# 浅谈 Gevent 与 Tornado

written on 2012年8月17日星期五

还是前几月的时候，几乎在同一时间，自己接触到了 Gevent 和 Tornado 这两个已经不新的东西，那时那个 思绪混乱啊！似乎都支持异步，似乎都是无阻塞（non-blocking），性能似乎都好到个不行（猛击）。知道两者虽是单线程， 但基于无阻塞的特性，战斗力那个是嗖嗖地上涨，运用得当的话，hold住上K个连接不是问题。虽然很感 兴趣，虽然完全没弄清楚两者内里的实质，但为了完成工作，略略了解了基本的应用后，卷起手袖就上啦。 当然，作为一个有志的程序员，在满足了现实的迫切需要后，一颗渴望知其所以然的心便开始蠢蠢欲动。

## 从 Tornado 说起

刚开始，对 Tornado 的感觉最为新鲜，在官网介绍里其是一个无阻塞的Web服务器以及相关工具的集合，但 个人更为倾向其为一个颇为完备的微型 web 框架。Tornado 性能好的关键是其无阻塞异步的特性，但这魔术 似的效果是如何达成的呢？迷思与困惑。我那小脑袋里的思维还停留于多进程（多线程）那样的并发模型中， 实在有点难以理解 Tornado 的异步机制。

通过查阅各式文章以及源代码，整体的框架脉络开始逐渐在脑海中显现出来。其实，Tornado 的异步模型 是由事件驱动以及特定的回调函数（callback）所组成的！一直没有弄明白，Tornado 具体是如何实现 无阻塞异步，当清楚了事件驱动和回调函数的概念后，事情似乎又变得简单起来了。

对于一般的程序，在执行阶段若遇到 I/O 事件，整个进程将被阻塞住，直到 I/O 事件结束，程序又继续执行。 接设我们对一些 I/O 事件进行了定制，使其可以立即返回（即无阻塞），那么程序将能立即继续执行。但 问题又来了，那当 I/O 事件完成后又该怎么办呢？此时，回调函数的威力就出来了，我只需要将进行特定 处理的回调函数与该 I/O 事件绑定起来，当该 I/O 事件完成后就调用绑定的回调函数，就可以处理具体的 I/O 事件啦。啊，似乎还有一个问题，回调函数要如何与 I/O 事件绑定起来？最简单的想法是，直接通过 一个 while True 循环不断的轮询，当检测到 I/O 事件完成了即触发回调函数。但是，这样的效率当然不会 高，利用系统中高效的 I/O 事件轮询机制（epoll on Linux, kqueue on most BSD）就是最明智的 解决方案。于是，无阻塞 I/O +事件驱动+高效轮询方式便组成了 Tornado 的异步模型。

Tornado 的核心是 ioloop 和 iostream 这两个模块，前者提供了一个高效的 I/O 事件循环，后者则封装了 一个无阻塞的 socket 。通过向 ioloop 中添加网络 I/O 事件，利用无阻塞的 socket ，再搭配相应的回调 函数，便可达到梦寐以求的高效异步执行啦。多说无益，来看一下具体的示例：

```
from tornado import ioloop
from tornado.httpclient import AsyncHTTPClient

urls = ['http://www.google.com', 'http://www.yandex.ru', 'http://www.python.or
g']

def print_head(response):
    print ('%s: %s bytes: %r' % (response.request.url,
                                 len(response.body),
                                 response.body[:50]))

http_client = AsyncHTTPClient()
for url in urls:
    print ('Starting %s' % url)
    http_client.fetch(url, print_head)
ioloop.IOLoop.instance().start()
```

因为使用了 AsyncHTTPClient 来处理请求操作，整个示例是异步执行的，即三个url请求无等待的依次发出。 我们可以看到 fetch 方法使用了 print_head 函数来作为回调函数，这意味着，当 fetch 完成了请求操作， 相应的 print_head 函数便会被触发调用。恩，... 额，...，乍看起来，使用 Tornado 进行异步编程似乎 并不难，让人跃跃欲试。但实际上，在现实生活中，事件驱动的编程还是会很费脑力，需要一定的创造性思维。不过，这也许是 Tornado 受欢迎的原因之一呢。 :)

## 再来看下 Gevent

Gevent 是基于协程（coroutine）实现的 Python 网络库，使用了轻量级的 greenlet 作为执行单元，并 基于 libevent 事件循环构建了直观的调用接口。

当时看到这样的描述，脑袋的第一反应是，协程？？稍稍了解后，发现协程其实也不是什么高深的概念，协程 也被称为微线程，一看这别名就知道跟线程应该很类似。作为类比倒也可以这么认为，两者关键的区别在于， 线程是由系统进行调度的，而协程是由用户自己进行调度的。当知道这一事实后，立刻想到，这自行调度灵活 肯定是会很灵活，但要调度的话可是很有难度的吧？调度的方法暂时不谈，除了更为灵活外，自行调度的直接 结果当然就是省去了系统调度（什么用户态转内核态，以及什么 context switch），因此协程间切换的资源 消耗很小，再配合协程生成成本很低的另一特点，这可真是相当的美妙。事实上，Python 语言本身就支持基础 的协程的概念，generator 是其中的产物（这里）。

对于 Gevent，其使用的协程实际上就是 greenlet 。当你使用 greenlet 生成了一些协程，就可以在这些 协程里不断跳转执行，两个 greenlet 之间的跳转被称为切换（switch）。通过切换，我们就可以实现对协程 的调度。还应该知道的是，每个 greenlet 都拥有一个父 greenlet ，这是在 greenlet 初始化时就确定的。 当一个 greenlet 执行完毕后，执行权会切换到其父 greenlet 中。实际上，所有的 greenlet 会被组织成 一颗树，树根便是最“老资格”的 greenlet ，这个老 greenlet 确定了各

greenlet 间的逻辑关系。

上面说到协程必须自行调度，不会是要自己构造一个调度器吧？这当然可以做到，但不是必须，因为 Gevent 已经基于 greenlet 和 libevent 封装了许多基础常用的库，例如 socket 、event 和 queue 等，只要使用 这些库进行开发，或者对使用的标准库或第三方库打一下补丁（ monket patch ），就能保证生成的各协程在 I/O 等待时正确地进行切换，从而实现无阻塞的异步执行。

刚接触 Gevent 时，感觉跟传统的并发编程很类似，但了解渐深后，才发现这货实际上跟 Tornado 更为类似。 因为， Gevent 本质上也是事件驱动。实现的策略可以是，在将要执行 I/O 阻塞事件时，先在事件循环中对该事件 进行注册，关联的回调函数便是对当前协程的切换操作（ **current_greenlet.switch()** ），注册成功后即 切换回当前协程的父协程中进行执行（ **current_greenlet.parent.switch()** ）。当注册的 I/O 事件被 触发后，事件循环在恰当时机便会执行该回调函数，也就是切换到原先的协程继续执行程序。从而，就实现 无阻塞的 I/O 事件处理。怎样，是否感觉相当的有趣？ :)

Gevent 了不得的地方还在于，我们能像编写一般程序那样来编写异步程序，这可是弥足珍贵。为了更直观的 显示，让我们来看一下具体的运行示例:

```
import gevent
from gevent import monkey
# patches stdlib (including socket and ssl modules) to cooperate with other gre
enlets
monkey.patch_all()

import urllib2

urls = ['http://www.google.com', 'http://www.yandex.ru', 'http://www.python.or
g']

def print_head(url):
    print ('Starting %s' % url)
    data = urllib2.urlopen(url).read()
    print ('%s: %s bytes: %r' % (url, len(data), data[:50]))

jobs = [gevent.spawn(print_head, url) for url in urls]

gevent.joinall(jobs)
```

上面示例做的事情实际上跟前面 Tornado 的示例是一样，同样是异步的对url进行请求。在我看来，使用 Gevent 进行编程，无论是可读性还是可操作性都能让人满意。但也要清楚，在实际操作中，为了达到较理想 效果，经常还是需要根据不同的情况对代码进行一些相应的“雕琢”。还有一点很常被人忽略， Gevent 是 基于协程实现的 Python 网络库，其适用面更多的是在于网络 I/O 频繁的需求里，很多情况下 Gevent 可能 并不是很好的选择。总的来说，Gevent 确实很讨人喜爱，性能好，开销小，代码易维护，是广大 pythoner 手中的一大利器。

# 总要总结一下

作为一名 Python 程序员，在探究和使用 Tornado 与 Gevent 的过程里，除了得到许多思考的乐趣外，最 让人高兴的是收获了一些全新的视野。使用 Python 编程的好处之一便是，可以很容易地跳出语言的框框去看 各式问题，从而提高自己对于程序设计的总体认识。人生苦短，我用Python！ :-)

This entry was tagged Gevent, Python, Tornado and coroutine