#### 

中国市场数据 API

**注意：如果您对我们的实时市场交易数据更感兴趣，请参考我们最新版的websocket API和FIX API文档：**

* <http://btcchina.org/websocket-api-market-data-documentation-zh>
* <http://btcchina.org/fix-api-documentation-zh>

市场数据API是公开使用的, 无需进行身份验证。

对于每个API，您可以提供您要查询的市场。默认为 “btccny”, 这是 BTC／CNY 的市场。需要 LTC／CNY，请使用 “market=ltccny”, 即 “<https://data.btcchina.com/data/ticker?market=ltccny>”. 需要 LTC／BTC，请使用 “market=ltcbtc”。

请注意， 过频访问可能导致IP拦截。

市场数据 API V1.3.3

2014-11-21 市场数据API v1.3.3: 增加了Q&A部分.

市场数据 API V1.3.2

2014-09-11 市场数据API v1.3.2: 为行情API增加了一个新的返回值”open”,增加了FIX API文档链接.

市场数据 API V1.3.1

2014-08-21 市场数据API v1.3.1: 为行情API增加了两个新的返回值”vwap”和”prev\_close”。

市场数据 API V1.3

2014-08-15 市场数据API v1.3为买卖订单API增加了“limit”参数。

市场数据 API V1.2

2014-07-31 市场数据API v1.2为交易历史API增加了“sincetype”参数。

市场数据 API V1.1

2014-07-09 市场数据API v1.1为交易历史API增加了limit参数.

市场数据 API V1

市场数据API v1提供了行情，交易，交易历史，买卖订单四种查询方法。

行 情

获取最新的行情数据。使用参数 market 来指明单个市场或所有市场。

* <https://data.btcchina.com/data/ticker?market=btccny>
* <https://data.btcchina.com/data/ticker?market=all>

## 单个市场 ##

{

"ticker": {

"high": "2894.97",

"low": "2850.08",

"buy": "2876.92",

"sell": "2883.80",

"last": "2875.66",

"vol": "4133.63800000",

"date": 1396412995,

"vwap": 2879.12,

"prev\_close": 2875.61,

"open": 2880.01

}

}

## 所有市场 ##

{

"ticker\_btccny": {

"high": "2894.97",

"low": "2850.08",

"buy": "2880.00",

"sell": "2883.86",

"last": "2880.00",

"vol": "4164.41040000",

"date": 1396412841,

"vwap": 2879.12,

"prev\_close": 2875.61,

"open": 2880.01

},

"ticker\_ltccny": {

"high": "78.80",

"low": "77.50",

"buy": "78.22",

"sell": "78.35",

"last": "78.35",

"vol": "56443.71000000",

"date": 1396412841,

"vwap": 78.12,

"prev\_close": 78.61,

"open": 78.62

},

"ticker\_ltcbtc": {

"high": "0.02800000",

"low": "0.02710000",

"buy": "0.02720000",

"sell": "0.02730000",

"last": "0.02720000",

"vol": "7715.69400000",

"date": 1396412841,

"vwap": 0.0274,

"prev\_close": 0.0273,

"open": 0.0272

}

}

|  |  |  |
| --- | --- | --- |
| **参数名** | **类型** | **描述** |
| high | string | 近24小时内最高价格 |
| low | string | 近24小时内最低价格 |
| buy | string | 最高出价 |
| sell | string | 最低要价 |
| last | string | 最新成交价格 |
| vol | string | 近24小时内比特币成交量 |
| date | number | 最新更新时间 |
| vwap | number | 近24小时内平均成交价 |
| prev\_close | number | 昨日收盘价 |
| open | number | 今日开盘价 |

交 易

获取过去24小时内的交易历史,注意,为了保证服务质量和响应速度,返回的交易数量上限为10000个。

* <https://data.btcchina.com/data/trades>

[{

"date":"1383286640",

"price":1264.31,

"amount":0.01,

"tid":"680350"

},{

"date":"1383286877",

"price":1264.33,

"amount":1,

"tid":"680351"

},{

"date":"1383286880",

"price":1264.79,

"amount":0.3,

"tid":"680352"

}]

|  |  |  |
| --- | --- | --- |
| **参数名** | **类型** | **描述** |
| date | string | Unix的时间（秒）自1970年1月1日 |
| price | string | 1个比特币的价格 |
| amount | string | 成交的比特币总量 |
| tid | string | 交易单号 |

交易历史

获取交易历史的清单，可通过设置since后的参数来获取较早的历史记录，可通过设置limit后的参数来获取指定数量的历史纪录，limit的默认值是100，有效区间是[0,5000]. 可通过设置sincetype参数为“id”或者“time”来指定since后的参数作用在哪个数据上，默认sincetype为id。

* <https://data.btcchina.com/data/historydata> //返回当前时间以前的100条交易记录.
* <https://data.btcchina.com/data/historydata?limit=10> //返回当前时间以前的10条交易记录.
* <https://data.btcchina.com/data/historydata?since=5000> //返回单号为5000以后的100条交易记录.
* <https://data.btcchina.com/data/historydata?since=5000&limit=10> //返回单号为5000以后的10条交易记录.
* <https://data.btcchina.com/data/historydata?since=1406794449&limit=10&sincetype=time> //返回unix时间为1406794449以后的10条交易记录.

[{

"date":"1383374245",

"price":1278.7,

"amount":0.4,

"tid":"684284",

"type":"sell"

},{

"date":"1383374245",

"price":1278.77,

"amount":0.35,

"tid":"684285",

"type":"buy"

}

]

|  |  |  |
| --- | --- | --- |
| **参数名** | **类型** | **描述** |
| date | string | Unix的时间（秒）自1970年1月1日 |
| price | string | 1个比特币的价格 |
| amount | string | 成交的比特币总量 |
| tid | string | 交易单号 |
| type | string | 表示交易为“买”或者“卖” |

买卖订单

订单数据默认包含所有公开的要价和出价。 可通过设置limit后的参数来获取指定数量的订单数据。

* <https://data.btcchina.com/data/orderbook> //返回BTCCNY市场所有订单.
* <https://data.btcchina.com/data/orderbook?limit=10> //返回BTCCNY市场各10个买卖订单，其中买单是价格最高的十个，卖单为价格最低的十个，返回值均按照价格降序排序.
* <https://data.btcchina.com/data/orderbook?market=ltccny&limit=10> //返回LTCCNY市场各10个买卖订单，其中买单是价格最高的十个，卖单为价格最低的十个，返回值均按照价格降序排序.

{"

asks":[

[1279.1,3.036],

[1278.8,7.9],

[1278.77,8.9],

[1278.7,2],

[1278.68,0.359]

],

"bids":[

[1278.5,1.056],

[1277.61,1.833],

[1277.6,7.7],

[1277.3,3],

[1277.28,9.99],

],

"date": 1395989337

}

|  |  |  |
| --- | --- | --- |
| **参数名** | **类型** | **描述** |
| asks | array | [0] 要价  [1] 买比特币数量 |
| bids | array | [0] 出价  [1] 卖比特币数量 |
| date | number | 最新更新时间 |

Q&A

Question 1. 为什么在交易历史中的交易的tid有时候是不连续的？

Answer: tid的不连续是由于MySql数据库本身的实现导致的，API没有隐藏任何交易数据或者删除过任何交易历史记录.

交易 API 访问认证

要进行交易 API 的访问认证，您需要将公开的访问**密匙 (Access Key)** 以及通过**秘密密匙 (Secret Key)** 签名得到的一个哈希字符串传递给交易 API。**签名**所采用的算法是 HMAC (Hash-based Message Authentication Code)。除了访问密匙和秘密密匙以外，您还需要传递一个“**Tonce**”参数。Tonce 是指以毫秒为单位的当前时间的时间戳 (Timestamp)。访问密匙和秘密密匙生成的签名通过 HTTP 基本身份验证 (HTTP Basic Authentication) 传递，Tonce 通过 HTTP 头 (HTTP Header) 传递。

详细步骤

1. 使用如下强制的参数创建签名字符串。将键值对使用“&”符号按照下面列表的顺序连接在一起。请注意连接顺序很重要。所有的键名必须添加，但是键值可以为空 (例如 params)。

* tonce (以毫秒为单位的时间戳，请确保您的系统时间准确)
* accesskey (访问密匙，您可以在您的账户管理页面申请)
* method (HTTP 请求方法，目前仅支持“post”)
* id (JSON-RPC 请求 id)
* method method (JSON-RPC 方法名称)
* params (JSON-RPC 方法参数)

### Example 1 ###

tonce=1377743828095093

&accesskey=1d87effa-e84d-48c1-a172-0232b86305dd

&requestmethod=post

&id=1

&method=getAccountInfo

&params=

### Example 2 ###

tonce=1377743828095093

&accesskey=1d87effa-e84d-48c1-a172-0232b86305dd

&requestmethod=post

&id=1

&method=buyOrder

&params=500,1

2. 使用您的秘密密匙用 HMAC 生成哈希签名。使用 **sha1** 作为哈希算法。

### PHP ###

hash\_hmac('sha1', $signature, $secretkey)

3. 使用 HTTP 基本身份验证传递身份验证信息。您可以通过如下两种方式实现：

* 在网址中传递。您可以在网址中直接传递访问密匙和哈希签名

https://<accesskey>:<hash>@api.btcchina.com/api\_trade\_v1.php

* HTTP 认证头。计算并创建认证头 (Authorization)，并和其他请求一起传递到 API。身份信息为用冒号 (:) 连接的访问密匙和哈希签名。最后的认证头的值需要用 Base64 编码。

### PHP ###

base64\_encode(<accesskey>:<hash>) //PGFjY2Vzc2tleT46PGhhc2g+

# HTTP HEADER

Authorization: Basic PGFjY2Vzc2tleT46PGhhc2g+

4. 最后您需要将 Tonce 也作为 HTTP 头传递给 API。Tonce 的值必须和您在第 1 步生成哈希签名时使用的值一致。

# HTTP HEADER

Json-Rpc-Tonce: 1377743828095093

5. 发起 API 请求。如果有错误，API 会返回 HTTP 401 Unauthorized。

API 示例代码

点击以下链接，获取更多示例源代码:

* PHP: <https://github.com/BTCChina/btcchina-api-php>
* Python: <https://github.com/BTCChina/btcchina-api-python>
* Java: <https://github.com/BTCChina/btcchina-api-java>
* .NET C#: <https://github.com/BTCChina/btcchina-api-csharp>
* C++: <https://github.com/BTCChina/btcchina-api-cpp>
* JavaScript: <https://github.com/BTCChina/btcchina-api-js>

PHP

**<?php**

**function** sign($method, $params = [array](http://www.php.net/array)()){

$accessKey = "YOUR\_ACCESS\_KEY";

$secretKey = "YOUR\_SECRET\_KEY";

$mt = [explode](http://www.php.net/explode)(' ', [microtime](http://www.php.net/microtime)());

$ts = $mt[1] . [substr](http://www.php.net/substr)($mt[0], 2, 6);

$signature = [urldecode](http://www.php.net/urldecode)([http\_build\_query](http://www.php.net/http_build_query)([array](http://www.php.net/array)(

'tonce' => $ts,

'accesskey' => $accessKey,

'requestmethod' => 'post',

'id' => 1,

'method' => $method,

'params' => '', *//implode(',', $params),*

)));

[var\_dump](http://www.php.net/var_dump)($signature);

$hash = [hash\_hmac](http://www.php.net/hash_hmac)('sha1', $signature, $secretKey);

return [array](http://www.php.net/array)(

'ts' => $ts,

'hash' => $hash,

'auth' => [base64\_encode](http://www.php.net/base64_encode)($accessKey.':'. $hash),

);

}

**function** request($method, $params){

$sign = sign($method, $params);

$options = [array](http://www.php.net/array)(

CURLOPT\_HTTPHEADER => [array](http://www.php.net/array)(

'Authorization: Basic ' . $sign['auth'],

'Json-Rpc-Tonce: ' . $sign['ts'],

),

);

$postData = [json\_encode](http://www.php.net/json_encode)([array](http://www.php.net/array)(

'method' => $method,

'params' => $params,

'id' => 1,

));

print($postData);

$headers = [array](http://www.php.net/array)(

'Authorization: Basic ' . $sign['auth'],

'Json-Rpc-Tonce: ' . $sign['ts'],

);

$ch = [curl\_init](http://www.php.net/curl_init)();

[curl\_setopt](http://www.php.net/curl_setopt)($ch, CURLOPT\_RETURNTRANSFER, true);

[curl\_setopt](http://www.php.net/curl_setopt)($ch, CURLOPT\_USERAGENT,

'Mozilla/4.0 (compatible; BTC China Trade Bot; '.[php\_uname](http://www.php.net/php_uname)('a').'; PHP/'.[phpversion](http://www.php.net/phpversion)().')'

);

[curl\_setopt](http://www.php.net/curl_setopt)($ch, CURLOPT\_URL, 'https://api.btcchina.com/api\_trade\_v1.php');

[curl\_setopt](http://www.php.net/curl_setopt)($ch, CURLOPT\_POSTFIELDS, $postData);

[curl\_setopt](http://www.php.net/curl_setopt)($ch, CURLOPT\_HTTPHEADER, $headers);

[curl\_setopt](http://www.php.net/curl_setopt)($ch, CURLOPT\_SSL\_VERIFYPEER, FALSE);

*// run the query*

$res = [curl\_exec](http://www.php.net/curl_exec)($ch);

return $res;

*/\*\*/*

}

try {

[var\_dump](http://www.php.net/var_dump)(request('getAccountInfo', [array](http://www.php.net/array)()));

} catch (Exception $e) {

echo "Error:".$e->getMessage();

}

?>

PYTHON

*#!/usr/bin/python*

*# -\*- coding: utf-8 -\*-*

import btcchina

access\_key="YOUR\_ACCESS\_KEY"

secret\_key="YOUR\_SECRET\_KEY"

bc = btcchina.BTCChina(access\_key,secret\_key)

''' These methods have no arguments '''

*#result = bc.get\_account\_info()*

*#print result*

*#result = bc.get\_market\_depth()*

*#print result*

*# NOTE: for all methods shown here, the transaction ID could be set by doing*

*#result = bc.get\_account\_info(post\_data={'id':'stuff'})*

*#print result*

''' buy and sell require price (CNY, 5 decimals) and amount (LTC/BTC, 8 decimals) '''

*#result = bc.buy(500,1)*

*#print result*

*#result = bc.sell(500,1)*

*#print result*

''' cancel requires id number of order '''

*#result = bc.cancel(2)*

*#print result*

''' request withdrawal requires currency and amount '''

*#result = bc.request\_withdrawal('BTC',0.1)*

*#print result*

''' get deposits requires currency. the optional "pending" defaults to true '''

*#result = bc.get\_deposits('BTC',pending=True)*

*#print result*

''' get orders returns status for one order if ID is specified,

otherwise returns all orders, the optional "open\_only" defaults to true '''

*#result = bc.get\_orders(2)*

*#print result*

*#result = bc.get\_orders(open\_only=True)*

*#print result*

''' get withdrawals returns status for one transaction if ID is specified,

if currency is specified it returns all transactions,

the optional "pending" defaults to true '''

*#result = bc.get\_withdrawals(2)*

*#print result*

*#result = bc.get\_withdrawals('BTC',pending=True)*

*#print result*

*#!/usr/bin/env python*

*# -\*- coding: utf-8 -\*-*

import time

import re

import hmac

import hashlib

import base64

import httplib

import json

class BTCChina():

def \_\_init\_\_(**self**,access=**None**,secret=**None**):

**self**.access\_key=access

**self**.secret\_key=secret

**self**.conn=httplib.HTTPSConnection("api.btcchina.com")

def \_get\_tonce(**self**):

return **int**(time.time()\*1000000)

def \_get\_params\_hash(**self**,pdict):

pstring=""

*# The order of params is critical for calculating a correct hash*

fields=['tonce','accesskey','requestmethod','id','method','params']

for f in fields:

if pdict[f]:

if f == 'params':

*# Convert list to string, then strip brackets and spaces*

*# probably a cleaner way to do this*

param\_string=**str**(pdict[f]);

*# Replace None with empty string*

param\_string=param\_string.replace('None', '')

param\_string=re.sub("[**\[\]** ]","",param\_string)

param\_string=re.sub("'",'',param\_string)

pstring+=f+'='+param\_string+'&'

else:

pstring+=f+'='+**str**(pdict[f])+'&'

else:

pstring+=f+'=&'

pstring=pstring.strip('&')

print pstring

*# now with correctly ordered param string, calculate hash*

phash = hmac.new(**self**.secret\_key, pstring, hashlib.sha1).hexdigest()

return phash

def \_private\_request(**self**,post\_data):

*#fill in common post\_data parameters*

tonce=**self**.\_get\_tonce()

post\_data['tonce']=tonce

post\_data['accesskey']=**self**.access\_key

post\_data['requestmethod']='post'

*# If ID is not passed as a key of post\_data, just use tonce*

if not 'id' in post\_data:

post\_data['id']=tonce

pd\_hash=**self**.\_get\_params\_hash(post\_data)

*# must use b64 encode*

auth\_string='Basic '+base64.b64encode(**self**.access\_key+':'+pd\_hash)

headers={'Authorization':auth\_string,'Json-Rpc-Tonce':tonce}

*#post\_data dictionary passed as JSON*

**self**.conn.request("POST",'/api\_trade\_v1.php',json.dumps(post\_data),headers)

response = **self**.conn.getresponse()

*# check response code, ID, and existence of 'result' or 'error'*

*# before passing a dict of results*

if response.status == 200:

*# this might fail if non-json data is returned*

resp\_dict = json.loads(response.read())

*# The id's may need to be used by the calling application,*

*# but for now, check and discard from the return dict*

if **str**(resp\_dict['id']) == **str**(post\_data['id']):

if 'result' in resp\_dict:

return resp\_dict['result']

elif 'error' in resp\_dict:

return resp\_dict['error']

else:

*# not great error handling....*

print "status:",response.status

print "reason:",response.reason

return **None**

def get\_account\_info(**self**,post\_data={}):

post\_data['method']='getAccountInfo'

post\_data['params']=[]

return **self**.\_private\_request(post\_data)

def get\_market\_depth(**self**,post\_data={}):

post\_data['method']='getMarketDepth'

post\_data['params']=[]

return **self**.\_private\_request(post\_data)

def buy(**self**,price,amount,post\_data={}):

post\_data['method']='buyOrder2'

if price is **None**:

post\_data['params']=[**None**,"{0:.4f}".format(**round**(amount,4))]

else:

post\_data['params']=["{0:.4f}".format(**round**(price,4)),"{0:.4f}".format(**round**(amount,4))]

return **self**.\_private\_request(post\_data)

def sell(**self**,price,amount,post\_data={}):

post\_data['method']='sellOrder2'

if price is **None**:

post\_data['params']=[**None**,"{0:.4f}".format(**round**(amount,4))]

else:

post\_data['params']=["{0:.4f}".format(**round**(price,4)),"{0:.4f}".format(**round**(amount,4))]

return **self**.\_private\_request(post\_data)

def cancel(**self**,order\_id,post\_data={}):

post\_data['method']='cancelOrder'

post\_data['params']=[order\_id]

return **self**.\_private\_request(post\_data)

def request\_withdrawal(**self**,currency,amount,post\_data={}):

post\_data['method']='requestWithdrawal'

post\_data['params']=[currency,amount]

return **self**.\_private\_request(post\_data)

def get\_deposits(**self**,currency='BTC',pending=**True**,post\_data={}):

post\_data['method']='getDeposits'

if pending:

post\_data['params']=[currency]

else:

post\_data['params']=[currency,'false']

return **self**.\_private\_request(post\_data)

def get\_orders(**self**,**id**=**None**,open\_only=**True**,post\_data={}):

*# this combines getOrder and getOrders*

if **id** is **None**:

post\_data['method']='getOrders'

if open\_only:

post\_data['params']=[]

else:

post\_data['params']=['false']

else:

post\_data['method']='getOrder'

post\_data['params']=[**id**]

return **self**.\_private\_request(post\_data)

def get\_withdrawals(**self**,**id**='BTC',pending=**True**,post\_data={}):

*# this combines getWithdrawal and getWithdrawls*

try:

**id** = **int**(**id**)

post\_data['method']='getWithdrawal'

post\_data['params']=[**id**]

except:

post\_data['method']='getWithdrawals'

if pending:

post\_data['params']=[**id**]

else:

post\_data['params']=[**id**,'false']

return **self**.\_private\_request(post\_data)

JAVA

import *javax.crypto.spec.SecretKeySpec*;

import *javax.crypto.Mac*;

import *javax.net.ssl.HttpsURLConnection*;

import *java.io.BufferedReader*;

import *java.io.DataOutputStream*;

import *java.io.InputStreamReader*;

import *java.net.URL*;

import *javax.xml.bind.DatatypeConverter*;

class BTCChinaApiAuthentication{

private static final [String](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+string) ACCESS\_KEY = "YOUR\_ACCESS\_KEY";

private static final [String](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+string) SECRET\_KEY = "YOUR\_SECRET\_KEY";

private static final [String](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+string) HMAC\_SHA1\_ALGORITHM = "HmacSHA1";

public static [String](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+string) getSignature([String](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+string) data,[String](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+string) key) throws [Exception](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+exception) {

*// get an hmac\_sha1 key from the raw key bytes*

SecretKeySpec signingKey = new SecretKeySpec(key.getBytes(), HMAC\_SHA1\_ALGORITHM);

*// get an hmac\_sha1 Mac instance and initialize with the signing key*

Mac mac = Mac.getInstance(HMAC\_SHA1\_ALGORITHM);

mac.init(signingKey);

*// compute the hmac on input data bytes*

byte[] rawHmac = mac.doFinal(data.getBytes());

return bytArrayToHex(rawHmac);

}

private static [String](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+string) bytArrayToHex(byte[] a) {

StringBuilder sb = new StringBuilder();

for(byte b: a)

sb.append([String](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+string).format("%02x", b&0xff));

return sb.toString();

}

public static void main([String](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+string) args[]) throws [Exception](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+exception){

[String](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+string) tonce = ""+([System](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+system).currentTimeMillis() \* 1000);

[String](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+string) params = "tonce="+tonce.toString()+"&accesskey="+ACCESS\_KEY+"&requestmethod=post&id=1&method=getAccountInfo&params=";

[String](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+string) hash = getSignature(params, SECRET\_KEY);

[String](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+string) url = "https://api.btcchina.com/api\_trade\_v1.php";

[URL](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+url) obj = new [URL](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+url)(url);

HttpsURLConnection con = (HttpsURLConnection) obj.openConnection();

[String](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+string) userpass = ACCESS\_KEY + ":" + hash;

[String](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+string) basicAuth = "Basic " + DatatypeConverter.printBase64Binary(userpass.getBytes());

*//add reuqest header*

con.setRequestMethod("POST");

con.setRequestProperty("Json-Rpc-Tonce", tonce.toString());

con.setRequestProperty ("Authorization", basicAuth);

[String](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+string) postdata = "{**\"**method**\"**: **\"**getAccountInfo**\"**, **\"**params**\"**: [], **\"**id**\"**: 1}";

*// Send post request*

con.setDoOutput(**true**);

[DataOutputStream](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+dataoutputstream) wr = new [DataOutputStream](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+dataoutputstream)(con.getOutputStream());

wr.writeBytes(postdata);

wr.flush();

wr.close();

int responseCode = con.getResponseCode();

[System](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+system).out.println("**\n**Sending 'POST' request to URL : " + url);

[System](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+system).out.println("Post parameters : " + postdata);

[System](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+system).out.println("Response Code : " + responseCode);

[BufferedReader](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+bufferedreader) in = new [BufferedReader](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+bufferedreader)(

new [InputStreamReader](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+inputstreamreader)(con.getInputStream()));

[String](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+string) inputLine;

[StringBuffer](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+stringbuffer) response = new [StringBuffer](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+stringbuffer)();

while ((inputLine = in.readLine()) != **null**) {

response.append(inputLine);

}

in.close();

*//print result*

[System](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+system).out.println(response.toString());

}

}

.NET C#

using System;

using System.Net;

using System.Text;

using System.Collections.Specialized;

using System.Collections.Generic;

using System.Security.Cryptography;

using System.IO;

using System.Net.Security;

using System.Security.Cryptography.X509Certificates;

namespace BTCChinaApiAuthentication {

class MainClass {

public static void Main (string[] args) {

*// For https.*

ServicePointManager.ServerCertificateValidationCallback = delegate { return true; };

*// Enter your personal API access key and secret here*

string accessKey = "YOUR\_ACCESS\_KEY";

string secretKey = "YOUR\_SECRET\_KEY";

string method = "getAccountInfo";

TimeSpan timeSpan = DateTime.UtcNow - [new](http://www.google.com/search?q=new+msdn.microsoft.com) DateTime (1970, 1, 1);

long milliSeconds = Convert.ToInt64(timeSpan.TotalMilliseconds \* 1000);

string tonce = Convert.ToString(milliSeconds);

NameValueCollection parameters = [new](http://www.google.com/search?q=new+msdn.microsoft.com) NameValueCollection() {

{ "tonce", tonce },

{ "accesskey", accessKey },

{ "requestmethod", "post" },

{ "id", "1" },

{ "method", method },

{ "params", "" }

};

string paramsHash = GetHMACSHA1Hash(secretKey, BuildQueryString(parameters));

string base64String = Convert.ToBase64String(

Encoding.ASCII.GetBytes(accessKey + ':' + paramsHash));

string url = "https://api.btcchina.com/api\_trade\_v1.php";

string postData = "{**\"**method**\"**: **\"**" + method + "**\"**, **\"**params**\"**: [], **\"**id**\"**: 1}";

SendPostByWebRequest(url, base64String, tonce, postData);

}

private static void SendPostByWebClient(string url, string base64,

string tonce, string postData) {

using (WebClient client = [new](http://www.google.com/search?q=new+msdn.microsoft.com) WebClient()) {

client.Headers["Content-type"] = "application/json-rpc";

client.Headers["Authorization"] = "Basic " + base64;

client.Headers["Json-Rpc-Tonce"] = tonce;

try {

byte[] response = client.UploadData(

url, "POST", Encoding.Default.GetBytes(postData));

Console.WriteLine("**\n**Response: {0}", Encoding.UTF8.GetString(response));

} catch (System.Net.WebException ex) {

Console.WriteLine(ex.Message);

}

}

}

public static void SendPostByWebRequest(string url, string base64,

string tonce, string postData) {

WebRequest webRequest = WebRequest.Create(url);

*//WebRequest webRequest = HttpWebRequest.Create(url);*

if (webRequest == null) {

Console.WriteLine("Failed to create web request for url: " + url);

return;

}

byte[] bytes = Encoding.ASCII.GetBytes(postData);

webRequest.Method = "POST";

webRequest.ContentType = "application/json-rpc";

webRequest.ContentLength = bytes.Length;

webRequest.Headers["Authorization"] = "Basic " + base64;

webRequest.Headers["Json-Rpc-Tonce"] = tonce;

try {

*// Send the json authentication post request*

using (Stream dataStream = webRequest.GetRequestStream()) {

dataStream.Write(bytes, 0, bytes.Length);

dataStream.Close();

}

*// Get authentication response*

using (WebResponse response = webRequest.GetResponse()) {

using (var stream = response.GetResponseStream()) {

using (var reader = [new](http://www.google.com/search?q=new+msdn.microsoft.com) StreamReader(stream)) {

Console.WriteLine("Response: " + reader.ReadToEnd());

}

}

}

} catch (WebException ex) {

Console.WriteLine(ex.Message);

}

}

private static string BuildQueryString(NameValueCollection parameters) {

List<string> keyValues = [new](http://www.google.com/search?q=new+msdn.microsoft.com) List<string>();

foreach (string key in parameters) {

keyValues.Add(key + "=" + parameters[key]);

}

return String.Join("&", keyValues.ToArray());

}

private static string GetHMACSHA1Hash(string secret\_key, string input) {

HMACSHA1 hmacsha1 = [new](http://www.google.com/search?q=new+msdn.microsoft.com) HMACSHA1(Encoding.ASCII.GetBytes(secret\_key));

MemoryStream stream = [new](http://www.google.com/search?q=new+msdn.microsoft.com) MemoryStream(Encoding.ASCII.GetBytes(input));

byte[] hashData = hmacsha1.ComputeHash(stream);

*// Format as hexadecimal string.*

StringBuilder hashBuilder = [new](http://www.google.com/search?q=new+msdn.microsoft.com) StringBuilder();

foreach (byte data in hashData) {

hashBuilder.Append(data.ToString("x2"));

}

return hashBuilder.ToString();

}

}

}

C++

*#include <iostream>*

*#include <math.h>*

*#include <netdb.h>*

*#include <openssl/ssl.h>*

*#include <openssl/err.h>*

*#include <stdio.h>*

*#include <sstream>*

*#include "HMAC\_SHA1.h"*

**using** **namespace** std;

*// Reference:*

*// http://www.codeproject.com/Articles/22118/C-Class-Implementation-of-HMAC-SHA*

string getHmacSha1(string key, string content) {

unsigned char digest[20];

CHMAC\_SHA1 HMAC\_SHA1;

HMAC\_SHA1.HMAC\_SHA1((unsigned char \*) content.c\_str(), content.size(),

(unsigned char \*) key.c\_str(), key.size(), digest);

stringstream output;

char result[3];

for (int i = 0; i < 20; i++) {

sprintf(result, "%02x", digest[i]);

output << result;

}

return output.str();

}

long long getMilliSeconds() {

struct timeval start, end;

gettimeofday( &start, **NULL** );

return start.tv\_sec \* 1000000LL + start.tv\_usec;

}

*// Connects and gets the socket handle. Returns 0 if any errors.*

int SocketConnect(string host\_name, int port) {

struct hostent\* host = gethostbyname(host\_name.c\_str());

int handle = socket(AF\_INET, SOCK\_STREAM, 0);

if (handle == -1) {

cout << "Error when creating socket to " << host\_name;

return 0;

}

struct sockaddr\_in server;

server.sin\_family = AF\_INET;

server.sin\_port = htons(port);

server.sin\_addr = \*((in\_addr \*) host->h\_addr);

bzero(&(server.sin\_zero), 8);

int error = connect(handle, (sockaddr \*) &server, sizeof(sockaddr));

if (error == -1) {

cout << "Error when connecting " << host\_name;

return 0;

}

return handle;

}

*// Establishes a SSL connection. Return false if errors.*

bool SslConnect(const string& server, int port,

int& socket, SSL\*& sslHandle, SSL\_CTX\*& sslContext) {

socket = SocketConnect(server, port);

if (socket == 0) {

return **false**;

}

SSL\_load\_error\_strings();

SSL\_library\_init();

*// Use SSL 2 or 3, bind SSL to socket, and then do the SSL connection.*

if ((sslContext = SSL\_CTX\_new(SSLv23\_client\_method())) == **NULL** ||

(sslHandle = SSL\_new(sslContext)) == **NULL** ||

SSL\_set\_fd(sslHandle, socket) != 1 ||

SSL\_connect(sslHandle) != 1) {

ERR\_print\_errors\_fp(**stderr**);

return **false**;

}

return **true**;

}

string ReadAllFromSSL(SSL\* ssl\_handle) {

const int BUCKET\_READ\_SIZE = 1024;

char buffer[BUCKET\_READ\_SIZE];

stringstream result;

int received\_bytes = std::numeric\_limits<int>::max();

while (received\_bytes >= BUCKET\_READ\_SIZE) {

received\_bytes = SSL\_read(ssl\_handle, buffer, BUCKET\_READ\_SIZE);

if (received\_bytes > 0) {

buffer[received\_bytes] = '\0';

result << buffer;

} else {

cout << "Error when read from SSL"; *// To get the error reason, use SSL\_get\_error.*

return "";

}

}

return result.str();

}

string Base64Encode(const string& message) {

*// Do encoding.*

BIO\* bio = BIO\_new(BIO\_f\_base64());

BIO\* bmem = BIO\_new(BIO\_s\_mem());

BIO\_push(bio, bmem);

BIO\_set\_flags(bio, BIO\_FLAGS\_BASE64\_NO\_NL); *// // Ignore newlines*

BIO\_write(bio, message.c\_str(), message.size());

BIO\_flush(bio);

*// Get results.*

char\* data;

int length = BIO\_get\_mem\_data(bmem, &data);

string encoded = string(data, length);

BIO\_free\_all(bio);

return encoded;

}

string GetBtcPostContent(const string& accessKey, const string& secretKey,

const string& method) {

*// Get authorization token.*

long long tonce = getMilliSeconds();

stringstream authInput;

authInput << "tonce=" << tonce

<< "&accesskey=" << accessKey

<< "&requestmethod=post&id=1&method=" << method

<< "&params=";

string paramsHash = getHmacSha1(secretKey, authInput.str());

string authToken = Base64Encode(accessKey + ":" + paramsHash);

*// Get post content.*

string json\_content = "{\"method\": \"getAccountInfo\", \"params\": [], \"id\": 1}";

stringstream postStream;

postStream << "POST /api\_trade\_v1.php HTTP/1.1\r\n"

<< "Content-Type: application/json-rpc\r\n"

<< "Authorization: Basic " << authToken << "\r\n"

<< "Json-Rpc-Tonce: " << tonce << "\r\n"

<< "Content-Length: " << json\_content.size() << "\r\n"

<< "Host: api.btcchina.com\r\n\r\n"

<< json\_content;

string postContent = postStream.str();

cout << "POST content: " << postStream.str() << endl;

return postContent;

}

int main(int argc, char \*\*argv) {

string accessKey = "YOUR\_ACCESS\_KEY";

string secretKey = "YOUR\_SECRET\_KEY";

string method = "getAccountInfo";

string postContent = GetBtcPostContent(accessKey, secretKey, method);

*// Start the real SSL request.*

int socket = 0;

SSL\* sslHandle = **NULL**;

SSL\_CTX\* sslContext = **NULL**;

if (SslConnect("api.btcchina.com", 443, socket, sslHandle, sslContext)) {

SSL\_write(sslHandle, postContent.c\_str(), postContent.size());

string response = ReadAllFromSSL(sslHandle);

cout << "Get response: " << response << endl;

}

*// Cleanups no matter connect succeed or not.*

if (socket != 0) {

close(socket);

}

if (sslHandle) {

SSL\_shutdown(sslHandle);

SSL\_free(sslHandle);

}

if (sslContext) {

SSL\_CTX\_free(sslContext);

}

return 0;

}

交易 API V2.0.1.3

2014-11-19 将[buyOrder2](http://btcchina.org/api-trade-documentation-zh#buyorder2)和[sellOrder2](http://btcchina.org/api-trade-documentation-zh#sellorder2)中的amount和price参数类型由number变为string。

交易 API V2.0.1.2

2014-11-17 为getAccountInfo增加了loan参数。

交易 API V2.0.1.1

2014-10-17 更新了[requestWithdrawal](http://btcchina.org/api-trade-documentation-zh#requestwithdrawal)方法的描述。

交易 API V2.0.1

2014-08-21 为[getTransactions](http://btcchina.org/api-trade-documentation-zh#gettransactions) API方法增加了两个参数: “since” and “sincetype”.

交易 API V2.0

2014-08-15 为所有市场发布了[止损止盈订单API方法](http://btcchina.org/api-trade-documentation-zh#止损止盈订单api方法) :

* [buyStopOrder](http://btcchina.org/api-trade-documentation-zh#buystoporder)
* [sellStopOrder](http://btcchina.org/api-trade-documentation-zh#sellstoporder)
* [getStopOrder](http://btcchina.org/api-trade-documentation-zh#getstoporder)
* [getStopOrders](http://btcchina.org/api-trade-documentation-zh#getstoporders)
* [cancelStopOrder](http://btcchina.org/api-trade-documentation-zh#cancelstoporder)

交易 API V1.9

2014-08-13 为 [getOrder](http://btcchina.org/api-trade-documentation-zh#getorder) 增加了withdetail参数支持; 为 [getOrders](http://btcchina.org/api-trade-documentation-zh#getorders) 增加了since和withdetail参数支持。

交易 API V1.8

2014-07-21 为 [getAccountInfo](http://btcchina.org/api-trade-documentation-zh#getaccountinfo) 增加了更多的参数支持: 增加了“all”，“balance”，“frozen”和“profile”等参数支持。

交易 API V1.7

2014-07-17 增加了所有市场的iceberg订单API支持:

* [buyIcebergOrder](http://btcchina.org/api-trade-documentation-zh#buyicebergorder)
* [sellIcebergOrder](http://btcchina.org/api-trade-documentation-zh#sellicebergorder)
* [getIcebergOrder](http://btcchina.org/api-trade-documentation-zh#geticebergorder)
* [getIcebergOrders](http://btcchina.org/api-trade-documentation-zh#geticebergorders)
* [cancelIcebergOrder](http://btcchina.org/api-trade-documentation-zh#cancelicebergorder)

交易 API V1.6

2014-07-10 增加了代码示例的Github链接。

交易 API V1.5.2

2014-04-02 本次交易API更新，减少了客户端请求量和网络带宽。

* 更新了[getOrders](http://btcchina.org/api-trade-documentation-zh#getorders) 和 [getMarketDepth2](http://btcchina.org/api-trade-documentation-zh#getmarketdepth2) ，现支持可将 ‘market’ 参数设置为 “ALL”。
* 增加了分页索引，可使 [getOrders](http://btcchina.org/api-trade-documentation-zh#getorders) 和 [getTransanctions](http://btcchina.org/api-trade-documentation-zh#gettransactions) 从索引开始返回结果。

交易 API V1.5.1

2014-03-28 本次更新包含一些小的新功能的完善以及缺陷的修复。

* 在getMarketDepth2里增加了“最新更新时间”；
* 标准化了 market currency参数，使之符合金融行业标准；
* getWithdrawals 和 getDeposits 返回空数组替换了原来返回Error的情况；
* 比特币交易现支持4位小数；
* 改进了 buyOrder2 和 sellOrder2 处理市场价（price =null）的方式；

交易 API V1.5

2014年3月18日，我们发布了比特币-莱特币（LTC/BTC）互换交易，请将货币参数修改为“ltcbtc”以进行交易。

交易 API V1.4

在API中增加对提现权限的控制。目前api权限（permission）的值为： 1-只读权限， 3-仅交易权限， 5-仅提现权限，7-交易以及提现权限

交易 API V1.3

交易 API v1.3将支持莱特币的交易，本次更新与现有的接口兼容。

更新日志:

* 增加了对莱特币的支持，所有受影响的方法增加了新的货币参数或者市场参数（null）。如果货币参数缺失，将以BTC和BTCCNY作为默认值。
* 增加了API密钥使用权限。
* 交易类型现增加了 “买莱特币”，“卖莱特币”，“莱特币充值”，“莱特币提现”，“莱特币退款”，文档更新“挂单返利”，”交易费用“。
* 比特币或莱特币提现新增状态 “cancel”, “refund”, “processing”

交易 API V1.2

交易 API v1.2 使用与 v1 相同的 URL：<https://api.btcchina.com/api_trade_v1.php>.

更新日志

* 普通列表项目新增方法buyOrder2 和 sellOrder2. 该方法将返回下单后的订单号. 原方法 buyOrder 和sellOrder 已过时。

交易 API V1.1

交易 API v1.1 使用与 v1 相同的 URL：<https://api.btcchina.com/api_trade_v1.php>.

更新日志

* 添加新的方法 [getTransanctions](http://btcchina.org/api-trade-documentation-zh#gettransactions)。
* [getMarketDepth](http://btcchina.org/api-trade-documentation-zh#getmarketdepth)方法过时了,请使用[getMarketDepth2](http://btcchina.org/api-trade-documentation-zh#getmarketdepth2)。
* 为所有 API 方法修正参数验证。
* 更新错误代码表。

交易 API V1

交易 API 是基于 JSON-RPC 2.0 实现的。 要了解更多关于 JSON-RCP 2.0 的规范，请参考其官方文档。

* <http://www.jsonrpc.org/specification>

各种流行编程语言的 JSON-RPC 实现可以在如下网址找到。

* <http://json-rpc.org/wiki/implementations>

错误代码

所有 API 方法调用在请求失败或遇到未知错误时会返回 JSON-RPC 错误对象。

|  |  |
| --- | --- |
| **Code** | **Message** |
| -32000 | 内部错误 |
| -32003 | 人民币账户余额不足 |
| -32004 | 比特币账户余额不足 |
| -32005 | 挂单没有找到 |
| -32006 | 无效的用户 |
| -32007 | 无效的货币 |
| -32008 | 无效的金额 |
| -32009 | 无效的钱包地址 |
| -32010 | 没有找到提现记录 |
| -32011 | 没有找到充值记录 |
| -32017 | 无效的类型 |
| -32018 | 无效的价格 |
| -32019 | 无效的参数 |
| -32025 | 订单已取消 |
| -32026 | 订单已完成 |
| -32062 | 市场深度不足以成交该市场交易 |
| -32065 | 无效的货币参数 |
| -32086 | 订单处理中 |

{"error":{

"code":-32003,

"message":"Insufficient CNY balance",

"id": 1

}

}

返回对象

成功的方法调用会返回如下对象。这些对象的示例会在方法文档中列出。

PROFILE

|  |  |  |
| --- | --- | --- |
| **名称** | **类型** | **描述** |
| username | string | 账户用户名 |
| trade\_password\_enabled | boolean | 指示该账户是否已设置交易密码 |
| otp\_enabled | boolean | 指示是否已启用双重认证 |
| trade\_fee | number | 比特币交易手续费 |
| trade\_fee\_cnyltc | number | 莱特币交易手续费 |
| trade\_fee\_btcltc | number | 比特币莱特币互换交易手续费 |
| daily\_btc\_limit | number | 账户每日比特币提现限额 |
| daily\_ltc\_limit | number | 账户每日莱特币提现限额 |
| btc\_deposit\_address | string | 比特币充值地址 |
| btc\_withdrawal\_address | string | 比特币提现地址 |
| ltc\_deposit\_address | string | 莱特币充值地址 |
| ltc\_withdrawal\_address | string | 莱特币提现地址 |
| api\_key\_permission | string | API交易权限， 1-只读权限， 3-仅交易权限， 5-仅提现权限，7-交易以及提现权限 |

BALANCE

|  |  |  |
| --- | --- | --- |
| **名称** | **类型** | **描述** |
| currency | string | 货币代码 |
| symbol | string | 货币符号 |
| amount | number | 账户余额 (小数精度) |
| amount\_integer | string | 账户余额 (整数精度) |
| amount\_decimal | integer | 在使用“amount\_integer”时小数点的位置 |

FROZEN

|  |  |  |
| --- | --- | --- |
| **名称** | **类型** | **描述** |
| currency | string | 货币代码 |
| symbol | string | 货币符号 |
| amount | number | 账户冻结金额 (小数精度) |
| amount\_integer | string | 账户冻结金额 (整数精度) |
| amount\_decimal | integer | 在使用“amount\_integer”时小数点的位置 |

LOAN

|  |  |  |
| --- | --- | --- |
| **名称** | **类型** | **描述** |
| currency | string | 货币代码 |
| symbol | string | 货币符号 |
| amount | number | 账户冻结金额 (小数精度) |
| amount\_integer | string | 账户冻结金额 (整数精度) |
| amount\_decimal | integer | 在使用“amount\_integer”时小数点的位置 |

ORDER

|  |  |  |  |
| --- | --- | --- | --- |
| **名称** | **类型** | **描述** |  |
| id | integer | 挂单 ID |  |
| type | string | 挂单类型。可能值：bid 或 ask |  |
| price | number | 挂单价格 |  |
| currency | string | 货币。可能值：CNY |  |
| amount | number | 挂单剩余数量。如果此值小于 amount\_original，说明此挂单仅有部分成交 |  |
| amount\_original | number | 初始挂单数量 |  |
| date | integer | Unix 时间戳。自1970年1月1日以来的秒数 |  |
| status | string | 状态。可能值：open、closed 、cancelled、pending 或 error |  |
| detail | object[] | 订单详情,可选返回值。返回对象数组：[order\_detail](http://btcchina.org/api-trade-documentation-zh#order_detail) |  |

WITHDRAWAL

|  |  |  |
| --- | --- | --- |
| **名称** | **类型** | **描述** |
| id | integer | 提现 ID |
| address | string | 比特币或者莱特币提现地址 |
| currency | string | 货币代码。可能值：BTC 或 LTC |
| amount | number | 提现数量 |
| date | integer | Unix 时间戳。自1970年1月1日以来的秒数 |
| transaction | string | 交易 ID |
| status | string | 提现状态。可能值：pending 、completed、 processing、 cancel、refund |

DEPOSIT

|  |  |  |
| --- | --- | --- |
| **名称** | **类型** | **描述** |
| id | integer | 存款 ID |
| address | string | 比特币或者莱特币充值地址 |
| currency | string | 货币代码。可能值：BTC 或 LTC |
| amount | number | 充值数量 |
| date | integer | Unix 时间戳。自1970年1月1日以来的秒数 |
| status | string | 充值状态。可能值：pending 或 completed |

MARKET\_DEPTH

|  |  |  |
| --- | --- | --- |
| **名称** | **类型** | **描述** |
| bid | object[] | 买单对象列表。价格从高到低排列 |
| ask | object[] | 卖单对象列表。价格从低到高排列 |

BID / ASK

|  |  |  |
| --- | --- | --- |
| **名称** | **类型** | **描述** |
| price | number | 1 BTC 或者 LTC 的价格 |
| amount | number | BTC 或者 LTC 数量 |

TRANSACTION

|  |  |  |
| --- | --- | --- |
| **名称** | **类型** | **描述** |
| id | integer | 交易 ID |
| type | string | 交易类型。'fundbtc | withdrawbtc | fundmoney | withdrawmoney | refundmoney | buybtc | sellbtc | refundbtc |tradefee| rebate | fundltc | refundltc | withdrawltc | buyltc | sellltc' |
| btc\_amount | number | 成交的 BTC 数量。负数表明是从账户余额中扣减 |
| btc\_amount | number | 成交的 LTC 数量。负数表明是从账户余额中扣减 |
| cny\_amount | number | 成交的 CNY 数量。负数表明是从账户余额中扣减 |
| date | integer | Unix 时间戳（自1970年1月1日以来经过的秒数） |

ICEBERG\_ORDER

|  |  |  |
| --- | --- | --- |
| **名称** | **类型** | **描述** |
| id | integer | Iceberg订单号. |
| type | string | [bid|ask] |
| price | number | 买/卖 1 BTC/LTC的价格. |
| market | string | [BTCCNY|LTCCNY|LTCBTC] |
| amount | number | Iceberg订单里剩余的未成交的BTC/LTC数量. |
| amount\_original | number | Iceberg订单里的BTC/LTC总量. |
| disclosed\_amount | number | 公开的每笔拆分订单的BTC/LTC数量，必须小于等于iceberg订单里的BTC/LTC总量. |
| variance | number | 为了使每笔拆分订单数量不同而设置的波动率. |
| date | integer | 精确到秒的Unix时间戳，起始值为1970年1月1日. |
| status | string | [ open | closed | cancelled | error ] |

ORDER\_DETAIL

|  |  |  |
| --- | --- | --- |
| **名称** | **类型** | **描述** |
| dateline | integer | Unix 时间戳. |
| price | number | 买/卖 BTC/LTC的成交单价. |
| amount | number | 成交的BTC/LTC数量. |

STOP\_ORDER

|  |  |  |
| --- | --- | --- |
| **名称** | **类型** | **描述** |
| id | integer | 止损止盈订单号. |
| type | string | [bid|ask] |
| stop\_price | number | 触发止损止盈订单的BTC/LTC单价. 如果设置了追单波动价格数量/率，触发单价将由系统自动调节. |
| trailing\_amount | number | 追单波动数量，设置后可动态决定止损止盈单价. |
| trailing\_percentage | number | 追单波动率，设置后可动态决定止损止盈单价. |
| price | number | 由止损止盈订单触发的订单中，所下订单的BTC/LTC单价. |
| market | string | [BTCCNY|LTCCNY|LTCBTC] |
| amount | number | 由止损止盈订单触发的订单中，所下订单的BTC/LTC数量. |
| date | integer | 止损止盈订单下单时间. |
| status | string | [ open | closed | cancelled | error ] |
| order\_id | integer | 由止损止盈订单触发的订单的订单号，如果止损止盈订单尚未触发或者已被取消则值为null. |

API 方法

BUYORDER

该方法已过时，请使用buyOrder2。

BUYORDER2

下比特币/莱特币买单。该方法将返回订单号。

参数

|  |  |  |  |
| --- | --- | --- | --- |
| **名称** | **类型** | **必选？** | **描述** |
| price | string | 是 | 买 1 比特币/莱特币所用人民币的价格，最多支持小数点后 2位精度。以市场价格交易，将 price 设置为 null |
| amount | string | 是 | 要买的比特币或者莱特币数量, BTC最多支持小数点后 4 位精度, LTC最多支持小数点后 3 位精度 |
| market | string | 否 | 可使用[BTCCNY],[LTCCNY],[LTCBTC] 默认值为BTCCNY |

JSON 请求示例

{"method":"buyOrder2","params":[\"500\",\"1\"],"id":1}

{"method":"buyOrder2","params":[\"500.01\",\"1.2312\",\"BTCCNY\"],"id":1}

## market order ##

{"method":"buyOrder2","params":[null,\"1\"],"id":1}

返回

|  |  |  |
| --- | --- | --- |
| **名称** | **类型** | **描述** |
| result | integer | 如果下单成功，返回订单号 |

JSON 响应示例

{"result":12345,"id":"1"}

CANCELORDER

取消一个还未完全成交的挂单，其状态应该为“open”。

参数

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **类型** | **必选？** | **描述** |
| id | number | 是 | 要取消的挂单的 ID |
| market | string | 否 | 可选值BTCCNY、LTCCNY、LTCBTC 默认值为BTCCNY |

JSON 请求示例

{"method":"cancelOrder","params":[2],"id":1}

返回

|  |  |  |
| --- | --- | --- |
| **Name** | **类型** | **描述** |
| result | boolean | 如果取消挂单成功，返回 true |

JSON 响应示例

{"result":true,"id":"1"}

GETACCOUNTINFO

获取账户信息和余额。

参数

|  |  |  |  |
| --- | --- | --- | --- |
| **名称** | **类型** | **必选？** | **描述** |
| 类型 | string | 否 | 参数可以是“all”，“balance”，“frozen”, “loan”或者“profile”，默认为“all”. |

JSON 请求示例

{"method":"getAccountInfo","params":[],"id":1}

## 只获取账户可用余额信息 ##

{"method":"getAccountInfo","params":["balance"],"id":1}

返回

|  |  |  |
| --- | --- | --- |
| **名称** | **类型** | **描述** |
| result | object[] | 包含或者是如下对象：[profile](http://btcchina.org/api-trade-documentation-zh#profile), [balance](http://btcchina.org/api-trade-documentation-zh#balance), [frozen](http://btcchina.org/api-trade-documentation-zh#frozen), [loan](http://btcchina.org/api-trade-documentation-zh#loan) |

JSON 响应示例

{

"result": {

"profile": {

"username": "btc",

"trade\_password\_enabled": true,

"otp\_enabled": true,

"trade\_fee": 0,

"trade\_fee\_cnyltc": 0,

"trade\_fee\_btcltc": 0,

"daily\_btc\_limit": 10,

"daily\_ltc\_limit": 300,

"btc\_deposit\_address": "123myZyM9jBYGw5EB3wWmfgJ4Mvqnu7gEu",

"btc\_withdrawal\_address": "123GzXJnfugniyy7ZDw3hSjkm4tHPHzHba",

"ltc\_deposit\_address": "L12ysdcsNS3ZksRrVWMSoHjJgcm5VQn2Tc",

"ltc\_withdrawal\_address": "L23GzXJnfugniyy7ZDw3hSjkm4tHPHzHba",

"api\_key\_permission": 3

},

"balance": {

"btc": {

"currency": "BTC",

"symbol": "\u0e3f",

"amount": "100.00000000",

"amount\_integer": "10000000000",

"amount\_decimal": 8

},

"ltc": {

"currency": "LTC",

"symbol": "\u0141",

"amount": "0.00000000",

"amount\_integer": "0",

"amount\_decimal": 8

},

"cny": {

"currency": "CNY",

"symbol": "\u00a5",

"amount": "50000.00000",

"amount\_integer": "5000000000",

"amount\_decimal": 5

}

},

"frozen": {

"btc": {

"currency": "BTC",

"symbol": "\u0e3f",

"amount": "0.00000000",

"amount\_integer": "0",

"amount\_decimal": 8

},

"ltc": {

"currency": "LTC",

"symbol": "\u0141",

"amount": "0.00000000",

"amount\_integer": "0",

"amount\_decimal": 8

},

"cny": {

"currency": "CNY",

"symbol": "\u00a5",

"amount": "0.00000",

"amount\_integer": "0",

"amount\_decimal": 5

}

}

"loan": {

"btc": {

"currency":"BTC",

"symbol":"\u0e3f",

"amount":"0.00000000",

"amount\_integer":"0",

"amount\_decimal":8

},

"cny":{

"currency":"CNY",

"symbol":"\u00a5",

"amount":"0.00000",

"amount\_integer":"0",

"amount\_decimal":5

}

}

},

"id": "1"

}

GETDEPOSITS

获得用户全部充值记录。

参数

|  |  |  |  |
| --- | --- | --- | --- |
| **名称** | **类型** | **必选？** | **描述** |
| currency | string | 是 | 目前支持“BTC”，“LTC” |
| pendingonly | boolean | 否 | 默认为“true”。如果为“true”，仅返回尚未入账的比特币或者莱特币充值 |

JSON 请求示例

{"method":"getDeposits","params":["BTC"],"id":1}

返回

|  |  |  |
| --- | --- | --- |
| **名称** | **类型** | **描述** |
| result | object[] | 包含对象：[deposit](http://btcchina.org/api-trade-documentation-zh#deposit) |

JSON 响应示例

{"result":{

"deposit":[{

"id":49751,

"address":"mufUAWHCius1jZpjB4zCUwzaRbYuwXCupC",

"currency":"BTC",

"amount":1,

"date":1376910685,

"status":"pending"

},{

"id":49749,

"address":"mkrmyZyM9jBYGw5EB3wWmfgJ4Mvqnu7gEu",

"currency":"BTC",

"amount":2,

"date":1376906645,

"status":"completed"

}]

}

}

GETMARKETDEPTH

方法过时了,请使用getMarketDepth2。

GETMARKETDEPTH2

获得完整的市场深度。返回全部尚未成交的买单和卖单。

参数

|  |  |  |  |
| --- | --- | --- | --- |
| **名称** | **类型** | **必选？** | **描述** |
| limit | integer | 否 | 限制返回的买卖单数目。默认是买单卖单各10条。 |
| market | string | 否 | 可选值BTCCNY、LTCCNY、LTCBTC、ALL 默认值为BTCCNY |

JSON 请求示例

## 单个市场 ##

{"method":"getMarketDepth2","params":[],"id":1}

## 所有市场 ##

{"method":"getMarketDepth2","params":[10,"ALL"],"id":1}

返回

|  |  |  |
| --- | --- | --- |
| **名称** | **类型** | **描述** |
| result | object | 对象数组：[market\_depth](http://btcchina.org/api-trade-documentation-zh#market_depth) |

JSON 响应示例

## 单个市场 ##

{"result":{

"market\_depth":{

"bid":[{

"price":99,

"amount":1

},{

"price":98,

"amount":2

}],

"ask":[{

"price":100,

"amount":0.997

},{

"price":101,

"amount":2

}]}

},

"id":"1"

}

## 所有市场 ##

{

"result": {

"market\_depth\_btccny": {

"bid": [

{

"price": 2878.01,

"amount": 1.2412

}

],

"ask": [

{

"price": 2882,

"amount": 0.1

}

],

"date": 1396415275

},

"market\_depth\_ltccny": {

"bid": [

{

"price": 78.1,

"amount": 0.088

}

],

"ask": [

{

"price": 78.3,

"amount": 0.1

}

],

"date": 1396415275

},

"market\_depth\_ltcbtc": {

"bid": [

{

"price": 0.0272,

"amount": 232.443

}

],

"ask": [

{

"price": 0.0273,

"amount": 251.578

}

],

"date": 1396415276

}

},

"id": "1"

}

GETORDER

获得挂单状态。当withdetail设置为true的时候，可以获取此笔订单的所包含的所有交易详情。

参数

|  |  |  |  |
| --- | --- | --- | --- |
| **名称** | **类型** | **必选？** | **描述** |
| id | number | 是 | 挂单 ID |
| market | string | 否 | 可选值BTCCNY、LTCCNY、LTCBTC 默认值为BTCCNY |
| withdetail | boolean | 否 | 是否返回订单内每笔交易详情。可选值true,false. 默认值为false，不返回交易详情 |

JSON 请求示例

{"method":"getOrder","params":[2],"id":1}

## 取BTCCNY市场订单号为2的订单，附带订单详情 ##

{"method":"getOrder","params":[2,"BTCCNY",true],"id":1}

返回

|  |  |  |
| --- | --- | --- |
| **名称** | **类型** | **描述** |
| result | object | 返回对象：[order](http://btcchina.org/api-trade-documentation-zh#order) |

JSON 响应示例

{"result":{

"order":{

"id":2,

"type":"ask",

"price":"46.84",

"currency":"CNY",

"amount":"0.00000000",

"amount\_original":"3.18400000",

"date":1406860694,

"status":"closed",

"details":[{

"dateline":"1406860696",

"price":"46.84",

"amount":3.184}]

}},

"id":"1"

}

GETORDERS

获得一组挂单的状态。

参数

|  |  |  |  |
| --- | --- | --- | --- |
| **名称** | **类型** | **必选？** | **描述** |
| openonly | boolean | 否 | 默认为“true”。如果为“true”，仅返回还未完全成交的挂单。 |
| market | string | 否 | 可选值BTCCNY、LTCCNY、LTCBTC、ALL 默认值为BTCCNY |
| limit | integer | 否 | 限制返回的交易记录数，默认为 1000。 |
| offset | integer | 否 | 分页索引, 默认为 0. |
| since | integer | 否 | 限制返回交易记录的起始时间. |
| withdetail | boolean | 否 | 是否返回订单内每笔交易详情。可选值true,false. 默认值为false，不返回交易详情 |

JSON 请求示例

{"method":"getOrders","params":[],"id":1}

{"method":"getOrders","params":[false],"id":1}

## 返回所有市场的最近两笔订单，不管订单是否成交 ##

{"method":"getOrders","params":[false,"ALL",2],"id":1}

## 返回所有市场的从1377671475时间点开始的最近10笔成交订单，并附带订单详情 ##

{"method":"getOrders","params":[true,"ALL",10,0,1377671475,true],"id":1}

返回

|  |  |  |
| --- | --- | --- |
| **名称** | **类型** | **描述** |
| result | object[] | 对象数组：[order](http://btcchina.org/api-trade-documentation-zh#order) |

JSON 响应示例

## 单个市场 ##

{"result":{

"order":[{

"id":2,

"type":"bid",

"price":500,

"currency":"cny",

"amount":0.9,

"amount\_original":0.9,

"date":1377671476,

"status":"cancelled"

},{

"id":3,

"type":"bid",

"price":501,

"currency":"cny",

"amount":0.8,

"amount\_original":0.8,

"date":1377671475,

"status":"cancelled"

}]

},

"id":"1"

}

## 所有市场 ##

{

"result": {

"order\_btccny": [

{

"id": 13942927,

"type": "bid",

"price": "2000.00",

"currency": "CNY",

"amount": "0.00100000",

"amount\_original": "0.00100000",

"date": 1396255376,

"status": "open"

},

{

"id": 13942807,

"type": "bid",

"price": "2000.00",

"currency": "CNY",

"amount": "0.00100000",

"amount\_original": "0.00100000",

"date": 1396255245,

"status": "open"

}

],

"order\_ltccny": [

],

"order\_ltcbtc": [

]

},

"id": "1"

}

GETTRANSACTIONS

获取交易记录。

参数

|  |  |  |  |
| --- | --- | --- | --- |
| **名称** | **类型** | **必选？** | **描述** |
| type | string | 否 | 按类型获取交易记录。默认为“all”（全部）。可用类型包括： 'all | fundbtc | withdrawbtc | fundmoney | withdrawmoney | refundmoney | buybtc | sellbtc | buyltc | sellltc | tradefee | rebate' |
| limit | integer | 否 | 限制返回的交易记录数，默认为 10。 |
| offset | integer | 否 | 分页索引, 默认为 0. |
| since | integer | 否 | 取从某一点开始的交易记录, 这一点可以是某个订单号或者Unix时间戳, 默认值为0. |
| sincetype | string | 否 | 指定since参数的类型，可以是“id”或者“time”，默认值为“time”. |

JSON 请求示例

{"method":"getTransactions","params":[],"id":1}

{"method":"getTransactions","params":["buybtc",2],"id":1}

# 取回从订单号101开始的100条交易记录 #

{"method":"getTransactions","params":["all",100,0,101,"id"],"id":1}

返回

|  |  |  |
| --- | --- | --- |
| **Name** | **Value** | **Description** |
| result | object[] | [transaction](http://btcchina.org/api-trade-documentation-zh#transaction) |

JSON 响应示例

{"result":{"

transaction":[{

"id":8,

"type":"buybtc",

"btc\_amount":"0.00100000",

"cny\_amount":"-0.10000",

"date":1383128749

},{

"id":7,

"type":"sellbtc",

"btc\_amount":"-0.00100000",

"cny\_amount":"0.10000",

"date":1383128749

}]

},

"id":"1"

}

GETWITHDRAWAL

获取提现状态。

参数

|  |  |  |  |
| --- | --- | --- | --- |
| **名称** | **类型** | **必选？** | **描述** |
| id | number | 是 | 提现 ID |
| currency | string | 否 | BTC 和 LTC 默认为“BTC” |

JSON 请求示例

{"method":"getWithdrawal","params":[1],"id":1}

返回

|  |  |  |
| --- | --- | --- |
| **名称** | **类型** | **描述** |
| result | object | 返回对象：[withdrawal](http://btcchina.org/api-trade-documentation-zh#withdrawal) |

JSON 响应示例

{"result":{

"withdrawal":{

"id":20351,

"address":"15MGzXJnfugniyy7ZDw3hSjkm4tHPHzHba",

"currency":"BTC",

"amount":0.1,

"date":1376891209,

"transaction":null,

"status":"pending"

}

},

"id":"1"

}

GETWITHDRAWALS

获取全部提现记录。

参数

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **类型** | **必选？** | **描述** |
| currency | string | 是 | 目前支持“BTC”，“LTC” |
| pendingonly | boolean | 否 | 默认为“true”。如果为“true”，仅返回尚未处理的提现记录 |

JSON REQUEST

{"method":"getWithdrawals","params":["BTC"],"id":1}

返回

|  |  |  |
| --- | --- | --- |
| **名称** | **类型** | **描述** |
| result | object[] | 对象数组：[withdrawal](http://btcchina.org/api-trade-documentation-zh#withdrawal) |

JSON 响应示例

{"result":{

"withdrawal":[{

"id":20351,

"address":"15MGzXJnfugniyy7ZDw3hSjkm4tHPHzHba",

"currency":"BTC",

"amount":0.1,

"date":1376891209,

"transaction":null,

"status":"pending"

},{

"id":20352,

"address":"15MGzXJnfugniyy7ZDw3hSjkm4tHPHzHba",

"currency":"BTC",

"amount":0.1,"date":1376891268,

"transaction":null,

"status":"pending"

}

}],

"id":"1"

}

REQUESTWITHDRAWAL

发起比特币提现请求,为了安全起见,本方法不提供提现地址参数，默认使用上一次提现的比特币地址。假如用户希望更改提现地址，需要首先去网站完成一笔提现，到最新更改的比特币收款地址。

参数

|  |  |  |  |
| --- | --- | --- | --- |
| **名称** | **类型** | **必选？** | **描述** |
| currency | string | 是 | 货币代码。可能值：BTC 或 LTC |
| amount | number | 是 | 提现金额 |

JSON 请求示例

{"method":"requestWithdrawal","params":["BTC",0.1],"id":1}

返回

|  |  |  |
| --- | --- | --- |
| **名称** | **类型** | **描述** |
| result | integer | 返回提现 ID |

JSON 响应示例

{"result":{"id":"20362"},"id":"1"}

SELLORDER

该方法已过时，请使用sellOrder2.

SELLORDER2

下比特币或莱特币卖单。

参数

|  |  |  |  |
| --- | --- | --- | --- |
| **名称** | **类型** | **必选？** | **描述** |
| price | string | 是 | 卖 1 比特币，莱特币所用人民币的价格，最多支持小数点后 2 位精度。若以市场价交易，将 price 设置为 null |
| amount | string | 是 | 要卖的比特币数量, BTC最多支持小数点后 4 位精度, LTC最多支持小数点后 3 位精度 |
| market | string | 否 | 可选值BTCCNY、LTCCNY、LTCBTC 默认值为BTCCNY |

JSON 请求示例

{"method":"sellOrder2","params":[\"500\",\"1\"],"id":1}

{"method":"sellOrder2","params":[\"500.01\",\"1.231\",\"LTCCNY\"],"id":1}

## market order ##

{"method":"sellOrder2","params":[null,\"1\"],"id":1}

返回

|  |  |  |
| --- | --- | --- |
| **名称** | **类型** | **描述** |
| result | integer | 如果下单成功，返回订单号 |

JSON 响应示例

{"result":12345,"id":"1"}

ICEBERG 订单API方法

Iceberg订单是指为了隐藏一笔较大订单而将其拆分为若干个小额订单分别下单的方法。 当一笔小额订单全部成交以后，下一笔被拆分的小额订单才会被放到市场里，如此下去，一直到整个iceberg订单完全成交。 为了更好的隐藏iceberg订单，我们引入了波动率这个参数，它可以用来控制每笔小额订单的数量波动范围。

你必须保证在下iceberg订单的时候，账户里有足够的可用余额，但是当iceberg订单下单成功以后，您的账户可用余额允许小于整个iceberg订单剩余的金额，因为iceberg订单每次只会冻结拆分的小订单的等值金额。

当一笔小额订单成交以后而你的可用余额不足以完成下一笔小额订单的时候，整个iceberg订单就会被取消。 对于每一笔小额订单，当有任何异常或错误发生的时候，整个iceberg订单也会被取消。 当你取消或者更新任何一笔小额订单的时候，整个iceberg订单也会被取消。

BUYICEBERGORDER

下iceberg的买单。此方法成功后返回iceberg订单号。

参数

|  |  |  |  |
| --- | --- | --- | --- |
| **名称** | **类型** | **必选？** | **描述** |
| price | number | 是 | 买 1 比特币/莱特币所用的价格，BTC/CNY和LTC/CNY市场最多支持小数点后 2 位精度，LTC/BTC市场最多支持小数点后 4 位精度。假如要以市场价格交易，将 price 设置为 null. |
| amount | number | 是 | 要买的比特币或者莱特币总量, BTC最多支持小数点后 4 位精度, LTC最多支持小数点后 3 位精度. |
| disclosed\_amount | number | 是 | 要买的公开的每笔拆分订单的比特币或者莱特币数量, BTC最多支持小数点后 4 位精度, LTC最多支持小数点后 3 位精度. |
| variance | number | 否 | 默认值为0，必须小于1. 作为被拆分订单数量的波动率，比如每笔拆分订单数量设置为5个，波动率为0.1时，拆分订单的实际数量为［5-5\*0.1，5+5\*0.1］之间的随机值. |
| market | string | 否 | 默认值为”BTCCNY”. [ BTCCNY | LTCCNY | LTCBTC ] |

JSON 请求

{"method":"buyIcebergOrder","params":[500,100,5,0.2],"id":1}

## 市场价格订单 ##

{"method":"buyIcebergOrder","params":[null,100,5,0.2],"id":1}

返回值

|  |  |  |
| --- | --- | --- |
| **名称** | **类型** | **描述** |
| result | integer | 下单成功后返回iceberg订单号. |

JSON 响应

{"result":12345,"id":"1"}

SELLICEBERGORDER

下iceberg的卖单。此方法成功后返回iceberg订单号。

参数

|  |  |  |  |
| --- | --- | --- | --- |
| **名称** | **类型** | **必选？** | **描述** |
| price | number | 是 | 卖 1 比特币/莱特币所用的价格，BTC/CNY和LTC/CNY市场最多支持小数点后 2 位精度，LTC/BTC市场最多支持小数点后 4 位精度。假如要以市场价格交易，将 price 设置为 null. |
| amount | number | 是 | 要卖的比特币或者莱特币总量, BTC最多支持小数点后 4 位精度, LTC最多支持小数点后 3 位精度. |
| disclosed\_amount | number | 是 | 要卖的公开的每笔拆分订单的比特币或者莱特币数量, BTC最多支持小数点后 4 位精度, LTC最多支持小数点后 3 位精度. |
| variance | number | 否 | 默认值为0，必须小于1. 作为被拆分订单数量的波动率，比如每笔拆分订单数量设置为5个，波动率为0.1时，拆分订单的实际数量为［5-5\*0.1，5+5\*0.1］之间的随机值. |
| market | string | 否 | 默认值为”BTCCNY”. [ BTCCNY | LTCCNY | LTCBTC ] |

JSON 请求

{"method":"sellIcebergOrder","params":[500,100,5,0.2],"id":1}

## 市场价格订单 ##

{"method":"sellIcebergOrder","params":[null,100,5,0.2],"id":1}

返回值

|  |  |  |
| --- | --- | --- |
| **名称** | **类型** | **描述** |
| result | integer | 下单成功后返回iceberg订单号. |

JSON 响应

{"result":12345,"id":"1"}

GETICEBERGORDER

获取一笔iceberg订单，包括所有在市场上公开过的拆分订单。

参数

|  |  |  |  |
| --- | --- | --- | --- |
| **名称** | **类型** | **必选？** | **描述** |
| id | number | 是 | iceberg订单号. |
| market | string | 否 | 默认值为”BTCCNY”. [ BTCCNY | LTCCNY | LTCBTC ] |

JSON 请求

{"method":"getIcebergOrder","params":[123],"id":1}

返回值

|  |  |  |
| --- | --- | --- |
| **名称** | **类型** | **描述** |
| result | object[] | [iceberg\_order](http://btcchina.org/api-trade-documentation-zh#iceberg_order) |

JSON 响应

{"result":

{"iceberg\_order":

{"id":1,

"type":"bid",

"price":"40.00",

"market":"BTCCNY",

"amount":"12.00000000",

"amount\_original":"12.00000000",

"disclosed\_amount":"5.00000000",

"variance":"0.10",

"date":1405412126,

"status":"open",

"order": [

{"id":3301,

"type":"bid",

"price":"40.00",

"currency":"CNY",

"amount":"4.67700000",

"amount\_original":"4.67700000",

"date":1405412126,

"status":"open"}]}},

"id":"1"}

GETICEBERGORDERS

获取多笔iceberg订单，包括每笔iceberg订单里的所有公开过的拆分订单。

参数

|  |  |  |  |
| --- | --- | --- | --- |
| **名称** | **类型** | **必选？** | **描述** |
| limit | integer | 否 | 想要获取的iceberg订单的数量，默认值为1000. |
| offset | integer | 否 | 获取的订单起始索引，默认值为0. |
| market | string | 否 | 默认值为”BTCCNY”. [ BTCCNY | LTCCNY | LTCBTC ] |

JSON 请求

{"method":"getIcebergOrders","params":[],"id":1}

## 获取从第11单开始的50个iceberg订单 ##

{"method":"getIcebergOrders","params":[50, 10],"id":1}

返回值

|  |  |  |
| --- | --- | --- |
| **名称** | **类型** | **描述** |
| result | object[] | [iceberg\_order](http://btcchina.org/api-trade-documentation-zh#iceberg_order) |

JSON 响应

{"result":

{"iceberg\_orders":

[

{"id":2,

"type":"ask",

"price":"40.00",

"market":"BTCCNY",

"amount":"0.00000000",

"amount\_original":"5.00000000",

"disclosed\_amount":"5.00000000",

"variance":"0.00",

"date":1405412293,

"status":"closed",

"order":

[

{"id":3304,

"type":"ask",

"price":"40.00",

"currency":"CNY",

"amount":"0.00000000",

"amount\_original":"5.00000000",

"date":1405412293,

"status":"closed"}]},

{"id":1,

"type":"bid",

"price":"40.00",

"market":"BTCCNY",

"amount":"7.00000000",

"amount\_original":"12.00000000",

"disclosed\_amount":"5.00000000",

"variance":"0.00",

"date":1405412292,

"status":"open",

"order":

[

{"id":3305,

"type":"bid",

"price":"40.00",

"currency":"CNY",

"amount":"5.00000000",

"amount\_original":"5.00000000",

"date":1405412294,

"status":"open"},

{"id":3303,

"type":"bid",

"price":"40.00",

"currency":"CNY",

"amount":"0.00000000",

"amount\_original":"5.00000000",

"date":1405412292,

"status":"closed"}]}]},

"id":"1"}

CANCELICEBERGORDER

取消一个iceberg订单。如果iceberg订单已经完全成交或者被取消了，则会返回错误。与被取消的iceberg订单相关的所有订单都会被取消。

参数

|  |  |  |  |
| --- | --- | --- | --- |
| **名称** | **类型** | **必选？** | **描述** |
| id | number | 是 | 要取消的iceberg订单号. |
| market | string | 否 | 默认值为”BTCCNY”. [ BTCCNY | LTCCNY | LTCBTC ] |

JSON 请求

{"method":"cancelIcebergOrder","params":[1234],"id":1}

返回值

|  |  |  |
| --- | --- | --- |
| **名称** | **类型** | **描述** |
| result | boolean | 如果取消成功，则返回true. |

JSON 响应

{"result":true,"id":"1"}

止损止盈订单API方法

用户在止损止盈订单中设置一个“触发单价”，当市场单价达到触发单价的时候，会触发系统自动为用户下一个市价单或者限价单。 如果“price”参数没有设置，那么当市场单价达到触发单价时，被触发的订单是一个市价单。这种类型的订单也叫做“止损订单”。 如果“price”参数被设置了，那么当市场单价达到触发单价时，被触发的订单是一个以“price”作为价格的限价单。这种类型的订单也叫做“限价止损订单”。

注意，当用户下一个止损止盈订单的时候，并不会检查用户的可用余额。可用余额只有在市场单价达到触发单价，并导致被触发的订单下单时才会被检查和冻结。

追踪止损单是止损止盈单的一种。在追踪止损单中，触发单价是随着市场单价的变动而动态变化的。用户通过设置追单波动价格量（trailing\_amount）或者追单波动价格率（trailing\_percentage），并将触发单价（stop\_price）设置为空（null），来使用追踪止损单。

当追单波动价格量被设置了以后，在一个止损卖单里，触发单价随着市场单价的上升单价量而上升相应的单价量；在止损买单里则是下降。 相似的，在一个止损卖单里，当追单波动价格率被设置了以后，触发单价随着市场价格的上升而上升到此时市场最高单价与（1-追单波动价格率）的乘积值；在止损买单里则是下降到市场最低单价与（1＋追单波动价格率）的乘积值。

在止损止盈单里，下面三个值中有且仅有一个值被设置为非空：触发单价（stop\_price）, 追单波动价格量（trailing\_amount）,追单波动价格率（trailing\_percentage）。

BUYSTOPORDER

设置一个止损止盈买单。此方法返回一个止损止盈订单号。

参数

|  |  |  |  |
| --- | --- | --- | --- |
| **名称** | **类型** | **必选？** | **描述** |
| stop\_price | number | 否 | 触发单价，当市场成交价大于等于此价格时，会触发系统自动为用户下一个市价单或者限价单。在BTCCNY和LTCCNY市场最多允许2位小数，在LTCBTC市场最多允许4位小数。只有追单波动价格量（trailing\_amount）和追单波动价格率（trailing\_percentage）同时被设置为空时才可设置此参数. |
| price | number | 是 | 当市场单价达到触发单价时，被触发的订单中的单价。在BTCCNY和LTCCNY市场最多允许2位小数，在LTCBTC市场最多允许4位小数。如若要下市价单，将此参数设置为'null'. |
| amount | number | 是 | 当市场单价达到触发单价时，被触发的订单中要买的BTC/LTC数量。BTC最多允许4位小数，LTC最多允许3位小数. |
| trailing\_amount | number | 否 | 追单波动价格量，被设置后将决定触发单价。触发单价等于追单波动价格量与最低市场单价的和。其中最低市场单价指的是从此追踪止损单被创建以来的市场最低单价。只有触发单价和追单波动价格率同时被设置为空时才可设置此参数. |
| trailing\_percentage | number | 否 | 追单波动价格率，被设置后将决定触发单价。触发单价等于最低市场单价\*（1＋追单波动价格率）。其中最低市场单价指的是从此追踪止损单被创建以来的市场最低单价。只有触发单价和追单波动价格率同时被设置为空时才可设置此参数. |
| market | string | 否 | 默认值为”BTCCNY”. [ BTCCNY | LTCCNY | LTCBTC ] |

JSON 请求

## 止损止盈订单，触发后下限价买单 ##

{"method":"buyStopOrder","params":[500,500.01,0.123],"id":1}

## 止损止盈订单，触发后下市价买单 ##

{"method":"buyStopOrder","params":[500,null,0.123],"id":1}

返回值

|  |  |  |
| --- | --- | --- |
| **名称** | **类型** | **描述** |
| result | integer | 如果止损止盈订单设置成功则返回止损止盈订单号. |

JSON 响应

{"result":12345,"id":"1"}

SELLSTOPORDER

设置一个止损止盈卖单。此方法返回一个止损止盈订单号。

参数

|  |  |  |  |
| --- | --- | --- | --- |
| **名称** | **类型** | **必选？** | **描述** |
| stop\_price | number | 否 | 触发单价，当市场成交价小于等于此价格时，会触发系统自动为用户下一个市价单或者限价单。在BTCCNY和LTCCNY市场最多允许2位小数，在LTCBTC市场最多允许4位小数。只有追单波动价格量（trailing\_amount）和追单波动价格率（trailing\_percentage）同时被设置为空时才可设置此参数. |
| price | number | 是 | 当市场单价达到触发单价时，被触发的订单中的单价。在BTCCNY和LTCCNY市场最多允许2位小数，在LTCBTC市场最多允许4位小数。如若要下市价单，将此参数设置为'null'. |
| amount | number | 是 | 当市场单价达到触发单价时，被触发的订单中要买的BTC/LTC数量。BTC最多允许4位小数，LTC最多允许3位小数. |
| trailing\_amount | number | 否 | 追单波动价格量，被设置后将决定触发单价。触发单价等于最高市场单价与追单波动价格量的差。其中最高市场单价指的是从此追踪止损单被创建以来的市场最高单价。只有触发单价和追单波动价格率同时被设置为空时才可设置此参数. |
| trailing\_percentage | number | 否 | 追单波动价格率，被设置后将决定触发单价。触发单价等于最高市场单价\*（1－追单波动价格率）。其中最高市场单价指的是从此追踪止损单被创建以来的市场最高单价。只有触发单价和追单波动价格率同时被设置为空时才可设置此参数. |
| market | string | 否 | 默认值为 “BTCCNY”. [ BTCCNY | LTCCNY | LTCBTC ] |

JSON 请求

## 止损止盈订单，触发后下限价卖单 ##

{"method":"sellStopOrder","params":[500,499.99,0.123],"id":1}

## 止损止盈订单，触发后下市价卖单 ##

{"method":"sellStopOrder","params":[500,null,0.123],"id":1}

返回值

|  |  |  |
| --- | --- | --- |
| **名称** | **类型** | **描述** |
| result | integer | 如果止损止盈订单设置成功则返回止损止盈订单号. |

JSON 响应

{"result":12345,"id":"1"}

GETSTOPORDER

获取一个止损止盈订单.

参数

|  |  |  |  |
| --- | --- | --- | --- |
| **名称** | **类型** | **必选？** | **描述** |
| id | number | 是 | 止损止盈订单号. |
| market | string | 否 | 默认值为”BTCCNY”. [ BTCCNY | LTCCNY | LTCBTC ] |

JSON 请求

{"method":"getStopOrder","params":[123],"id":1}

## 获取订单号为123的LTCCNY订单 ##

{"method":"getStopOrder","params":[123, "LTCCNY"],"id":1}

返回值

|  |  |  |
| --- | --- | --- |
| **名称** | **类型** | **描述** |
| result | object | [stop\_order](http://btcchina.org/api-trade-documentation-zh#stop_order) |

JSON 响应

{"result":

{"stop\_order":

{"id":1,

"type":"bid",

"stop\_price":"50.00",

"trailing\_amount":"10.00000000",

"trailing\_percentage":null,

"price":"50.00",

"market":"LTCCNY",

"amount":"2.00000000",

"date":1407489603,

"status":"open",

"order\_id":null}},

"id":"1"}

GETSTOPORDERS

获取一组符合筛选条件的止损止盈订单.

参数

|  |  |  |  |
| --- | --- | --- | --- |
| **名称** | **类型** | **必选？** | **描述** |
| status | string | 否 | 要筛选的订单状态: [ open | closed | cancelled | error ] |
| type | string | 否 | 要筛选的订单类型: [ ask | bid ] |
| stop\_price | float | 否 | 要筛选的触发单价. 对于止损止盈买单来说，将返回所有触发单价小于等于此参数的订单；对于止损止盈卖单来说，将返回所有触发单价大于等于此参数的订单. |
| limit | integer | 否 | 限制返回的订单数量，默认值为1000. |
| offset | integer | 否 | 起始分页索引，默认值为0. |
| market | string | 否 | 默认值为”BTCCNY”. [ BTCCNY | LTCCNY | LTCBTC ] |

JSON 请求

{"method":"getStopOrders","params":[],"id":1}

返回值

|  |  |  |
| --- | --- | --- |
| **名称** | **类型** | **描述** |
| result | object[] | 对象数组：[stop\_order](http://btcchina.org/api-trade-documentation-zh#stop_order) |

JSON 响应

{"result":

{"stop\_orders":

[

{"id":1,

"type":"bid",

"stop\_price":"50.00",

"trailing\_amount":"10.00000000",

"trailing\_percentage":null,

"price":"50.00",

"market":"LTCCNY",

"amount":"2.00000000",

"date":1407492143,

"status":"open",

"order\_id":null},

{"id":2,

"type":"bid",

"stop\_price":"44.00",

"trailing\_amount":null,

"trailing\_percentage":"0.10",

"price":"51.00",

"market":"LTCCNY",

"amount":"2.00000000",

"date":1407492144,

"status":"open",

"order\_id":null}]},

"id":"1"}

CANCELSTOPORDER

取消一个止损止盈订单. 如果订单已经成交或者已被取消则操作失败.

参数

|  |  |  |  |
| --- | --- | --- | --- |
| **名称** | **类型** | **必选？** | **描述** |
| id | number | 是 | 要取消的订单号. |
| market | string | 否 | 默认值为”BTCCNY”. [ BTCCNY | LTCCNY | LTCBTC ] |

JSON 请求

{"method":"cancelStopOrder","params":[123],"id":1}

## 取消订单号为123的LTCCNY订单 ##

{"method":"cancelStopOrder","params":[123, "LTCCNY"],"id":1}

返回值

|  |  |  |
| --- | --- | --- |
| **名称** | **类型** | **描述** |
| result | boolean | 如果取消成功，则返回true. |

JSON 响应

{"result":true,"id":"1"}

API 常见问题

1. 使用交易API时为什么会出现401 UNAUTHORIZED 错误?

* 请校准系统时间，确保 tonce 为标准Linux16位值；
* 确保Access Key有效；
* 确保Params有效；请查看以下关于[getAccountInfo](http://btcchina.org/api-trade-documentation-zh#getaccountinfo中参数如何配置)，[buyOrder2/sellOrder2](http://btcchina.org/api-trade-documentation-zh#buyorder2_sellorder2_的参数如何设置)以及[getOrders](http://btcchina.org/api-trade-documentation-zh#getorders_中_参数如何配置)的说明；
* 确保HTTP Authorization Header 提交给服务器了；

2. GETACCOUNTINFO中参数如何配置？

Signature 哈希值为：

"tonce=<timestamp>

&accesskey=<yourkey>

&requestmethod=post

&id=1

&method=getAccountInfo

&params="

JSON Request格式为:

{"method":"getAccountInfo","params":[],"id":1}

3. BUYORDER2/SELLORDER2 的参数如何设置？

1. 限价单下单方式：

Signature 哈希值为：

"tonce=<timestamp>

&accesskey=<yourkey>

&requestmethod=post

&id=1

&method=buyOrder2

&params=0.0001,0.005,LTCBTC"

JSON Request格式为:

1. LTC/BTC

{"method":"buyOrder2","params":[0.0001,0.005,"LTCBTC"],"id":1}

2. LTC/CNY

{"method":"buyOrder2","params":[100.00,0.001,"LTCCNY"],"id":1}

3. BTC/CNY

{"method":"buyOrder2","params":[4000.00,0.005,"BTCCNY"],"id":1}

2. 市场价下单方式：

Signature 哈希值为：

"tonce=<timestamp>

&accesskey=<yourkey>

&requestmethod=post

&id=1

&method=buyOrder2

&params=,0.2,ltccny"

JSON Request格式为:

{"method":"buyOrder2","params":[null,0.2,"ltccny"],"id":1}

4. GETORDERS 中 参数如何配置?

1. OPENONLY 为FALSE:

Signature 哈希值为：

"tonce=<timestamp>

&accesskey=<yourkey>

&requestmethod=post

&id=1

&method=getOrders

&params=,LTCCNY"

JSON Request格式为:

{"method":"getOrders","params":[false,"LTCCNY"],"id":1}

2. OPENONLY 为TRUE:

Signature 哈希值为：

"tonce=<timestamp>

&accesskey=<yourkey>

&requestmethod=post

&id=1

&method=getOrders

&params=1,LTCCNY"

JSON Request格式为:

{"method":"getOrders","params":[true,"LTCCNY"],"id":1}

备注:

1. 以上实列中为“LTCCNY”（莱特币交易查询），若想查询其他类型，改为相应类型“BTCCNY”，“LTCBTC”即可。
2. 当前Signature Hash的方式未采用JSON-RPC标准，如果为true则会解析是1，false会解析为空（’’）。以后开发完支持JSON RPC的格式去Hash后，会对文档做相关更新。

5. 为什么会出现403 FORBIDDEN错误？

首先，请确认你是否有“允许交易”或者“允许提现”权限，出现该错误是因为你创建的API密钥没有交易、提现的权限，你可以新创建一个有“允许交易”或者“允许提现“的API密钥。

6. 为什么出现”INVALID AMOUNT”和”INVALID PRICE”错误？

出现这个问题是你传递的price/amount参数为0或者round（n）后为0。其中n为price、amount的精度；

* BTC/CNY: Price：n=2—表示人民币的精度，Amount：n=4-表示比特币售出或购买数量；
* LTC/CNY: Price：n=2—表示人民币的精度，Amount：n=3-表示莱特币售出或购买数量；
* LTC/BTC: Price：n=4—表示比特币的精度，Amount：n=3-表示莱特币售出或购买数量；

7. MARKET DATA API 多久刷新一次？

每隔5秒刷新一次。

8. 调用GETMARKETDEPTH2 查询市场深度会出现“ 503 SERVICE TEMPORARILY UNAVAILABLE ”错误?

可能是由于我们系统正在升级，请稍等一段时间即可进行查询。

9. 使用API有限制么？

* 交易API每秒最多提交5个请求。
* 数据API是不能不停地连续访问的。

10. 使用PYTHON 时会出现 “HTTPLIB HAS NO ATTRIBUTE HTTPSCONNECTION”错误？

使用Python出现HTTPSConnection error，请确认你是否import httplib 包.

11. 可以从哪里下载源代码？

请访问 <https://github.com/BTCChina>.

12. GETORDER返回结果的PRICE是下单价格还是成交价格？

GETORDER返回结果的Price是下单价格，若想查询成交价格、成交均价请查询GETTRANSACTIONS。