```
def __init__(self, environ, start_response):
self.environ = environ
self.start = start_response

def __iter__(self):
status = '200 OK'
response_headers = [('Content-type','text/plain')]
self.start(status, response_headers)
yield "Hello world!\n"
```

服务器接口

服务器/gateway为每一个http客户端发来的请求都会请求应用程序可调用者一次。为了说明这里有一个CGI gateway，以一个获取应用程序对象的函数实现，请注意，这个例子拥有有限的错误处理，因为默认情况下没有被捕获的异常都会被输出到sys.stderr并被服务器记录下来。

```
import os, sys

def run_with_cgi(application):

environ = dict(os.environ.items())

environ['wsgi.input']        = sys.stdin

environ['wsgi.errors']       = sys.stderr

environ['wsgi.version']      = (1,0)

environ['wsgi.multithread']  = False

environ['wsgi.multiprocess'] = True
```

John_ABC的博客

http://blog.sina.com.cn/u/2316878653 [订阅] [手机订阅]

首页 | 博文目录 | 图片 | 关于我

John_ABC

微博

加好友 发纸条

写留言 加关注

博客等级：16

博客积分：909 新

博客访问：118,722

关注人气：34

获赠金笔：4

赠出金笔：0

荣誉徽章：

正文

PEP 333（不是3333）的翻译 (2013-07-22 17:28:15)

分类：Python

简介

本文档描述一份在web服务器与web应用/web框架之间的标准接口，此接口的目的

性。

基本原理与目标

python目前拥有大量的web框架，比如 Zope, Quixote, Webware, SkunkWeb, PSO,
为总得来说，框架的选择都会限制web服务器的选择。

对比之下，虽然java也拥有许多web框架，但是java的" servlet" API使得使用任何
API的web服务器上运行。服务器中这种针对python的API（不管服务器是用pytho
python）的使用和普及，将分离人们对web框架和对web服务器的选择，用户可以
的开发者也能够把精力集中到各自的领域。

因此，这份PEP建议在web服务器和web应用/web框架之间建立一种简单的通用的
(WSGI).

但是光有这么一份规范对于改变web服务器和web应用/框架的现状是不够的，只
起应有的效果。

然而，既然还没有任何框架或服务器实现了WSGI，对实现WSGI也没有什么直接
作者的初始投资。

服务器和框架两边接口的实现的简单性，对于WSGI的作用来说，绝对是非常重

对于框架作者来说，实现的简单和使用的方便是不一样的。WSGI为框架作者展

```
environ['wsgi.run_once'] = True

if environ.get('HTTPS','off') in ('on','1'):

environ['wsgi.url_scheme'] = 'https'

else:

environ['wsgi.url_scheme'] = 'http'

headers_set = []

headers_sent = []

def write(data):

if not headers_set:

raise AssertionError("write() before start_response()")

elif not headers_sent:

# Before the first output, send the stored headers

status, response_headers = headers_sent[:] = headers_set

sys.stdout.write('Status: %s\r\n' % status)

for header in response_headers:

sys.stdout.write('%s: %s\r\n' % header)

sys.stdout.write('\r\n')

sys.stdout.write(data)

sys.stdout.flush()

def start_response(status,response_headers,exc_info=None):

if exc_info:
```

相关博文

- 韩国姑娘春日街头长腿秀
  Korea
- 台南夜市人气爆棚的美食
  BoBo
- 女星透视装若隐若现朦胧美
  娱娱玉玉
- 点翠头面及其他
  傩送
- 下周密切注意量能变化
  紫气东来dhm
- 人口与投资
  梁建章
- [转载] 价值投资就是一种保
  佐罗投资札记
- 男性身体大滑坡的三个精准信号
  梁发汇
- 国产新车电池给力，特斯拉你就
  概念爱好者
- ST博元:庄春虹何许人也?
  ST老大
- 生态文明的要害在保护荒凉
  冯永锋
- 科技股的诱惑---牛市开新仓
  黑熊捕鱼
- 只重眼前是散户的通病
  anmin0001

更多>>

和对cookie的处理这些问题和框架现有的对这些问题的处理是矛盾的。再次重申
间轻松互连，而不是创建一套新的web框架。

同时也要注意到，这个目标使得WSGI不能依赖任何在当前已部署版本的python
标准模块，并且WSGI并不需要2.2.2以上版本的python(当然，在以后的python标
注意)

不光要让现有的或将要出现的框架和服务器容易实现，也应该容易创建请求预处
件，对于服务器来说他们是应用程序，而对于他们包含的应用程序来说他们是服

如果中间件既简单又健壮，而且WSGI广泛得实现在服务器和框架中，那么就有
个WSGI中间件组件组成。甚至现有框架的作者都会选择重构将以实现的服务以
而不是一个独立的框架。这样web应用开发这就可以根据特定功能选择最适合的

当然，这一天无疑还要等很久，在这之间，一个合适的短期目标就是让任何框架

最后，需要指出的是当前版本的WSGI并没有规定一个应用具体以何种方式部署
器或gateway的具体实现来定义。如果足够多实现了WSGI的服务器或gateway通
PEP来描述WSGI服务器和应用框架的部署标准。

概述

WSGI接口有两种形式：一个是针对服务器或gateway的，另一个针对应用程序或
用的对象，至于该对象是如何被请求的取决与服务器或gateway。我们假定一些
个简短的脚本来启动一个服务器或gateway的实例，并把应用程序对象提供得服
他机制来指定从哪里导入或者或得应用程序对象。

除了纯粹的服务器/gateway和应用程序/框架，还可以创建实现了这份规格说明书
程序，而对于他们包含的应用程序来说他们是服务器，他们可以用来提供可扩展

在整个规格说明书中，我们使用短语"一个可调用者"意思是"一个函数，方法，
与服务器，gateway，应用程序根据需要而选择的合适实现方式。相反服务器，g
具体的实现方式，not introspected upon.

应用接口

一个应用程序对象是一个简单的接受两个参数的可调用对象，这里的对象并不是
者带有 __call__ 方法的对象实例都可以用来做应用程序对象。应用程序对象必须
确实会产生这样的重复请求。

(注意：虽然我们把他叫做"应用程序"对象，但并不是说程序员要把WSGI当做AP
面上的框架服务来开发应用程序，WSGI是提供给框架和服务器开发者使用的工

这里有两个应用程序对象的示例，一个是函数，另一个是类:

```
def simple_app(environ, start_response):

    """也许是最简单的应用程序对象"""

    status = '200 OK'

    response_headers = [('Content-type','text/plain')]

    start_response(status, response_headers)
```

```
try:
if headers_sent:
# Re-raise original exception if headers sent
raise exc_info[0], exc_info[1], exc_info[2]
finally:
exc_info = None     # avoid dangling circular ref
elif headers_set:
raise AssertionError("Headers already set!")
headers_set[:] = [status,response_headers]
return write
result = application(environ, start_response)
try:
for data in result:
if data:    # body 出现以前不发送headers
write(data)
if not headers_sent:
write('')   # 如果这个时候body为空则发送header
finally:
if hasattr(result,'close'):
result.close()
```

```
    return ['Hello world!\n']


class AppClass:

    """产生同样的输出，不过是使用一个类来实现


    (注意: 'AppClass' 在这里就是 "application" ,所以对它的调用会'AppClass'的一个

    这个实例做为迭代器再返回"application callable"应该返回的那些值)


    如果我们想使用 'AppClass' 的实例直接作为应用程序对象, 我们就必须实现 ``_

    外部通过调用这个方法来执行应用程序, 并且我们需要创建一个实例给服务器

    """


    def __init__(self, environ, start_response):

        self.environ = environ

        self.start = start_response


    def __iter__(self):

        status = '200 OK'

        response_headers = [('Content-type','text/plain')]

        self.start(status, response_headers)

        yield "Hello world!\n"
```

服务器接口

服务器/gateway为每一个http客户端发来的请求都会请求应用程序可调用者一次。
程序对象的函数实现，请注意，这个例子拥有有限的错误处理，因为默认情况下
器记录下来。

```
import os, sys
```

▪ 中国首脑雅加达会谈的意义何在？

▪ 风青杨：什么是“有偿异性陪侍”

▪ 直面疯狂的股市，政府该做什么

▪ 自主研发与拿来主义的不同结局

▪ “抗日神剧”走了，“足球神剧”

印加之路色彩盛宴

夜游京都祇园

地震中损毁的世界遗产

漫步东方瑞士博[illegible]

乌来养在深闺人未识

去皇家园林赏盛世牡丹

查看更多>>

谁看过这篇博文

BamBoo	4月20日
671324100	4月19日
我是一只...	4月15日
阳光下的...	4月14日
[illegible]	4月13日
gudy	4月11日
JustDoTufei	4月6日
善良的饭...	3月31日
leeway89	3月30日
sudo	3月30日
落木萧条	3月29日
qqliangqi	3月26日

```
def run_with_cgi(application):

    environ = dict(os.environ.items())

    environ['wsgi.input']        = sys.stdin

    environ['wsgi.errors']       = sys.stderr

    environ['wsgi.version']      = (1,0)

    environ['wsgi.multithread']  = False

    environ['wsgi.multiprocess'] = True

    environ['wsgi.run_once']    = True

    if environ.get('HTTPS','off') in ('on','1'):

        environ['wsgi.url_scheme'] = 'https'

    else:

        environ['wsgi.url_scheme'] = 'http'

    headers_set = []

    headers_sent = []

    def write(data):

        if not headers_set:

            raise AssertionError("write() before start_response()")

        elif not headers_sent:

            # Before the first output, send the stored headers

            status, response_headers = headers_sent[:] = headers_set

            sys.stdout.write('Status: %s\r\n' % status)

            for header in response_headers:
```

中间件：同时扮演两种角色的组件

注意到单个对象可以作为请求应用程序的服务器存在，也可以作为被服务器调用的应用程序存在。这样的中间件可以执行这样一些功能：

重写前面提到的 environ 之后，可以根据目标URL将请求传递到不同的应用程序对象

允许多个应用程序和框架在同一个进程中运行

通过在网络传递请求和响应，实现负载均衡和远程处理

对内容进行后加工，比如附加xsl样式表

中间件的存在对于服务器接口和应用接口来说都应该是透明的，并且不需要特别的支持。希望在应用程序中加入中间件的用户只需简单得把中间件当作应用提供给服务器，并配置中间件足以服务器的身份来请求应用程序。

当然，中间件组件包裹的可能是包裹应用程序的另一个中间件组件，这样循环下去就构成了我们称为"中间件堆栈"的东西了。for the most part,中间件要符合应用接口和服务器接口提出的一些限制和要求，有些时候这样的限制甚至比纯粹的服务器或应用程序还要严格，这些地方我们会特别指出。

这里有一个中间件组件的例子，他用Joe Strout的piglatin.py将text/plain的响应转换成pig latin（注意：真正的中间件应该使用更加安全的方式——应该检查内容的类型和内容的编码，同样这个简单的例子还忽略了一个单词might be split across a block boundary的可能性）。

```
from piglatin import piglatin

class LatinIter:

"""如果可以的话，将输出转换为piglatin格式

Note that the "okayness" can change until the application yields

its first non-empty string, so 'transform_ok' has to be a mutable

truth value."""

def __init__(self,result,transform_ok):

if hasattr(result,'close'):
```

```
self.close = result.close

self._next = iter(result).next

self.transform_ok = transform_ok

def __iter__(self):

return self

def next(self):

if self.transform_ok:

return piglatin(self._next())

else:

return self._next()

class Latinator:

# by default, don't transform output

transform = False

def __init__(self, application):

self.application = application

def __call__(environ, start_response):

transform_ok = []

def start_latin(status,response_headers,exc_info=None):

# Reset ok flag, in case this is a repeat call

transform_ok[:]=[]
```

```
                 sys.stdout.write('%s: %s\r\n' % header)

             sys.stdout.write('\r\n')


        sys.stdout.write(data)

        sys.stdout.flush()


    def start_response(status,response_headers,exc_info=None):

        if exc_info:

            try:

                if headers_sent:

                    # Re-raise original exception if headers sent

                    raise exc_info[0], exc_info[1], exc_info[2]

            finally:

                exc_info = None     # avoid dangling circular ref

        elif headers_set:

            raise AssertionError("Headers already set!")


        headers_set[:] = [status,response_headers]

        return write


    result = application(environ, start_response)

    try:

        for data in result:

            if data:    # body 出现以前不发送headers

                write(data)

        if not headers_sent:

            write('')   # 如果这个时候body为空则发送header
```

```
for name,value in response_headers:

if name.lower()=='content-type' and value=='text/plain':

transform_ok.append(True)

# Strip content-length if present, else it'll be wrong

response_headers = [(name,value)

for name,value in response_headers

if name.lower()<>'content-length'

]

break

write = start_response(status,response_headers,exc_info)

if transform_ok:

def write_latin(data):

write(piglatin(data))

return write_latin

else:

return write

return LatinIter(self.application(environ,start_latin),transform_ok)

# Run foo_app under a Latinator's control, using the example CGI gateway

from foo_app import foo_app

run_with_cgi(Latinator(foo_app))
```

详细说明

```
finally:

    if hasattr(result,'close'):

        result.close()
```

中间件：同时扮演两种角色的组件

注意到单个对象可以作为请求应用程序的服务器存在，也可以作为被服务器调用
功能:

- 重写前面提到的 environ 之后，可以根据目标URL将请求传递到不同的应用程
- 允许多个应用程序和框架在同一个进程中运行
- 通过在网络传递请求和响应，实现负载均衡和远程处理
- 对内容进行后加工，比如附加xsl样式表

中间件的存在对于服务器接口和应用接口来说都应该是透明的，并且不需要特别
简单得把中间件当作应用提供给服务器，并配置中间件足见以服务器的身份来请

当然，中间件组件包裹的可能是包裹应用程序的另一个中间件组件，这样循环下
most part,中间件要符合应用接口和服务器接口提出的一些限制和要求，有些时候
格，这些地方我们会特别指出。

这里有一个中间件组件的例子，他用Joe Strout的piglatin.py将text/plain的响应转换
的方式——应该检查内容的类型和内容的编码，同样这个简单的例子还忽略了一
性)。

```
from piglatin import piglatin



class LatinIter:



    """如果可以的话，将输出转换为piglatin格式



    Note that the "okayness" can change until the application yields

    its first non-empty string), so 'transform_ok' has to be a mutable

    truth value."""



    def __init__(self,result,transform_ok):

        if hasattr(result,'close'):

            self.close = result.close
```

详细说明

应用程序对象必须接受两个参数，为了方便说明我们不妨分别命名为 environ 和 start_response ，但并非必须取这个名字。服务器或gateway必须用这两个参数请求应用程序对象(比如象上面展示的,这样调用 result = application(environ,start_response))

参数 environ 是个字典对象，包含CGI风格的环境变量。这个对象必须是一个python内建的字典对象(不能是子类、UserDict或其他对字典对象的模仿)，应用程序可以以任何他愿意的方式修改这个字典， environ 还应该包含一些特定的WSGI需要的变量(在后面的节里会描述)，有可以包含一些服务器特定的扩展变量，通过下面提高的约定命名。

start_response 参数是一个接受两个必须参数和一个可选参数的可调用者。方便说明，我们分别把他们命名为 status, response_headers ,和 exc_info 。应用程序必须用这些参数来请求可调用者 start_response (比如象这样 start_response(status,response_headers))

参数 status 是一个形式象"999 Message here"的状态字符串。而 response_headers 参数是元组(header_name,header_value)的列表,描述http响应头。可选的 exc_info 参数会在下面的 `The start_response() Callable`_ 和 Error Handling 两节中描述，他只有在应用程序产生了错误并希望在浏览器上显示错误的时候才有用。

start_response 可调用者必须返回一个 write(body_data) 可调用者，他接受一个可选参数：一个将要被做为http响应体的一部分输出的字符串(注意：提供可调用者 write() 只是为了支持现有框架的必要的输出API，新的应用程序或框架尽量避免使用，详细情况请看 Buffering and Streaming 一节。)

当被服务器请求的时候，应用程序对象必须返回一个0或多个可迭代的字符串，这可以通过多种方法完成，比如返回一个字符串的列表，或者应用程序本身是一个生产字符串的函数，或者应用程序是一个类而他的实例是可迭代的，不管怎么完成，应用程序对象必须总是返回0或多个可迭代的字符串。

服务器必须将产生的字符串以一种无缓冲的方式传送到客户端，每次传完一个字符串再去获取下一个。(换句话说，应用程序应该实现自己的缓冲，更多关于应用程序输出必须如何处理的细节请阅读下面的 Buffering and Streaming 节。)

服务器或gateway应该把产生的字符串当字节流对待：特别地，他必须保证没修改行的结尾。应用程序负责确保字符串是以与客户端匹配的编码输出(服务器/gateway可能会附加HTTP传送编码，或者为了实现一些http的特性而进行一些转换比如byte-range transmission，更多细节请看下面的 Other HTTP Features)

```
        self._next = iter(result).next
        self.transform_ok = transform_ok
    def __iter__(self):
        return self
    def next(self):
        if self.transform_ok:
            return piglatin(self._next())
        else:
            return self._next()
class Latinator:
    # by default, don't transform output
    transform = False
    def __init__(self, application):
        self.application = application
    def __call__(self, environ, start_response):
        transform_ok = []
        def start_latin(status, response_headers, exc_info=None):
            # Reset ok flag, in case this is a repeat call
```

如果调 len(iterable) 成功，服务器将认为结果是正确的。也就是说，应用程序返回的可迭代的字符串提供了一个有用 的__len__() 方法，么肯定返回了正确的结果(关于这个方法正常情况下如何被使用的请阅读 Handling the Content-Length Header)

如果应用程序返回的可迭代者有close()方法，则不管该请求是正常结束还是由于错误而终止，服务器/gateway都**必须**在结束该请求之前调用这个方法，（这是用来支持应用程序对资源的释放，This protocol is intended to complement PEP 325's generator support, and other common iterables with close() methods.)

（注意：应用程序必须在可迭代者产生第一个字符串之间请求 start_response() 可调用者，这样服务器才能在发送任何主体内容之前发送响应头，然而这一步也可以在可迭代者第一次迭代的时候执行,所以服务器不能假定开始迭代之前 start_response() 已经被调用过了）

最后，服务器或gateway不能应用程序返回的可迭代者的任何其他属性，除非是针对服务器或gateway特定类型的实例，比如wsgi.file_wrapper返回的“file wrapper”（阅读 Optional Platform-Specific File Handling)。通常情况下，只有在这里指定的属性，或者通过PEP 234 iteration APIs才是可以访问的。
environ 变量

environ 字典被用来包含这些在Common Gateway Interface specification [2]_中定义了的CGI环境变量。下面这些变量 必须 呈现出来, 除非其值是空字符串,这种情况下如果下面没有特别指出的话他们 可能 会被忽略

REQUEST_METHOD
　HTTP请求的方式, 比如 "GET" 或者 "POST". 这个不可能是空字符串并且也是必须给出的。
SCRIPT_NAME
　请求URL中路径的开始部分，对应应用程序对象，这样应用程序就知道它的虚拟位置。如果该应用程序对应服务器的 根 的话， 它 可能 是为空字符串。
PATH_INFO
　请求URL中路径的剩余部分，指定请求的目标在应用程序内部的虚拟位置。如果请求的目标是应用程序跟并且没有trailing slash的话，可能为空字符串 。
QUERY_STRING
　请求URL中跟在"?"后面的那部分,可能为空或不存在.
CONTENT_TYPE
　HTTP请求中任何 Content-Type 域的内容。
CONTENT_LENGTH
　HTTP请求中任何 Content-Length 域的内容。可能为空或不存在.

```
            transform_ok = []
            for name,value in response_headers:
                if name.lower()=='content-type' and value=='text/plain':
                    transform_ok.append(True)
                    # Strip content-length if present, else it'll be wrong
                    response_headers = [(name,value)
                        for name,value in response_headers
                            if name.lower() <> 'content-length'
                    ]
                    break
            write = start_response(status,response_headers,exc_info)
            if transform_ok:
                def write_latin(data):
                    write(piglatin(data))
                return write_latin
            else:
                return write
        return LatinIter(self.application(environ,start_latin),transform_ok)
# Run foo_app under a Latinator's control, using the example CGI gateway
from foo_app import foo_app
run_with_cgi(Latinator(foo_app))
```

SERVER_NAME, SERVER_PORT

这些变量可以和 SCRIPT_NAME、PATH_INFO 一起组成完整的URL。然而要注意的是，重建请求URL的时候应该优先使用 HTTP_HOST 而非 SERVER_NAME。详细内容请阅读下面的 URL Reconstruction 。SERVER_NAME 和 SERVER_PORT 永远是空字符串，也总是必须存在的。

SERVER_PROTOCOL

客户端发送请求所使用协议的版本。通常是类似 "HTTP/1.0" 或 "HTTP/1.1" 的东西可以被用来判断如何处理请求headers。(既然这个变量表示的是请求中使用的协议，而且和服务器响应时使用的协议无关，也许它应该被叫做 REQUEST_PROTOCOL 。然后，为了保持和CGI的兼容性，我们还是使用已有的名字。)

HTTP_ 变量

对应客户端提供的HTTP请求headers (也就是说名字以 "HTTP_" 开头的变量)。这些变量的存在与否应该和请求中的合适的HTTP header一致。

服务器或gateway 应该 尽可能提供其他可用的CGI变量。另外，如果用了SSL，服务器或gateway也 应该 尽可能提供可用的Apache SSL环境变量 [5] ，比如 HTTPS=on 和SSL_PROTOCOL``。不过要注意，使用了任何上面没有列出的变量的应用程序对不支持相关扩展的服务器来说就有点necessarily non-portable。(比如，不发布文件的web服务器就不能提供一个有意义的 ``DOCUMENT_ROOT 或 PATH_TRANSLATED 。)

一个支持WSGI的服务器或gateway 应该 在描述它们自己的同时说明它们可以提供些什么变量应用程序 应该 对所有他们需要的变量的存在性进行检查，并且在某变量不存在的时候有备用的措施

注意: 不需要的变量 (比如在不需要验证的情况下的 REMOTE_USER) 应该被移出 environ字典。同样注意CGI定义的变量如果存在的话必须是字符串。任何 str 类型以外的CGI变量的存在都是对本规范的违反

除了CGI定义的变量， environ 字典也可以包含任意操作系统的环境变量，并且 必须包含下面这些WSGI定义的变量:

变量 值

wsgi.version 元组 (1,0), 表明WSGI版本 1.0.

wsgi.url_scheme A string representing the "scheme" portion ofthe URL at which the application is beinginvoked. Normally, this will have the value"http" or "https", as appropriate.

wsgi.input An input stream (file-like object) from whichthe HTTP request body can be read. (The serveror gateway may perform reads on-demand asrequested by the application, or it may pre-read the client's request body and buffer itin-memory or on disk, or use any othertechnique for providing such an

详细说明

应用程序对象必须接受两个参数，为了方便说明我们不妨分别命名为 environ 和
gateway必须用这两个参数请求应用程序对象(比如象上面展示的,这样调用 result =

参数 environ 是个字典对象，包含CGI风格的环境变量。这个对象必须是一个pyth
字典对象的模仿)，应用程序可以以任何他愿意的方式修改这个字典， environ 还
[illegible]描述，[illegible]一些服务器特定的扩展变量，通过下面[illegible]高的约定命名。

start_response 参数是一个接受两个必须参数和一个可选参数的可调用者。方便说
,和 exc_info 。应用程序必须用这些参数来请求可调用者 start_response (比如象这

参数 status 是一个形式象"999 Message here"的状态字符串。而 response_headers 参
http响应头。可选的 exc_info 参数会在下面的 `The start_response() Callable`_ 和 E
了错误并希望在浏览器上显示错误的时候才有用。

start_response 可调用者必须返回一个 write(body_data) 可调用者，他接受一个可迭
字符串(注意：提供可调用者 write() 只是为了支持现有框架的必要的输出API，新
Buffering and Streaming 一节。)

当被服务器请求的时候，应用程序对象必须返回一个0或多个可迭代的字符串，这
表，或者应用程序本身是一个生产字符串的函数，或者应用程序是一个类而他的
须总是返回0或多个可迭代的字符串。

服务器必须将产生的字符串以一种无缓冲的方式传送到客户端，每次传完一个字
现自己的缓冲，更多关于应用程序输出必须如何处理的细节请阅读下面的 Buffer

服务器或gateway应该把产生的字符串当字节流对待：特别地，他必须保证没修改
匹配的编码输出(服务器/gateway可能会附加HTTP传送编码，或者为了实现一些h
transmission，更多细节请看下面的 Other HTTP Features)

如果调 len(iterable) 成功，服务器将认为结果是正确的。也就是说，应用程序返回
法，么肯定返回了正确的结果(关于这个方法正常情况下如何被使用的请阅读 Ha

如果应用程序返回的可迭代者有close()方法，则不管该请求是正常结束还是由于
[illegible] This protocol is i
and other common iterables with close() methods.)

（注意：应用程序必须在可迭代者产生第一个字符串之间请求 start_response() 可
[illegible]行,所以服务器不
了）

最后，服务器或gateway不能应用程序返回的可迭代者的任何其他属性，除非是针
wsgi.file_wrapper返回的“file wrapper”（阅读 Optional Platform-Specific File Handli
过PEP 234 iteration APIs才是可以访问的。

environ 变量

environ 字典被用来包含这些在Common Gateway Interface specification [2]_中定义
除非其值是空字符串,这种情况下如果下面没有特别指出的话他们 可能 会被忽略

REQUEST_METHOD

HTTP请求的方式, 比如 "GET" 或者 "POST". 这个不可能是空字符串并且也是必

SCRIPT_NAME

input stream,according to its preference.)

wsgi.errors

An output stream (file-like object) to whicherror output can be written, for the purpose ofrecording program or other errors in astandardized and possibly centralized location.This should be a "text mode" stream; i.e.,applications should use "\n" as a lineending, and assume that it will be converted tothe correct line ending by the server/gateway.

For many servers, wsgi.errors will be theserver's main error log. Alternatively, thismay be sys.stderr, or a log file of somesort. The server's documentation shouldinclude an explanation of how to configure thisor where to find the recorded output. A serveror gateway may supply different error streamsto different applications, if this is desired.

wsgi.multithread This value should evaluate true if theapplication object may be simultaneouslyinvoked by another thread in the same process,and should evaluate false otherwise.

wsgi.multiprocess This value should evaluate true if anequivalent application object may besimultaneously invoked by another process,and should evaluate false otherwise.

wsgi.run_once This value should evaluate true if the serveror gateway expects (but does not guarantee!)that the application will only be invoked thisone time during the life of its containingprocess. Normally, this will only be true fora gateway based on CGI (or something similar).

最后 environ 字典也可以包含服务器定义的变量。这些变量的名字必须是小写字母、数字、点和下划线，并且应该带一个能唯一代表服务器或gateway的前缀。比如， mod_python 可能会定义象这样的一些变量:mod_python.some_variable.

输入和错误流

服务器提供的输入和错误流必须提供以下方法:

方法	流	注解
read(size)	input	1
readline()	input	1,2
readlines(hint)	input	1,3
__iter__()	input	
flush()	errors	4
write(str)	errors	
writelines(seq)	errors	

每个方法的语义如果上面没有特别指出均和Python Library Reference记载的一样:

请求URL中路径的开始部分，对应应用程序对象，这样应用程序就知道它的虚
它 可能 是为空字符串。

PATH_INFO

请求URL中路径的剩余部分，指定请求的目标在应用程序内部的虚拟位置。如
话，可能为空字符串。

QUERY_STRING

请求URL中跟在"?"后面的那部分,可能为空或不存在.

CONTENT_TYPE

HTTP请求中任何 Content-Type 域的内容。

CONTENT_LENGTH

HTTP请求中任何 Content-Length 域的内容。可能为空或不存在.

SERVER_NAME, SERVER_PORT

这些变量可以和 SCRIPT_NAME、PATH_INFO 一起组成完整的URL。然而要
HTTP_HOST 而非 SERVER_NAME 。详细内容请阅读下面的 URL Reconstructio
字符串，也总是必须存在的。

SERVER_PROTOCOL

客户端发送请求所使用协议的版本。通常是类似 "HTTP/1.0" 或 "HTTP/1.1" 的
个变量表示的是请求中使用的协议，而且和服务器响应时使用的协议无关，也许
了保持和CGI的兼容性，我们还是使用已有的名字。)

HTTP_ 变量

对应客户端提供的HTTP请求headers (也就是说名字以 "HTTP_" 开头的变量)。
header一致。

服务器或gateway 应该尽可能提供其他可用的CGI变量。另外，如果用了SSL，刖
SSL 环境变量 [5] ，比如 HTTPS=on 和SSL_PROTOCOL''。不过要注意，使用了
扩展的服务器来说就有点necessarily non-portable。(比如，不发布文件的web服务
或 PATH_TRANSLATED 。)

一个支持WSGI的服务器或gateway 应该 在描述它们自己的同时说明它们可以提
的存在性进行检查，并且在某变量不存在的时候有备用的措施

注意：不需要的变量 (比如在不需要验证的情况下的 REMOTE_USER) 应该被移出
话必须是字符串。任何 str 类型以外的CGI变量的存在都是对本规范的违反

除了CGI定义的变量， environ 字典也可以包含任意操作系统的环境变量，并且
变量 值
wsgi.version (1,0) 表明WSGI版本 1.0
wsgi.url_scheme A string representing the "scheme" portion ofthe URL at which the
the value"http" or "https", as appropriate.
wsgi.input An input stream (file-like object) from whichthe HTTP request body can
demand as requested by the application, or it may pre-read the client's request body and
othertechnique for providing such an input stream,according to its preference.)
wsgi.errors

An output stream (file-like object) to whicherror output can be written, for the purpose
and possibly centralized location.This should be a "text mode" stream; i.e.,applications
be converted tothe correct line ending by the server/gateway.

For many servers, wsgi.errors will be theserver's main error log. Alternatively, thismay
documentation shouldinclude an explanation of how to configure thisor where to find t
different error streamsto different applications, if this is desired.
wsgi.multithread This value should evaluate true if theapplication object may be sim
process,and should evaluate false otherwise.
wsgi.multiprocess This value should evaluate true if anequivalent application object

The server is not required to read past the client's specifiedContent-Length, and is allowed to simulate an end-of-filecondition if the application attempts to read past that point.The application should not attempt to read more data than isspecified by the CONTENT_LENGTH variable.

The optional "size" argument to readline() is not supported,as it may be complex for server authors to implement, and is notoften used in practice.

Note that the hint argument to readlines() is optional forboth caller and implementer. The application is free not tosupply it, and the server or gateway is free to ignore it.

Since the errors stream may not be rewound, servers and gatewaysare free to forward write operations immediately, without buffering.In this case, the flush() method may be a no-op. Portableapplications, however, cannot assume that output is unbufferedor that flush() is a no-op. They must call flush() ifthey need to ensure that output has in fact been written. (Forexample, to minimize intermingling of data from multiple processeswriting to the same error log.)

The methods listed in the table above must be supported by allservers conforming to this specification. Applications conformingto this specification must not use any other methods or attributesof the input or errors objects. In particular, applicationsmust not attempt to close these streams, even if they possessclose() methods.

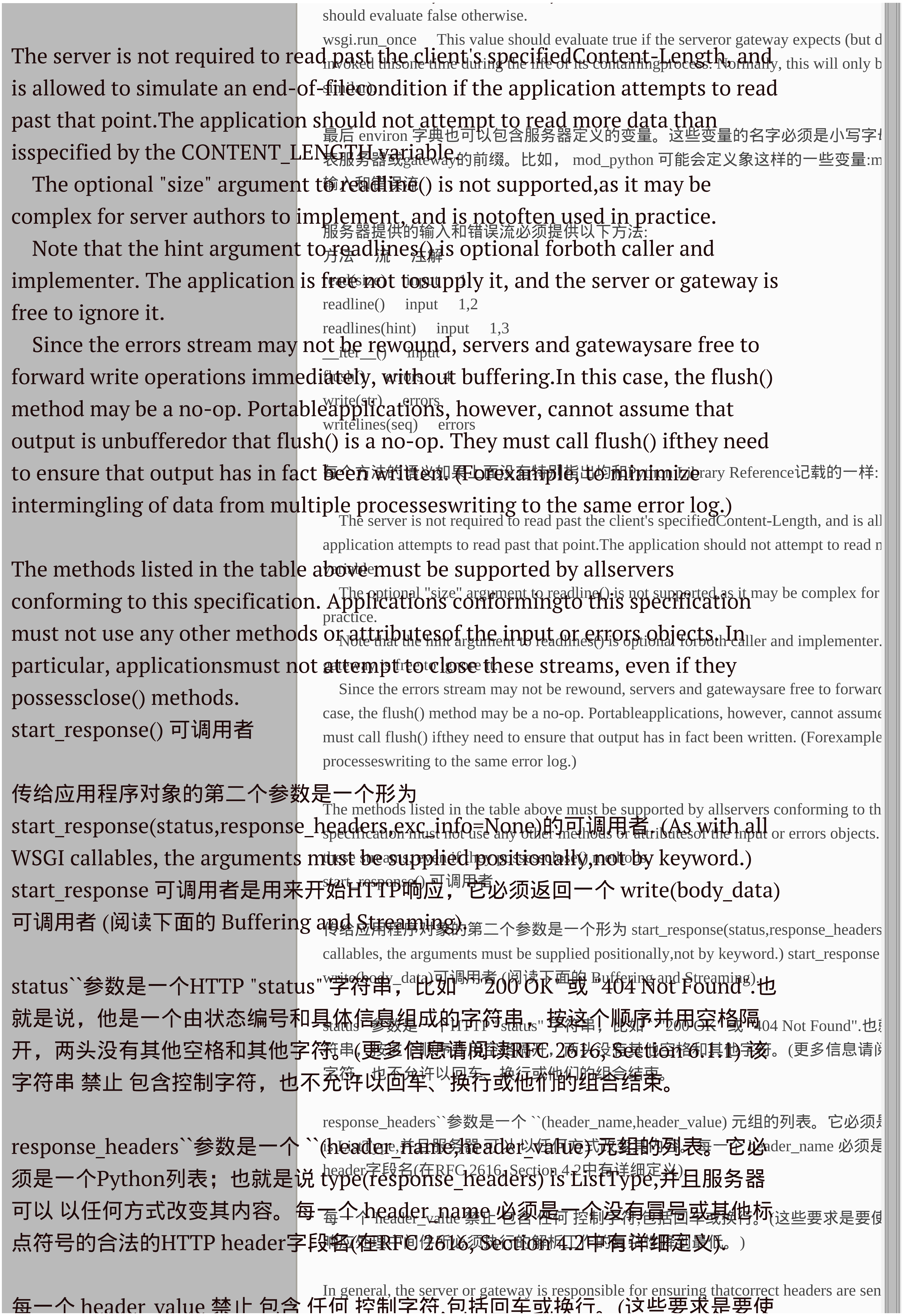
start_response() 可调用者

传给应用程序对象的第二个参数是一个形为 start_response(status,response_headers,exc_info=None)的可调用者. (As with all WSGI callables, the arguments must be supplied positionally,not by keyword.) start_response 可调用者是用来开始HTTP响应，它必须返回一个 write(body_data) 可调用者 (阅读下面的 Buffering and Streaming).

status``参数是一个HTTP "status" 字符串，比如``"200 OK" 或 "404 Not Found".也就是说，他是一个由状态编号和具体信息组成的字符串，按这个顺序并用空格隔开，两头没有其他空格和其他字符。(更多信息请阅读RFC 2616, Section 6.1.1) 该字符串 禁止 包含控制字符，也不允许以回车、换行或他们的组合结束。

response_headers``参数是一个 ``(header_name,header_value) 元组的列表。它必须是一个Python列表；也就是说 type(response_headers) is ListType,并且服务器 可以 以任何方式改变其内容。 每一个 header_name 必须是一个没有冒号或其他标点符号的合法的HTTP header字段名(在RFC 2616, Section 4.2中有详细定义).

每一个 header_value 禁止 包含 任何 控制字符,包括回车或换行。(这些要求是要使



should evaluate false otherwise.

wsgi.run_once This value should evaluate true if the serveror gateway expects (but d

invoked thisone time during the life of its containingprocess. Normally, this will only b

similar.)

最后 environ 字典也可以包含服务器定义的变量。这些变量的名字必须是小写字

表服务器或gateway的前缀。比如， mod_python 可能会定义象这样的一些变量:m

输入和错误流

服务器提供的输入和错误流必须提供以下方法:

方法	流	注解
read(size)	input	1
readline()	input	1,2
readlines(hint)	input	1,3
__iter__()	input	
flush()	errors	4
write(str)	errors	
writelines(seq)	errors	

每个方法的语义如果上面没有特别指出均和Python Library Reference记载的一样:

The server is not required to read past the client's specifiedContent-Length, and is all

application attempts to read past that point.The application should not attempt to read m

variable.

The optional "size" argument to readline() is not supported,as it may be complex for

practice.

Note that the hint argument to readlines() is optional forboth caller and implementer.

gateway is free to ignore it.

Since the errors stream may not be rewound, servers and gatewaysare free to forward

case, the flush() method may be a no-op. Portableapplications, however, cannot assume

must call flush() ifthey need to ensure that output has in fact been written. (Forexample

processeswriting to the same error log.)

The methods listed in the table above must be supported by allservers conforming to th

specification must not use any other methods or attributesof the input or errors objects.

these streams, even if they possessclose() methods.

start_response() 可调用者

传给应用程序对象的第二个参数是一个形为 start_response(status,response_headers

callables, the arguments must be supplied positionally,not by keyword.) start_response

write(body_data)可调用者 (阅读下面的 Buffering and Streaming).

status``参数是一个HTTP "status" 字符串，比如``"200 OK" 或 "404 Not Found".也

符串，按这个顺序并用空格隔开，两头没有其他空格和其他字符。(更多信息请阅

字符，也不允许以回车、换行或他们的组合结束。

response_headers``参数是一个 ``(header_name,header_value) 元组的列表。它必须是

is ListType,并且服务器 可以 以任何方式改变其内容。每一个 header_name 必须是

header字段名(在RFC 2616, Section 4.2中有详细定义).

每一个 header_value 禁止 包含 任何 控制字符,包括回车或换行。(这些要求是要使

所必须执行的解析工作的复杂性降到最低。)

In general, the server or gateway is responsible for ensuring thatcorrect headers are sen

得那些必须检查或修改响应头的服务器、gateway、响应处理中间件所必须执行的解析工作的复杂性降到最低。）

In general, the server or gateway is responsible for ensuring thatcorrect headers are sent to the client: if the application omitsa header required by HTTP (or other relevant specifications that are ineffect), the server or gateway must add it. For example, the HTTPDate: and Server: headers would normally be supplied by theserver or gateway.

(A reminder for server/gateway authors: HTTP header names arecase-insensitive, so be sure to take that into consideration whenexamining application-supplied headers!)

Applications and middleware are forbidden from using HTTP/1.1"hop-by-hop" features or headers, any equivalent features in HTTP/1.0,or any headers that would affect the persistence of the client'sconnection to the web server. These features are theexclusive province of the actual web server, and a server or gatewayshould consider it a fatal error for an application to attemptsending them, and raise an error if they are supplied tostart_response(). (For more specifics on "hop-by-hop" features andheaders, please see the Other HTTP Features section below.)

The start_response callable must not actually transmit theresponse headers. Instead, it must store them for the server orgateway to transmit only after the first iteration of theapplication return value that yields a non-empty string, or uponthe application's first invocation of the write() callable. Inother words, response headers must not be sent until there is actualbody data available, or until the application's returned iterable isexhausted. (The only possible exception to this rule is if theresponse headers explicitly include a Content-Length of zero.)

This delaying of response header transmission is to ensure that bufferedand asynchronous applications can replace their originally intendedoutput with error output, up until the last possible moment. Forexample, the application may need to change the response status from"200 OK" to "500 Internal Error", if an error occurs while the body isbeing generated within an application buffer.

The exc_info argument, if supplied, must be a Pythonsys.exc_info() tuple. This argument should be supplied by theapplication only if start_response is being called by an errorhandler. If exc_info is supplied, and no HTTP headers have beenoutput yet, start_response should replace the currently-storedHTTP

by HTTP (or other relevant specifications that are ineffect), the server or gateway must
headers would normally be supplied by theserver or gateway.

(A reminder for server/gateway authors: HTTP header names arecase-insensitive, so be
application-supplied headers!)

Applications and middleware are forbidden from using HTTP/1.1"hop-by-hop" feature
headers that would affect the persistence of the client'sconnection to the web server. Th
server, and a server or gatewayshould consider it a fatal error for an application to atter
tostart_response(). (For more specifics on "hop-by-hop" features andheaders, please see

The start_response callable must not actually transmit theresponse headers. Instead, it r
after the first iteration of theapplication return value that yields a non-empty string, or u
callable. Inother words, response headers must not be sent until there is actualbody data
isexhausted. (The only possible exception to this rule is if theresponse headers explicitl

This delaying of response header transmission is to ensure that bufferedand asynchrono
intendedoutput with error output, up until the last possible moment. Forexample, the ap
from"200 OK" to "500 Internal Error", if an error occurs while the body isbeing genera

The exc_info argument, if supplied, must be a Pythonsys.exc_info() tuple. This argume
start_response is being called by an errorhandler. If exc_info is supplied, and no HTTP
replace the currently-storedHTTP response headers with the newly-supplied ones, thus
output when an error hasoccurred.

However, if exc_info is provided, and the HTTP headers have alreadybeen sent, start_r
exc_info tuple. That is:

raise exc_info[0],exc_info[1],exc_info[2]

This will re-raise the exception trapped by the application, and inprinciple should abort
attempt error output to the browser once the HTTPheaders have already been sent.) The
start_response, if it calledstart_response with exc_info. Instead, it should allowsuch exc
See Error Handling below, for more details.

The application may call start_response more than once, if andonly if the exc_info argu
call start_response without the exc_infoargument if start_response has already been cal
the example CGI gateway above for an illustration of the correct logic.)

Note: servers, gateways, or middleware implementing start_responseshould ensure that
the duration of the function's execution, to avoidcreating a circular reference through th
this is something like:

def start_response(status,response_headers,exc_info=None):

 if exc_info:

  try:

   # do stuff w/exc_info here

  finally:

beenoutput yet, start_response should replace the currently-storedHTTP response headers with the newly-supplied ones, thus allowing theapplication to "change its mind" about the output when an error hasoccurred.

However, if exc_info is provided, and the HTTP headers have alreadybeen sent, start_response must raise an error, and shouldraise the exc_info tuple. That is:

```
raise exc_info[0],exc_info[1],exc_info[2]
```

This will re-raise the exception trapped by the application, and inprinciple should abort the application. (It is not safe for theapplication to attempt error output to the browser once the HTTPheaders have already been sent.) The application must not trapany exceptions raised by start_response, if it calledstart_response with exc_info. Instead, it should allowsuch exceptions to propagate back to the server or gateway. SeeError Handling below, for more details.

The application may call start_response more than once, if andonly if the exc_info argument is provided. More precisely, it isa fatal error to call start_response without the exc_infoargument if start_response has already been called within thecurrent invocation of the application. (See the example CGIgateway above for an illustration of the correct logic.)

Note: servers, gateways, or middleware implementing start_responseshould ensure that no reference is held to the exc_infoparameter beyond the duration of the function's execution, to avoidcreating a circular reference through the traceback and framesinvolved. The simplest way to do this is something like:

```
def start_response(status,response_headers,exc_info=None):

if exc_info:

try:

# do stuff w/exc_info here

finally:

exc_info = None    # Avoid circular ref.
```

```
exc_info = None    # Avoid circular ref.
```

The example CGI gateway provides another illustration of thistechnique.

Handling the Content-Length Header

If the application does not supply a Content-Length header, a server or gateway may ch simplest of these is to close the client connection whenthe response is completed.

Under some circumstances, however, the server or gateway may beable to either genera close the client connection. If the applicationdoes not call the write() callable, and retur automatically determineContent-Length by taking the length of the first string yieldedb

And, if the server and client both support HTTP/1.1 "chunkedencoding" [3], then the se write() call or string yielded by the iterable, thus generating a Content-Length header fo connection alive, if it wishesto do so. Note that the server must comply fully with RFC other strategies fordealing with the absence of Content-Length.

(Note: applications and middleware must not apply any kind ofTransfer-Encoding to th hop" operations, these encodings are the province of theactual web server/gateway. See

Buffering and Streaming

Generally speaking, applications will achieve the best throughputby buffering their (mo common approach in existing frameworks such asZope: the output is buffered in a Strin with the response headers.

The corresponding approach in WSGI is for the application to simplyreturn a single-ele body as a single string. This is the recommended approachfor the vast majority of appli easily fits in memory.

For large files, however, or for specialized uses of HTTP streaming(such as multipart " in smaller blocks (e.g. to avoid loading a large file intomemory). It's also sometimes th produce, but it would be useful to send ahead theportion of the response that precedes i

In these cases, applications will usually return an iterator (oftena generator-iterator) tha blocks may be broken to coincide with mulitpartboundaries (for "server push"), or just block of an on-disk file).

WSGI servers, gateways, and middleware must not delay thetransmission of any block; guarantee that they will continuetransmission even while the application is producing it provide this guarantee in one ofthree ways:

- Send the entire block to the operating system (and requestthat any O/S buffers be flu
- Use a different thread to ensure that the block continuesto be transmitted while the a
- (Middleware only) send the entire block to its parentgateway/server

By providing this guarantee, WSGI allows applications to ensurethat transmission will data. This is critical for proper functioningof e.g. multipart "server push" streaming, wh transmitted in full to the client.

Middleware Handling of Block Boundaries

In order to better support asynchronous applications and servers,middleware componen from an application iterable. If the middlewareneeds to accumulate more data from the yield an empty string.

The example CGI gateway provides another illustration of thistechnique.
Handling the Content-Length Header

If the application does not supply a Content-Length header, aserver or gateway may choose one of several approaches to handlingit. The simplest of these is to close the client connection whenthe response is completed.

Under some circumstances, however, the server or gateway may beable to either generate a Content-Length header, or at leastavoid the need to close the client connection. If the applicationdoes not call the write() callable, and returns an iterablewhose len() is 1, then the server can automatically determineContent-Length by taking the length of the first string yieldedby the iterable.

And, if the server and client both support HTTP/1.1 "chunkedencoding" [3], then the server may use chunked encoding to senda chunk for each write() call or string yielded by the iterable,thus generating a Content-Length header for each chunk. Thisallows the server to keep the client connection alive, if it wishesto do so. Note that the server must comply fully with RFC 2616when doing this, or else fall back to one of the other strategies fordealing with the absence of Content-Length.

(Note: applications and middleware must not apply any kind ofTransfer-Encoding to their output, such as chunking or gzipping;as "hop-by-hop" operations, these encodings are the province of theactual web server/gateway. See Other HTTP Features below, formore details.)
Buffering and Streaming

Generally speaking, applications will achieve the best throughputby buffering their (modestly-sized) output and sending it all atonce. This is a common approach in existing frameworks such asZope: the output is buffered in a StringIO or similar object, thentransmitted all at once, along with the response headers.

The corresponding approach in WSGI is for the application to simplyreturn a single-element iterable (such as a list) containing theresponse body as a single string. This is the recommended approachfor the vast majority of application functions, that renderHTML pages whose text easily fits in memory.

For large files, however, or for specialized uses of HTTP streaming(such as multipart "server push"), an application may need to provideoutput in smaller

To put this requirement another way, a middleware component mustyield at least one v
If the middleware cannot yield any other value,it must yield an empty string.

This requirement ensures that asynchronous applications and serverscan conspire to red
given number of application instances simultaneously.

Note also that this requirement means that middleware mustreturn an iterable as soon a
forbidden for middleware to use thewrite() callable to transmit data that is yielded by ar
parentserver's write() callable to transmit data that theunderlying application sent using
The write() Callable

Some existing application framework APIs support unbufferedoutput in a different mar
function or method of some kind to write an unbuffered block of data, or else they provi
to flush the buffer.

Unfortunately, such APIs cannot be implemented in terms ofWSGI's "iterable" applicat
mechanisms are used.

Therefore, to allow these frameworks to continue using animperative API, WSGI inclu
start_response callable.

New WSGI applications and frameworks should not use thewrite() callable if it is possi
hack to support imperativestreaming APIs. In general, applications should produce thei
possiblefor web servers to interleave other tasks in the same Python thread,potentially p

The write() callable is returned by the start_response()callable, and it accepts a single p
response body, that is treated exactlyas though it had been yielded by the output iterabl
guarantee that thepassed-in string was either completely sent to the client, orthat it is bu
upward.

An application must return an iterable object, even if ituses write() to produce all or par
empty (i.e. yield no non-emptystrings), but if it does yield non-empty strings, that outpu
(i.e., it must be sent or queued immediately). Applications must not invokewrite() from
yielded by the iterable are transmitted after all stringspassed to write() have been sent to
Unicode Issues

HTTP does not directly support Unicode, and neither does this interface. All encoding/d
passed to or from the server must be standard Python bytestrings, not Unicode objects.
object is required, is undefined.

Note also that strings passed to start_response() as a status or as response headers must
must either be ISO-8859-1 characters, or use RFC 2047MIME encoding.

On Python platforms where the str or StringType type is infact Unicode-based (e.g. Jyt
to in this specification must contain only code points representable in ISO-8859-1 encod
for an application tosupply strings containing any other Unicode character or code poin
to an application containing any other Unicode characters.

Again, all strings referred to in this specification must beof type str or StringType, and
if a given platform allowsfor more than 8 bits per character in str/StringType objects,or
to inthis specification as a "string".

Error Handling

blocks (e.g. to avoid loading a large file intomemory). It's also sometimes the case that part of a response maybe time-consuming to produce, but it would be useful to send ahead theportion of the response that precedes it.

In these cases, applications will usually return an iterator (oftena generator-iterator) that produces the output in a block-by-blockfashion. These blocks may be broken to coincide with mulitpartboundaries (for "server push"), or just before time-consumingtasks (such as reading another block of an on-disk file).

WSGI servers, gateways, and middleware must not delay thetransmission of any block; they must either fully transmitthe block to the client, or guarantee that they will continuetransmission even while the application is producing its next block.A server/gateway or middleware may provide this guarantee in one ofthree ways:

Send the entire block to the operating system (and requestthat any O/S buffers be flushed) before returning controlto the application, OR

Use a different thread to ensure that the block continuesto be transmitted while the application produces the nextblock.

(Middleware only) send the entire block to its parentgateway/server

By providing this guarantee, WSGI allows applications to ensurethat transmission will not become stalled at an arbitrary pointin their output data. This is critical for proper functioningof e.g. multipart "server push" streaming, where data betweenmultipart boundaries should be transmitted in full to the client.

Middleware Handling of Block Boundaries

In order to better support asynchronous applications and servers,middleware components must not block iteration waiting formultiple values from an application iterable. If the middlewareneeds to accumulate more data from the application before it canproduce any output, it must yield an empty string.

To put this requirement another way, a middleware component mustyield at least one value each time its underlying applicationyields a value. If the middleware cannot yield any other value,it must yield an empty string.

This requirement ensures that asynchronous applications and serverscan conspire to reduce the number of threads that are requiredto run a given number of application instances simultaneously.

In general, applications should try to trap their own, internal errors, and display a helpfu
decide what "helpful" means in this context.)

However, to display such a message, the application must not haveactually sent any dat
response. WSGI therefore provides a mechanism to either allow theapplication to send
exc_info argument to start_response. Here is an exampleof its use:

```
try:
    # regular application code here
    status = "200 Froody"
    response_headers = [("content-type","text/plain")]
    start_response(status, response_headers)
    return ["normal body goes here"]
except:
    # XXX should trap runtime issues like MemoryError, KeyboardInterrupt
    # in a separate handler before this bare 'except:'...
    status = "500 Oops"
    response_headers = [("content-type","text/plain")]
    start_response(status, response_headers, sys.exc_info())
    return ["error body goes here"]
```

If no output has been written when an exception occurs, the call tostart_response will r
body to be sent to the browser. However, if any outputhas already been sent to the brov
This exception should not be trapped bythe application, and so the application will abor
exception and abort the response.

Servers should trap and log any exception that aborts anapplication or the iteration of it
written to the browser when an application error occurs, the server or gateway may atter
sent headers indicate atext/* content type that the server knows how to modify cleanly.

Some middleware may wish to provide additional exception handlingservices, or interc
cases, middleware may choose to not re-raise the exc_info supplied to start_response, bu
simply return without an exception after storing the supplied arguments. This will then c
invoke write()), allowing the middlewareto capture and modify the error output. These

- Always provide exc_info when beginning an error response
- Never trap errors raised by start_response when exc_info isbeing provided

HTTP 1.1 Expect/Continue

Servers and gateways that implement HTTP 1.1 must providetransparent support for H

Note also that this requirement means that middleware mustreturn an iterable as soon as its underlying application returnsan iterable. It is also forbidden for middleware to use thewrite() callable to transmit data that is yielded by anunderlying application. Middleware may only use their parentserver's write() callable to transmit data that theunderlying application sent using a middleware-provided write()callable.

The write() Callable

Some existing application framework APIs support unbufferedoutput in a different manner than WSGI. Specifically, theyprovide a "write" function or method of some kind to writean unbuffered block of data, or else they provide a buffered"write" function and a "flush" mechanism to flush the buffer.

Unfortunately, such APIs cannot be implemented in terms ofWSGI's "iterable" application return value, unless threadsor other special mechanisms are used.

Therefore, to allow these frameworks to continue using animperative API, WSGI includes a special write() callable,returned by the start_response callable.

New WSGI applications and frameworks should not use thewrite() callable if it is possible to avoid doing so. Thewrite() callable is strictly a hack to support imperativestreaming APIs. In general, applications should produce theiroutput via their returned iterable, as this makes it possiblefor web servers to interleave other tasks in the same Python thread,potentially providing better throughput for the server as a whole.

The write() callable is returned by the start_response()callable, and it accepts a single parameter: a string to bewritten as part of the HTTP response body, that is treated exactlyas though it had been yielded by the output iterable. In otherwords, before write() returns, it must guarantee that thepassed-in string was either completely sent to the client, orthat it is buffered for transmission while the applicationproceeds onward.

An application must return an iterable object, even if ituses write() to produce all or part of its response body.The returned iterable may be empty (i.e. yield no non-emptystrings), but if it does yield non-empty strings, that outputmust be treated normally by the server or gateway (i.e., it must besent or queued immediately). Applications must not invokewrite() from within their return iterable, and therefore anystrings yielded by the iterable are transmitted after all stringspassed to write() have been sent to the client.

Unicode Issues

done in any of several ways:

Respond to requests containing an Expect: 100-continue requestwith an immediate "

Proceed with the request normally, but provide the applicationwith a wsgi.input strea the application first attempts to read from theinput stream. The read request must then r

Wait until the client decides that the server does not supportexpect/continue, and sen is not recommended.)

Note that these behavior restrictions do not apply for HTTP 1.0requests, or for requests information on HTTP 1.1 Expect/Continue, see RFC2616, sections 8.2.3 and 10.1.1.

Other HTTP Features

In general, servers and gateways should "play dumb" and allow theapplication complet that do not alter the effective semantics of the application'sresponse. It is always possib components to supply additional features, so server/gatewaydevelopers should be conse should consider itself to be like an HTTP "gateway server",with the application being a the definition of these terms.)

However, because WSGI servers and applications do not communicate viaHTTP, what WSGIinternal communications. WSGI applications must not generate any"hop-by-hop' wouldrequire them to generate such headers, or rely on the content ofany incoming "ho servers must handle any supported inbound "hop-by-hop" headerson their own, such as chunked encoding if applicable.

Applying these principles to a variety of HTTP features, it should beclear that a server If-Modified-Since request headers and theLast-Modified and ETag response headers. H should perform itsown cache validation if it wants to support that feature, sincethe serv

Similarly, a server may re-encode or transport-encode anapplication's response, but the own, and must not apply atransport encoding. A server may transmit byte ranges of the theapplication doesn't natively support byte ranges. Again, however,the application sho

Note that these restrictions on applications do not necessarily meanthat every applicatio HTTPfeatures can be partially or fully implemented by middlewarecomponents, thus fr fromimplementing the same features over and over again.

Thread Support

Thread support, or lack thereof, is also server-dependent.Servers that can run multiple r running an application in a single-threadedfashion, so that applications or frameworks t

Implementation/Application Notes

Server Extension APIs

Some server authors may wish to expose more advanced APIs, thatapplication or frame example, a gateway based on mod_python might wish to exposepart of the Apache API

In the simplest case, this requires nothing more than defining anenviron variable, such a possible presence of middleware can make this difficult.For example, an API that offer environ variables, might return different data ifenviron has been modified by middlewa

In general, any extension API that duplicates, supplants, or bypassessome portion of W incompatiblewith middleware components. Server/gateway developers should notassur frameworkdevelopers specifically intend to organize or reorganize theirframeworks to

So, to provide maximum compatibility, servers and gateways thatprovide extension AP

HTTP does not directly support Unicode, and neither does thisinterface. All encoding/decoding must be handled by the application;all strings passed to or from the server must be standard Python bytestrings, not Unicode objects. The result of using a Unicode objectwhere a string object is required, is undefined.

Note also that strings passed to start_response() as a status oras response headers must follow RFC 2616 with respect to encoding.That is, they must either be ISO-8859-1 characters, or use RFC 2047MIME encoding.

On Python platforms where the str or StringType type is infact Unicode-based (e.g. Jython, IronPython, Python 3000, etc.), all"strings" referred to in this specification must contain onlycode points representable in ISO-8859-1 encoding (\u0000 through\u00FF, inclusive). It is a fatal error for an application tosupply strings containing any other Unicode character or code point.Similarly, servers and gateways must not supplystrings to an application containing any other Unicode characters.

Again, all strings referred to in this specification must beof type str or StringType, and must not be of typeunicode or UnicodeType. And, even if a given platform allowsfor more than 8 bits per character in str/StringType objects,only the lower 8 bits may be used, for any value referred to inthis specification as a "string".

Error Handling

In general, applications should try to trap their own, internalerrors, and display a helpful message in the browser. (It is upto the application to decide what "helpful" means in this context.)

However, to display such a message, the application must not haveactually sent any data to the browser yet, or else it risks corruptingthe response. WSGI therefore provides a mechanism to either allow theapplication to send its error message, or be automatically aborted:the exc_info argument to start_response. Here is an exampleof its use:

```
try:

# regular application code here

status = "200 Froody"
```

those APIs so that they are invoked using the portion of theAPI that they replace. For e headers must require the application to pass in its currentenviron, so that the server/gate API have not been altered by middleware. If theextension API cannot guarantee that it HTTP headers, it must refuse service to the application, e.g. by raising an error, returnin appropriate to the API.

Similarly, if an extension API provides an alternate means of writingresponse data or h passed in, before the application can obtain theextended service. If the object passed in supplied to the application, it cannotguarantee correct operation and must refuse to prov

These guidelines also apply to middleware that adds information suchas parsed cookies Specifically, such middleware should provide thesefeatures as functions which operate environ. This helps ensure that informationis calculated from environ after any middlev modifications.

It is very important that these "safe extension" rules be followed byboth server/gateway in which middleware developers are forced to delete any and allextension APIs from en by applications using those extensions!

Application Configuration

This specification does not define how a server selects or obtains anapplication to invol server-specific matters. It is expected that server/gatewayauthors will document how to object, and with what options (such asthreading options).

Framework authors, on the other hand, should document how to create anapplication ol Theuser, who has chosen both the server and the application framework,must connect t andthe server now have a common interface, this should be merely amechanical matter server/framework pair.

Finally, some applications, frameworks, and middleware may wish touse the environ di Servers and gateways should support this by allowingan application's deployer to speci simplest case, this support can consist merely ofcopying all operating system-supplied dictionary, since the deployer inprinciple can configure these externally to the server, o server's configurationfiles.

Applications should try to keep such required variables to aminimum, since not all serv even in the worst case, persons deploying anapplication can create a script to supply the

```
from the_app import application

def new_app(environ,start_response):

    environ['the_app.configval1'] = 'something'

    return application(environ,start_response)
```

But, most existing applications and frameworks will probably only needa single config their application or framework-specific configuration file(s). (Ofcourse, applications sh read it upon each invocation.)

URL Reconstruction

```
status = "200 Froody"

response_headers = [("content-type","text/plain")]

start_response(status, response_headers)

return ["normal body goes here"]

except:

# XXX should trap runtime issues like MemoryError, KeyboardInterrupt

#     in a separate handler before this bare 'except:'...

status = "500 Oops"

response_headers = [("content-type","text/plain")]

start_response(status, response_headers, sys.exc_info())

return ["error body goes here"]
```

If no output has been written when an exception occurs, the call tostart_response will return normally, and the application willreturn an error body to be sent to the browser. However, if any outputhas already been sent to the browser, start_response will reraisethe provided exception. This exception should not be trapped bythe application, and so the application will abort. The server orgateway can then trap this (fatal) exception and abort the response.

Servers should trap and log any exception that aborts anapplication or the iteration of its return value. If a partialresponse has already been written to the browser when an applicationerror occurs, the server or gateway may attempt to add an errormessage to the output, if the already-sent headers indicate atext/* content type that the server knows how to modify cleanly.

Some middleware may wish to provide additional exception handlingservices, or intercept and replace application error messages. Insuch cases, middleware may choose to not re-raise the exc_infosupplied to start_response, but instead raise a middleware-specificexception, or simply return without an exception after storing thesupplied arguments. This will then cause the application to returnits error body iterable (or invoke write()), allowing the middlewareto capture and modify the error output. These techniques will work aslong as application

If an application wishes to reconstruct a request's complete URL, itmay do so using the

```
from urllib import quote

url = environ['wsgi.url_scheme']+'://'


if environ.get('HTTP_HOST'):

   url += environ['HTTP_HOST']

else:

   url += environ['SERVER_NAME']



   if environ['wsgi.url_scheme'] == 'https':

      if environ['SERVER_PORT'] != '443':

        url += ':' + environ['SERVER_PORT']

   else:

      if environ['SERVER_PORT'] != '80':

        url += ':' + environ['SERVER_PORT']


url += quote(environ.get('SCRIPT_NAME',''))

url += quote(environ.get('PATH_INFO',''))


if environ.get('QUERY_STRING'):

   url += '?' + environ['QUERY_STRING']
```

Note that such a reconstructed URL may not be precisely the same URIas requested by modified the client's originally requested URL to place it in a canonical form.

Supporting Older (<2.2) Versions of Python

Some servers, gateways, or applications may wish to support older(<2.2) versions of Py platform, since as of this writing a production-ready version of Python 2.2 is not yet ava

For servers and gateways, this is relatively straightforward:servers and gateways target themselves to using only a standard "for" loop toiterate over any iterable returned by ar compatibility with both the pre-2.2 iterator protocol (discussed further below) and "toda

(Note that this technique necessarily applies only to servers,gateways, or middleware th iterator protocol(s) correctly from other languages isoutside the scope of this PEP.)

modify the error output. These techniques will work aslong as application authors:

Always provide exc_info when beginning an error response
Never trap errors raised by start_response when exc_info isbeing provided

HTTP 1.1 Expect/Continue

Servers and gateways that implement HTTP 1.1 must providetransparent support for HTTP 1.1's "expect/continue" mechanism. Thismay be done in any of several ways:

Respond to requests containing an Expect: 100-continue requestwith an immediate "100 Continue" response, and proceed normally.
Proceed with the request normally, but provide the applicationwith a wsgi.input stream that will send the "100 Continue"response if/when the application first attempts to read from theinput stream. The read request must then remain blocked until theclient responds.
Wait until the client decides that the server does not supportexpect/continue, and sends the request body on its own. (Thisis suboptimal, and is not recommended.)

Note that these behavior restrictions do not apply for HTTP 1.0requests, or for requests that are not directed to an applicationobject. For more information on HTTP 1.1 Expect/Continue, see RFC2616, sections 8.2.3 and 10.1.1.
Other HTTP Features

In general, servers and gateways should "play dumb" and allow theapplication complete control over its output. They should only makechanges that do not alter the effective semantics of the application'sresponse. It is always possible for the application developer to addmiddleware components to supply additional features, so server/gatewaydevelopers should be conservative in their implementation. In a sense,a server should consider itself to be like an HTTP "gateway server",with the application being an HTTP "origin server". (See RFC 2616,section 1.3, for the definition of these terms.)

However, because WSGI servers and applications do not communicate viaHTTP, what RFC 2616 calls "hop-by-hop" headers do not apply to WSGIinternal communications. WSGI applications must not generate any"hop-by-hop" headers [4], attempt to use HTTP features that wouldrequire them to generate such headers, or rely on the content ofany incoming "hop-by-hop" headers in

For applications, supporting pre-2.2 versions of Python is slightlymore complex:

You may not return a file object and expect it to work as an iterable,since before Pyt
do this anyway, because it will peform quite poorly mostof the time!) Use wsgi.file_wr
Optional Platform-Specific File Handlingfor more on wsgi.file_wrapper, and an examp
If you return a custom iterable, it must implement the pre-2.2iterator protocol. That i
key, and raises IndexError when exhausted.(Note that built-in sequence types are also a

Finally, middleware that wishes to support pre-2.2 versions of Python,and iterates over
both), must follow the appropriate recommendations above.

(Note: It should go without saying that to support pre-2.2 versionsof Python, any serve
only language features available in the target version, use1 and 0 instead of True and F
Optional Platform-Specific File Handling

Some operating environments provide special high-performance file-transmission facil
gateways may expose this functionality via an optionalwsgi.file_wrapper key in the env
convert a file or file-like objectinto an iterable that it then returns, e.g.:

```
if 'wsgi.file_wrapper' in environ:

  return environ['wsgi.file_wrapper'](filelike, block_size)

else:

  return iter(lambda: filelike.read(block_size), '')
```

If the server or gateway supplies wsgi.file_wrapper, it must bea callable that accepts on
positional parameter. The first parameter is the file-likeobject to be sent, and the second
the server/gateway need not use). Thecallable must return an iterable object, and must r
server/gateway actuallyreceives the iterable as a return value from the application.(To c
to interpretor override the response data.)

To be considered "file-like", the object supplied by the applicationmust have a read() m
a close() method, and if so, the iterable returnedby wsgi.file_wrapper must have a close
close() method. If the "file-like" object has any other methods or attributes with namesm
fileno()), the wsgi.file_wrapper may assume that these methods orattributes have the sa

The actual implementation of any platform-specific file handlingmust occur after the ap
see if a wrapper object was returned. (Again,because of the presence of middleware, er
wrapper created will actually be used.)

Apart from the handling of close(), the semantics of returning afile wrapper from the ap
returned iter(filelike.read, ''). In other words,transmission should begin at the current po
begins, and continue until the end isreached.

Of course, platform-specific file transmission APIs don't usuallyaccept arbitrary "file-li
introspect the supplied object forthings such as a fileno() (Unix-like OSes) or ajava.nio
file-like object is suitable for use with the platform-specificAPI it supports.

Note that even if the object is not suitable for the platform API,the wsgi.file_wrapper m
so that applications using file wrappersare portable across platforms. Here's a simple pl

the environ dictionary.WSGI servers must handle any supported inbound "hop-by-hop" headerson their own, such as by decoding any inbound Transfer-Encoding,including chunked encoding if applicable.

Applying these principles to a variety of HTTP features, it should beclear that a server may handle cache validation via theIf-None-Match and If-Modified-Since request headers and theLast-Modified and ETag response headers. However, it isnot required to do this, and the application should perform itsown cache validation if it wants to support that feature, sincethe server/gateway is not required to do such validation.

Similarly, a server may re-encode or transport-encode anapplication's response, but the application should use asuitable content encoding on its own, and must not apply atransport encoding. A server may transmit byte ranges of theapplication's response if requested by the client, and theapplication doesn't natively support byte ranges. Again, however,the application should perform this function on its own if desired.

Note that these restrictions on applications do not necessarily meanthat every application must reimplement every HTTP feature; many HTTPfeatures can be partially or fully implemented by middlewarecomponents, thus freeing both server and application authors fromimplementing the same features over and over again.

Thread Support

Thread support, or lack thereof, is also server-dependent.Servers that can run multiple requests in parallel, should alsoprovide the option of running an application in a single-threadedfashion, so that applications or frameworks that are not thread-safemay still be used with that server.

Implementation/Application Notes

Server Extension APIs

Some server authors may wish to expose more advanced APIs, thatapplication or framework authors can use for specialized purposes.For example, a gateway based on mod_python might wish to exposepart of the Apache API as a WSGI extension.

In the simplest case, this requires nothing more than defining anenviron variable, such as mod_python.some_api. But, in manycases, the possible presence of middleware can make this difficult.For example, an API that offers access to the same HTTP headers thatare found in environ variables, might

2.2) and new Pythons alike:

```
class FileWrapper:

    def __init__(self, filelike, blksize=8192):

        self.filelike = filelike

        self.blksize = blksize

        if hasattr(filelike,'close'):

            self.close = filelike.close

    def __getitem__(self,key):

        data = self.filelike.read(self.blksize)

        if data:

            return data

        raise IndexError
```

and here is a snippet from a server/gateway that uses it to provideaccess to a platform-s

```
environ['wsgi.file_wrapper'] = FileWrapper

result = application(environ, start_response)

try:

    if isinstance(result,FileWrapper):

        # check if result.filelike is usable w/platform-specific

        # API, and if so, use that API to transmit the result.

        # If not, fall through to normal iterable handling

        # loop below.

    for data in result:
```

return different data ifenviron has been modified by middleware.

In general, any extension API that duplicates, supplants, or bypassessome portion of WSGI functionality runs the risk of being incompatiblewith middleware components. Server/gateway developers should notassume that nobody will use middleware, because some frameworkdevelopers specifically intend to organize or reorganize theirframeworks to function almost entirely as middleware of various kinds.

So, to provide maximum compatibility, servers and gateways thatprovide extension APIs that replace some WSGI functionality, mustdesign those APIs so that they are invoked using the portion of theAPI that they replace. For example, an extension API to access HTTPrequest headers must require the application to pass in its currentenviron, so that the server/gateway may verify that HTTP headersaccessible via the API have not been altered by middleware. If theextension API cannot guarantee that it will always agree withenviron about the contents of HTTP headers, it must refuse serviceto the application, e.g. by raising an error, returning Noneinstead of a header collection, or whatever is appropriate to the API.

Similarly, if an extension API provides an alternate means of writingresponse data or headers, it should require the start_responsecallable to be passed in, before the application can obtain theextended service. If the object passed in is not the same one thatthe server/gateway originally supplied to the application, it cannotguarantee correct operation and must refuse to provide the extendedservice to the application.

These guidelines also apply to middleware that adds information suchas parsed cookies, form variables, sessions, and the like toenviron. Specifically, such middleware should provide thesefeatures as functions which operate on environ, rather than simplystuffing values into environ. This helps ensure that informationis calculated from environ after any middleware has done any URLrewrites or other environ modifications.

It is very important that these "safe extension" rules be followed byboth server/gateway and middleware developers, in order to avoid afuture in which middleware developers are forced to delete any and allextension APIs from environ to ensure that their mediation isn'tbeing bypassed by applications using those extensions!

Application Configuration

```
    # etc.

finally:

    if hasattr(result,'close'):

        result.close()
```

Questions and Answers

Why must environ be a dictionary? What's wrong with using asubclass?

The rationale for requiring a dictionary is to maximize portabilitybetween servers. Th
dictionary's methods as being the standard and portableinterface. In practice, however,
their needs, and thus framework authors will come to expect the fullset of dictionary fe
than not. But, if some serverchooses not to use a dictionary, then there will beinteropera
tospec. Therefore, making a dictionary mandatory simplifies thespecification and guara

Note that this does not prevent server or framework developers fromoffering speciali
dictionary. This is the recommended approach forofffering any such value-added servic

Why can you call write() and yield strings/return aniterable? Shouldn't we pick just o

If we supported only the iteration approach, then currentframeworks that assume the
pushing via write(), then server performancesuffers for transmission of e.g. large files (
until all of the output has beensent). Thus, this compromise allows an application frame
with only a little more burden to the server implementor than a push-only approachwoul

What's the close() for?

When writes are done during the execution of an applicationobject, the application ca
block. But, if the application returns aniterable, any resources used will not be released
allows anapplication to release critical resources at the end of a request,and it's forward
that's proposed by PEP 325.

Why is this interface so low-level? I wantfeature X! (e.g.cookies, sessions, persisten

This isn't Yet Another Python Web Framework. It's just a way forframeworks to talk
features, you need to pick a web framework that provides thefeatures you want. And if
should be able to run it in most WSGI-supportingservers. Also, some WSGI servers ma
environ dictionary; see theapplicable server documentation for details. (Of course,appli
otherWSGI-based servers.)

Why use CGI variables instead of good oldHTTP headers? And whymix them in wi

Many existing web frameworks are built heavily upon the CGI spec,and existing web
In contrast, alternative ways of representing inbound HTTP informationare fragmented
seems like a good way to leverage existingimplementations. As for mixing them with W
dictionary arguments to bepassed around, while providing no real benefits.

What about the status string? Can't we just use the number,passing in 200 instead of

This specification does not define how a server selects or obtains anapplication to invoke. These and other configuration options arehighly server-specific matters. It is expected that server/gatewayauthors will document how to configure the server to execute aparticular application object, and with what options (such asthreading options).

Framework authors, on the other hand, should document how to create anapplication object that wraps their framework's functionality. Theuser, who has chosen both the server and the application framework,must connect the two together. However, since both the framework andthe server now have a common interface, this should be merely amechanical matter, rather than a significant engineering effort foreach new server/framework pair.

Finally, some applications, frameworks, and middleware may wish touse the environ dictionary to receive simple string configurationoptions. Servers and gateways should support this by allowingan application's deployer to specify name-value pairs to be placed inenviron. In the simplest case, this support can consist merely ofcopying all operating system-supplied environment variables fromos.environ into the environ dictionary, since the deployer inprinciple can configure these externally to the server, or in theCGI case they may be able to be set via the server's configurationfiles.

Applications should try to keep such required variables to aminimum, since not all servers will support easy configuration ofthem. Of course, even in the worst case, persons deploying anapplication can create a script to supply the necessary configurationvalues:

```
from the_app import application

def new_app(environ,start_response):

environ['the_app.configval1'] = 'something'

return application(environ,start_response)
```

But, most existing applications and frameworks will probably only needa single configuration value from environ, to indicate the locationof their application or framework-specific configuration file(s). (Ofcourse, applications should cache such configuration, to avoid havingto re-read it upon each invocation.)

URL Reconstruction

Doing this would complicate the server or gateway, by requiringthem to have a table
contrast, it is easy for an application or frameworkauthor to type the extra text to go wi
frameworks often already have a tablecontaining the needed messages. So, on balance i
responsible, rather than the serveror gateway.

Why is wsgi.run_once not guaranteed to run the app only once?

Because it's merely a suggestion to the application that it should"rig for infrequent ru
have multiple modes of operation for caching sessions, and so forth. In a "multiple run"
not write e.g. logs or session data todisk after each request. In "single run" mode, such
writes after each request.

However, in order to test an application or framework to verifycorrect operation in th
to invoke it more than once. Therefore, anapplication should not assume that it will def
wsgi.run_once set toTrue.

Feature X (dictionaries, callables, etc.) are ugly for use inapplication code; why don'

All of these implementation choices of WSGI are specificallyintended to decouple fe
encapsulated objects makes it somewhat harder towrite servers or gateways, and an ord
or modifies only small portions ofthe overall functionality.

In essence, middleware wants to have a "Chain of Responsibility"pattern, whereby it
allowing others to remain unchanged. This is difficult to dowith ordinary Python object
one must use __getattr__ or __getattribute__ overrides, to ensure that extensions (such a
through.

This type of code is notoriously difficult to get 100% correct, andfew people will wa
people's implementations, but fail to update them whenthe person they copied from cor

Further, this necessary boilerplate would be pure excise, adeveloper tax paid by midd
application framework developers. But,application framework developers will typically
in a very limited part oftheir framework as a whole. It will likely be their first (andmayb
likelyimplement with this specification ready to hand. Thus, the effortof making the AP
likely be wasted for this audience.

We encourage those who want a prettier (or otherwise improved) WSGIinterface for
opposedto web framework development) to develop APIs or frameworks thatwrap WSG
thisway, WSGI can remain conveniently low-level for server andmiddleware authors, w

Proposed/Under Discussion

These items are currently being discussed on the Web-SIG and elsewhere,or are on the

Should wsgi.input be an iterator instead of a file? This wouldhelp for asynchronous a
Optional extensions are being discussed for pausing iteration of anapplication's ouptu
Add a section about synchronous vs. asynchronous apps and servers,the relevant thre

Acknowledgements

Thanks go to the many folks on the Web-SIG mailing list whosethoughtful feedback m

Gregory "Grisha" Trubetskoy, author of mod_python, who beat upon the first draft a
encouraging me to look for a better approach.

If an application wishes to reconstruct a request's complete URL, itmay do so using the following algorithm, contributed by Ian Bicking:

```
from urllib import quote

url = environ['wsgi.url_scheme']+'://'

if environ.get('HTTP_HOST'):

url += environ['HTTP_HOST']

else:

url += environ['SERVER_NAME']

if environ['wsgi.url_scheme'] == 'https':

if environ['SERVER_PORT'] != '443':

url += ':' + environ['SERVER_PORT']

else:

if environ['SERVER_PORT'] != '80':

url += ':' + environ['SERVER_PORT']

url += quote(environ.get('SCRIPT_NAME',''))

url += quote(environ.get('PATH_INFO',''))

if environ.get('QUERY_STRING'):

url += '?' + environ['QUERY_STRING']
```

Note that such a reconstructed URL may not be precisely the same URIas requested by the client. Server rewrite rules, for example, mayhave modified the client's originally requested URL to place it in acanonical form.

Supporting Older (<2.2) Versions of Python

Ian Bicking, who helped nag me into properly specifying themultithreading and mult
mechanism for servers to supply custom extension data toan application.

Tony Lownds, who came up with the concept of a start_responsefunction that took t
contributed/guided the design of the exception handlingfacilities, especially in the area o
error messages.

Alan Kennedy, whose courageous attempts to implement WSGI-on-Jython(well befo
"supportingolder versions of Python" section, as well as the optionalwsgi.file_wrapper

Mark Nottingham, who reviewed the spec extensively for issues withHTTP RFC cor
thatI didn't even know existed until he pointed them out.

References

[1] The Python Wiki "Web Programming" topic(http://www.python.org/cgi-bin/moi
[2] The Common Gateway Interface Specification, v 1.1, 3rd Draft(http://cgi-spec.go
[3] "Chunked Transfer Coding" -- HTTP/1.1, section 3.6.1(http://www.w3.org/Proto
[4] "End-to-end and Hop-by-hop Headers" -- HTTP/1.1, Section 13.5.1(http://www.
sec13.html#sec13.5.1)
[5] mod_ssl Reference, "Environment Variables"(http://www.modssl.org/docs/2.8/ss

Copyright

0 喜欢

0 赠

分享：

阅读(392)┆ 评论 (0)┆ 收藏(0) ┆转载(0) ┆ 喜欢▼ ┆打印┆举报

前一篇：Python-WSGI详解汇总

后一篇：Python web 开发摆脱不了框架?

评论 重要提示：警惕虚假中奖信息

评论加载中，请稍候...

发评论

Supporting Older (<2.2) Versions of Python

Some servers, gateways, or applications may wish to support older(<2.2) versions of Python. This is especially important if Jythonis a target platform, since as of this writing a production-readyversion of Jython 2.2 is not yet available.

For servers and gateways, this is relatively straightforward:servers and gateways targeting pre-2.2 versions of Python mustsimply restrict themselves to using only a standard "for" loop toiterate over any iterable returned by an application. This is theonly way to ensure source-level compatibility with both the pre-2.2iterator protocol (discussed further below) and "today's" iteratorprotocol (see PEP 234).

(Note that this technique necessarily applies only to servers,gateways, or middleware that are written in Python. Discussion ofhow to use iterator protocol(s) correctly from other languages isoutside the scope of this PEP.)

For applications, supporting pre-2.2 versions of Python is slightlymore complex:

You may not return a file object and expect it to work as an iterable,since before Python 2.2, files were not iterable. (In general, youshouldn't do this anyway, because it will peform quite poorly mostof the time!) Use wsgi.file_wrapper or an application-specificfile wrapper class. (See Optional Platform-Specific File Handlingfor more on wsgi.file_wrapper, and an example class you can useto wrap a file as an iterable.)

If you return a custom iterable, it must implement the pre-2.2iterator protocol. That is, provide a __getitem__ method thataccepts an integer key, and raises IndexError when exhausted.(Note that built-in sequence types are also acceptable, since theyalso implement this protocol.)

Finally, middleware that wishes to support pre-2.2 versions of Python,and

登录名： 密码： 找回密码 注册

分享到微博 评论并转载此博文

验证码： 请点击后输入验证码 收听验证码

发评论

以上网友发言只代表其个人观点，不代表新浪

< 前一篇

Python-WSGI详解汇总