**基于 IB 盈透证券的原生 Python API 连接**

<https://developer.51cto.com/art/202108/679004.htm>

## 基于 IB 盈透证券的原生 Python API 连接

在本文中，我们将介绍如何为 Interactive Brokers Native Python API 提供的 EClient 和 EWrapper 类派生子类。然后，我们将提供端到端连接测试脚本，以确保我们能够与 IB 进行对话。

作者：佚名来源：[Python中文社区](https://mp.weixin.qq.com/s/240r2EiZI6Lyt20jwcjxfw)|2021-08-24 10:42

[收藏](javascript:favorBox('open');)

[分享](javascript:;)

[](https://s2.51cto.com/oss/202108/24/87a2f0985ddde5f7be5f65fa6f6b0322.jpeg?x-oss-process=image/format,jpg,image/resize,w_651)

Python中文社区 （ID:python-china）

在本文中，我们将介绍如何为 Interactive Brokers Native Python API 提供的 EClient 和 EWrapper 类派生子类。然后，我们将提供端到端连接测试脚本，以确保我们能够与 IB 进行对话。

盈透证券（Interactive Brokers）一直是受交易者欢迎的经纪商。最初，这可能部分归因于 IB 提供了一个应用程序编程接口 (API)，允许量化交易者获取市场数据并直接在代码中进行交易。许多相互竞争的经纪商花了一些时间来开发自己的 API，使 IB 在零售量化交易领域获得了合理的先发优势。

虽然最初的 IB API 以接口复杂而著称，但近年来随着 IB Native Python API 库的发布，情况发生了变化。

在这个新系列文章中，我们将使用 ibapi 库，来解释如何通过“原生 Python”接口与Interactive Brokers API交互。

最终我们将学习如何请求市场数据、定义合同和处理订单。本文将重点介绍接口本身并测试基本连接。

本文假设您有一个可运行的 Python 虚拟环境（例如 Anaconda 个人版）并已成功将 IB Python API 安装到该环境中。安装说明是特定于操作系统的。可以在Interactive Brokers API 站点上找到最新的说明。

### 概述

IB API 通过异步'request-response'模型进行工作。消息通过客户端类发送到 IB 服务器（通过 Trader Workstation 或IB Gateway），而响应（称为“errors”）由包装类单独处理。

大部分内部连接处理都通过 Python API 从最终用户那里抽取出来，允许最少必要的'boilerplate'代码进行连接。但是，连接到 IB 的原始遗留机制仍然部分地影响了 API 的设计。因此，对于那些不习惯面向对象设计原则的人来说，这可能会令人困惑。

虽然最初似乎不清楚所有组件如何组合在一起，但在对以下类进行编码后，应该开始了解 API 的构建方式。

为了连接到 IB API，需要对四个主要组件进行编码。

第一个是 IB API EWrapper 类的派生子类。 EWrapper 用于处理来自 IB 服务器的所有响应（'errors'）。

第二个是IB API EClient 类的派生子类。 EClient 用于将所有消息发送到 IB 服务器。

第三个是从 EWrapper 和 EClient 派生的子类的多重继承类，用作 IB API 应用程序的基础，它将所有通讯联系在一起。

最后会有一个 if \_\_name\_\_ == "\_\_main\_\_": 入口点，旨在允许从命令行执行脚本。

### 初始化

第一步是导入脚本中使用的必要库组件。

我们将需要 IB API EWrapper 和 EClient 类，这将在下面描述。我们还需要分别来自线程 和队列库的Thread和Queue标准库组件。最后，我们将导入 datetime 以将 Unix 时间戳转换为更易读的格式：

1. # ib\_api\_connection.py
2. import datetime
3. import queue
4. import threading
5. from ibapi.client import EClient
6. from ibapi.wrapper import EWrapper

我们现在可以定义 IBAPIWrapper 类。

### IBAPIWrapper 类

EWrapper 类提供了一个接口处理来自 IB 服务器的响应（描述为'errors'）。接口指定可以在派生子类中实现的方法。通过继承这个类，我们可以覆盖这些方法以适应我们自己特定的数据处理方法。

让我们首先创建 EWrapper 的 IBAPIWrapper 派生子类并覆盖一些方法。显示此组件的完整代码段,并将依次介绍每种方法：

1. # ib\_api\_connection.py
2. class IBAPIWrapper(EWrapper):
3. """
4. A derived subclass of the IB API EWrapper interface
5. that allows more straightforward response processing
6. from the IB Gateway or an instance of TWS.
7. """
8. def init\_error(self):
9. """
10. Place all of the error messages from IB into a
11. Python queue, which can be accessed elsewhere.
12. """
13. error\_queue = queue.Queue()
14. self.\_errors = error\_queue
15. def is\_error(self):
16. """
17. Check the error queue for the presence
18. of errors.
19. Returns
20. -------
21. `boolean`
22. Whether the error queue is not empty
23. """
24. return not self.\_errors.empty()
25. def get\_error(self, timeout=5):
26. """
27. Attempts to retrieve an error from the error queue,
28. otherwise returns None.
29. Parameters
30. ----------
31. timeout : `float`
32. Time-out after this many seconds.
33. Returns
34. -------
35. `str` or None
36. A potential error message from the error queue.
37. """
38. if self.is\_error():
39. try:
40. return self.\_errors.get(timeouttimeout=timeout)
41. except queue.Empty:
42. return None
43. return None
44. def error(self, id, errorCode, errorString):
45. """
46. Format the error message with appropriate codes and
47. place the error string onto the error queue.
48. """
49. error\_message = (
50. "IB Error ID (%d), Error Code (%d) with "
51. "response '%s'" % (id, errorCode, errorString)
52. )
53. self.\_errors.put(error\_message)
54. def init\_time(self):
55. """
56. Instantiates a new queue to store the server
57. time, assigning it to a 'private' instance
58. variable and also returning it.
59. Returns
60. -------
61. `Queue`
62. The time queue instance.
63. """
64. time\_queue = queue.Queue()
65. self.\_time\_queue = time\_queue
66. return time\_queue
67. def currentTime(self, server\_time):
68. """
69. Takes the time received by the server and
70. appends it to the class instance time queue.
71. Parameters
72. ----------
73. server\_time : `str`
74. The server time message.
75. """
76. self.\_time\_queue.put(server\_time)

init\_error 的任务是创建一个 Python Queue队列并将其附加一个名为\_errors 的“私有”实例变量。该队列将在整个类中用于存储 IB 错误消息以供以后处理。

is\_error 是一个简单的方法，用于判断\_errors 队列是否为空。

get\_error尝试从队列中检索错误消息，规定的超时时间以秒为单位。如果队列为空或超时，则该方法不会返回任何内容。

error 将提供的错误代码与错误消息一起格式化为适当的字符串格式，然后将其放入 \_errors 队列。此方法用于在针对 API 执行代码时在控制台上提供更好的调试信息。

这四种方法完成了对来自盈透证券的响应（'errors'）的处理。需要注意的是，ibapi 库中有很多内部机器执行此处理。从我们的派生子类中无法直接看到大部分工作。

其余两个方法 init\_time 和 currentTime 用于执行连接“健全性检查”（'sanity check'）。确定我们是否连接成功的一种简单方法是检索 IB 服务器上的本地时间。

这两种方法只是创建一个新队列来存储服务器时间消息，并在请求时将新时间消息放置到这个队列中。

我们的 EWrapper 简单子类到此结束。我们现在能够处理来自 IB 服务器的某些响应。下一个任务是实际向 IB 服务器发送消息。为此，我们需要覆盖 EClient 类。

### IBAPIClient 类

EClient的 IBAPIClient 派生子类用于向 IB 服务器发送消息。

需要特别注意的是，我们派生子类的构造函数 \_\_init\_\_方法接受一个包装参数，然后将其传递给EClient父构造函数。这意味着在 IBAPIClient类中没有覆盖本地 IB API 方法。相反，我们提供了包装器实例（从 IBAPIWrapper实例化）来处理响应。

1. # ib\_api\_connection.py
2. class IBAPIClient(EClient):
3. """
4. Used to send messages to the IB servers via the API. In this
5. simple derived subclass of EClient we provide a method called
6. obtain\_server\_time to carry out a 'sanity check' for connection
7. testing.
8. Parameters
9. ----------
10. wrapper : `EWrapper` derived subclass
11. Used to handle the responses sent from IB servers
12. """
13. MAX\_WAIT\_TIME\_SECONDS = 10
14. def \_\_init\_\_(self, wrapper):
15. EClient.\_\_init\_\_(self, wrapper)
16. def obtain\_server\_time(self):
17. """
18. Requests the current server time from IB then
19. returns it if available.
20. Returns
21. -------
22. `int`
23. The server unix timestamp.
24. """
25. # Instantiate a queue to store the server time
26. time\_queue = self.wrapper.init\_time()
27. # Ask IB for the server time using the EClient method
28. self.reqCurrentTime()
29. # Try to obtain the latest server time if it exists
30. # in the queue, otherwise issue a warning
31. try:
32. server\_time = time\_queue.get(
33. timeout=IBAPIClient.MAX\_WAIT\_TIME\_SECONDS
34. )
35. except queue.Empty:
36. print(
37. "Time queue was empty or exceeded maximum timeout of "
38. "%d seconds" % IBAPIClient.MAX\_WAIT\_TIME\_SECONDS
39. )
40. server\_time = None
41. # Output all additional errors, if they exist
42. while self.wrapper.is\_error():
43. print(self.get\_error())
44. return server\_time

在obtain\_server\_time中，我们首先创建一个队列来保存来自服务器的时间戳消息。然后我们调用原生 EClient 方法 reqCurrentTime 从服务器获取时间。

随后，我们在 try...except 块中包装了一个从时间队列中获取值的调用。我们提供10秒的超时时间。如果超时或队列为空，我们将服务器时间设置为None。

我们运行一个 while 循环来检查 EWrapper 派生子类中定义的错误队列中的任何其他响应。如果它们存在，它们将打印到控制台。

最后我们返回服务器时间。

下一阶段是创建一种机制来实例化 IBAPIWrapper和 IBAPIClient，以及实际连接到 IB 服务器。

### IBAPIApp

本文中要派生的最后一个类是 IBAPIApp 类。

此类利用多重继承从 IBAPIWrapper 和 IBAPIClient 类继承。在初始化时，这两个类也被初始化。但是，请注意 IBAPIClient 类将 wrapper=self 作为初始化关键字参数，因为 IBAPIApp也是从 IBAPIWrapper 派生的。

在初始化两个父类之后，使用适当的连接参数调用connect原生方法。

下一部分代码初始化应用程序所需的各种线程。客户端实例有一个线程，另一个用于将响应消息添加到各个队列。

最后调用 init\_error 方法开始监听 IB 响应。

1. # ib\_api\_connection.py
2. class IBAPIApp(IBAPIWrapper, IBAPIClient):
3. """
4. The IB API application class creates the instances
5. of IBAPIWrapper and IBAPIClient, through a multiple
6. inheritance mechanism.
7. When the class is initialised it connects to the IB
8. server. At this stage multiple threads of execution
9. are generated for the client and wrapper.
10. Parameters
11. ----------
12. ipaddress : `str`
13. The IP address of the TWS client/IB Gateway
14. portid : `int`
15. The port to connect to TWS/IB Gateway with
16. clientid : `int`
17. An (arbitrary) client ID, that must be a positive integer
18. """
19. def \_\_init\_\_(self, ipaddress, portid, clientid):
20. IBAPIWrapper.\_\_init\_\_(self)
21. IBAPIClient.\_\_init\_\_(self, wrapper=self)
22. # Connects to the IB server with the
23. # appropriate connection parameters
24. self.connect(ipaddress, portid, clientid)
25. # Initialise the threads for various components
26. thread = threading.Thread(target=self.run)
27. thread.start()
28. setattr(self, "\_thread", thread)
29. # Listen for the IB responses
30. self.init\_error()

现在定义了前三个类，我们就可以创建脚本入口点了。

### 执行代码

我们首先设置连接参数，包括主机 IP 地址、连接到 TWS/IB 网关的端口和（任意）正整数客户端 ID。

然后我们使用适当的连接参数实例化一个应用程序实例。

我们使用该应用程序从 IB 获取服务器时间，然后使用 datetime 库的 utcfromtimestamp 方法将 Unix 时间戳适当地格式化为更具可读性的日期格式。

最后我们断开与IB服务器的连接并结束程序。

1. # ib\_api\_connection.py
2. if \_\_name\_\_ == '\_\_main\_\_':
3. # Application parameters
4. host = '127.0.0.1'  # Localhost, but change if TWS is running elsewhere
5. port = 7497  # Change to the appropriate IB TWS account port number
6. client\_id = 1234
7. print("Launching IB API application...")
8. # Instantiate the IB API application
9. app = IBAPIApp(host, port, client\_id)
10. print("Successfully launched IB API application...")
11. # Obtain the server time via the IB API app
12. server\_time = app.obtain\_server\_time()
13. server\_time\_readable = datetime.datetime.utcfromtimestamp(
14. server\_time
15. ).strftime('%Y-%m-%d %H:%M:%S')
16. print("Current IB server time: %s" % server\_time\_readable)
17. # Disconnect from the IB server
18. app.disconnect()
19. print("Disconnected from the IB API application. Finished.")

在这个阶段，我们准备运行 ib\_api\_connection.py。只需导航到您存储文件的目录，确保带有 ibapi的虚拟环境处于活动状态，并且 TWS（或 IB 网关）已正确加载和配置，然后键入以下内容：

1. python ib\_api\_connection.py

您应该会看到类似于以下内容的输出：

1. Launching IB API application...
2. Successfully launched IB API application...
3. IB Error ID (-1), Error Code (2104) with response 'Market data farm connection is OK:usfarm'
4. IB Error ID (-1), Error Code (2106) with response 'HMDS data farm connection is OK:ushmds'
5. IB Error ID (-1), Error Code (2158) with response 'Sec-def data farm connection is OK:secdefnj'
6. Current IB server time: 2020-07-29 13:27:18
7. Disconnected from the IB API application. Finished.
8. unhandled exception in EReader thread
9. Traceback (most recent call last):
10. File "/home/mhallsmoore/venv/qstrader/lib/python3.6/site-packages/ibapi/reader.py", line 34, in run
11. data = self.conn.recvMsg()
12. File "/home/mhallsmoore/venv/qstrader/lib/python3.6/site-packages/ibapi/connection.py", line 99, in recvMsg
13. buf = self.\_recvAllMsg()
14. File "/home/mhallsmoore/venv/qstrader/lib/python3.6/site-packages/ibapi/connection.py", line 119, in \_recvAllMsg
15. buf = self.socket.recv(4096)
16. OSError: [Errno 9] Bad file descriptor

第一组输出是带有代码 2104、2106 和 2158 的 IB 'errors'。这些实际上是说明与各种服务器的连接正常运行的响应。也就是说，它们不是'errors'！

服务器时间也从 Unix 时间戳正确转换为更易读的格式和输出。在此阶段，应用程序断开连接。

但是请注意，在 EReader 线程中引发了 OSError 异常。这是 IB API 本身的一个内部问题，目前还没有一个修复程序。出于本教程的目的，它可以被忽略。

现在完成了连接到 IB Python Native API 的教程。 ib\_api\_connection.py 的完整代码请扫描下方二维码获取。

我们已经成功连接到IB服务器，并通过调用检查连接，获取当前服务器时间。后面我们将确定如何从 IB API 检索股票的市场数据。

[鸿蒙官方战略合作共建——HarmonyOS技术社区](https://harmonyos.51cto.com/#kfxqy)