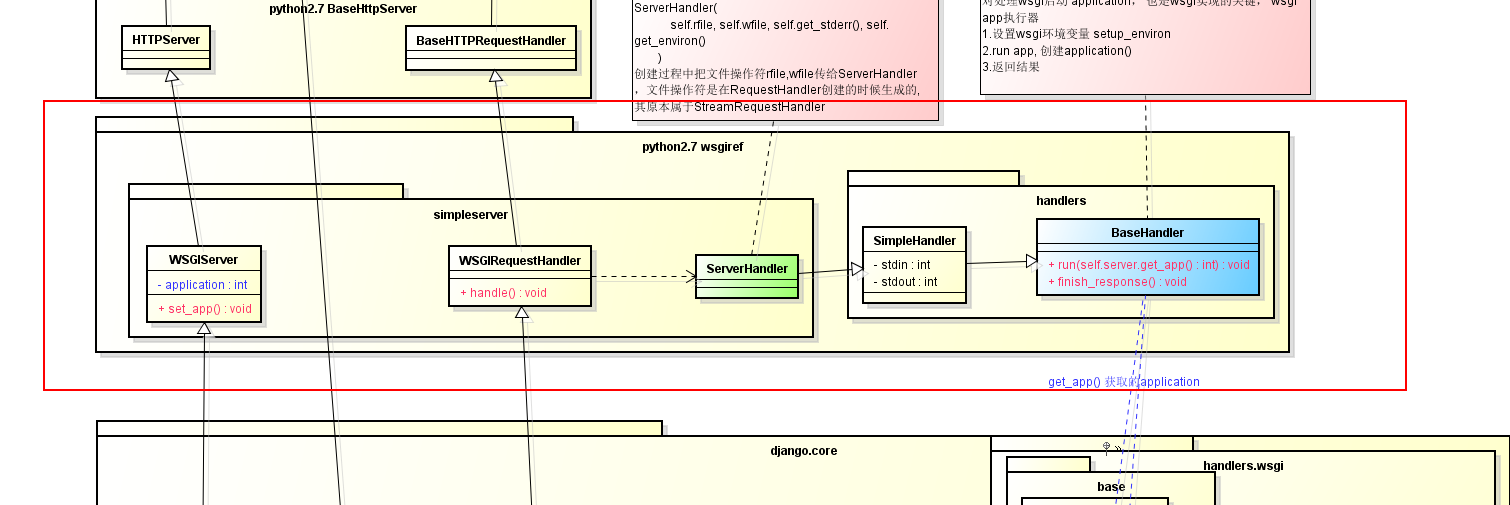
# python使用WSGIServer和WSGIRequestHandler构建wsgi协议的服务。



Python的wigiref模块提供了demo\_app，来演示如何使用wsgi协议。

wsgiref的simple\_server.py文件中

def demo\_app(environ,start\_response):

from StringIO import StringIO

stdout = StringIO()

print >>stdout, "Hello world!"

print >>stdout

h = environ.items(); h.sort()

for k,v in h:

print >>stdout, k,'=', repr(v)

start\_response("200 OK", [('Content-Type','text/plain')])

return [stdout.getvalue()]

demo\_app即是一个标准的wsgi app。它接受两个参数，一个包含cgi服务器的环境变量，另外一个参数是一个函数，这个函数也接受两个函数，一个是http状态，另外是http协议的header信息。最后该app返回一个可迭代对象，这个对象即发送给客户端的body内容。demo\_app有一些对StringIO的操作，这些都是把environ进行格式化输出。

Python的web框架，都是一个wsgi app。通过自己构造wsgi应用，很容易写出一个框架的骨架。python定义了wsgi，让web框架几乎可以大一统了。下面就来分析，python是如何实现这个协议的。

## WSGIServer

class BaseServer:

"......"

def finish\_request(self, request, client\_address):

"""Finish one request by instantiating RequestHandlerClass."""

**self.RequestHandlerClass(request, client\_address, self)**

class TCPServer(BaseServer):

"""Base class for various socket-based server classes.

Defaults to synchronous IP stream (i.e., TCP).

Methods for the caller:

- \_\_init\_\_(server\_address, RequestHandlerClass, bind\_and\_activate=True)

- serve\_forever(poll\_interval=0.5)

- shutdown()

- handle\_request() # if you don't use serve\_forever()

- fileno() -> int # for select()

Methods that may be overridden:

- server\_bind()

- server\_activate()

- get\_request() -> request, client\_address

- handle\_timeout()

- verify\_request(request, client\_address)

- process\_request(request, client\_address)

- shutdown\_request(request)

- close\_request(request)

- handle\_error()

Methods for derived classes:

- finish\_request(request, client\_address)

Class variables that may be overridden by derived classes or

instances:

- timeout

- address\_family

- socket\_type

- request\_queue\_size (only for stream sockets)

- allow\_reuse\_address

Instance variables:

- server\_address

- RequestHandlerClass

- socket

"""

class HTTPServer(SocketServer.TCPServer):

allow\_reuse\_address = 1 # Seems to make sense in testing environment

def server\_bind(self):

"""Override server\_bind to store the server name."""

SocketServer.TCPServer.server\_bind(self)

host, port = self.socket.getsockname()[:2]

self.server\_name = socket.getfqdn(host)

self.server\_port = port

-------------------------------------wsgiref-------------------------------------------

wsgiref.simple\_server.py

class WSGIServer(HTTPServer):

application = None

def server\_bind(self):

HTTPServer.server\_bind(self)

self.setup\_environ()

def setup\_environ(self):

# 甚至环境变量

env = self.base\_environ = {}

env['SERVER\_NAME'] = self.server\_name

env['GATEWAY\_INTERFACE'] = 'CGI/1.1'

env['SERVER\_PORT'] = str(self.server\_port)

env['REMOTE\_HOST']=''

env['CONTENT\_LENGTH']=''

env['SCRIPT\_NAME'] = ''

def get\_app(self):

return self.application

def set\_app(self,application):

self.application = application

-------------------------------------------django-----------------------------------------------

class WSGIServer(simple\_server.WSGIServer, object):

"""BaseHTTPServer that implements the Python WSGI protocol"""

request\_queue\_size = 10

def \_\_init\_\_(self, \*args, \*\*kwargs):

if kwargs.pop('ipv6', False):

self.address\_family = socket.AF\_INET6

super(WSGIServer, self).\_\_init\_\_(\*args, \*\*kwargs)

def server\_bind(self):

"""Override server\_bind to store the server name."""

super(WSGIServer, self).server\_bind()

self.setup\_environ()

WSGIServer继承HTTPServer，并重写了server\_bind；

WSGIServer主要做的事情：

1.第一次启动，创建socket，绑定socket

2.setup\_environ设置了一些服务的环境变量，比如服务名称，端口号等， django这里有点不好 WSGIServer.setup\_environ 在server\_bind里面调用了，django下的WSGIServer. server\_bind里面又调了一次

3. socket.listen 开始监听服务

4.循环检测是否有新的请求

5.如果有新请求， socket.accept() 接受获取到该请求

6.处理该请求

6.1创建一个线程

6.2 在线程中创建 WSGIRequestHandler对象，来真正的处理这个request

Server只是处理socket连接相关的逻辑，后面的事情WSGIRequestHandler来做了。

## WSGIRequestHandler

WSGIRequestHandler才是处理客户端请求逻辑，获取并给环境变量设置 针对这条请求的所有数据，比如CONTENT\_TYPE, SERVER\_PROTOCOL

-----------------------------socket----------------------------------------------------

class BaseRequestHandler:

class StreamRequestHandler(BaseRequestHandler):

class BaseHTTPRequestHandler(SocketServer.StreamRequestHandler):

""

-----------------------------wsgiref-----------------------------------------------

class WSGIRequestHandler(BaseHTTPRequestHandler):

server\_version = "WSGIServer/" + \_\_version\_\_

def get\_environ(self):

env = self.server.base\_environ.copy()

env['SERVER\_PROTOCOL'] = self.request\_version

env['REQUEST\_METHOD'] = self.command

if '?' in self.path:

path,query = self.path.split('?',1)

else:

path,query = self.path,''

env['PATH\_INFO'] = urllib.unquote(path)

env['QUERY\_STRING'] = query

host = self.address\_string()

if host != self.client\_address[0]:

env['REMOTE\_HOST'] = host

env['REMOTE\_ADDR'] = self.client\_address[0]

if self.headers.typeheader is None:

env['CONTENT\_TYPE'] = self.headers.type

else:

env['CONTENT\_TYPE'] = self.headers.typeheader

length = self.headers.getheader('content-length')

if length:

env['CONTENT\_LENGTH'] = length

for h in self.headers.headers:

k,v = h.split(':',1)

k=k.replace('-','\_').upper(); v=v.strip()

if k in env:

continue # skip content length, type,etc.

if 'HTTP\_'+k in env:

env['HTTP\_'+k] += ','+v # comma-separate multiple headers

else:

env['HTTP\_'+k] = v

return env

def get\_stderr(self):

return sys.stderr

def handle(self):

"""Handle a single HTTP request"""

self.raw\_requestline = self.rfile.readline()

if not self.parse\_request(): # An error code has been sent, just exit

return

handler = ServerHandler(

self.rfile, self.wfile, self.get\_stderr(), self.get\_environ()

)

handler.request\_handler = self # backpointer for logging

handler.run(self.server.get\_app())

---------------------------------------django------------------------------------------------------

class WSGIRequestHandler(simple\_server.WSGIRequestHandler, object):

def \_\_init\_\_(self, \*args, \*\*kwargs):

self.style = color\_style()

super(WSGIRequestHandler, self).\_\_init\_\_(\*args, \*\*kwargs)

def address\_string(self):

# Short-circuit parent method to not call socket.getfqdn

return self.client\_address[0]

def log\_message(self, format, \*args):

msg = "[%s] %s\n" % (self.log\_date\_time\_string(), format % args)

# Utilize terminal colors, if available

if args[1][0] == '2':

# Put 2XX first, since it should be the common case

msg = self.style.HTTP\_SUCCESS(msg)

elif args[1][0] == '1':

msg = self.style.HTTP\_INFO(msg)

elif args[1] == '304':

msg = self.style.HTTP\_NOT\_MODIFIED(msg)

elif args[1][0] == '3':

msg = self.style.HTTP\_REDIRECT(msg)

elif args[1] == '404':

msg = self.style.HTTP\_NOT\_FOUND(msg)

elif args[1][0] == '4':

msg = self.style.HTTP\_BAD\_REQUEST(msg)

else:

# Any 5XX, or any other response

msg = self.style.HTTP\_SERVER\_ERROR(msg)

sys.stderr.write(msg)

def get\_environ(self):

# Strip all headers with underscores in the name before constructing

# the WSGI environ. This prevents header-spoofing based on ambiguity

# between underscores and dashes both normalized to underscores in WSGI

# env vars. Nginx and Apache 2.4+ both do this as well.

for k, v in self.headers.items():

if '\_' in k:

del self.headers[k]

return super(WSGIRequestHandler, self).get\_environ()

工作：

1.接连初始化：RequestHandler setup，主要是调用StreamRequestHandler.setup

设置tcp连接的超时时间和读写文件描述符， 调用setsockopt 设置套接字的一些属性

2. self.raw\_requestline = self.rfile.readline()

参考：

Django:

http://www.django-rest-framework.org/tutorial/6-viewsets-and-routers/