# Application

**注意：针对tornado是2.4.1版本 以下代码有删减，具体见源代码**

application是request handlers 的集合 ， 用来分发请求到对应的处理类。

## 例子

class MainHandler(tornado.web.RequestHandler):

def get(self):

self.write("Hello, world")

application = tornado.web.Application([

(r"/", MainHandler),

])

http\_server = httpserver.HTTPServer(application)

http\_server.listen(8080)

ioloop.IOLoop.instance().start()

**上面是**Application 的一个例子，前面我们已经了解了HTTPServer，当有连接请求是，他会调用回调函数 ——application( request ) ,而此处application是Application的一个对象实例，他必须是可以调用的，即一定会有 Application.\_\_call\_\_(request) 方法。

Application 对象里面有个 self.handlers = [

( host\_pattern1, handlers) , ( host\_pattern2 , handlers) ,… ]

host 和 对应的处理 的集合 ， 其中

handlers = [URLSpec (pattern1, handler, kwargs) , URLSpec (pattern,2 handler, kwargs) ..]

当一个request来时 （Application.\_\_call\_\_(request) 大致流程）

1、先找host对应的handlers （域名对应的处理集合）

def \_get\_host\_handlers(self, request):

host = request.host.lower().split(':')[0]

for pattern, handlers in self.handlers:

if pattern.match(host):

return handlers **#逻辑是找第一个**

2、匹配路径，找到具体handler

for spec in handlers:

match = spec.regex.match(request.path)

if match:

handler = spec.handler\_class(*self*, request, \*\*spec.kwargs)

break

3、调用handler的\_ execute 方法

handler.\_execute(transforms, \*args, \*\*kwargs)

**add\_handlers:**可以添加handlers

application.add\_handlers(r"www\.myhost\.com", [

(r"/article/([0-9]+)", ArticleHandler), **#可以是一个2元组，没有kwargs**

])

def **add\_handlers**(*self*, host\_pattern, host\_handlers):

if not host\_pattern.endswith(*"$"*):

host\_pattern += *"$"*

handlers = []

#如果最后一个是all（’.\*$’） 新添加的放到他前面

if *self*.handlers and *self*.handlers[-1][0].pattern == *'.\*$'*:

*self*.handlers.insert(-1, (re.compile(host\_pattern), handlers))

else:

*self*.handlers.append((re.compile(host\_pattern), handlers))

。。。。

初始化函数：

def \_\_init\_\_(self, handlers=None, default\_host="", transforms=None,

wsgi=False, \*\*settings):

….

self.handlers = []

…

if handlers:

self.add\_handlers(**".\*$"**, handlers) 如果有handler host\_patter = ‘.\*$’

**上面**的例子，当有连接请求是，他会调用回调函数Application.\_\_call\_\_(request) 方法, 而该方法找到对应的处理类后，生成一个处理类对象handler，

调用handler.\_execute(transforms, \*args, \*\*kwargs)

而该handler 类型是MainHandler。

即tornado.web.RequestHandler

看一下class **RequestHandler**(object) 的\_execute 方法

def **\_****execute**(*self*, transforms, \*args, \*\*kwargs):

*self*.\_transforms = transforms

try:

。。。

if not *self*.\_finished:

args = [*self*.decode\_argument(arg) for arg in args]

kwargs = dict((k, *self*.decode\_argument(v, name=k))

for (k, v) in kwargs.iteritems())

**getattr(*self*, *self*.request.method.lower())(\*args, \*\*kwargs) #***调用对应的get、post方法*

if *self*.\_auto\_finish and not *self*.\_finished:

*self*.finish()

except Exception, e:

*self*.\_handle\_request\_exception(e)

def **finish**(*self*, chunk=None):

*"""Finishes this response, ending the HTTP request."""*

。。。

if chunk is not None:

*self*.write(chunk)

。。。

if not *self*.application.\_wsgi:

*self*.flush(include\_footers=True) #本地缓存写到网络（IOStream里面）

*self*.request.finish() #close 或者继续读请求if connection

*self*.\_log()

*self*.\_finished = True

*self*.on\_finish() # 空函数

*#只写到本地缓存，调用flush 才写到网络上*

def **write**(*self*, chunk):

。。。

if isinstance(chunk, dict):

chunk = escape.json\_encode(chunk)

*self*.set\_header(*"Content-Type"*, *"application/json; charset=UTF-8"*)

chunk = utf8(chunk)

*self*.\_write\_buffer.append(chunk)

def **flush**(*self*, include\_footers=False, callback=None):

*"""Flushes the current output buffer to the network.*

*The ``callback`` argument, if given, can be used for flow control:*

*it will be run when all flushed data has been written to the socket.*

*Note that only one flush callback can be outstanding at a time;*

*if another flush occurs before the previous flush's callback*

*has been run, the previous callback will be discarded.*

*"""*

if *self*.application.\_wsgi:

raise Exception(*"WSGI applications do not support flush()"*)

chunk = b(*""*).join(*self*.\_write\_buffer)

*self*.\_write\_buffer = []

if not *self*.\_headers\_written:

*self*.\_headers\_written = True

for transform in *self*.\_transforms:

*self*.\_status\_code, *self*.\_headers, chunk = \

transform.transform\_first\_chunk(

*self*.\_status\_code, *self*.\_headers, chunk, include\_footers)

headers = *self*.\_generate\_headers()

else:

for transform in *self*.\_transforms:

chunk = transform.transform\_chunk(chunk, include\_footers)

headers = b(*""*)

if *self*.request.method == *"HEAD"*:

if headers:

*self*.request.write(headers, callback=callback)

return

*self*.request.write(headers + chunk, callback=callback)