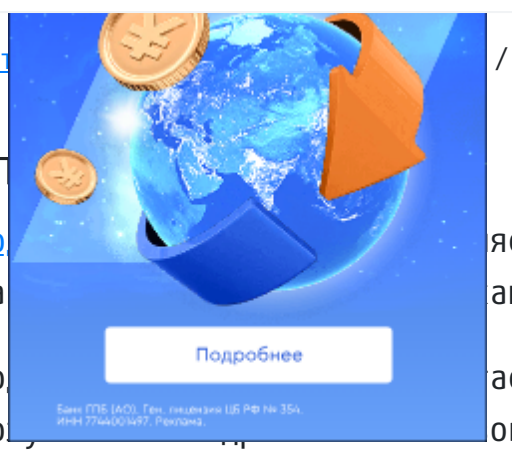
Модуль urllib.request в Python



От GET запросов. Получение данных из URL-адресов

Мо яет функции и классы, которые помогают открывать URL-адреса (в основном HTTP), имеет ба ацию, перенаправления, файлы cookie и многое другое.

Мо ает очень простой интерфейс в виде функции urllib.request.urlopen(), что позволяет по ованием множества различных протоколов. Также этот модуль предлагает немного более сложный интерфейс для обработки обычных ситуаций, таких как базовая аутентификация, файлы cookie, прокси и т. д. Они предоставляются объектами, называемыми обработчиками и "*открывателями*".

Модуль поддерживает получение для многих схем URL-адресов (идентифицируемых строкой перед : в URL-адресе - например, ftp: - это схема URL-адресов ftp://python.org/) с использованием связанной с ними сети.

В простых ситуациях функция urllib.request.urlopen() очень проста в использовании. При появлении ошибок или нетривиальных случаев при открытии URL-адресов, необходимое некоторое понимание протокола передачи гипертекста. Цель этого вводного материала - проиллюстрировать использование модуля urllib.request с достаточно подробным описанием HTTP.

Так же, рассмотрите возможность использования в своем коде, написанного для людей, стороннего модуля HTTP запросов requests (требует дополнительной установки).

Содержание:

Получение данных с сайта по URL-адресу.

Самый простой способ использовать модуль urllib.request следующий:

```
import urllib.request
with urllib.request.urlopen('http://python.org/') as resp:
   html = resp.read()
```

Если необходимо получить ресурс по URL-адресу и сохранить его во временном объекте или файле, то это можно сделать при помощи функций shutil.copyfileobj() и tempfile.NamedTemporaryFile():

```
import shutil
import tempfile
import urllib.request

with urllib.request.urlopen('http://python.org/') as response:
    with tempfile.NamedTemporaryFile(delete=False) as tmp_file:
        shutil.copyfileobj(response, tmp_file)

wi (tmp_file.name) as html:
    pass
```

Многие варианты использования urllib.request будут такими простыми (обратите внимание, что вместо URL-адреса http: можно использовать URL-адрес, начинающийся с ftp:, file: и т. д.

HTTP основан на запросах и ответах - клиент делает запросы, а серверы отправляют ответы. Модуль urllib.request отр это с помощью объекта Request, который представляет собой HTTP-запрос, который делается. В простейшей форме - создается объект Request с указанием URL-адреса, который надо получить получить. Вызов функции urllib.request.urlopen() с объектом Request возвращает объект ответа для запрошенного URL. Этот ответ является файловым объектом, для чтения которого можно вызвать, например, метод .read():

```
import urllib.request

req = urllib.request.Request('http://www.voidspace.org.uk')
with urllib.request.urlopen(req) as response:
   the_page = response.read()
```

Обратите внимание, что модуль urllib.request использует один и тот же интерфейс запроса для обработки всех схем URL. Например, можно сделать такой FTP-запрос:

```
req = urllib.request.Request('ftp://example.com/')
```

В случае с HTTP есть две дополнительные вещи, которые позволяют создавать объекты Request:

1. можно передавать данные для отправки на сервер.
2. можно передать на сервер дополнительную информацию - *метаданные* о самом запросе - эта информация отправляется в виде *заголовков* HTTP.

Создание и отправка POST запросов модулем urllib.request.

Иногда необходимо отправить данные по URL-адресу (часто URL-адрес будет относиться к сценарию CGI или другому веб-приложению). В HTTP это делается с помощью так называемого запроса POST запроса. Это то, что делает браузер, когда отправляется HTML-форма. Не все POST запросы поступают из заполненных форм, которые заполняются пользователями на сайтах. POST запросы могут использоваться для передачи произвольных данных в собственное приложение. В общем случае HTML-форм данные должны быть закодированы стандартным способом, а затем переданы объекту Request в качестве аргумента данных. Кодирование выполняется с помощью функции из модуля urllib.parse.

```
import urllib.parse
import urllib.request

# URL - адрес, получающий данные
url = 'http://www.someserver.com/cgi-bin/register.cgi'
# Отправляемые данные
values = {'name' : 'Michael Foord',
          'location' : 'Northampton',
          'language' : 'Python' }

# Кодирование данных
data = urllib.parse.urlencode(values)
# данные должны быть байтами
data = data.encode('utf-8')
req = urllib.request.Request(url, data)
with urllib.request.urlopen(req) as response:
   the_page = response.read()
```

Обратите внимание, что иногда, для кодирования данных требуется другая кодировка.

Создание и отправка GET запросов модулем urllib.request.

Если объект Request не получает аргумент с данными, то модуль urllib.request использует GET запрос. Одним из отличий запросов GET и POST является то, что запросы POST часто имеют *побочные эффекты*, они каким-то образом меняют состояние системы.

Произвольные данные также можно передать в GET-запросе, закодировав их в самом URL. Это делается следующим образом:


```
>>> import urllib.request
>>> import urllib.parse
```

```
>>> data = {}
>>> data['name'] = 'Somebody Here'
>>> data['location'] = 'Northampton'
>>> data['language'] = 'Python'
>>> url_values = urllib.parse.urlencode(data)
# Порядок может отличаться от приведенного ниже.
>>> print(url_values)
# name=Somebody+Here&language=Python&location=Northampton
>>> url = 'http://www.example.com/example.cgi'
>>> full_url = '?'.join([url, url_values])
>>> data = urllib.request.urlopen(full_url)
```


Обратите внимание, что полный URL-адрес создается путем добавления '?' к URL-адресу, за которым следуют закодированные значения.

Отправка заголовков HEADERS при помощи urllib.request.

Некоторым веб-сайтам не нравится, когда их просматривают различные роботы или сканеры, к тому же, по умолчанию модуль urllib.request идентифицирует себя как Python-urllib/x.y (где x и y - это старший и дополнительный номера версий версии Python, например Python-urllib/3.9). Браузер идентифицирует себя через заголовок User-Agent. При создании объекта Request, ему можно передать словарь заголовков. В следующем примере выполняется тот же запрос, что и выше, но идентифицируется как версия Internet Explorer .

```
import urllib.parse
import urllib.request

url = 'http://www.someserver.com/cgi-bin/register.cgi'
user_agent = 'Mozilla/5.0 (Windows NT 6.1; Win64; x64)'
values = {'name': 'Michael Foord',
          'location': 'Northampton',
          'language': 'Python' }
headers = {'User-Agent': user_agent}

data = urllib.parse.urlencode(values)
data = data.encode('ascii')
# создание объекта Request с указанием, в качестве
# аргументов, параметров запроса и заголовков
req = urllib.request.Request(url, data, headers)
# отправка POST запроса
with urllib.request.urlopen(req) as response:
   the_page = response.read()
```

Обработка исключений при использовании urllib.request.

Функция urllib.request.urlopen() вызывает URLError, когда не может обработать ответ (хотя, как обычно с API Python, также могут возникать встроенные исключения, такие как ValueError, TypeError и т. д.).

HTTPError - это подкласс URLError, возникающий в конкретном случае при запросе URL-адресов по HTTP протоколу.

Классы исключений экспортируются из модуля urllib.error.

Обработка ошибок URLError.

Часто URLError возникает из-за отсутствия сетевого подключения (нет маршрута к указанному серверу) или из-за того, что указанный сервер не существует. В этом случае возникшее исключение будет иметь атрибут, который представляет собой кортеж, содержащий код ошибки и текстовое сообщение об ошибке.

```
>>> req = urllib.request.Request('http://www.pretend_server.org')
>>> try: urllib.request.urlopen(req)
... except urllib.error.URLError as e:
...     print(e.reason)
...
# (4, 'getaddrinfo failed')
```

Вверх

Обработка ошибок HTTPError.

Каждый ответ HTTP от сервера содержит числовой *код ответа сервера*. Иногда код этот состояния указывает на то, что сервер не может выполнить запрос. Обработчики по умолчанию будут обрабатывать некоторые из этих ответов за вас (на), если ответ представляет собой *перенаправление*, которое запрашивает у клиента выборку документа с другого URL-адреса, то модуль urllib.request сделает это за вас). Для тех URL-адресов, с которыми модуль не может справиться, он вызывает ошибки HTTPError. Типичные ошибки включают 404 (страница не найдена), 403 (запрос запрещен) и 401 (требуется аутентификация).

Возникший экземпляр HTTPError будет иметь целочисленный атрибут code, который соответствует ошибке, отправленной сервером. Поскольку обработчики по умолчанию обрабатывают перенаправления (коды в диапазоне 300), а коды в диапазоне 100–299 указывают на успех, то необходимо обрабатывать только коды ошибок в диапазоне 400–599.

Когда возникает ошибка, сервер отвечает, возвращая код ошибки HTTP и страницу с ошибкой. Можете использовать экземпляр HTTPError в качестве ответа на возвращенной странице. Это означает, что помимо атрибута code он также имеет методы .read(), .geturl() и .info(), возвращаемые модулем urllib.response:

```
>>> req = urllib.request.Request('http://www.python.org/fish.html')
>>> try:
...     urllib.request.urlopen(req)
... except urllib.error.HTTPError as e:
...     print(e.code)
...     print(e.read())
...
# 404
# b'<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
#   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">\n\n\n<html
#   ...
#   <title>Page Not Found</title>\n
#   ...
```

Итак, если необходимо обрабатывать ошибки HTTPError или URLError, есть два основных подхода. Мы предпочитаем второй подход.

Первый способ:

```
from urllib.request import Request, urlopen
from urllib.error import URLError, HTTPError
req = Request(someurl)
try:
    response = urlopen(req)
except HTTPError as e:
    print('The server couldn\'t fulfill the request.')
    print('Error code: ', e.code)
except URLError as e:
    print('We failed to reach a server.')
    print('Reason: ', e.reason)
else:
    # Все в порядке
```

Примечание. Исключение HTTPError должно быть первым, иначе исключение URLError также перехватит HTTPError.

Второй способ:

```
from urllib.request import Request, urlopen
from urllib.error import URLError
req = Request(someurl)
try:
    response = urlopen(req)
except URLError as e:
    if hasattr(e, 'reason'):
        print('We failed to reach a server.')
        print('Reason: ', e.reason)
    elif hasattr(e, 'code'):
        print('The server couldn\'t fulfill the request.')
        print('Error code: ', e.code)
```

```
else:
    # все в порядке
```

Содержание раздела: