# IOStream

——从sockets到流

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

它的目的是提供一个对非阻塞式sockets的轻量级抽象，

用来处理socket的异步读写

IOStream --read/write --ioloop  （异步io基于IOLoop）

源文件：

tornado/iostream.py

## IOStream主要提供的接口有：

连接

connect(self, address, callback=None):

读数据的：

read\_until\_regex(self, regex, callback) # 获取 以regex结尾(正则表达式)的数据

read\_until(self, delimiter, callback) # 获取 以delimiter结尾的数据

read\_bytes(self, num\_bytes, callback, streaming\_callback=None) # 读取多少个字节数据

read\_until\_close(self, callback, streaming\_callback=None) # 读直到流关闭

写：

write(self, data, callback=None):

关闭连接回调

close()

set\_close\_callback(self, callback):

所有上述的方法都可以通过异步方式在它们完成时触发回调函数。

## 实现原理

IOstream 里面有2个缓冲队列 读和写，所有外部读写操作都是直接操作缓存区，底层socket异步回写。

class IOStream(object):

def \_\_init\_\_(self, socket, io\_loop=None, max\_buffer\_size=104857600,

read\_chunk\_size=4096):

# 2个缓冲队列

self.\_read\_buffer = collections.deque()

self.\_write\_buffer = collections.deque()

由于使用缓冲队列，当还在连接过程中，调用IOStream.write 是安全的；

但是Calling IOStream read 不是所有平台都是可用（works on some platforms but is non-portable.）

异步的IOStream是基于IOloop类

\_read\_buffer

\_read\_callback

socket.recv

**READ/\_handle\_read**

IOLoop

WRITE**/**\_handle\_write

Sock.send

\_write\_buffer

\_write\_callback

下面看read\_xxx 实现

先读缓存区，如果数据不够socke.recv, 如果还不够数据，IOLoop注册READ事件

如果读到数据调用回调

Write实现：

分片写 到缓存区

Sock.send 发送缓存区数据

如果缓存区还有数据 IOLoop注册Write事件

否则回调函数

IOStream回调函数的调用：它不是直接调用，而是callback 丢到下一个ioloop迭代

def \_run\_callback(self, callback, \*args):

def wrapper():

self.\_pending\_callbacks -= 1

try:

callback(\*args)

except Exception:

logging.error("Uncaught exception, closing connection.",

exc\_info=True)

self.close()

raise

self.\_maybe\_add\_error\_listener()

with stack\_context.NullContext():

self.\_pending\_callbacks += 1

self.io\_loop.add\_callback(wrapper) # 下一个迭代后执行

## 例子

文档例子A very simple (and broken) HTTP client using this class::

def send\_request():

stream.write("GET / HTTP/1.0\r\nHost: friendfeed.com\r\n\r\n")

stream.read\_until("\r\n\r\n", on\_headers)

def on\_headers(data):

headers = {}

for line in data.split("\r\n"):

parts = line.split(":")

if len(parts) == 2:

headers[parts[0].strip()] = parts[1].strip()

stream.read\_bytes(int(headers["Content-Length"]), on\_body)

def on\_body(data):

print data

stream.close()

ioloop.IOLoop.instance().stop()

s = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM, 0)

stream = iostream.IOStream(s)

stream.connect(("friendfeed.com", 80), send\_request)

ioloop.IOLoop.instance().start()

# tornado 3.0.1 改变

3.x 版本对IOStream 进行重构。把它拆成2个类，

class **BaseIOStream**(object):

class **IOStream**(BaseIOStream):

Much of [**IOStream**](http://www.tornadoweb.org/en/stable/iostream.html#tornado.iostream.IOStream) has been refactored into a separate class [**BaseIOStream**](http://www.tornadoweb.org/en/stable/iostream.html#tornado.iostream.BaseIOStream).

简单重构，改变的不多

* New class [**tornado.iostream.PipeIOStream**](http://www.tornadoweb.org/en/stable/iostream.html#tornado.iostream.PipeIOStream) provides the IOStream interface on pipe file descriptors.
* 详细见
* <http://www.tornadoweb.org/en/stable/releases/v3.0.0.html>