

kidney

Career(1)

Engineering(1)

HTML&CSS(4)

JavaScript(8)

Python(1)

View(2)

用 Python 撸一个区块链

本文翻译自 Daniel van Flymen 的文章 Learn Blockchains by Building One

略有删改。原文地址：<https://hackernoon.com/learn-blockchains-by-building-one-117428612f46>

相信你和我一样对数字货币的崛起感到新奇，并且想知道其背后的技术——区块链是怎样实现的。

但是理解区块链并非易事，至少对于我来说是如此。晦涩难懂的视频、漏洞百出的教程以及示例的匮乏令我倍受挫折。

我喜欢在实践中学习，通过写代码来学习技术会掌握得更牢固。如果你也这样做，那么读完本文，你将获得一个可用的区块链以及对区块链的深刻理解。

哔哔肾

博客园

首页

新随笔

联系

订阅

管理

开始之前...

如果你不了解 Hash，这里有个例子 <https://learn cryptography.com/hash-functions/what-are-hash-functions>

其次，你需要安装 Python3.6+，Flask，Request

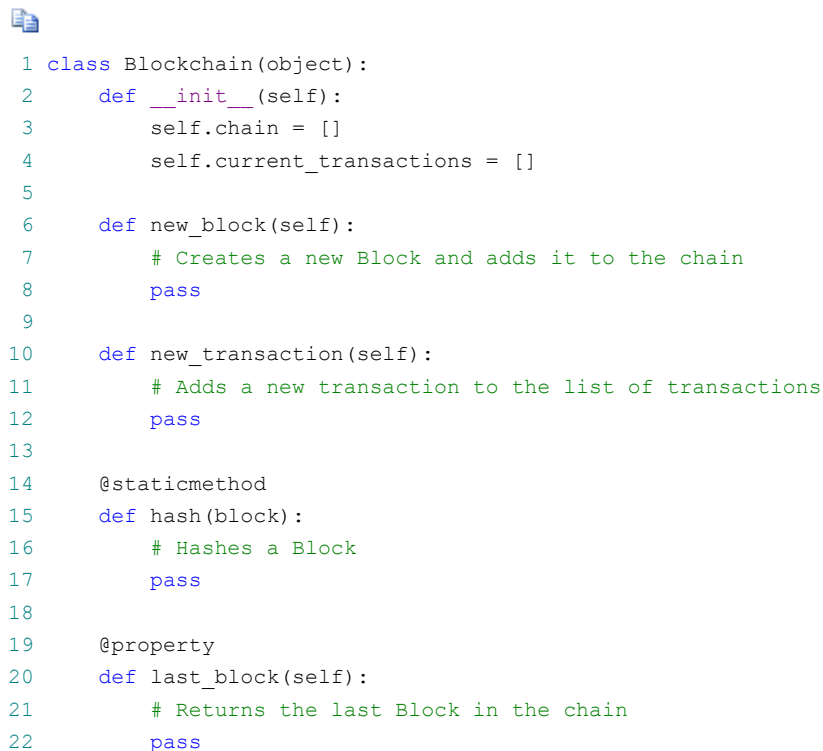
```
pip install Flask==0.12.2 requests==2.18.4
```

同时你还需要一个 HTTP 客户端，比如 Postman，cURL 或任何其它客户端。

最终的源代码在这里：<https://github.com/dvf/blockchain>

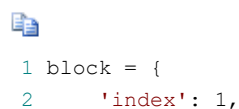
第一步：打造一个 Blockchain

新建一个文件 blockchain.py，本文所有的代码都写在这一个文件中。首先创建一个 Blockchain 类，在构造函数中我们创建了两个列表，一个用于储存区块链，一个用于储存交易。



```
1 class Blockchain(object):
2     def __init__(self):
3         self.chain = []
4         self.current_transactions = []
5
6     def new_block(self):
7         # Creates a new Block and adds it to the chain
8         pass
9
10    def new_transaction(self):
11        # Adds a new transaction to the list of transactions
12        pass
13
14    @staticmethod
15    def hash(block):
16        # Hashes a Block
17        pass
18
19    @property
20    def last_block(self):
21        # Returns the last Block in the chain
22        pass
```

一个区块有五个基本属性：index，timestamp（in Unix time），transaction 列表，工作量证明（稍后解释）以及前一个区块的 Hash 值。



```
1 block = {
2     'index': 1,
```

```

6         'sender': "852/14/te1fb426f9dd54bde4b2/ee00",
7         'recipient': "a77f5cdfa2934df3954a5c7c7da5df1f",
8         'amount': 5,
9     }
10 ],
11 'proof': 324984774000,
12 'previous_hash': "2cf24dba5fb0a30e26e83b2ac5b9e29e1b161e5c1fa7425e73043362938b9824"
13 }

```

到这里，区块链的概念应该比较清楚了：每个新的区块都会包含上一个区块的 Hash 值。这一点非常关键，它是区块链不可变性的根本保障。如果攻击者破坏了前面的某个区块，那么后面所有区块的 Hash 都会变得不正确。不理解？慢慢消化~

我们需要一个向区块添加交易的方法：

```

1 class Blockchain(object):
2     ...
3
4     def new_transaction(self, sender, recipient, amount):
5         """
6         Creates a new transaction to go into the next mined Block
7         :param sender: <str> Address of the Sender
8         :param recipient: <str> Address of the Recipient
9         :param amount: <int> Amount
10        :return: <int> The index of the Block that will hold this transaction
11        """
12
13        self.current_transactions.append({
14            'sender': sender,
15            'recipient': recipient,
16            'amount': amount,
17        })
18
19        return self.last_block['index'] + 1

```

`new_transaction()` 方法向列表中添加一个交易记录，并返回该记录将被添加到的区块——下一个待挖掘的区块——的索引，稍后在用户提交交易时会有用。

当 `Blockchain` 实例化后，我们需要创建一个初始的区块（创世块），并且给它预设一个工作量证明。

除了添加创世块的代码，我们还需要补充 `new_block()`, `new_transaction()` 和 `hash()` 方法：

```

1 import hashlib
2 import json

```

```

6     def __init__(self):
7         self.current_transactions = []
8         self.chain = []
9
10        # Create the genesis block
11        self.new_block(previous_hash=1, proof=100)
12
13    def new_block(self, proof, previous_hash=None):
14        block = {
15            'index': len(self.chain) + 1,
16            'timestamp': time(),
17            'transactions': self.current_transactions,
18            'proof': proof,
19            'previous_hash': previous_hash or self.hash(self.chain[-1]),
20        }
21
22        # Reset the current list of transactions
23        self.current_transactions = []
24
25        self.chain.append(block)
26        return block
27
28    def new_transaction(self, sender, recipient, amount):
29        self.current_transactions.append({
30            'sender': sender,
31            'recipient': recipient,
32            'amount': amount,
33        })
34
35        return self.last_block['index'] + 1
36
37    @property
38    def last_block(self):
39        return self.chain[-1]
40
41    @staticmethod
42    def hash(block):
43        block_string = json.dumps(block, sort_keys=True).encode()
44        return hashlib.sha256(block_string).hexdigest()

```



上面的代码应该很直观，我们基本上有了区块链的雏形。但此时你肯定很想知道一个区块究竟是怎样被创建或挖掘出来的。

新的区块来自工作量证明（PoW）算法。PoW 的目标是计算出一个符合特定条件的数字，这个数字对于所有人而言必须在计算上非常困难，但易于验证。这就是工作量证明的核心思想。

举个例子：

假设一个整数 x 乘以另一个整数 y 的积的 Hash 值必须以 0 结尾，即 $\text{hash}(x * y) = \text{ac23dc}...0$ 。设 $x = 5$ ，求 y ？

```

1 from hashlib import sha256
2 x = 5
3 y = 0 # We don't know what y should be yet...

```

结果是 $y = 21$ // $\text{hash}(5 * 21) = 1253e9373e...5e3600155e860$

在比特币中，工作量证明算法被称为 Hashcash，它和上面的问题很相似，只不过计算难度非常大。这就是矿工们为了争夺创建区块的权利而争相计算的问题。通常，计算难度与目标字符串需要满足的特定字符的数量成正比，矿工算出结果后，就会获得一定数量的比特币奖励（通过交易）。

网络要验证结果，当然非常容易。

让我们来实现一个 PoW 算法，和上面的例子非常相似，规则是：寻找一个数 p ，使得它与前一个区块的 proof 拼接成的字符串的 Hash 值以 4 个零开头。

```
1 import hashlib
2 import json
3 from time import time
4 from uuid import uuid4
5
6 class Blockchain(object):
7     ...
8
9     def proof_of_work(self, last_proof):
10         proof = 0
11         while self.valid_proof(last_proof, proof) is False:
12             proof += 1
13
14         return proof
15
16     @staticmethod
17     def valid_proof(last_proof, proof):
18         guess = f'{last_proof}{proof}'.encode()
19         guess_hash = hashlib.sha256(guess).hexdigest()
20         return guess_hash[:4] == "0000"
```

衡量算法复杂度的办法是修改零的个数。4 个零足够用于演示了，你会发现哪怕多一个零都会大大增加计算出结果所需的时间。

我们的 Blockchain 基本已经完成了，接下来我们将使用 HTTP requests 来与之交互。

第二步：作为 API 的 Blockchain

我们将使用 Flask 框架，它十分轻量并且很容易将网络请求映射到 Python 函数。

我们将创建三个接口：

`/transactions/new` 创建一个交易并添加到区块

我们的服务器将扮演区块链网络中的一个节点。我们先添加一些常规代码：

```

1 import hashlib
2 import json
3 from textwrap import dedent
4 from time import time
5 from uuid import uuid4
6 from flask import Flask, jsonify, request
7
8 class Blockchain(object):
9     ...
10
11 # Instantiate our Node
12 app = Flask(__name__)
13
14 # Generate a globally unique address for this node
15 node_identifier = str(uuid4()).replace('-', '')
16
17 # Instantiate the Blockchain
18 blockchain = Blockchain()
19
20 @app.route('/mine', methods=['GET'])
21 def mine():
22     return "We'll mine a new Block"
23
24 @app.route('/transactions/new', methods=['POST'])
25 def new_transaction():
26     return "We'll add a new transaction"
27
28 @app.route('/chain', methods=['GET'])
29 def full_chain():
30     response = {
31         'chain': blockchain.chain,
32         'length': len(blockchain.chain),
33     }
34     return jsonify(response), 200
35
36 if __name__ == '__main__':
37     app.run(host='127.0.0.1', port=5000)

```

这是用户发起交易时发送到服务器的请求：

```

1 {
2     "sender": "my address",
3     "recipient": "someone else's address",
4     "amount": 5
5 }

```

我们已经有了向区块添加交易的方法，因此剩下的部分就很简单了：

```

2 def new_transaction():
3     values = request.get_json()
4
5     # Check that the required fields are in the POST'ed data
6     required = ['sender', 'recipient', 'amount']
7     if not all(k in values for k in required):
8         return 'Missing values', 400
9
10    # Create a new Transaction
11    index = blockchain.new_transaction(values['sender'], values['recipient'], values['amount'])
12
13    response = {'message': f'Transaction will be added to Block {index}'}
14    return jsonify(response), 201

```



挖掘端正是奇迹发生的地方，它只做三件事：计算 PoW；通过新增一个交易授予矿工一定数量的比特币；构造新的区块并将其添加到区块链中。

```

1 @app.route('/mine', methods=['GET'])
2 def mine():
3     # We run the proof of work algorithm to get the next proof...
4     last_block = blockchain.last_block
5     last_proof = last_block['proof']
6     proof = blockchain.proof_of_work(last_proof)
7
8     # We must receive a reward for finding the proof.
9     # The sender is "0" to signify that this node has mined a new coin.
10    blockchain.new_transaction(
11        sender="0",
12        recipient=node_identifier,
13        amount=1,
14    )
15
16    # Forge the new Block by adding it to the chain
17    block = blockchain.new_block(proof)
18
19    response = {
20        'message': "New Block Forged",
21        'index': block['index'],
22        'transactions': block['transactions'],
23        'proof': block['proof'],
24        'previous_hash': block['previous_hash'],
25    }
26    return jsonify(response), 200

```



需注意交易的接收者是我们自己的服务器节点，目前我们做的大部分事情都只是围绕 Blockchain 类进行交互。到此，我们的区块链就算完成了。

使用 Python 展示，如下。

第四步：一致性

这真的很棒，我们已经有了一个基本的区块链可以添加交易和挖矿。但是，整个区块链系统必须是分布式的。既然是分布式的，那么我们究竟拿什么保证所有节点运行在同一条链上呢？这就是一致性问题，我们要想在网络中添加新的节点，就必须实现保证一致性的算法。

在实现一致性算法之前，我们需要找到一种方式让一个节点知道它相邻的节点。每个节点都需要保存一份包含网络中其它节点的记录。让我们新增几个接口：

1. /nodes/register 接收以 URL 的形式表示的新节点的列表
2. /nodes/resolve 用于执行一致性算法，用于解决任何冲突，确保节点拥有正确的链

```
1 ...
2 from urllib.parse import urlparse
3 ...
4
5 class Blockchain(object):
6     def __init__(self):
7         ...
8         self.nodes = set()
9         ...
10
11     def register_node(self, address):
12         parsed_url = urlparse(address)
13         self.nodes.add(parsed_url.netloc)
```

注意到我们用 set 来储存节点，这是一种避免重复添加节点的简便方法。

前面提到的冲突是指不同的节点拥有的链存在差异，要解决这个问题，我们规定最长的合规的链就是最有效的链，换句话说，只有最长且合规的链才是实际存在的链。

让我们再添加两个方法，一个用于添加相邻节点，另一个用于解决冲突。

```
1 ...
2 import requests
3
4 class Blockchain(object):
5     ...
6
7     def valid_chain(self, chain):
8         last_block = chain[0]
9         current_index = 1
10
```



```

14         print(f'{block}')
15         print("\n-----\n")
16         # Check that the hash of the block is correct
17         if block['previous_hash'] != self.hash(last_block):
18             return False
19
20         # Check that the Proof of Work is correct
21         if not self.valid_proof(last_block['proof'], block['proof']):
22             return False
23
24         last_block = block
25         current_index += 1
26
27     return True
28
29     def resolve_conflicts(self):
30         neighbours = self.nodes
31         new_chain = None
32
33         # We're only looking for chains longer than ours
34         max_length = len(self.chain)
35
36         # Grab and verify the chains from all the nodes in our network
37         for node in neighbours:
38             response = requests.get(f'http://{node}/chain')
39
40             if response.status_code == 200:
41                 length = response.json()['length']
42                 chain = response.json()['chain']
43
44                 # Check if the length is longer and the chain is valid
45                 if length > max_length and self.valid_chain(chain):
46                     max_length = length
47                     new_chain = chain
48
49         # Replace our chain if we discovered a new, valid chain longer than ours
50         if new_chain:
51             self.chain = new_chain
52             return True
53
54     return False

```



现在你可以新开一台机器，或者在本机上开启不同的网络接口来模拟多节点的网络，或者邀请一些朋友一起来测试你的区块链。

我希望本文能激励你创造更多新东西。我之所以对数字货币入迷，是因为我相信区块链会很快改变我们看待事物的方式，包括经济、政府、档案管理等。

译者补充参考：[比特币是什么](#)

好文要顶

关注我

收藏该文



kidney

关注 - 1

粉丝 - 52

+加关注

« 上一篇: [Vue.js 计算属性的秘密](#)

» 下一篇: [读懂源码: 一步一步实现一个 Vue](#)

4

0

posted @ 2017-10-04 20:50 kidney 阅读(4709) 评论(2) 编辑 收藏

#1楼 2017-10-06 19:24 夜里码码

相当有用

支持(0) 反对(0)

#2楼 2017-10-10 14:46 木土家的田童

你在本地运行起来了么?

支持(0) 反对(0)

[刷新评论](#) [刷新页面](#) [返回顶部](#)

注册用户登录后才能发表评论, 请 [登录](#) 或 [注册](#), [访问网站首页](#)。

【推荐】超50万VC++源码: 大型工控、组态\仿真、建模CAD源码2018!

【活动】世界各地的青年团体, 将在 2050 握手团聚

【抢购】新注册用户域名抢购1元起

腾讯云

小程序普惠节

精美模板1元选购
开发套餐30元/月起

立即选购

阿里云

告别高昂运维费用 云计算全面助力

40+款核心产品免费半年 再+8000津贴任意采购

立即申请

