









- [首页](#)
- [所有文章](#)
- [观点与动态](#)
- [基础知识](#)
- [系列教程](#)
- [实践项目](#)
- [工具与框架](#)
- [工具资源](#)
- [Python小组](#)

一起写一个 Web 服务器 (2)

2015/06/06 · [实践项目](#) · [10 评论](#) · [Web服务器](#)

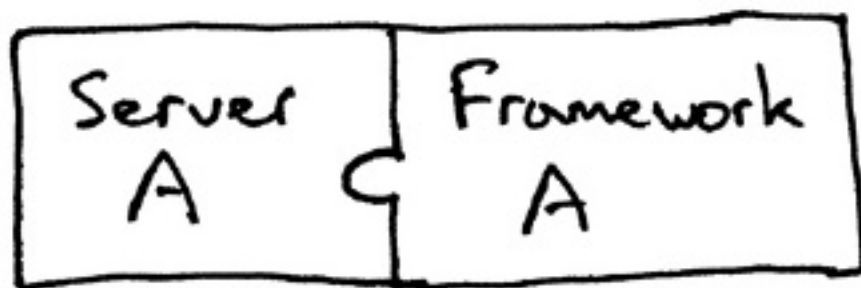
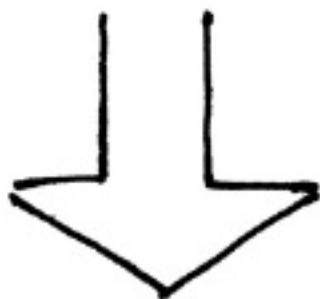
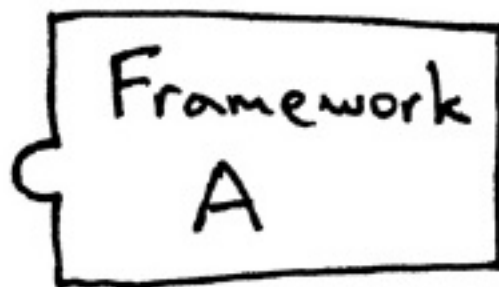
分享到:         9 本文由 [伯乐在线 - 高世界](#) 翻译, [艾凌风](#) 校稿。未经许可, 禁止转载!

英文出处: [ruslan spivak](#)。欢迎加入[翻译组](#)。

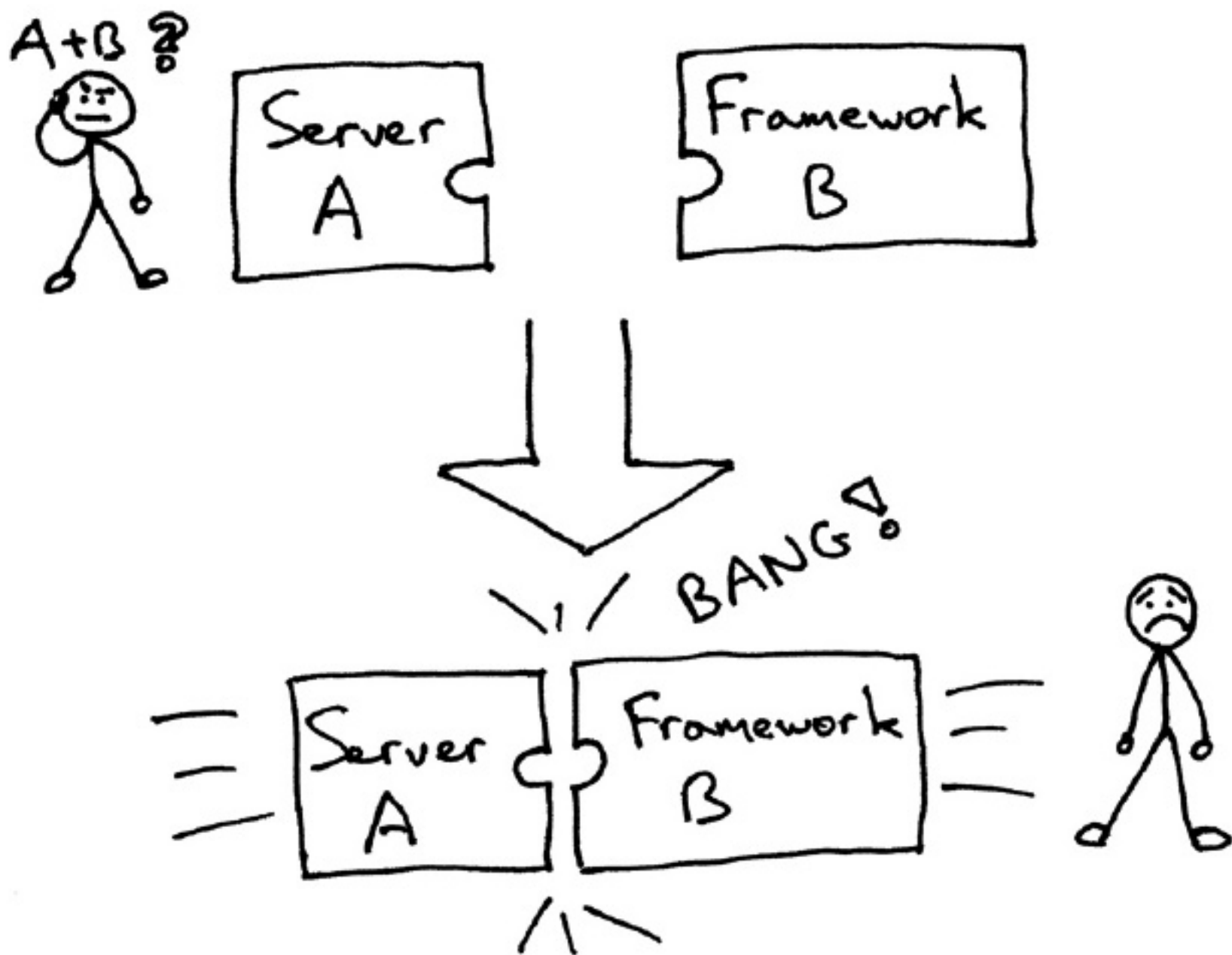
还记得吗? [在本系列第一部分我问过你](#): “怎样在你的刚完成的WEB服务器下运行 Django 应用、Flask 应用和 Pyramid 应用? 在不单独修改服务器来适应这些不同的WEB框架的情况下。”往下看, 来找出答案。

过去, 你所选择的一个Python Web框架会限制你选择可用的Web服务器, 反之亦然。如果框架和服务器设计的是可以一起工作的, 那就很好:

A+A ?



但是，当你试着结合没有设计成可以一起工作的服务器和框架时，你可能要面对（可能你已经面对了）下面这种问题：

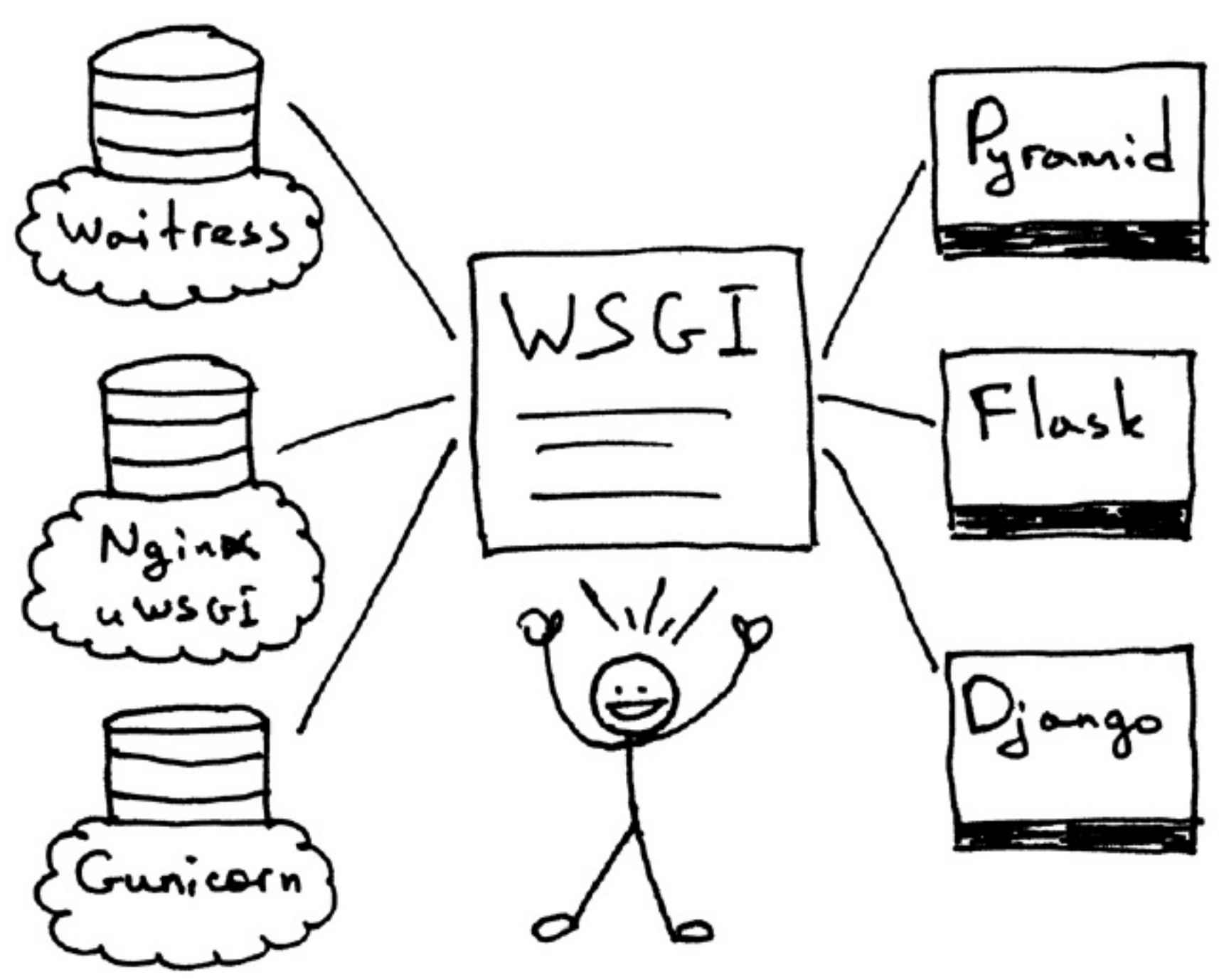


基本上，你只能用可以在一起工作的部分，而不是你想用的部分。

那么，怎样确保在不修改Web服务器和Web框架下，用你的Web服务器运行不同的Web框架？答案就是Python Web服务器网关接口（或者缩写为WSGI，读作“wizgy”）。



WSGI允许开发者把框架的选择和服务器的选择分开。现在你可以真正地混合、匹配Web服务器和Web框架了。例如，你可以在Gunicorn或者Nginx/uWSGI或者Waitress上面运行Django，Flask，或Pyramid。真正的混合和匹配哟，感谢WSGI服务器和框架两者都支持：



就这样，WSGI成了我在本系列第一部分和本文开头重复问的问题的答案。你的Web服务器必须实现WSGI接口的服务器端，所有的现代Python Web框架已经实现了WSGI接口的框架端了，这就让你可以不用修改服务器代码，适应某个框架。

现在你了解了Web服务器和Web框架支持的WSGI允许你选择一对儿合适的（服务器和框架），它对服务器和框架的开发者也有益，因为他们可以专注于他们特定的领域，而不是越俎代庖。其他语言也有相似的接口：例如，Java有Servlet API，Ruby有Rack。

一切都还不错，但我打赌你会说：“秀代码给我看！”好吧，看看这个漂亮且简约的WSGI服务器实现：

```
Python
1 # Tested with Python 2.7.9, Linux & Mac OS X
2 import socket
3 import StringIO
4 import sys
5
6
7 class WSGIServer(object):
```

```

8
9 address_family = socket.AF_INET
10 socket_type = socket.SOCK_STREAM
11 request_queue_size = 1
12
13 def __init__(self, server_address):
14     # Create a listening socket
15     self.listen_socket = listen_socket = socket.socket(
16         self.address_family,
17         self.socket_type
18     )
19     # Allow to reuse the same address
20     listen_socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
21     # Bind
22     listen_socket.bind(server_address)
23     # Activate
24     listen_socket.listen(self.request_queue_size)
25     # Get server host name and port
26     host, port = self.listen_socket.getsockname()[:2]
27     self.server_name = socket.getfqdn(host)
28     self.server_port = port
29     # Return headers set by Web framework/Web application
30     self.headers_set = []
31
32 def set_app(self, application):
33     self.application = application
34
35 def serve_forever(self):
36     listen_socket = self.listen_socket
37     while True:
38         # New client connection
39         self.client_connection, client_address = listen_socket.accept()
40         # Handle one request and close the client connection. Then
41         # loop over to wait for another client connection
42         self.handle_one_request()
43
44 def handle_one_request(self):
45     self.request_data = request_data = self.client_connection.recv(1024)
46     # Print formatted request data a la 'curl -v'
47     print(''.join(
48         '< {line}\n'.format(line=line)
49         for line in request_data.splitlines()
50     ))
51
52     self.parse_request(request_data)
53
54     # Construct environment dictionary using request data
55     env = self.get_environ()
56
57     # It's time to call our application callable and get
58     # back a result that will become HTTP response body
59     result = self.application(env, self.start_response)
60
61     # Construct a response and send it back to the client
62     self.finish_response(result)
63
64 def parse_request(self, text):
65     request_line = text.splitlines()[0]
66     request_line = request_line.rstrip('\r\n')
67     # Break down the request line into components
68     (self.request_method, # GET
69      self.path, # /hello
70      self.request_version # HTTP/1.1
71      ) = request_line.split()
72
73 def get_environ(self):
74     env = {}
75     # The following code snippet does not follow PEP8 conventions

```



```

76 # but it's formatted the way it is for demonstration purposes
77 # to emphasize the required variables and their values
78 #
79 # Required WSGI variables
80 env['wsgi.version'] = (1, 0)
81 env['wsgi.url_scheme'] = 'http'
82 env['wsgi.input'] = StringIO.StringIO(self.request_data)
83 env['wsgi.errors'] = sys.stderr
84 env['wsgi.multithread'] = False
85 env['wsgi.multiprocess'] = False
86 env['wsgi.run_once'] = False
87 # Required CGI variables
88 env['REQUEST_METHOD'] = self.request_method # GET
89 env['PATH_INFO'] = self.path # /hello
90 env['SERVER_NAME'] = self.server_name # localhost
91 env['SERVER_PORT'] = str(self.server_port) # 8888
92 return env

```

```

94 def start_response(self, status, response_headers, exc_info=None):
95     # Add necessary server headers
96     server_headers = [
97         ('Date', 'Tue, 31 Mar 2015 12:54:48 GMT'),
98         ('Server', 'WSGIServer 0.2'),
99     ]
100     self.headers_set = [status, response_headers + server_headers]
101     # To adhere to WSGI specification the start_response must return
102     # a 'write' callable. We simplicity's sake we'll ignore that detail
103     # for now.
104     # return self.finish_response

```

```

106 def finish_response(self, result):
107     try:
108         status, response_headers = self.headers_set
109         response = 'HTTP/1.1 {status}\r\n'.format(status=status)
110         for header in response_headers:
111             response += '{0}: {1}\r\n'.format(*header)
112         response += '\r\n'
113         for data in result:
114             response += data
115         # Print formatted response data a la 'curl -v'
116         print(''.join(
117             '> {line}\n'.format(line=line)
118             for line in response.splitlines()
119         ))
120         self.client_connection.sendall(response)
121     finally:
122         self.client_connection.close()

```

```

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

```

```

128 def make_server(server_address, application):
129     server = WSGIServer(server_address)
130     server.set_app(application)
131     return server

```

```

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

```

它明显比本系列第一部分中的服务器代码大，但为了方便你理解，而不陷入具体细节，它也足够小了（只有150行不到）。上面的服务器还做了别的事 - 它可以运行你喜欢的Web框架写的基本的Web应用，可以是Pyramid，Flask，Django，或者其他的Python WSGI框架。

不信？自己试试看。把上面的代码保存成webserver2.py或者直接从Github上下载。如果你不带参数地直接运行它，它就会报怨然后退出。

```
Python
1 $ python webserver2.py
2 Provide a WSGI application object as module:callable
```

它真的想给Web框架提供服务，从这开始有趣起来。要运行服务器你唯一需要做的是安装Python。但是要运行使用Pyramid，Flask，和Django写的的应用，你得先安装这些框架。一起安装这三个吧。我比较喜欢使用virtualenv。跟着以下步骤来创建和激活一个虚拟环境，然后安装这三个Web框架。

```
Python
1 $ [sudo] pip install virtualenv
2 $ mkdir ~/envs
3 $ virtualenv ~/envs/lsbaws/
4 $ cd ~/envs/lsbaws/
5 $ ls
6 bin include lib
7 $ source bin/activate
8 (lsbaws) $ pip install pyramid
9 (lsbaws) $ pip install flask
10 (lsbaws) $ pip install django
```

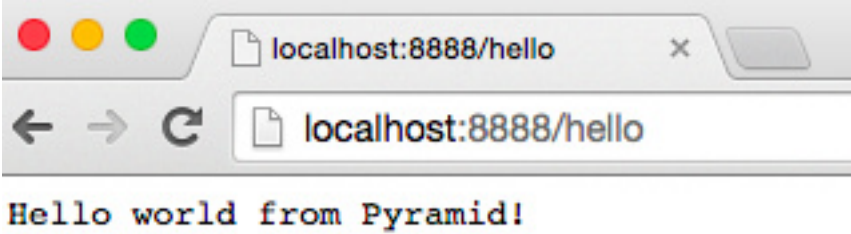
此时你需要创建一个Web应用。我们先拿Pyramid开始吧。保存以下代码到保存webserver2.py时相同的目录。命名为pyramidapp.py。或者直接从Github上下载：

```
Python
1 from pyramid.config import Configurator
2 from pyramid.response import Response
3
4
5 def hello_world(request):
6     return Response(
7         'Hello world from Pyramid!\n',
8         content_type='text/plain',
9     )
10
11 config = Configurator()
12 config.add_route('hello', '/hello')
13 config.add_view(hello_world, route_name='hello')
14 app = config.make_wsgi_app()
```

现在你已经准备好用完全属于自己的Web服务器来运行Pyramid应用了：

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

刚才你告诉你的服务器从python模块‘pyramidapp’中加载可调用的‘app’，现在你的服务器准备好了接受请求然后转发它们给你的Pyramid应用。目前应用只处理一个路由： /hello 路由。在浏览器里输入http://localhost:8888/hello地址，按回车键，观察结果：



你也可以在命令行下使用‘curl’工具来测试服务器：

```
1 $ curl -v http://localhost:8888/hello
2 ...
```

检查服务器和curl输出了什么到标准输出。

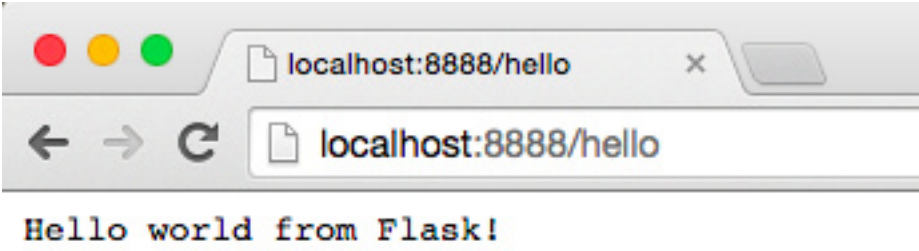
现在弄Flask。按照相同的步骤。

```
1 from flask import Flask
2 from flask import Response
3 flask_app = Flask('flaskapp')
4
5
6 @flask_app.route('/hello')
7 def hello_world():
8     return Response(
9         'Hello world from Flask!\n',
10        mimetype='text/plain'
11    )
12
13 app = flask_app.wsgi_app
```

保存以上代码为flaskapp.py或者从Github上下载它。然后像这样运行服务器：

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

现在在浏览器里输入http://localhost:8888/hello然后按回车：



再一次，试试‘curl’，看看服务器返回了一条Flask应用产生的消息：

```
1 $ curl -v http://localhost:8888/hello
2 ...
```

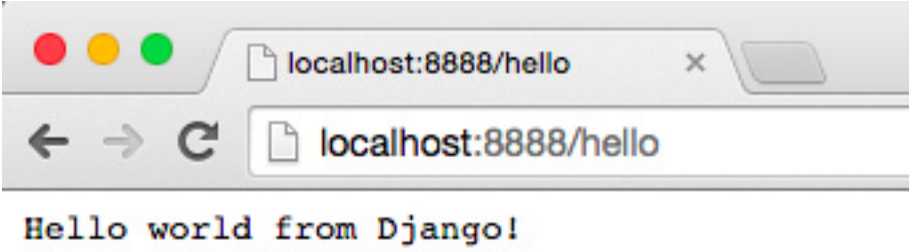

服务器也能处理Django应用吗？试试吧！尽管这有点复杂，但我还是推荐克隆整个仓库，然后使用djangoapp.py，它是GitHub仓库的一部分。以下的源码，简单地把Django ‘helloworld’ 工程（使用Django的django-admin.py启动项目预创建的）添加到当前Python路径，然后导入了工程的WSGI应用。

```
1 import sys
2 sys.path.insert(0, './helloworld')
3 from helloworld import wsgi
4
5 app = wsgi.application
```

把以上代码保存为djangoapp.py，然后用你的Web服务器运行Django应用：

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

输入下面的地址，然后按回车键：



虽然你已经做过两次啦，你还是可以再在命令行测试一下，确认一下，这次是Django应用处理了请求。

```
1 $ curl -v http://localhost:8888/hello
2 ...
```

你试了吧？你确定服务器可以和这三个框架一起工作吧？如果没试，请试一下。阅读挺重要，但这个系列是关于重建的，也就是说，你要自己动手。去动手试试吧。别担心，我等你哟。你必须试下，最好呢，你亲自输入所有的东西，确保它工作起来像你期望的那样。

很好，你已经体验到了WSGI的强大：它可以让你把Web服务器和Web框架结合起来。WSGI提供了Python Web服务器和Python Web框架之间的一个最小接口。它非常简单，在服务器和框架端都可以轻易实现。下面的代码片段展示了（WSGI）接口的服务器和框架端：

```
1 def run_application(application):
2     """Server code."""
3     # This is where an application/framework stores
4     # an HTTP status and HTTP response headers for the server
5     # to transmit to the client
6     headers_set = []
7     # Environment dictionary with WSGI/CGI variables
8     environ = {}
9
10    def start_response(status, response_headers, exc_info=None):
11        headers_set[:] = [status, response_headers]
12
```

```

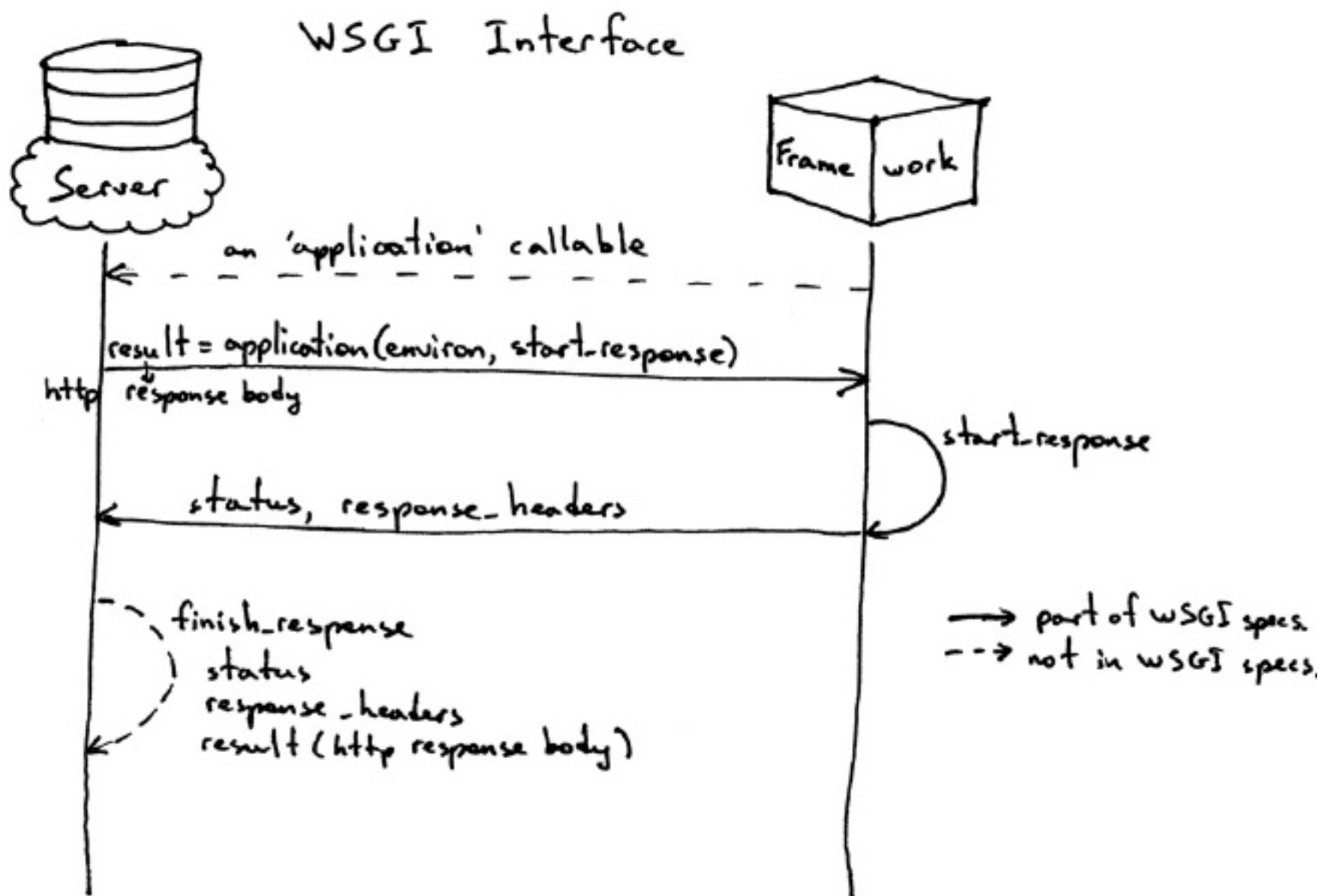
13 # Server invokes the 'application' callable and gets back the
14 # response body
15 result = application(environ, start_response)
16 # Server builds an HTTP response and transmits it to the client
17 ...
18
19 def app(environ, start_response):
20     """A barebones WSGI app."""
21     start_response('200 OK', [('Content-Type', 'text/plain')])
22     return ['Hello world!']
23
24 run_application(app)

```

以下是它如何工作的：

- 1.框架提供一个可调用的'应用'（WSGI规格并没有要求如何实现）
- 2.服务器每次接收到HTTP客户端请求后，执行可调用的'应用'。服务器把一个包含了WSGI/CGI变量的字典和一个可调用的'start_response'做为参数给可调用的'application'。
- 3.框架/应用生成HTTP状态和HTTP响应头，然后把它们传给可调用的'start_response'，让服务器保存它们。框架/应用也返回一个响应体。
- 4.服务器把状态，响应头，响应体合并到HTTP响应里，然后传给（HTTP）客户端（这步不是（WSGI）规格里的一部分，但它是后面流程中的一步，为了解释清楚我加上了这步）

以下是接口的视觉描述：



目前为止，你已经了解了Pyramid，Flask，和Django Web应用，你还了解了实现了WSGI规范服务器端的服务器代码。你甚至已经知道了不使用任何框架的基本的WSGI应用代码片段。

问题就在于，当你使用这些框架中的一个来写Web应用时，你站在一个比较高的层次，并不直接和WSGI打交道，但我知道你对WSGI接口的框架端好奇，因为你在读本文。所以，咱们一起写个极简的

WSGI Web应用/Web框架吧，不用Pyramid，Flask，或者Django，然后用你的服务器运行它：

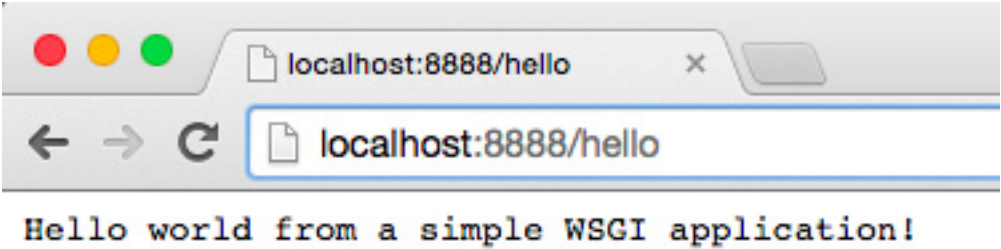
```
def app(environ, start_response):
    """A barebones WSGI application.

    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']
```

再次，保存以上代码到wsgiapp.py文件，或者直接从GitHub上下载，然后像下面这样使用你的Web服务器运行应用：

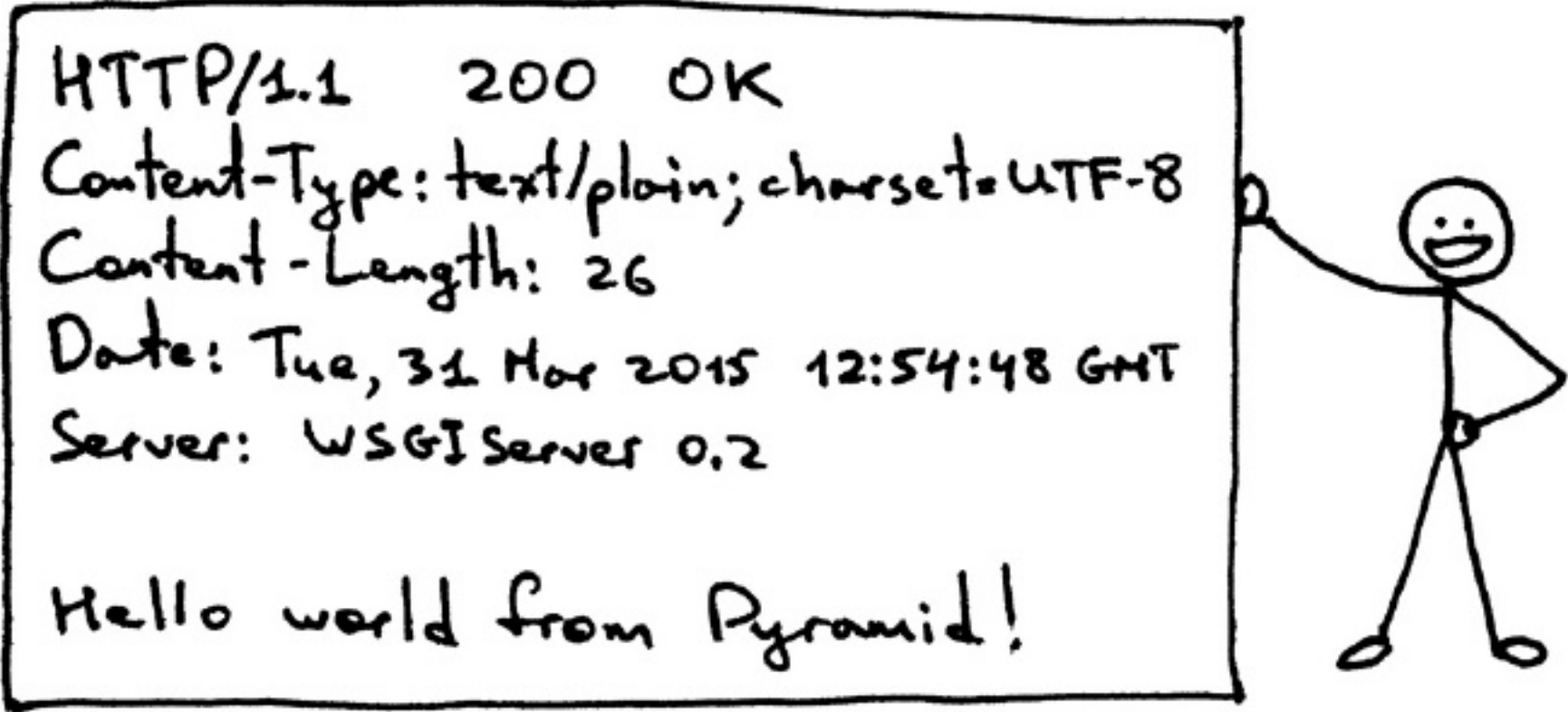
```
(lsbaws) $ python webserver2.py wsgiapp:app
WSGIServer: Serving HTTP on port 8888 ...
```

输入下面地址，敲回车。你应该就看到下面结果了：



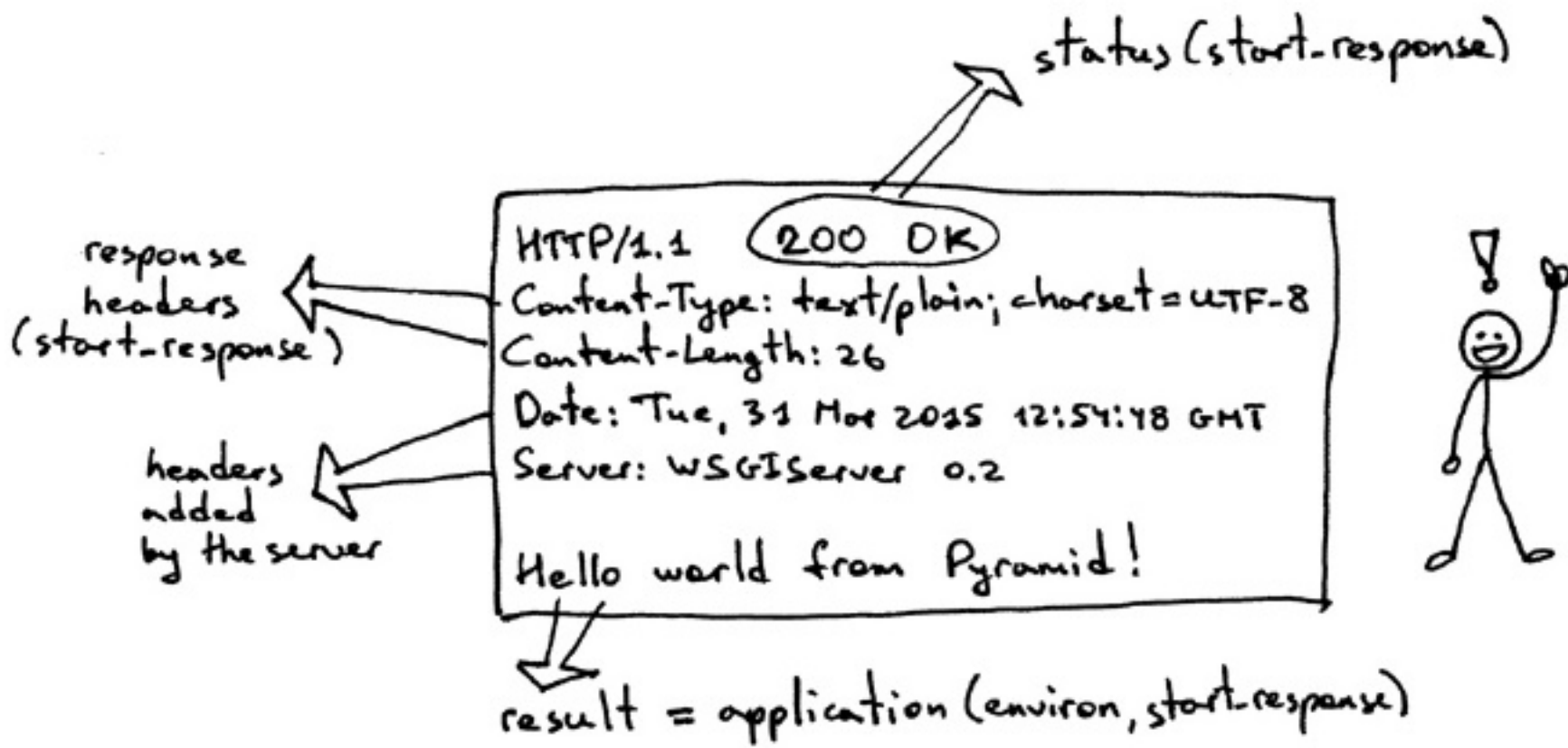
在你学习怎样写一个Web服务器时，你刚刚写了一个你自己的极简的WSGI Web框架！棒极啦。

现在，让我们回头看看服务器传输了什么给客户端。以下就是使用HTTP客户端调用Pyramid应用时生成的HTTP响应：



这个响应跟你在本系列第一部分看到的有一些相近的部分，但也有一些新东西。例如，你以前没见过的4个HTTP头：Content-Type, Content-Length, Date, 和Server。这些头是Web服务器生成的响应应该有的。虽然他们并不是必须的。头的目的传输HTTP请求/响应的额外信息。

现在你对WSGI接口了解的更多啦，同样，以下是带有更多信息的HTTP响应，这些信息表示了哪些部件产生的它（响应）：



我还没有介绍'environ'字典呢，但它基本上就是一个Python字典，必须包含WSGI规范规定的必要的WSGI和CGI变量。服务器在解析请求后，从HTTP请求拿到了字典的值，字典的内容看起来像下面这样：

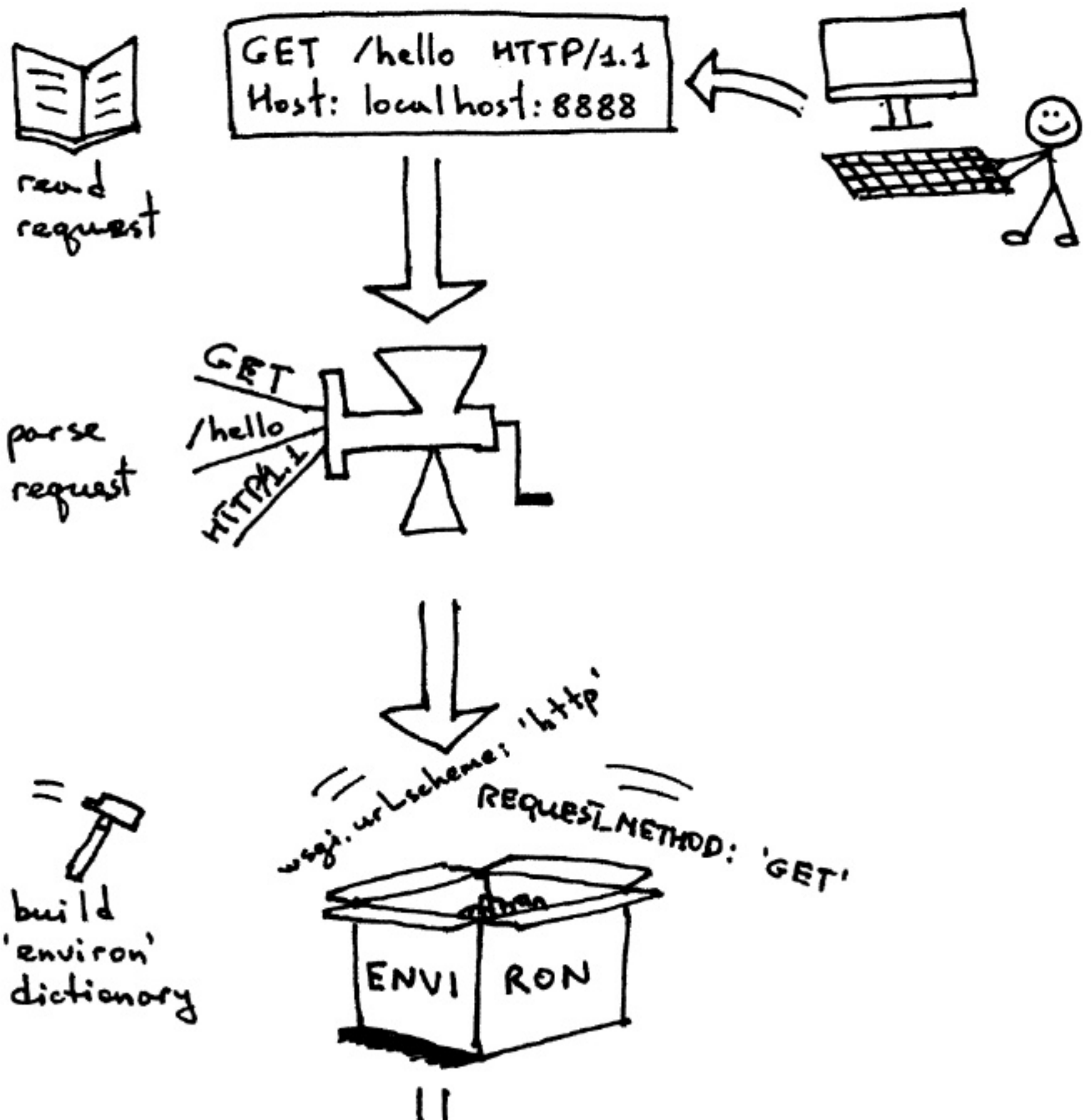
ENVIRON

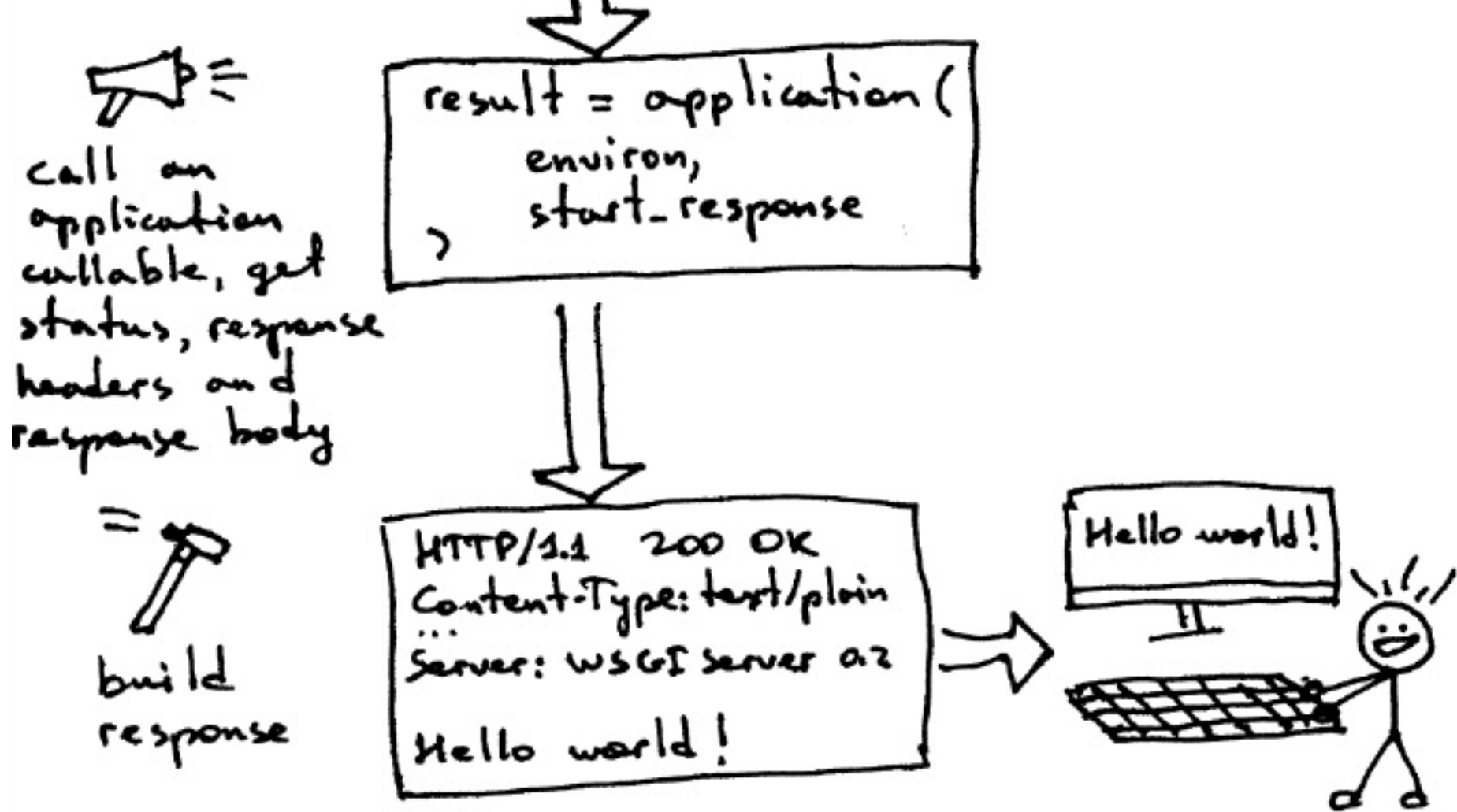
wsgi.version	(1, 0)
wsgi.url_scheme	'http'
wsgi.input	StringIO(request_data)
wsgi.errors	sys.stderr
wsgi.multiprocess	False
wsgi.multithread	False
wsgi.run_once	False
REQUEST_METHOD	'GET'
PATH_INFO	'/hello'
SERVER_NAME	'localhost'
SERVER_PORT	8888

Web框架使用字典里的信息来决定使用哪个视图，基于指定的路由，请求方法等，从哪里读请求体，错误写到哪里去，如果有的话。

现在你已经创建了你自己的WSGI Web服务器，使用不同的Web框架写Web应用。还有，你还顺手写了个简单的Web应用/Web框架。真是段难忘的旅程。咱们简要重述下WSGI Web服务器必须做哪些工作才能处理发给WSGI应用的请求吧：

- 首先，服务器启动并加载一个由Web框架/应用提供的可调用的'application'
- 然后，服务器读取请求
- 然后，服务器解析它
- 然后，服务器使用请求的数据创建了一个'environ'字典
- 然后，服务器使用'environ'字典和'start_response'做为参数调用'application'，并拿到返回的响应体。
- 然后，服务器使用调用'application'返回的数据，由'start_response'设置的状态和响应头，来构造HTTP响应。
- 最终，服务器把HTTP响应传回给客户端。





这就是全部啦。现在你有了一个可工作的WSGI服务器，它可以处理使用像Django，Flask，Pyramid或者你自己的WSGI框架这样的兼容WSGI的Web框架写的基本的Web应用。最优秀的地方是，服务器可以在不修改代码的情况下，使用不同的Web框架。

在你离开之前，还有个问题请你想一下，“该怎么做才能让服务器同一时间处理多个请求呢？”

保持关注，我会在本系列第三部分秀给你看实现它的一种方式。欢呼！

顺便说下，我在写一本书《一起构建WEB服务器：第一步》，它解释了从零开始写一个基本的WEB服务器，还更详细地讲解了我上面提到的话题。订阅邮件组来获取关于书籍和发布时间和最近更新。

2 赞

27 收藏

关于作者：高世界

我翻译得越多，发现知道的越少，我就要更多地翻译。论得的地正确用法。我是php开发者，对python，c/c++，linux感兴趣。

个人主页 · 我的文章 · 17 ·



相关文章

- [从零开始搭建论坛 \(2\)：Web服务器网关接口](#)
- [从零开始搭建论坛 \(1\)：Web服务器与Web框架](#) · [🗨️ 2](#)
- [一起写一个Web服务器 \(3\)](#) · [🗨️ 8](#)
- [一起写一个 Web 服务器 \(1\)](#) · [🗨️ 14](#)

可能感兴趣的话题

- [Java中的陷阱题--找奇数](#) · [🗨️ 2](#)
- [请大家各抒己见谈谈自己对现代战争中信息化的利用](#) · [🗨️ 1](#)
- [在北京做了5年开发，感觉无法突破自己，想去南方深圳发展，求建议](#) · [🗨️ 4](#)
- [父类和子类如何使用同一个装饰器呢，下面代码应该怎么改](#) · [🗨️ 2](#)
- [请杭州的程序员朋友帮推一份前端工程师的工作](#)
- [2016年链家网校招笔试（JAVA研发）：二叉树遍历](#)

登录后评论

新用户注册

直接登录



最新评论



[SeeYouAgain](#) (🎓 1)

2015/06/15

请问part3 好久能出呢

👍 赞 回复 ↩



[高世界](#) (🎓 17 · 🔒 🗨️)

2015/06/17

好问题。。。我抓紧译。。。稍等。

👍 赞 回复 ↩



[gatspy](#) (🎓 2)

2015/07/24

谢谢马儿的翻译啊。太棒了

👍 赞 回复 ↩



[生牛好白](#) (🎓 1)

2015/08/21

马儿 你吃的草从哪里来啊
：)
开玩笑 这些文章原文在哪

👍 赞 回复 ↩



小编 (10 · 0)
编辑

2015/08/21

文章开头已经注明了

赞 回复



smith alex

2016/01/02

发现翻译时候的代码有点小问题..去原博客复制的代码解决了问题...

赞 回复



HatBoy (1)

2016/06/25

用Python3编码问题调试了半天才调试通

赞 回复



no game no life (1 · 0)
程序员

2016/07/05

代码显示有点问题， '<'和'>' 被转义成<和> 还有字符串中的\n中的转义符被吞了，变成n了

赞 回复



黄利民 (97 · 0)
站长

2016/07/06

已修复，谢谢反馈

赞 回复



TDream (1)

2016/11/06

好文章，我喜欢，但是英文原文链接打不开，不知道为什么。。。

赞 回复



Python小组话题

我有新话题



[有没有非互联网行业的小伙伴自学编程...叫我小K咯](#) 发起 • 176 回复



[随着Python越来越火，自己也慢慢入了py...、O.o?](#) 发起 • 19 回复



[class的作用域](#)
[day_day_up](#) 发起 • 3 回复



[明年找工作，求python大神指条明路太懒~](#) 发起 • 15 回复



[父类和子类如何使用同一个装饰器呢，...加瓦](#) 发起 • 2 回复



[如何使用多线程逐套下载多套图片？~桂~](#) 发起 • 1 回复



- [本周热门Python文章](#)
- [本月热门](#)
- [热门标签](#)

0 [python logging日志模块以及多进程...](#)

1 [Python标准库系列之Redis模块](#)



[Tryton：一个通用商务框架](#) [杂项](#)



[NLTK：一个先进的用来处理自然语言数据的Python程序。](#) [自然语言处理](#)



[PyMC：马尔科夫链蒙特卡洛采样工具](#) [科学计算与分析](#)



[statsmodels：统计建模和计量经济学](#) [科学计算与分析](#)



[Pylearn2：一个基于Theano的机器学习库](#) [机器学习](#) · [Q1](#)

关于 Python 频道

Python频道分享 Python 开发技术、相关的行业动态。

快速链接

[网站使用指南 »](#)

[加入我们 »](#)

[问题反馈与求助 »](#)

[网站积分规则 »](#)

关注我们

新浪微博: [@Python开发者](#)

RSS: [订阅地址](#)

推荐微信号



合作联系

Email: bd@Jobbole.com

QQ: 2302462408 (加好友请注明来意)

更多频道

- [小组](#) - 好的话题、有启发的回复、值得信赖的圈子
- [头条](#) - 分享和发现有价值的内容与观点
- [相亲](#) - 为IT单身男女服务的征婚传播平台
- [资源](#) - 优秀的工具资源导航
- [翻译](#) - 翻译传播优秀的外文文章
- [文章](#) - 国内外的精选文章
- [设计](#) - UI,网页, 交互和用户体验
- [iOS](#) - 专注iOS技术分享
- [安卓](#) - 专注Android技术分享
- [前端](#) - JavaScript, HTML5, CSS
- [Java](#) - 专注Java技术分享
- [Python](#) - 专注Python技术分享

