

## 34 | RESTful & Socket: 搭建交易执行层核心

2019-07-26 景霄

Python核心技术与实战

[进入课程 >](#)



讲述：冯永吉

时长 17:33 大小 16.08M



你好，我是景霄。

上一节，我们简单介绍了量化交易的历史、严谨的定义和它的基本组成结构。有了这些高层次的基本知识，接下来我们就分模块，开始讲解量化交易系统中具体的每部分。

从这节课开始，我们将实打实地从代码出发，一步步设计出一套清晰完整、易于理解的量化交易系统。

一个量化交易系统，可以说是一个黑箱。这个黑箱连接交易所获取到的数据，通过策略运算，然后再连接交易所进行下单操作。正如我们在输入输出那节课说的那样，黑箱的特性是输入和输出。每一个设计网络交互的同学，都需要在大脑中形成清晰的交互状态图：

知道包是怎样在网络间传递的；

知道每一个节点是如何处理不同的输入包，然后输出并分发给下一级的。

在你搞不明白的时候，可以先在草稿纸上画出交互拓扑图，标注清楚每个节点的输入和输出格式，然后想清楚网络是怎么流动的。这一点，对网络编程至关重要。


现在，我假设你对网络编程只有很基本的了解。所以接下来，我将先从 REST 的定义讲起，然后过渡到具体的交互方式——如何通过 Python 和交易所进行交互，从而执行下单、撤单、查询订单等网络交互方式。

## REST 简介

什么是 REST API？什么是 Socket？有过网络编程经验的同学，一定对这两个词汇不陌生。


REST 的全称是表征层状态转移（REpresentational State Transfer），本意是指一种操作资源方法。不过，你不用纠结于这个绕口的名字。换种方式来说，REST 的实质可以理解为：通过 URL 定位资源，用 GET、POST、PUT、DELETE 等动词来描述操作。而满足 REST 要求的接口，就被称为 RESTful 的接口。

为了方便你更容易理解这些概念，这里我举个例子来类比。小明同学不是很聪明但很懂事，每天会在他的妈妈下班回来后给妈妈泡茶。刚开始，他的妈妈会发出这样的要求：

 复制代码

```
1 用红色杯子，去厨房泡一杯放了糖的 37.5 度的普洱茶。  
2
```

可是小明同学不够聪明，很难理解这个定语很多的句子。于是，他妈妈为了让他更简单明白需要做的事情，把这个指令设计成了更简洁的样子：

 复制代码

```
1 泡厨房的茶，  
2  
3  
4 要求：  
5 类型 = 普洱；
```


```
6 杯子 = 红色;
7 放糖 =True;
8 温度 =37.5 度。
```

这里的“茶”就是资源，“**厨房的茶**”就是资源的地址（URI）；“**泡**”是动词；后面的要求，都是接口参数。这样的接口，就是小明提供的一个 REST 接口。

如果小明是一台机器，那么解析这个请求就会非常容易；而我们作为维护者，查看小明的代码也很简单。当小明把这个接口暴露到网上时，这就是一个 RESTful 的接口。

总的来说，RESTful 接口通常以 HTTP GET 和 POST 形式出现。但并非所有的 GET、POST 请求接口，都是 RESTful 的接口。


这话可能有些拗口，我们举个例子来看。上节课中，我们获取了 Gemini 交易所中，BTC 对 USD 价格的 ticker 接口：

 复制代码

```
1 GET https://api.gemini.com/v1/pubticker/btcusd
```


这里的“GET”是动词，后边的 URI 是“Ticker”这个资源的地址。所以，这是一个 RESTful 的接口。

但下面这样的接口，就不是一个严格的 RESTful 接口：

 复制代码


```
1 POST https://api.restful.cn/accounts/delete/:username
```

因为 URI 中包含动词“delete”（删除），所以这个 URI 并不是**指向一个资源**。如果要修改成严格的 RESTful 接口，我们可以把它改成下面这样：

 复制代码

```
1 DELETE https://api.rest.cn/accounts/:username
```

然后，我们带着这个观念去看 Gemini 的取消订单接口：

 复制代码

```
1 POST https://api.gemini.com/v1/order/cancel
```

参数	类型	参数描述
request	string	The literal string <code>"/v1/order/cancel"</code>
nonce	integer	一个递增的随机数字，用来保证服务器接收到的请求是有序的。参见： <a href="#">Private API Invocation</a>
order_id	integer	创建订单时生成的订单号。

注：Private API Invocation 的网址链接为<https://docs.gemini.com/rest-api/#private-api-invocation>

你会发现，这个接口不够“RESTful”的地方有：

动词设计不准确，接口使用“POST”而不是重用 HTTP 动词“DELETE”；

URI 里包含动词 cancel；

ID 代表的订单是**资源**，但订单 ID 是放在**参数列表**而不是**URI**里的，因此 URI 并没有指向资源。

所以严格来说，这不是一个 RESTful 的接口。

此外，如果我们去检查 Gemini 的其他私有接口（Private，私有接口是指需要附加身份验证信息才能访问的接口），我们会发现，那些接口的设计都不是严格 RESTful 的。不仅如此，大部分的交易所，比如 Bitmex、Bitfinex、OKCoin 等等，它们提供的“REST 接口”，也都不是严格 RESTful 的。这些接口之所以还能被称为“REST 接口”，是因为他们大部分满足了 REST 接口的另一个重要要求：**无状态**。

无状态的意思是，每个 REST 请求都是独立的，不需要服务器在会话（Session）中缓存中间状态来完成这个请求。简单来说，如果服务器 A 接收到请求的时候宕机了，而此时把这个请求发送给交易所的服务器 B，也能继续完成，那么这个接口就是无状态的。

这里，我再给你举一个简单的有状态的接口的例子。服务器要求，在客户端请求取消订单的时候，必须发送两次不一样的 HTTP 请求。并且，第一次发送让服务器“等待取消”；第二次发送“确认取消”。那么，就算这个接口满足了 RESTful 的动词、资源分离原则，也不是一个 REST 接口。

当然，对于交易所的 REST 接口，你并不需要过于纠结“RESTful”这个概念，否则很容易就被这些名词给绕晕了。你只需要把握住最核心的一点：**一个 HTTP 请求完成一次完整操作。**

## 交易所 API 简介

现在，你对 REST 和 Web Socket 应该有一个大致了解了吧。接下来，我们就开始做点有意思的事情。

首先，我来介绍一下交易所是什么。区块链交易所是个撮合交易平台：它兼容了传统撮合规则撮合引擎，将资金托管和交割方式替换为区块链。数字资产交易所，则是一个中心化的平台，通过 Web 页面或 PC、手机客户端的形式，让用户将数字资产充值到指定钱包地址（交易所创建的钱包），然后在平台挂买单、卖单以实现数字资产之间的兑换。

通俗来说，交易所就是一个买和卖的菜市场。有人在摊位上大声喊着：“二斤羊肉啊，二斤羊肉，四斤牛肉来换！”这种人被称为 maker（挂单者）。有的人则游走于不同摊位，不动声色地掏出两斤牛肉，顺手拿走一斤羊肉。这种人被称为 taker（吃单者）。

交易所存在的意义，一方面是为 maker 和 taker 提供足够的空间活动；另一方面，让一个名叫撮合引擎的玩意儿，尽可能地把单子撮合在一起，然后收取一定比例的保护费...啊不对，是手续费，从而保障游戏继续进行下去。

市场显然是个很伟大的发明，这里我们就不进行更深入的哲学讨论了。

然后，我再来介绍一个叫做 Gemini 的交易所。Gemini，双子星交易所，全球首个获得合法经营许可的、首个推出期货合约的、专注于撮合大宗交易的数字货币交易所。Gemini 位

于纽约，是一家数字货币交易所和托管机构，允许客户交易和存储数字资产，并直接受纽约州金融服务部门（NYDFS）的监管。

Gemini 的界面清晰，API 完整而易用，更重要的是，还提供了完整的测试网络，也就是说，功能和正常的 Gemini 完全一样。但是他家的交易采用虚拟币，非常方便从业者在平台上进行对接测试。

另一个做得很好的交易所，是 Bitmex，他家的 API UI 界面和测试网络也是币圈一流。不过，鉴于这家是期货交易所，对于量化初学者来说有一定的门槛，我们还是选择 Gemini 更方便一些。

在进入正题之前，我们最后再以比特币和美元之间的交易为例，介绍四个基本概念（orderbook 的概念这里就不介绍了，你也不用深究，你只需要知道比特币的价格是什么就行了）。

买（buy）：用美元买入比特币的行为。

卖（sell）：用比特币换取美元的行为。

市价单（market order）：给交易所一个方向（买或者卖）和一个数量，交易所把给定数量的美元（或者比特币）换成比特币（或者美元）的单子。

限价单（limit order）：给交易所一个价格、一个方向（买或者卖）和一个数量，交易所在价格达到给定价格的时候，把给定数量的美元（或者比特币）换成比特币（或者美元）的单子。

这几个概念都不难懂。其中，市价单和限价单，最大的区别在于，限价单多了一个给定价格。如何理解这一点呢？我们可以来看下面这个例子。

大宝在某一天中午 12:00:00，告诉交易所，我要用一千美元买比特币。交易所收到消息，在 12:00:01 回复小明，现在你的账户多了 0.099 个比特币，少了 1000 美元，交易成功。这是一个市价买单。

而小宝在某一天中午 11:59:00，告诉交易所，我要挂一个单子，数量为 0.1 比特币，价格为 1000 美元，低于这个价格不卖。交易所收到消息，在 11:59:01 告诉小强，挂单成功，你的账户余额中 0.1 比特币的资金被冻结。又过了一分钟，交易所告诉小强，你的单子被

完全执行了（fully executed），现在你的账户多了 1000 美元，少了 0.1 个比特币。这就是一个限价卖单。

（这里肯定有人发现不对了：貌似少了一部分比特币，到底去哪儿了呢？嘿嘿，你不妨自己猜猜看。）

显然，市价单，在交给交易所后，会立刻得到执行，当然执行价格也并不受你的控制。它很快，但是也非常不安全。而限价单，则限定了交易价格和数量，安全性相对高很多。缺点呢，自然就是如果市场朝相反方向走，你挂的单子可能没有任何人去接，也就变成了干吆喝却没人买。因为我没有讲解 orderbook，所以这里的说辞不完全严谨，但是对于初学者理解今天的内容，已经够用了。

储备了这么久的基础知识，想必你已经跃跃欲试了吧？下面，我们正式进入正题，手把手教你使用 API 下单。

## 手把手教你使用 API 下单

手动挂单显然太慢，也不符合量化交易的初衷。我们就来看看如何用代码实现自动化下单吧。

第一步，你需要做的是，注册一个 Gemini Sandbox 账号。请放心，这个测试账号不需要你充值任何金额，注册后即送大量虚拟现金。这口吻是不是听着特像网游宣传语，接下来就是“快来贪玩蓝月里找我吧”？哈哈，不过这个设定确实如此，所以赶紧来注册一个吧。

注册后，为了满足好奇，你可以先尝试着使用 web 界面自行下单。不过，事实上，未解锁的情况下是无法正常下单的，因此这样尝试并没啥太大意义。


所以第二步，我们需要来配置 API Key。User Settings，API Settings，然后点 GENERATE A NEW ACCOUNT API KEY.，记下 Key 和 Secret 这两串字符。因为窗口一旦消失，这两个信息就再也找不到了，需要你重新生成。

配置到此结束。接下来，我们来看具体实现。

先强调一点，在量化系统开发的时候，你的心中一定要有清晰的数据流图。下单逻辑是一个很简单的 RESTful 的过程，和你在网页操作的一样，构造你的请求订单、加密请求，然后 post 给 gemini 交易所即可。



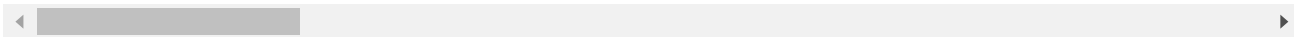
不过，因为涉及到的知识点较多，带你一步一步从零来写代码显然不太现实。所以，我们采用“先读懂后记忆并使用”的方法来学，下面即为这段代码：

 复制代码

```
1 import requests
2 import json
3 import base64
4 import hmac
5 import hashlib
6 import datetime
7 import time
8
9 base_url = "https://api.sandbox.gemini.com"
10 endpoint = "/v1/order/new"
11 url = base_url + endpoint
12
13 gemini_api_key = "account-zmidXEwP72yLSSybXVvn"
14 gemini_api_secret = "375b97HfE7E4tL8YaP3SJ239Pky9".encode()
15
16 t = datetime.datetime.now()
17 payload_nonce = str(int(time.mktime(t.timetuple())*1000))
18
19 payload = {
20     "request": "/v1/order/new",
21     "nonce": payload_nonce,
22     "symbol": "btcusd",
23     "amount": "5",
24     "price": "3633.00",
25     "side": "buy",
26     "type": "exchange limit",
27     "options": ["maker-or-cancel"]
28 }
29
30 encoded_payload = json.dumps(payload).encode()
31 b64 = base64.b64encode(encoded_payload)
32 signature = hmac.new(gemini_api_secret, b64, hashlib.sha384).hexdigest()
33
34 request_headers = {
35     'Content-Type': "text/plain",
36     'Content-Length': "0",
37     'X-GEMINI-APIKEY': gemini_api_key,
38     'X-GEMINI-PAYLOAD': b64,
39     'X-GEMINI-SIGNATURE': signature,
40     'Cache-Control': "no-cache"
41 }
42
43 response = requests.post(url,
44                           data=None,
45                           headers=request_headers)
46
```



```
47 new_order = response.json()
48 print(new_order)
49
50
51 ##### 输出 #####
52
53 {'order_id': '239088767', 'id': '239088767', 'symbol': 'btcusd', 'exchange': 'gemini',
```



我们来深入看一下这段代码。

RESTful 的 POST 请求，通过 `requests.post` 来实现。post 接受三个参数，url、data 和 headers。

这里的 url 等价于 <https://api.sandbox.gemini.com/v1/order/new>，但是在代码中分两部分写。第一部分是交易所 API 地址；第二部分，以斜杠开头，用来表示统一的 API endpoint。我们也可以在其他交易所的 API 中看到类似的写法，两者连接在一起，就构成了最终的 url。

而接下来大段命令的目的，是为了构造 `request_headers`。

这里我简单说一下 HTTP request，这是互联网中基于 TCP 的基础协议。HTTP 协议是 Hyper Text Transfer Protocol（超文本传输协议）的缩写，用于从万维网（WWW:World Wide Web）服务器传输超文本到本地浏览器的传送协议。而 TCP（Transmission Control Protocol）则是面向连接的、可靠的、基于字节流的传输层通信协议。

多提一句，如果你开发网络程序，建议利用闲暇时间认真读一读《计算机网络：自顶向下方》这本书，它也是国内外计算机专业必修课中广泛采用的课本之一。一边学习，一边应用，对于初学者的能力提升是全面而充分的。

回到 HTTP，它的主要特点是，连接简单、灵活，可以使用“简单请求，收到回复，然后断开连接”的方式，也是一种无状态的协议，因此充分符合 RESTful 的思想。

HTTP 发送需要一个请求头（request header），也就是代码中的 `request_headers`，用 Python 的语言表示，就是一个 str 对 str 的字典。

这个字典里，有一些字段有特殊用途，`'Content-Type': "text/plain"` 和 `'Content-Length': "0"` 描述 Content 的类型和长度，这里的 Content 对应于参数 data。但是 Gemini 这里的 request 的 data 没有任何用处，因此长度为 0。

还有一些其他字段，例如 `'keep-alive'` 来表示连接是否可持续化等，你也可以适当注意一下。要知道，网络编程很多 bug 都会出现在不起眼的细节之处。

继续往下走看代码。payload 是一个很重要的字典，它用来存储下单操作需要的所有的信息，也就是业务逻辑信息。这里我们可以下一个 limit buy，限价买单，价格为 3633 刀。

另外，请注意 nonce，这是个很关键并且在网络通信中很常见的字段。

因为网络通信是不可靠的，一个信息包有可能会丢失，也有可能重复发送，在金融操作中，这两者都会造成很严重的后果。丢包的话，我们重新发送就行了；但是重复的包，我们需要去重。虽然 TCP 在某种程度上可以保证，但为了在应用层面进一步减少错误发生的机会，Gemini 交易所要求所有的通信 payload 必须带有 nonce。

nonce 是个单调递增的整数。当某个后来的请求的 nonce，比上一个成功收到的请求的 nonce 小或者相等的时候，Gemini 便会拒绝这次请求。这样一来，重复的包就不会被执行两次了。另一方面，这样也可以在一定程度上防止中间人攻击：

- 一则是因为 nonce 的加入，使得加密后的同样订单的加密文本完全混乱；

- 二则是因为，这会使得中间人无法通过“发送同样的包来构造重复订单”进行攻击。

这样的设计思路是不是很巧妙呢？这就相当于每个包都增加了一个身份识别，可以极大地提高安全性。希望你也可以多注意，多思考一下这些巧妙的用法。

接下来的代码就很清晰了。我们要对 payload 进行 base64 和 sha384 算法非对称加密，其中 `gemini_api_secret` 为私钥；而交易所存储着公钥，可以对你发送的请求进行解密。最后，代码再将加密后的请求封装到 `request_headers` 中，发送给交易所，并收到 response，这个订单就完成了。

## 总结

这节课我们介绍了什么是 RESTful API，带你了解了交易所的 RESTful API 是如何工作的，以及如何通过 RESTful API 来下单。同时，我简单讲述了网络编程中的一些技巧操作，希望你在网络编程中要注意思考每一个细节，尽可能在写代码之前，对业务逻辑和具体的技术细节有足够清晰的认识。

下一节，我们同样将从 Web Socket 的定义开始，讲解量化交易中数据模块的具体实现。

## 思考题

最后留一个思考题。今天的内容里，能不能使用 timestamp 代替 nonce？为什么？欢迎留言写下你的思考，也欢迎你把这篇文章分享出去。

 极客时间

# Python 核心技术与实战

---

系统提升你的 Python 能力

---

景霄

Facebook 资深工程师



新版升级：点击「 请朋友读」，20位好友免费读，邀请订阅更有**现金**奖励。

© 版权归极客邦科技所有，未经许可不得传播售卖。页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 33 | 带你初探量化世界

下一篇 35 | RESTful & Socket: 行情数据对接和抓取

## 精选留言 (11)

 写留言



**SCAR**

2019-07-26

思考题：

1. 纯粹使用timestamp应该不行，虽然timestamp也是递增的，但是在python里timestamp是float而不是int。

2. 但如果基于timestamp抽取出部分应该是可以，比如老师例子中的：

```
payload_nonce = str(int(time.mktime(t.timetuple())*1000))...
```

展开 ▾



6



**Monroe He**

2019-07-26

我想问一下老师，有针对国内股票的虚拟交易平台吗

可以提供一下相关方面的书籍资料吗



2



**小侠龙旋风**

2019-07-27

知识点很多，整理一下。

1. 非对称加密：

加密：公钥加密，私钥解密；

签名：私钥签名，公钥验签。

2. hmac.new(key, str, digestmod)...

展开 ▾



1



**瞳梦**

2019-07-26

请问gemini sandbox账号怎么注册呢？我在官网只找到了Open a Personal Account和Represent an Institution

展开 ▾



1



**Geek\_adeba6**

2019-07-26

想请问如果想实现秒级别的市场行情获取，生产环境下的最佳实践是什么？



**Xg huang**

2019-07-26

哈哈，深入浅出，赞一个

不过有个地方是否写错？"而小宝在某一天中午 11:59:00，告诉交易所，我要挂一个单子，数量为 0.1 比特币，价格为 10000 美元，低于这个价格不卖。"

...

展开 ▾

编辑回复: 是的，我修改了



**蜉蝣**

2019-07-26

大家为什么都不运行一下 `time.mktime(datetime.datetime.now().timetuple())` 就在说不能代替的问题



**许童童**

2019-07-26

老师讲得好啊，妙啊！

展开 ▾



**code2**

2019-07-26

HTTP 协议是 Hyper Text Transfer Protocol 翻译过来不是超文本传输协议，提出这个协议的作者在其博士论文中有明确说明。

展开 ▾



**程序员人生**



2019-07-26

timestamp应该不能代替nonce。

当某个后请求的nonce，比上一个成功收到请求的nonce小或者等于时候，服务器会拒绝接收。

但timestamp不行，因为后请求的timestamp，可能会由于各种原因先到服务器，先请求的可能会晚到，并不能体现先后次序。...

展开 ▾



**SuQiu**

2019-07-26

timestamp也属于自增长，猜测是由于他的可预见性，所以不能代替nonce

