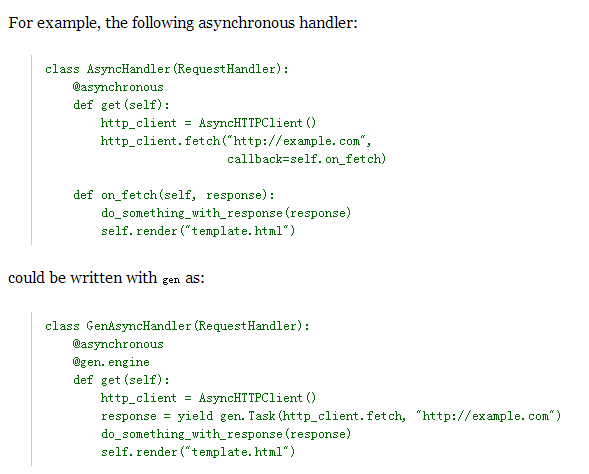
# tornado.gen

— Simplify asynchronous code

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

## 例子

使用 装饰器gen.engine 和 类gen.Task 封装异步回调



## 装饰器@asynchronous

Tornado/web.py

Asynchronous 封装异步处理请求，用@*asynchronous* 装饰的请求方法 不会立即返回Http请求，只有当处理对象调用self.finish()的时候，才返回结果。

其实现很简单，就设置了一下 *self*.\_auto\_finish 的值，让其，不直接返回结果，要掉用finish函数才返回结果。

def **asynchronous**(method):

*@functools.wraps*(method)

def **wrapper**(*self*, \*args, \*\*kwargs):

if *self*.application.\_wsgi:

raise Exception(*"@asynchronous is not supported for WSGI apps"*)

***self*.\_auto\_finish = False**

with stack\_context.ExceptionStackContext(

*self*.\_stack\_context\_handle\_exception):

return method(*self*, \*args, \*\*kwargs)

return wrapper

## 装饰器@engine

*tornado/gen.py*

*基于生成器接口，可以让异步回调分片代码 合成在一个生成器函数里面。*

gen.Task 类

上面例子 gen.Task 封装了异步回调函数http\_client.fetch("http://example.com")

在调用该函数之前传人 callback参数(注意，这就要求所有异步函数的回调参数名字一定要叫"callback" )

即 调用 http\_client.fetch("http://example.com" , callback=task\_callback)

这样 gen.Task 就可以得到 异步回调的结果了

具体实现

*tornado/gen.py*

def engine(func):

@functools.wraps(func)

def wrapper(\*args, \*\*kwargs):

gen = func(\*args, \*\*kwargs)

if isinstance(gen, types.GeneratorType):

runner = Runner(gen, deactivate)

runner.run()

return

# no yield, so we're done

return wrapper

*Runner类*

调用生成器，直到生成器结束

注：yield\_point 即是gen.Task

*def run(self):*

*while True:*

#是否已经运行完回调---否有结果

*if not self.yield\_point.is\_ready():*

*return*

# 获取task 结果

*next = self.yield\_point.get\_result()*

*try:*

# 继续运行生成器，并把 结果(task 异步回调参数) 传给yield 返回值

*yielded = self.gen.send(next)*

*except StopIteration:*

*self.finished = True*

*return*

#如果还有yield Task , 开始运行

*self.yield\_point = yielded*

*self.yield\_point.start(self)*

上面在while True 第一个if 语句 如何Task 还没有结束（活动结果），会退出循环，那循环是如何恢复的？

是在运行回调，保存结果后调用run

*def set\_result(self, key, result):*

*self.results[key] = result*

*self.run()*