Министерство образования и науки Российской Федерации

Федеральное государственное автономное образовательное учреждение

высшего профессионального образования «Уральский федеральный

университет имени первого президента России Б. Н. Ельцина»

Институт фундаментального образования

Кафедра интеллектуальных информационных технологий

**WebOb**

Пояснительная записка

Руководитель:

Студент:

Д. М. Свинцов

К. Е. Шимановская

Екатеринбург 2016

Оглавление

[Введение 2](#_Toc473237652)

[Описание 3](#_Toc473237653)

[Практическое применение 7](#_Toc473237654)

[Заключение 9](#_Toc473237655)

[Список источников 10](#_Toc473237656)

# Введение

В наш век, когда каждый человек хочет облегчить себе жизнь. В языках программирования, специально для современных людей, предусмотрены различные библиотеки, фреймворки, и тд.

В этом разделе мы изучим WebOb.

Что это?

WebOb – это библиотека, преобразующая *HTTP* запрос (текст) в объект и позволяющая генерировать *HTTP* ответы. В частности работает с окружением *WSGI*.

Проще говоря, WebOb — это обертка для WSGI-окружения.

Преимущества WebOb

* Терпимость большинства HTTP-спецификаций к присоединённым структурам данных.
* Проверенное временем окружение, которое скрывает все известные WSGI причуды.
* Нулевые известные проблемы (сообщившиеся ошибки всегда фиксируется, как можно скорее).
* 100% тестового покрытия.
* Никаких внешних зависимостей.
* Поддержка Python 3.

# Описание

Итак, немного теории с примерами о том, что вообще есть в этой библиотеке.

В библиотеке WebOb есть множество классов, которые помогают решить большинство задач.

Вот некоторые из них:

**Объект Request**

Класс [Request](http://docs.webob.org/en/stable/api/request.html#webob.request.Request) оборачивает окружение, пришедшее от Веб-сервера, в случае HTTP-запроса. Мы можем сами создать окружение для класса [Request](http://docs.webob.org/en/stable/api/request.html#webob.request.Request) и получить объект запроса, как в примере ниже.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | environ = {  'HTTP\_HOST': 'localhost:80',  'PATH\_INFO': '/article',  'QUERY\_STRING': 'id=1',  'REQUEST\_METHOD': 'GET',  'SCRIPT\_NAME': ''  }  **from** **webob** **import** Request  req = Request(environ)  **from** **pprint** **import** pprint  pprint(req.environ) |
| 1  2  3  4  5 | Ответ:  {'HTTP\_HOST': 'localhost:80',  'PATH\_INFO': '/article',  'QUERY\_STRING': 'id=1',  'REQUEST\_METHOD': 'GET',  'SCRIPT\_NAME': ''} |

Можно немного облегчить себе задачу, используя основные методы объекта Request:

|  |  |
| --- | --- |
| **Свойство** | **Цель** |
| host | Имя хоста, используемого для этого запроса. |
| domain(n=2) | Первые n сегментов имени хоста, начиная справа (домен верхнего уровня). |
| format | Тип содержимого, запрошенного с клиента. |
| method | Метод HTTP, использованного для запроса. |
| get?, post?, patch?, put?, delete?, head? | Возвращает true, если метод HTTP – это GET/POST/PATCH/PUT/DELETE/HEAD. |
| headers | Возвращает хэш, содержащий заголовки, связанные с запросом. |
| port | Номер порта (число), использованного для запроса. |
| protocol | Возвращает строку, содержащую использованный протокол плюс “://”, например “http://”. |
| query\_string | Часть URL со строкой запроса, т.е. все после “?”. |
| remote\_ip | Адрес IP клиента. |
| url | Полный URL, использованный для запроса. |

Пример работы методов Request:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | **from** **webob** **import** Request  req = Request.blank('/blog?page=4')  **print**(req.method)  **print**(req.scheme)  **print**(req.path\_info)  **print**(req.host)  **print**(req.host\_url)  **print**(req.application\_url)  **print**(req.path\_url)  **print**(req.url)  **print**(req.path)  **print**(req.path\_qs)  **print**(req.query\_string) |
| 1  2  3  4  5  6  7  8  9  10  11 | Ответ:  GET  http  /blog  localhost:80  http://localhost  http://localhost  http://localhost/blog  http://localhost/blog?page=4  /blog  /blog?page=4  page=4 |

Изучая класс Request, нельзя обойти стороной Mock запрос. Как он используется? [Request](http://docs.webob.org/en/stable/api/request.html#webob.request.Request) имеет конструктор, который создает минимальное окружение запроса. При помощи метода blank можно имитировать HTTP запрос:

|  |  |
| --- | --- |
| 1  2  3  4  5 | **from** **webob** **import** Request  req = Request.blank('/blog?page=4')  **from** **pprint** **import** pprint  pprint(req.environ) |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | Ответ:  {'HTTP\_HOST': 'localhost:80',  'PATH\_INFO': '/blog',  'QUERY\_STRING': 'page=4',  'REQUEST\_METHOD': 'GET',  'SCRIPT\_NAME': '',  'SERVER\_NAME': 'localhost',  'SERVER\_PORT': '80',  'SERVER\_PROTOCOL': 'HTTP/1.0',  'wsgi.errors': <open file '<stderr>', mode 'w' at 0x7f4ff5d111e0>,  'wsgi.input': <\_io.BytesIO object at 0x7f4ff3b622f0>,  'wsgi.multiprocess': False,  'wsgi.multithread': False,  'wsgi.run\_once': False,  'wsgi.url\_scheme': 'http',  'wsgi.version': (1, 0)} |

Немного подробнее рассмотрим метод Get.

Когда вы делаете request.GET['pref'], вы получите обратно только последнее значение pref. Иногда возвращаются строки и другие времена, возвращающие список причин частых исключений. Если вы хотите получить все значения, используйте request.GET.getall('pref'). Если вы хотите быть уверены, что есть одно и только одно значение, используйте request.GET.getone('pref'), которое будет вызывать исключение, если есть ноль или более чем одно значение pref.

При использовании таких операций , как request.GET.items(), вы получите обратно все пары ключ/значение.

Пример использования ниже:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | **from** **webob** **import** Request  req = Request.blank('/test?check=a&check=b&name=Bob')  **print**(req.GET)  **print**(req.GET['check'])  **print**(req.GET.getall('check'))  **print**(list(req.GET.items())) |
| 1  2  3  4 | Ответ:  GET([('check', 'a'), ('check', 'b'), ('name', 'Bob')])  b  ['a', 'b']  [('check', 'a'), ('check', 'b'), ('name', 'Bob')] |

Запуск WSGI-приложений с помощью Request

[webob.request.Request](http://docs.webob.org/en/stable/api/request.html#webob.request.Request) умеет запускать *WSGI*-приложения. Это может понадобиться, например, при написании тестов.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21 | **from** **webob** **import** Request  **def** wsgi\_app(environ, start\_response):  request = Request(environ)  **if** request.path == '/test':  start\_response('200 OK', [('Content-type', 'text/plain')])  **return** ['Hi!']  start\_response('404 Not Found', [('Content-type', 'text/plain')])  req = Request.blank('/test')  status, headers, app\_iter = req.call\_application(wsgi\_app)  **print**(status)  **print**(headers)  **print**(app\_iter)  **print**  req = Request.blank('/bar')  status, headers, app\_iter = req.call\_application(wsgi\_app)  **print**(status)  **print**(headers)  **print**(app\_iter) |
| 1  2  3  4  5  6  7 | Ответ:  200 OK  [('Content-type', 'text/plain')]  ['Hi!']  404 Not Found  [('Content-type', 'text/plain')]  None |

**Объект response**

Класс response обычно не используется напрямую, но он создается в течение исполнения экшна и рендеринга данных, которые посылаются обратно пользователю, но иногда бывает полезно иметь доступ к отклику напрямую. Некоторые из этих методов доступа имеют настройки, позволяющие изменять их значения.

Вот основные методы класса Respons:

|  |  |
| --- | --- |
| **Свойство** | **Назначение** |
| body | Это строка данных, которая будет возвращена клиенту. Часто это HTML. |
| status | Код статуса HTTP для отклика, например 200 для успешного отклика или 404 для ненайденного файла. |
| location | URL, по которому клиент будет перенаправлен, если указан. |
| content\_type | Тип содержимого отклика. |
| charset | Кодировка, используемая для отклика. По умолчанию это “utf-8”. |
| headers | Заголовки, используемые для отклика. |

По идее, это класс который содержит все необходимое для создания ответа *WSGI*-приложения.

Конструктор класса [Response](http://docs.webob.org/en/stable/api/response.html#webob.response.Response) имеет минимальный набор для HTTP ответа:

>>> from webob import Response

>>> res = Response()

>>> res.status

'200 OK'

>>> res.headerlist

[('Content-Type', 'text/html; charset=UTF-8'), ('Content-Length', '0')]

>>> res.body

''

# Практическое применение

Ниже представлены несколько примеров того, как WebOb помогает уменьшить количество строчек в коде.

BlogRead

Пример класса BlogRead без webob

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | **class** **BlogRead**(BaseArticle):  **def** \_\_iter\_\_(self):  **if** **not** self.article:  self.start('404 Not Found', [('content-type', 'text/plain')])  **yield** b'not found'  **return**  self.start('200 OK', [('Content-Type', 'text/html')])  **yield** str.encode(  env.get\_template('read.html').render(article=self.article)  ) |

Класс BlogRead с использованием библиотеки WebOb

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | **class** **BlogRead**(BaseArticle):  **def** response(self):  **from** **webob** **import** Response  **if** **not** self.article:  **return** Response(status=404)  **return** Response(env.get\_template('read.html')  .render(article=self.article)) |

BlogUpdate

Пример класса BlogUpdate без webob

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | **class** **BlogUpdate**(BaseArticle):  **def** \_\_iter\_\_(self):  **if** self.environ['REQUEST\_METHOD'].upper() == 'POST':  **from** **urllib.parse** **import** parse\_qs  values = parse\_qs(self.environ['wsgi.input'].read())  self.article['title'] = values[b'title'].pop().decode()  self.article['content'] = values[b'content'].pop().decode()  self.start('302 Found',  [('Content-Type', 'text/html'),  ('Location', '/')])  **return**  self.start('200 OK', [('Content-Type', 'text/html')])  **yield** str.encode(  env.get\_template('create.html').render(article=self.article)  ) |

Класс BlogUpdate с использованием WebOb

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | **class** **BlogUpdate**(BaseArticle):  **def** response(self):  **from** **webob** **import** Response  **if** self.request.method == 'POST':  self.article['title'] = self.request.POST['title']  self.article['content'] = self.request.POST['content']  **return** Response(status=302, location='/')  **return** Response(env.get\_template('create.html')  .render(article=self.article)) |

BlogDelete

Пример класса BlogDelete без webob

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | **class** **BlogDelete**(BaseArticle):  **def** \_\_iter\_\_(self):  self.start('302 Found', *# '301 Moved Permanently',*  [('Content-Type', 'text/html'),  ('Location', '/')])  ARTICLES.pop(self.index)  **yield** b'' |

Класс BlogDelete с использованием WebOb

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | **class** **BlogDelete**(BaseArticle):  **def** response(self):  **from** **webob** **import** Response  ARTICLES.pop(self.index)  **return** Response(status=302, location='/') |

# Заключение

Кто-то может спросить, а зачем вообще нужен WebOb?

Отвечу так – эта библиотека позволяет создавать богатые приложения и действительное промежуточное программное обеспечение, не зная всех сложностей WSGI и HTTP.

# Список источников

<http://lectureswww.readthedocs.io/6.www.sync/2.codding/6.webob.html#>

<http://docs.webob.org/en/stable/#webob>

<http://www.webob.org/>

<https://ru.wikipedia.org/wiki/Python_Paste#cite_note-3>

<http://softwaremaniacs.org/blog/2008/04/21/wsgi-framework/>