명시되어있어야 합니다.
'text/html'
>>> response.encoding      # charset 정보가 없어서, ISO-8859-1 인코딩으로 오해한 상황. :(
'ISO-8859-1'
```

requests 라이브러리가
이렇게 구현되어있어요.

한글이 깨졌다고 할 것이 아니라,
인코딩을 다른 인코딩으로 오해했다.

```
class Response(object):
    # 중략 ...
    @property
    def text(self):
        content = None
        encoding = self.encoding

        if not self.content:
            return str('')

        if self.encoding is None:
            encoding = self.apparent_encoding  # 자동 예측한 인코딩.

        try:
            content = str(self.content, encoding, errors='replace')
        except (LookupError, TypeError):
            content = str(self.content, errors='replace')

        return content
```

Content-Type 헤더

`text/html` (BAD)

`text/html; charset=EUC-KR` (GOOD)

응답 헤더에 인코딩 정보가 없어서,
자동예측을 할 수 밖에 없는 상황.
그런데 엉뚱한 인코딩으로 예측이 된다면? ☹

https://github.com/kennethreitz/requests/blob/v2.21.0/requests/models.py#L836

해결방법은?

근본적인 방법은 서버 측에서
응답헤더에 encoding정보를 명확히 넣어주는 것

```
# encoding을 커스텀 지정하고, .text 속성에 접근
>>> response.encoding = 'euc-kr'
>>> response.text[90:125]
':::: 한국유치원총연합회 부산지회 :::</title>\r\n<me'

# 혹은 .content를 직접 디코딩
>>> text = response.content.decode('euc-kr')
>>> text[90:125]
':::: 한국유치원총연합회 부산지회 :::</title>\r\n<me'
```

멜론 차트 웹페이지를 받아봅시다.

```
>>> import requests
>>> from bs4 import BeautifulSoup

>>> res = requests.get('http://www.melon.com/chart/index.htm')
>>> html = res.text
>>> html
''
```

응?

```
>>> res.status_code
406
```

Not Acceptable 상태 코드

```
>>> res.ok
False
```

요청 시에 커스텀 헤더 지정

```
request_headers = {
    'User-Agent': ('Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_6) '
                   'AppleWebKit/537.36 (KHTML, like Gecko) '
                   'Chrome/58.0.3029.110 Safari/537.36'),
    'Referer': 'http://www.melon.com',  # 멜론 홈
}

url = 'http://www.melon.com/chart/index.htm'
res = requests.get(url, headers=request_headers)
```

requests 라이브러리에서의 기본 User-Agent값은 'python-requests/버전'입니다. 서버에 따라 User-Agent값으로 요청 거부여부를 결정하기도 합니다.

손쉬운 HTTP 요청 (POST)

주로 데이터 추가/수정/삭제 목적으로 사용합니다. → 데이터의 변화를 요구하는 요청

POST 요청의 활용

ex) 게시판에 글 쓰기/수정/삭제

웹서비스 성격에 따라, 그에 맞춰 방법을 적용해야 합니다.

필요하다면 파일 업로드

웹 API로의 요청

```
# GET 요청 예시
response = requests.get(url, params=params, headers=headers)

# POST 주요 인자
response = requests.post(
    url,
    params=params,     # QueryString 인자가 필요할 때
    headers=headers,   # 커스텀 헤더 지정이 필요할 때
    data=data,         # body에 데이터를 실어보내야할 때
    files=files)       # body에 추가로 파일 업로드가 필요할 때
```

data 인자

POST 요청에서 사용

일반적인 Form 요청

dict/tuple/list 타입으로 data 지정

```
data = {'k1': 'v1', 'k2': 'v2'}
# data = (('k1', 'v1'), ('k1', 'v3'), ('k2', 'v2'))
response = requests.post('http://httpbin.org/post', data=data)
```

JSON API

JSON 문자열로 직렬화하여 요청이 필요할 때

```
# 방법1) json포맷 문자열로 변환한 후, data인자로 지정
json_string = json.dumps(data, ensure_ascii=False) # import json 필요
response = requests.post('http://httpbin.org/post', data=json_string)

# 방법2) 객체를 json인자로 지정하면, 내부적으로 json.dumps 처리
response = requests.post('http://httpbin.org/post', json=data)
```

파일 업로드 요청

```
# multipart/form-data 인코딩
files = {
    'photo1': open('f1.jpg', 'rb'),  # 데이터만 전송
    'photo2': open('f2.jpg', 'rb'),
    'photo3': ('f3.jpg', open('f3.jpg', 'rb'), 'image/jpeg', {'Expires': '0'}),
}
data = {'k1': 'v1'}
response = requests.post('http://httpbin.org/post', files=files, data=data)
```

인생은 짧습니다.
파이썬/장고를 쓰세요.

여러분의 파이썬/장고 페이스메이커가 되겠습니다.

- Ask Company