

RSS TAG 邮件列表

Linux.Cn

开源中文社区

招募「技术大牛导师」

用QQ帐号登录

用新浪微博登录

帐号 用户名/Email 自动登录 找回密码

密码 登录 骑士注册

技术 ◆ 学习 新闻 ◆ 快讯 观点 ◆ 热议 软件 ◆ 分享 论坛 投稿

Locez 新手指南： 下载 Linux » 安装 Linux » 安装软件 » 基础命令 »

请注册后再搜索 搜索

Qiniu

七牛云，实践40万用户，针对七大行业推出一站式数据服务

技术 ◆ 学习 查看内容

自己动手开发一个 Web 服务器（二）

2016-1-2 10:00 收藏: 2

来源：编程派 参考原文：http://ruslanspivak.com/lsbaws-part2/ 编译文章：http://codingpy.com/article/build-a-simple-web-server-part-two/ 文章地址：https://linux.cn/article-6816-1.html

作者：Ruslan

在《自己动手开发一个 Web 服务器（一）》中，我给大家留了一个问题：如何在不对服务器代码作任何修改的情况下，通过该服务器运行Django应用、Flask应用和Pyramid应用，同时满足这些不同网络框架的要求？读完这篇文章，你就可以回答这个问题了。

以前，你选择的Python网络框架将会限制所能使用的 Web 服务器，反之亦然。如果框架和服务器在设计时就是可以相互匹配的，那你就不会面临这个问题：

野心时代  
AGE OF AMBITION

互联网招聘风暴周

100offer  
互联网人才拍卖

告别盲目投简历  
让海量好机会主动来找你

了解更多

相关阅读

服务器 web

淘宝Web服务器，Tengine-1.2.3 正式发布 2012-3-1

4月全球Web服务器份额：Apache居首 Nginx 2012-4-16

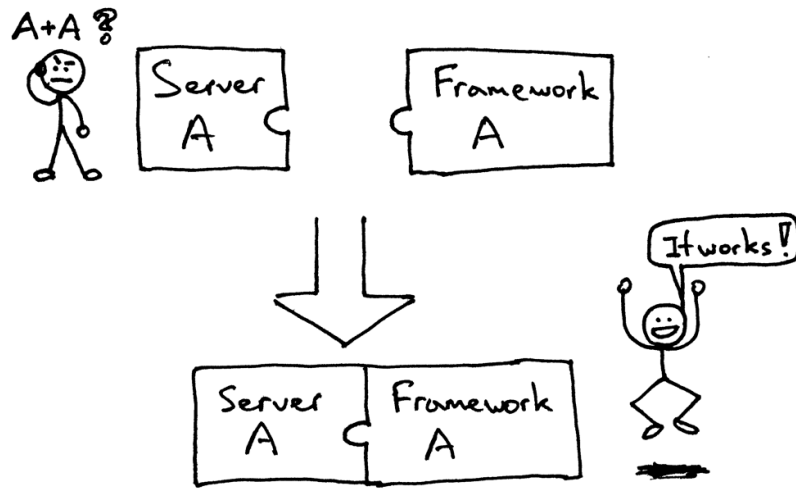
5月全球Web服务器市场份额：Nginx升至 2012-5-18

淘宝web服务器Tengine-1.3.0 版本发布 2012-5-29

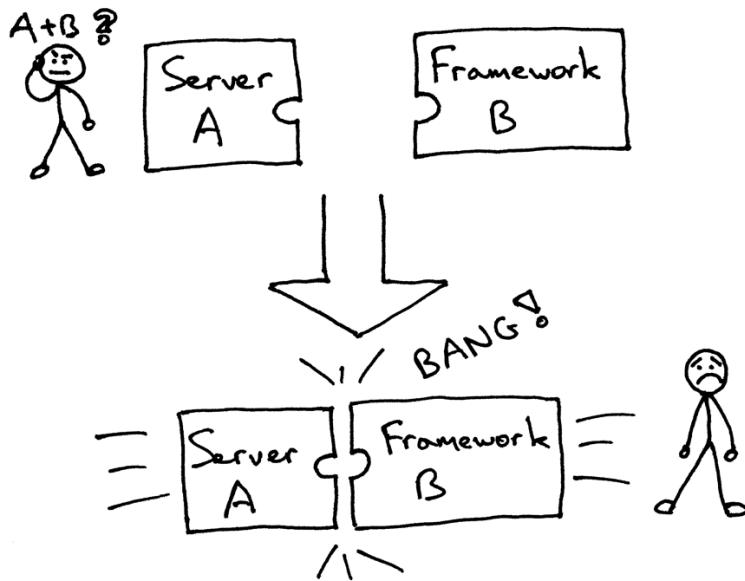
6月全球Web服务器市场份额：Apache升至 2012-6-14

IETF正式开始开发HTTP 2.0标准 2012-10-12

1 of 1201/03/2016 04:09 PM



但是如果你试图将设计不匹配的服务器与框架相结合，那么你一定就会碰到下面这张图所展示的这个问题：



这意味着，你基本上只能使用能够正常运行的服务器与框架组合，而不能选择你希望使用的服务器或框架。

那么，你怎样确保可以在不修改 Web 服务器代码或网络框架代码的前提下，使用自己选择的服务器，并且匹配多个不同的网络框架呢？为了解决这个问题，就出现了Python Web Server Gateway Interface, WSGI Web 服务器网关接口。



**WSGI** <<https://www.python.org/dev/peps/pep-0333/>> 的出现，让开发者可以将网络框架与 Web 服务器的选择分隔开来，不再相互限制。现在，你可以真正地将不同的 Web 服务器与网络开发框架进行混合搭配，选择满足自己需求的组合。例如，你可以使用Gunicorn或Nginx/uWSGI或Waitress服务器来运行Django、Flask或Pyramid应用。正是由于服务器和框架均支持WSGI，才真正得以实现二者之间的自由混合搭配。

所以，WSGI就是我在上一篇文章中所留问题的答案。你的 Web 服务器必须实现一个服务器端的WSGI接口，而目前所有现代Python网络框架都已经实现了框架端的WSGI接口，这样开发者不需要修改服务器的代码，就可以支持某个网络框架。

Web 服务器和网络框架支持WSGI协议，不仅让应用开发者选择符合自己需求的组合，同时也有利于服务器和框架的开发者，因为他们可以将注意力集中在自己擅长的领域，而不是相互倾轧。其他编程语言也拥有类似的接口：例如Java的Servlet API和Ruby的Rack。

口说无凭，我猜你肯定在想：“无代码无真相！”既然如此，我就在这里给出一个非常简单的WSGI服务器实现：

```
# Tested with Python 2.7.9, Linux & Mac OS X
import socket
import StringIO
import sys

class WSGIServer(object):

    address_family = socket.AF_INET
    socket_type = socket.SOCK_STREAM
    request_queue_size = 1

    def __init__(self, server_address):
        # Create a listening socket
        self.listen_socket = listen_socket = socket.socket(
            self.address_family,
            self.socket_type
        )
        # Allow to reuse the same address
        listen_socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
        # Bind
        listen_socket.bind(server_address)
        # Activate
        listen_socket.listen(self.request_queue_size)
        # Get server host name and port
        host, port = self.listen_socket.getsockname()[1:2]
        self.server_name = socket.getfqdn(host)
        self.server_port = port
        # Return headers set by Web framework/Web application
        self.headers_set = []

    def set_app(self, application):
        self.application = application
```

```
def serve_forever(self):
    listen_socket = self.listen_socket
    while True:
        # New client connection
        self.client_connection, client_address = listen_socket.accept()
        # Handle one request and close the client connection. Then
        # loop over to wait for another client connection
        self.handle_one_request()

def handle_one_request(self):
    self.request_data = request_data = self.client_connection.recv(1024)
    # Print formatted request data a la 'curl -v'
    print(''.join(
        '< {line}\n'.format(line=line)
        for line in request_data.splitlines()
    ))

    self.parse_request(request_data)

    # Construct environment dictionary using request data
    env = self.get_environ()

    # It's time to call our application callable and get
    # back a result that will become HTTP response body
    result = self.application(env, self.start_response)

    # Construct a response and send it back to the client
    self.finish_response(result)

def parse_request(self, text):
    request_line = text.splitlines()[0]
    request_line = request_line.rstrip('\r\n')
    # Break down the request line into components
    (self.request_method, # GET
     self.path,           # /hello
     self.request_version # HTTP/1.1
    ) = request_line.split()

def get_environ(self):
    env = {}
    # The following code snippet does not follow PEP8 conventions
    # but it's formatted the way it is for demonstration purposes
    # to emphasize the required variables and their values
    #
    # Required WSGI variables
    env['wsgi.version']      = (1, 0)
    env['wsgi.url_scheme']   = 'http'
    env['wsgi.input']        = StringIO.StringIO(self.request_data)
    env['wsgi.errors']       = sys.stderr
    env['wsgi.multithread']  = False
    env['wsgi.multiprocess'] = False
    env['wsgi.run_once']     = False
    # Required CGI variables
    env['REQUEST_METHOD']    = self.request_method # GET
    env['PATH_INFO']         = self.path           # /hello
    env['SERVER_NAME']       = self.server_name    # localhost
    env['SERVER_PORT']       = str(self.server_port) # 8888
    return env

def start_response(self, status, response_headers, exc_info=None):
    # Add necessary server headers
```

```
server_headers = [
    ('Date', 'Tue, 31 Mar 2015 12:54:48 GMT'),
    ('Server', 'WSGIServer 0.2'),
]
self.headers_set = [status, response_headers + server_headers]
# To adhere to WSGI specification the start_response must return
# a 'write' callable. We simplicity's sake we'll ignore that detail
# for now.
# return self.finish_response

def finish_response(self, result):
    try:
        status, response_headers = self.headers_set
        response = 'HTTP/1.1 {status}\r\n'.format(status=status)
        for header in response_headers:
            response += '{0}: {1}\r\n'.format(*header)
        response += '\r\n'
        for data in result:
            response += data
        # Print formatted response data a la 'curl -v'
        print(''.join(
            '> {line}\n'.format(line=line)
            for line in response.splitlines()
        ))
        self.client_connection.sendall(response)
    finally:
        self.client_connection.close()

SERVER_ADDRESS = (HOST, PORT) = '', 8888

def make_server(server_address, application):
    server = WSGIServer(server_address)
    server.set_app(application)
    return server

if __name__ == '__main__':
    if len(sys.argv) < 2:
        sys.exit('Provide a WSGI application object as module:callable')
    app_path = sys.argv[1]
    module, application = app_path.split(':')
    module = __import__(module)
    application = getattr(module, application)
    httpd = make_server(SERVER_ADDRESS, application)
    print('WSGIServer: Serving HTTP on port {port} ...\n'.format(port=PORT))
    httpd.serve_forever()
```

上面的代码比第一部分的服务器实现代码要长的多,但是这些代码实际也不算太长,只有不到150行,大家理解起来并不会太困难。上面这个服务器的功能也更多——它可以运行你使用自己喜欢的框架所写出来的网络应用,无论你选择Pyramid、Flask、Django或是其他支持WSGI协议的框架。

你不信?你可以自己测试一下,看看结果如何。将上述代码保存为 `webserver2.py`, 或者直接从我的[Github仓库](https://github.com/rspivak/lbaws/blob/master/part2/webserver2.py) [<https://github.com/rspivak/lbaws/blob/master/part2/webserver2.py>](https://github.com/rspivak/lbaws/blob/master/part2/webserver2.py) 下载。如果你运行该文件时没有提供任何参数,那么程序就会报错并退出。

```
$ python webserver2.py
Provide a WSGI application object as module:callable
```

上述程序设计的目的,就是运行你开发的网络应用,但是你还满足一些它的要求。要运行服务器,你只需要安装Python即可。但是要运行使用Pyramid、Flask和Django等框架开发的网络应用,你还需要先安装这些框架。我们接下来

安装这三种框架。我倾向于使用 `virtualenv` 安装。请按照下面的提示创建并激活一个虚拟环境，然后安装这三个网络框架。

```
$ [sudo] pip install virtualenv
$ mkdir ~/envs
$ virtualenv ~/envs/lsbaws/
$ cd ~/envs/lsbaws/
$ ls
bin include lib
$ source bin/activate
(lsbaws) $ pip install pyramid
(lsbaws) $ pip install flask
(lsbaws) $ pip install django
```

接下来，你需要创建一个网络应用。我们首先创建Pyramid应用。将下面的代码保存为 `pyramidapp.py` 文件，放至 `webserver2.py` 所在的文件夹中，或者直接从我的Github仓库 <<https://github.com/rspivak/lsbaws/blob/master/part2/pyramidapp.py>> 下载该文件：

```
from pyramid.config import Configurator
from pyramid.response import Response

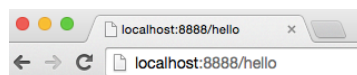
def hello_world(request):
    return Response(
        'Hello world from Pyramid!\n',
        content_type='text/plain',
    )

config = Configurator()
config.add_route('hello', '/hello')
config.add_view(hello_world, route_name='hello')
app = config.make_wsgi_app()
```

现在，你可以通过自己开发的 Web 服务器来启动上面的Pyramid应用。

```
(lsbaws) $ python webserver2.py pyramidapp:app
WSGIServer: Serving HTTP on port 8888 ...
```

在运行 `webserver2.py` 时，你告诉自己的服务器去加载 `pyramidapp` 模块中的 `app` 可调用对象 (callable)。你的服务器现在可以接收HTTP请求，并将请求中转至你的Pyramid应用。应用目前只能处理一个路由 (route)：/hello。在浏览器的地址栏输入 `http://localhost:8888/hello`，按下回车键，观察会出现什么情况：



```
localhost:8888/hello
Hello world from Pyramid!
```

你还可以在命令行使用 `curl` 命令，来测试服务器运行情况：

```
$ curl -v http://localhost:8888/hello
...
```

接下来我们创建Flask应用。重复上面的步骤。

```
from flask import Flask
from flask import Response
flask_app = Flask('flaskapp')

@flask_app.route('/hello')
def hello_world():
```

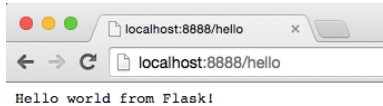
```
    return Response(
        'Hello world from Flask!\n',
        mimetype='text/plain'
    )

app = flask_app.wsgi_app
```

将上面的代码保存为 `flaskapp.py`，或者直接从我[的Github仓库](https://github.com/rspivak/lbaws/blob/master/part2/flaskapp.py) <<https://github.com/rspivak/lbaws/blob/master/part2/flaskapp.py>> 下载文件，并运行：

```
(lsbaws) $ python webserver2.py flaskapp:app
WSGIServer: Serving HTTP on port 8888 ...
```

然后在浏览器地址栏输入 `http://localhost:8888/hello`，并按下回车：



同样，在命令行使用 `curl` 命令，看看服务器是否会返回Flask应用生成的信息：

```
$ curl -v http://localhost:8888/hello
...
```

这个服务器是不是也能支持Django应用？试一试就知道了！不过接下来的操作更为复杂一些，我建议大家克隆整个仓库，并使用其中的 `djangoapp.py` 文件。下面的代码将一个名叫 `helloworld` 的Django应用添加至当前的Python路径中，然后导入了该项目的WSGI应用。

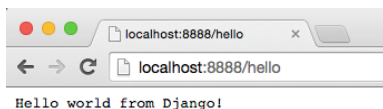
```
import sys
sys.path.insert(0, './helloworld')
from helloworld import wsgi

app = wsgi.application
```

将上面的代码保存为 `djangoapp.py`，并使用你开发的服务器运行这个Django应用。

```
(lsbaws) $ python webserver2.py djangoapp:app
WSGIServer: Serving HTTP on port 8888 ...
```

同样，在浏览器中输入 `http://localhost:8888/hello`，并按下回车键：



接下来，和前面几次一样，你通过命令行使用 `curl` 命令进行测试，确认了这个Django应用成功处理了你发出的请求：

```
$ curl -v http://localhost:8888/hello
...
```

你有没有按照上面的步骤测试？你做到了让服务器支持全部三种框架吗？如果没有，请尽量自己动手操作。阅读代码很重要，但这系列文章的目的在于重新开发，而这意味着你需要自己亲自动手。最好是你自己重新输入所有的代码，并确保代码运行结果符合预期。

经过上面的介绍，你应该已经认识到了WSGI的强大之处：它可以让你自由混合搭配 Web 服务器和框架。WSGI为Python Web 服务器与Python网络框架之间的交互提供了一个极简的接口，而且非常容易在服务器端和框架端实现。下面的代码段分别展示了服务器端和框架端的WSGI接口：



```
def run_application(application):
    """Server code."""
    # This is where an application/framework stores
    # an HTTP status and HTTP response headers for the server
    # to transmit to the client
    headers_set = []
    # Environment dictionary with WSGI/CGI variables
    environ = {}

    def start_response(status, response_headers, exc_info=None):
        headers_set[:] = [status, response_headers]

    # Server invokes the 'application' callable and gets back the
    # response body
    result = application(environ, start_response)
    # Server builds an HTTP response and transmits it to the client
    ...

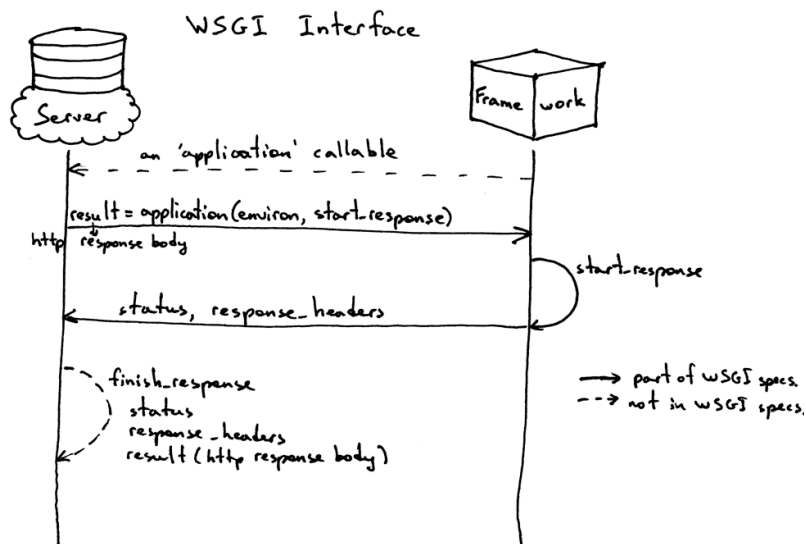
def app(environ, start_response):
    """A barebones WSGI app."""
    start_response('200 OK', [('Content-Type', 'text/plain')])
    return ['Hello world!']

run_application(app)
```

下面给大家解释一下上述代码的工作原理：

1. 网络框架提供一个命名为 `application` 的可调用对象（WSGI协议并没有指定如何实现这个对象）。
2. 服务器每次从HTTP客户端接收请求之后，调用 `application`。它会向可调用对象传递一个名叫 `environ` 的字典作为参数，其中包含了WSGI/CGI的诸多变量，以及一个名为 `start_response` 的可调用对象。
3. 框架/应用生成HTTP状态码以及HTTP 响应报头，然后将二者传递至 `start_response`，等待服务器保存。此外，框架/应用还将返回响应的正文。
4. 服务器将状态码、响应报头和响应正文组合成HTTP响应，并返回给客户端（这一步并不属于WSGI协议）。

下面这张图直观地说明了WSGI接口的情况：



有一点要提醒大家，当你使用上述框架开发网络应用的时候，你处理的是更高层级的逻辑，并不会直接处理WSGI协议相关的要求，但是我很清楚，既然你正在看这篇文章，你一定对框架端的WSGI接口很感兴趣。所以，我们接下来在不使用Pyramid、Flask或Django框架的前提下，自己开发一个极简的WSGI网络应用/网络框架，并使用WSGI服务器运行该应用：



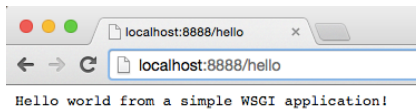
```
def app(environ, start_response):
    """A barebones WSGI application.

    This is a starting point for your own Web framework :)
    """
    status = '200 OK'
    response_headers = [('Content-Type', 'text/plain')]
    start_response(status, response_headers)
    return ['Hello world from a simple WSGI application!\n']
```

将上述代码保存为 `wsgiapp.py` 文件，或者直接从我的 [Github仓库 <https://github.com/rspivak/lbaws/blob/master/part2/wsgiapp.py>](https://github.com/rspivak/lbaws/blob/master/part2/wsgiapp.py) 下载，然后利用 Web 服务器运行该应用：

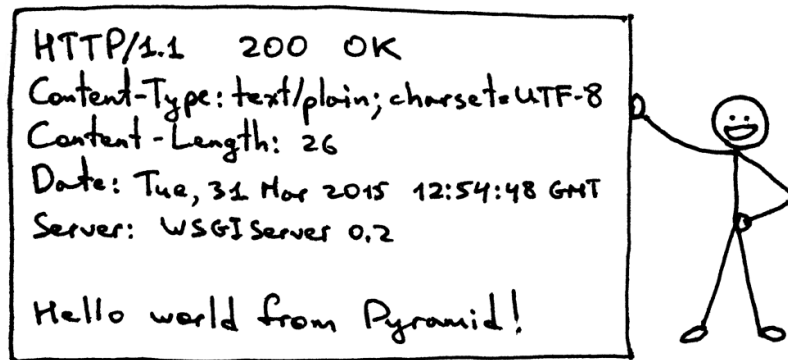
```
(lsbaws) $ python webserver2.py wsgiapp:app
WSGIServer: Serving HTTP on port 8888 ...
```

在浏览器中输入下图中的地址，然后按回车键。结果应该是这样的：



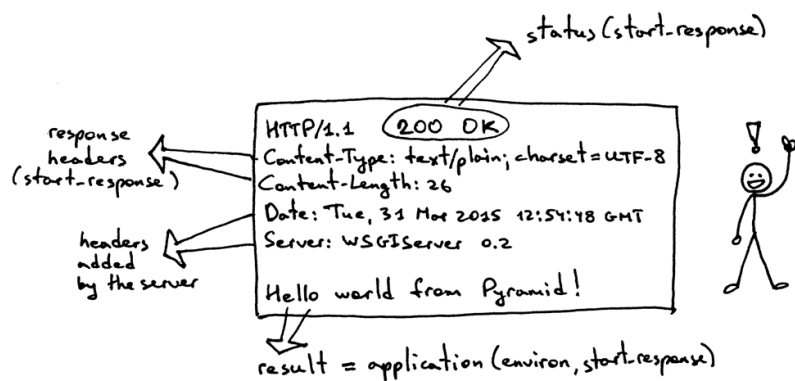
你刚刚自己编写了一个极简的WSGI网络框架！太不可思议了。

接下来，我们重新分析服务器返回给客户端的对象。下面这张图展示的是你通过HTTP客户端调用Pyramid应用后，服务器生成的HTTP响应：



上图中的响应与你在第一篇中看到的有些类似，但是也有明显不同之处。举个例子，其中就出现了你之前没有看到过的4个HTTP报头：Content-Type，Content-Length，Date和Server。这些事 Web 服务器返回的响应对象通常都会包含的报头。不过，这四个都不是必须的。报头的目的是传递有关HTTP请求/响应的额外信息。

既然你已经对WSGI接口有了更深的理解，下面这张图对响应对象的内容进行了更详细的解释，说明了每条内容是如何产生的。



到目前为止,我还没有介绍过 `environ` 字典的具体内容,但简单来说,它是一个必须包含着WSGI协议所指定的某些WSGI和CGI变量。服务器从HTTP请求中获取字典所需的值。下面这张图展示的是字典的详细内容:

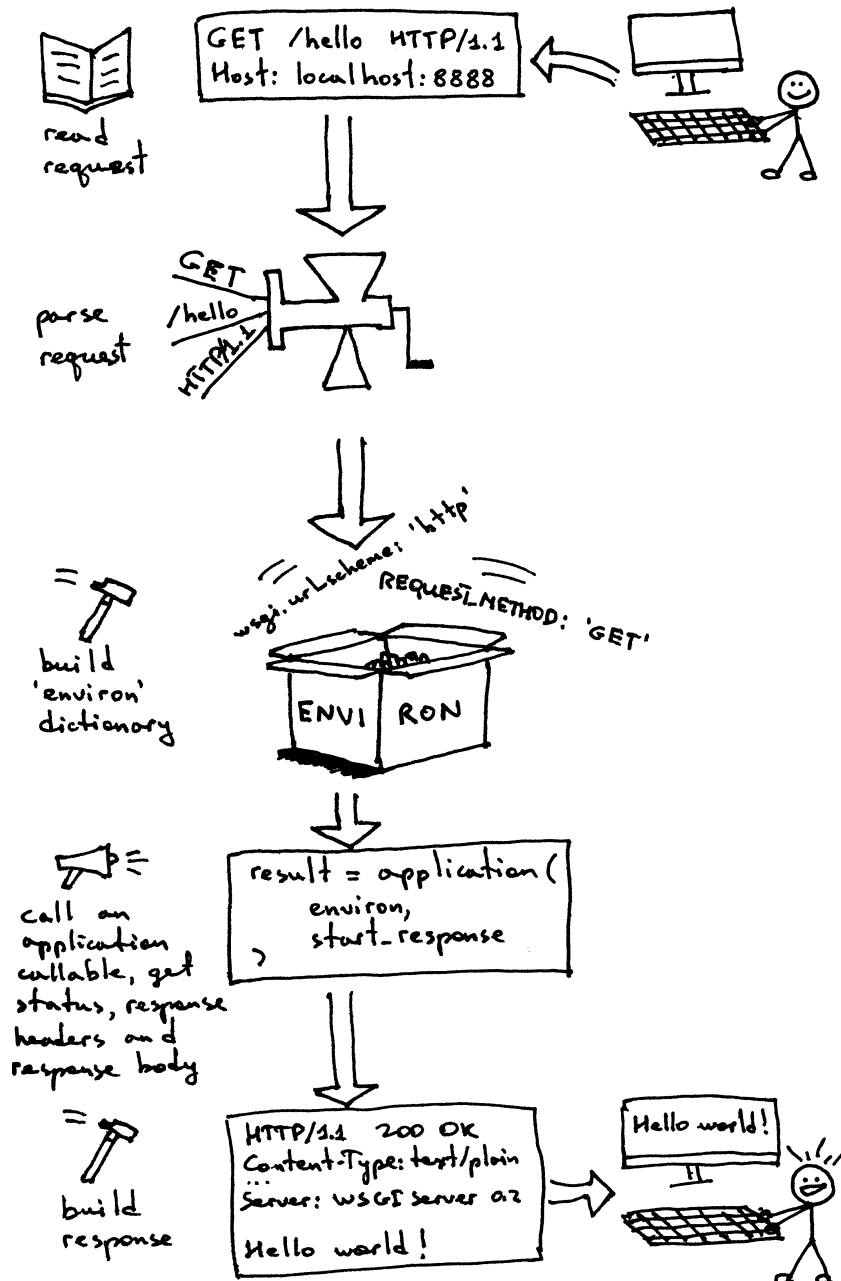
## ENVIRON

<code>wsgi.version</code>	<code>(1, 0)</code>
<code>wsgi.url_scheme</code>	<code>'http'</code>
<code>wsgi.input</code>	<code>StringIO(request_data)</code>
<code>wsgi.errors</code>	<code>sys.stderr</code>
<code>wsgi.multithread</code>	<code>False</code>
<code>wsgi.multiprocess</code>	<code>False</code>
<code>wsgi.run_once</code>	<code>False</code>
<code>REQUEST_METHOD</code>	<code>'GET'</code>
<code>PATH_INFO</code>	<code>'/hello'</code>
<code>SERVER_NAME</code>	<code>'localhost'</code>
<code>SERVER_PORT</code>	<code>8888</code>

网络框架通过该字典提供的信息,根据指定的路由和请求方法等参数来决定使用哪个 `视图`,从哪里读取请求正文,以及如何输出错误信息。

截至目前,你已经成功创建了自己的支持WSGI协议的 Web 服务器,还利用不同的网络框架开发了多个网络应用。另外,你还自己开发了一个极简的网络框架。本文介绍的内容可谓不丰富。我们接下来回顾一下WSGI Web 服务器如何处理HTTP请求:

- 首先,服务器启动并加载网络框架/应用提供的 `application` 可调对象
- 然后,服务器读取一个请求信息
- 然后,服务器对请求进行解析
- 然后,服务器使用请求数据创建一个名叫 `environ` 的字典
- 然后,服务器以 `environ` 字典和 `start_response` 可调对象作为参数,调用 `application`,并获得应用生成的响应正文。
- 然后,服务器根据调用 `application` 对象后返回的数据,以及 `start_response` 设置的状态码和响应标头,构建一个HTTP响应。
- 最后,服务器将HTTP响应返回至客户端。



以上就是第二部分的所有内容。你现在已经拥有了一个正常运行的WSGI服务器，可以支持通过遵守WSGI协议的网络框架所写的网络应用。最棒的是，这个服务器可以不需要作任何代码修改，就可以与多个网络框架配合使用。

最后，我再给大家留一道思考题：怎样让服务器一次处理多个请求？

来源：编程派 参考原文：<http://ruslanspivak.com/lbaws-part2/>  
编译文章：<http://codingpy.com/article/build-a-simple-web-server-part-two/>

作者：Ruslan

本文为转载，如需再次转载，请查看源站“编程派”的要求。如果我们的工作有侵犯到您的权益，请及时联系我们。  
文章仅代表作者的知识和看法，如有不同观点，请楼下排队吐槽 :D

上一篇：[黑客利用 Wi-Fi 攻击你的七种方法](#)

发表评论

评论

最新评论

[我也要发表评论](#)

Linux.CN © 2003-2015 Linux中国 | Powered by **DX** | 图片存储于七牛云存储

京ICP备05083684号-1 京公网安备110105001595

服务条款 | 除特别申明外，本站原创内容版权遵循 CC-BY-NC-SA 协议规定

