如何用Python快速实现区块链？

原创 *2018-03-25 Adil Moujahid* CSDN

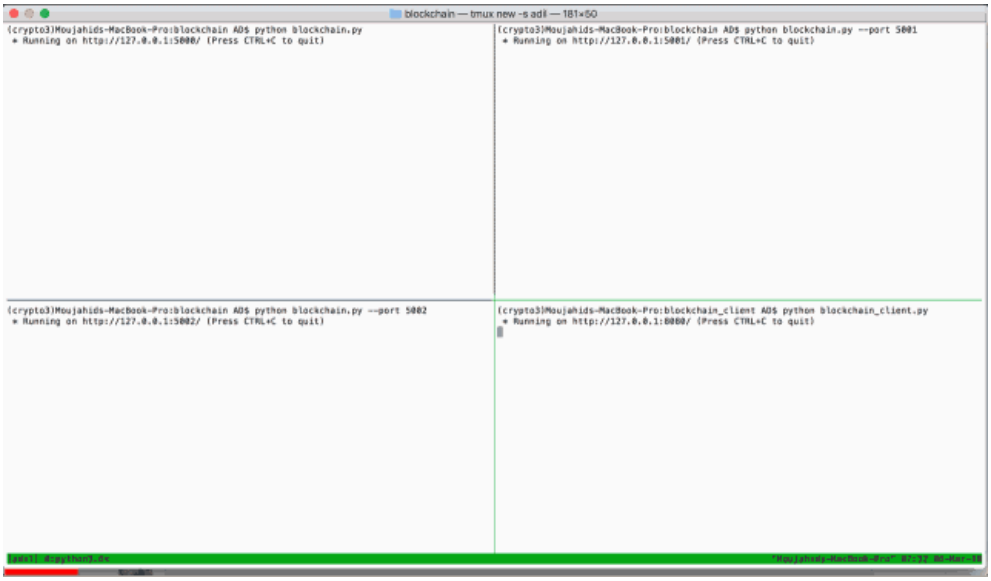
【编者按】区块链自诞生那天起就是互联网业界最富有争议与分歧的技术之一。它是比特币和其他加密货币幕后的核心技术，最近几年来引起了广泛的关注。

区块链的核心是分布式数据库，任何两方之间可以进行直接交易，而无需经过权威的中间机构。这个简单而又强大的概念对银行、政府以及市场等各种机构带来了很大影响。任何以中心化数据库作为核心竞争优势的商家或组织都有可能受到来自区块链技术的威胁。

这篇文章的主要目标是从实用的角度介绍区块链，不会讨论比特币等加密货币的价格等炒作。第一小节和第二小节将介绍区块链背后的核心概念，第三小节将介绍使用Python实现区块链的方法。同时还会介绍两个web应用程序，帮助终端用户轻松地与区块链交互。

请注意我在这里使用比特币作为例子向大家介绍更为通用的“区块链”技术，本文中所介绍的大多数概念也同样适用于其他区块链以及加密货币。

以下是我们将在第三节中创建的web应用程序的动画截图。

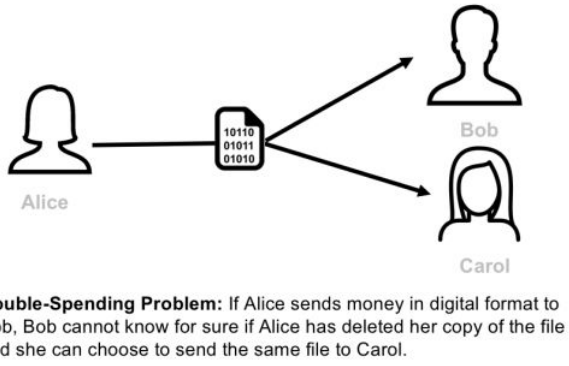


1. 区块链速成

区块链的一切源自2008年，一位匿名人士或组织以中本聪的名义发表的一篇白皮书。这篇白皮书的题目为《比特币：端对端的电子货币系统》（原文请参照这里：https://bitcoin.org/bitcoin.pdf），为后世著名的区块链奠定了基础。在最初的比特币白皮书中，中本聪阐述了如何建立端对端的数字货币系统，实现在线支付直接从一方传输到另一方，而无需经过中间机构。这个系统解决了电子货币中重要的问题：即“双花”（double-spending）。

1.1 双花是什么？

假设A（Alice）要付给B（Bob）1块钱。如果用现金，那么在交易完成之后，A手里的1块钱就没有了。如果A和B使用数字货币，那么问题就复杂了。数字货币是数字形式并且很容易复制。比如A通过邮件给B发送了一个价值为1块钱的数字文件，B却不知道是否A已经删除了这份文件的复制品。如果A手里还有这个1块钱的文件，那么她可以继续把相同的文件发给C（Carol）。这个问题称之为“双花”。



Double-Spending Problem: If Alice sends money in digital format to Bob, Bob cannot know for sure if Alice has deleted her copy of the file and she can choose to send the same file to Carol.

图片：双花问题：如果A（Alice）发送电子货币给B（Bob）。B无法知道是否A已经删除了文件的复制品，且A可以将相同的文件继续发给C（Carol）。

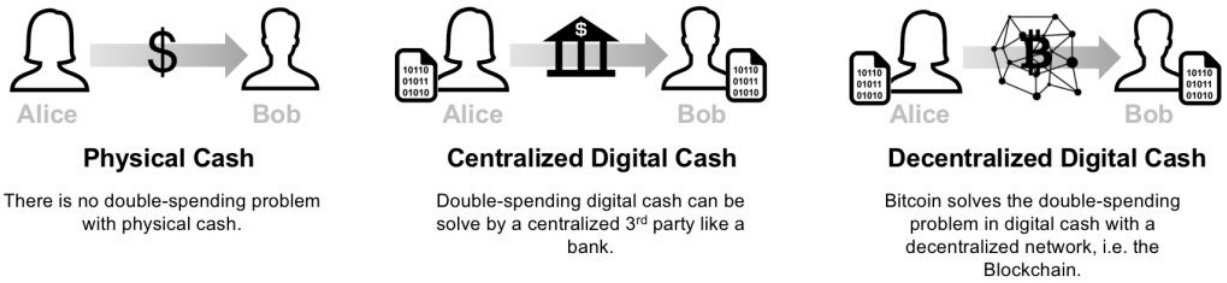
解决双花问题的一种方法是在A、B以及所有其他参与者的网络内找一个可信任的第三方（比如银行）。这个第三方负责管理中心化的账本，记录和验证该网络中的所有交易。这个解决方案的缺点在于：系统需要一个中心化的第三方提供信任。

1.2 比特币：双花问题的去中心化解决方案

为了解决双花问题，中本聪提出了一个公共账本，即比特币的区块链可以记录网络中的所有交易。比特币区块链

具有以下特征：

- **分布式：这个账本并没有保存在中心服务器上，而是在多台计算机之间复制。任何连接了互联网的计算机都可以下载完整的区块链。**
- **密码学：利用密码学保证发送方确实拥有他要发送的比特币，以及决定将交易添加到区块链的方式。**
- **不可修改：区块链只允许追加，换句话说，只能向区块链添加交易，但是不可以删除或修改。**
- **使用工作量证明（Proof of Work：PoW）：网络中有一种特殊的参与者，称为矿工，他们互相竞争去试图解决一个密码学难题，以获得向比特币区块链添加区块的权利。这个过程称为工作量证明，可以确保系统的安全性（后面将详细介绍）。**



图片1：实体现金

实体现金不存在双花的问题。

图片2：中心化的数字货币

可以通过中心化的第三方解决双花问题，比如银行。

图片3：去中心化的数字货币

比特币通过去中心化网络解决了数字货币中的双花问题，即区块链。

发送比特币的具体流程如下：

- **第一步（只需一次）：创建比特币钱包。比特币的发送方或接收方需要创建比特币钱包。比特币钱包存储了两方面的信息：私钥和公钥。私钥包含一组秘密的数字，货币主人可以使用私钥将比特币发送给另一个用户，或消费接受比特币支付的服务。公钥包含的数字则负责接收比特币。公钥也称为比特币的地址（这么说也不完全正确，但是为了简单起见，我们可以假定公钥和比特币的地址相同）。请注意钱包并没有存储比特币本身，比特币的余额信息保存在比特币区块链中。**
- **第二步：创建比特币交易。如果A想发送1比特币（BTC）给B，那么A需要通过私钥连接她的比特币钱包，然后创建一笔交易，其中包含了她要发送的比特币金额，以及发送地址（即B的公共地址）。**
- **第三步：在比特币网络上广播该交易。在创建好比特币交易后，A需要将该交易广播到整个比特币网络上。**
- **第四步：确认交易。一个监听比特币网络的矿工用A的公钥鉴定该交易，确认A的钱包中有足够的比特币余额（本例中A的钱包中最少要有1BTC），并在比特币区块链中添加一个新的记录存储该交易的详细信息。**
- **第五步：向所有矿工广播区块链的变更。交易一经确认，该矿工需要向所有矿工广播区块链的变化，确认所有矿工将该变化同步到自己的副本中。**

2. 深入区块链技术

本小节的目标是深入介绍构成区块链的组成部分。我们将介绍公开密钥加密、hash函数、挖矿，以及区块链的安全。

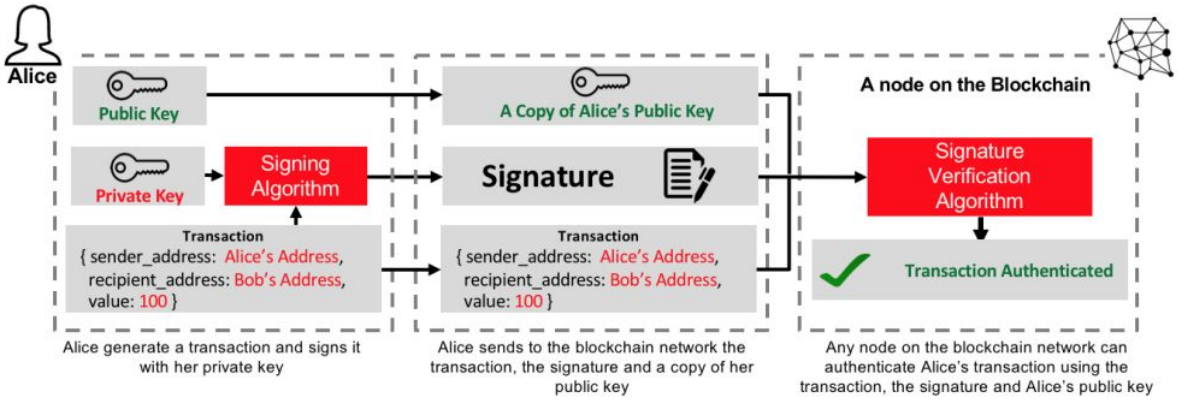
2.1 公开密钥加密

公开密钥加密，或称非对称加密，指的是使用了一对密钥的加密系统，一个密钥为公钥，可以广泛发放，另一个密钥为私钥，只有持有者才知道。这种加密有两方面的应用：第一是身份验证，公钥可以验证信息确实是私钥持有者发出来的；第二是加密，用公钥加密的信息，只有私钥持有者可以解密。

RSA加密算法和椭圆曲线数字签名(ECDSA)算法是最流行的公开密钥加密算法。

具体到比特币，它采用ECDSA算法生成比特币钱包。比特币使用各种密钥和地址，但是为了简单起见，我们在本文中假定每个比特币钱包持有一对公钥/私钥，且比特币地址就是钱包的公钥。如果你对比特币钱包的技术细节很感兴趣，那么我推荐你可以阅读这篇文章：
https://en.bitcoin.it/wiki/Technical_background_of_version_1_Bitcoin_addresses。

发送或接收BTC时，用户首先要生成包含了私钥和公钥的钱包。如果A想发送一些BTC给B，那么她可以输入自己和B的公钥，以及发送的金额，然后创建一笔交易。接下来A需要用自己的私钥在交易上签名。区块链上的一台计算机使用A的公钥验证该交易是真实的，并向区块链添加该交易记录。



Authentication Process for Transactions on the Blockchain

区块链上交易的认证过程

图片1：A生成了一笔交易并用私钥签名

图片2：A将该交易、签名和公钥的副本发送到区块链网络上

图片3：区块链网络上的任何一台计算机都可以使用该交易、签名和A的公钥验证交易

2.2 hash函数和挖矿

所有比特币交易会被分组记录到文件中，称为区块。比特币每10分钟新增一个交易区块。一旦有新的区块进入区块链，那么该区块就不可改变了，它不能被删除或修改。网络中有一种特殊的用户叫做矿工，他们负责为交易创建新区块。矿工必须使用发送者的公钥验证每笔交易，确认发送者有足够的余额支付，然后将交易加入到区块。矿工可以自由选择将哪个交易加入到区块中，因此发送者需要向矿工支付交易费，作为他们将交易添加到区块的奖励。

区块需要经过“开采”才能被区块链接受。为了开采一个区块，矿工需要给一个密码学难题找到一个非常非常稀

有的解。如果一个经过开采的区块被区块链接受，那么该矿工可以收到交易费之外的额外奖励。挖矿的过程也被称作工作量证明（Proof of Work：PoW），其主要机制就是保证区块链的无信任和安全（我们将在后面详细讨论区块链的安全问题）。

Hash和区块链的加密机制

为了更好地理解区块链的这个密码学难题，我们需要先介绍hash函数。hash函数用于将任意长度的数据映射成固定长度。hash函数的返回值称之为hash。hash函数常常通过探查重复记录加速数据库的查询，它们还被广泛用于加密。加密hash函数可以轻松地验证输入数据是否与给定的hash值相符，但是如果输入数据未知，用已知的hash值推导输入却非常困难。

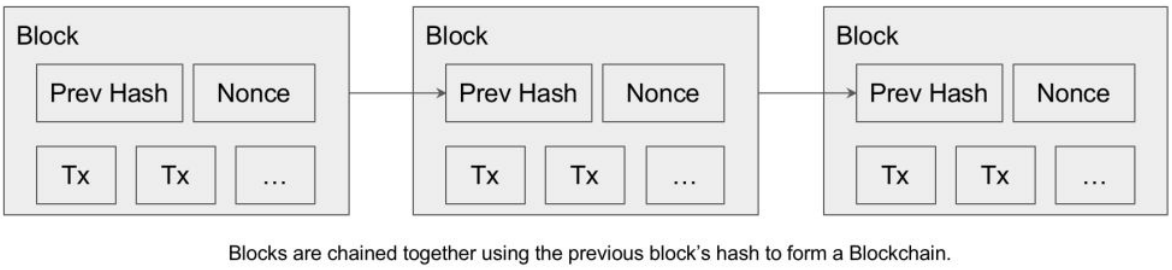
比特币使用的加密hash函数叫做SHA-256。区块数据（比特币交易）与一个随机数（称为nonce）组合后计算SHA-256。改变区块数据或nonce，我们可以得到完全不同的hash值。对于一个合法或“经过开采”的区块，它与nonce的hash值需要满足一定的条件。例如， hash值的头4位数字必须等于“0000”。我们可以通过增加条件的复杂度来增加挖矿的难度，比如我们可以增加hash值开头所需的0的个数。

矿工需要求解的密码学难题就是找到一个nonce以确保生成的hash值符合挖矿条件。如下所示的应用可以模仿挖矿。当在“Data”文本框中输入数字或改变Nonce时，就能看到hash值的变化。点击“挖矿（Mine）”的时候，应用从0开始逐个代入随机数字，计算hash值并检查开头的4位数字是否等于“0000”。如果开头的4位数字不等于“0000”，那么将随机数字+1，然后重复整个计算过程，直到找到一个nonce值满足条件。如果该区块开采成功，那么背景色会变成绿色。

2.3 从区块到区块链

正如我们在前面小节所讨论的，交易分组保存在区块中，然后再把区块添加到区块链。为了建立区块的链，每个新的区块都必须使用上一个区块hash值作为自己的数据的一部分。创建新区块的时候，矿工需要选择一组交易，把上一个区块的hash值加进来，然后按照上述的方式开采当前区块。

任何区块数据的变化都会影响所有后继区块的hash值，导致它们失效。这赋予了区块链不可改变的特性。

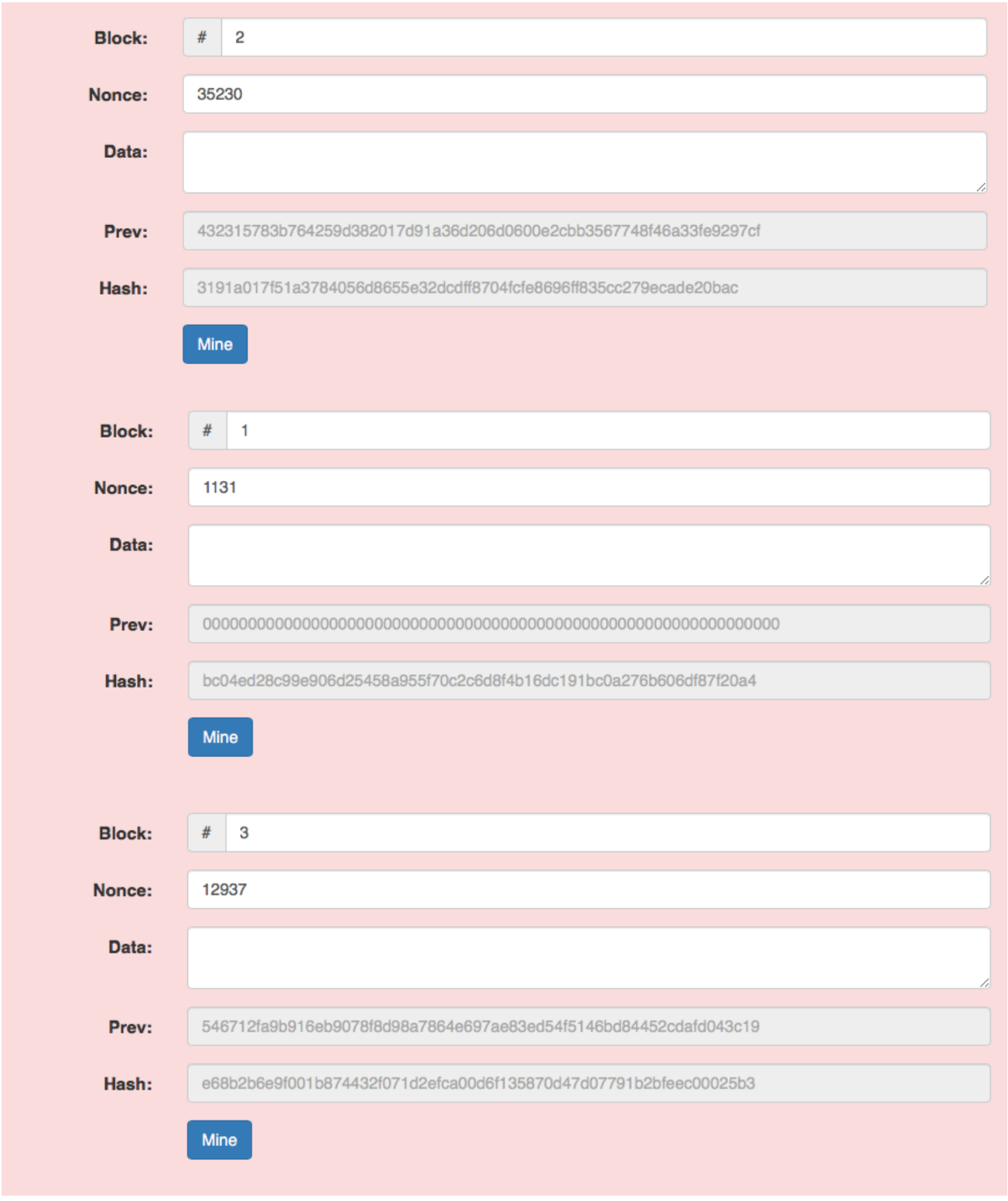


Blocks are chained together using the previous block's hash to form a Blockchain.

图片：结成链的区块使用上一个区块的hash，从而组成区块链。

如下所示的应用可以模拟由3个区块构成的区块链。在“Data”文本框输入数据或改变“Nouce”时，hash值将发生变化，且“Prev”（上一个hash）的值也会发生变化。你可以分别选择每个区块，然后点击“开采（Mine）”按钮模拟挖矿的过程。开采完所有3个区块之后，如果变更区块1或2中的数据，就会看到后续的所有区

块都将失效。



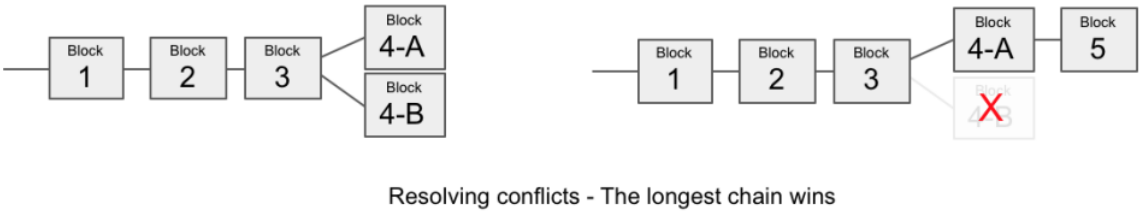
图片：上述挖矿模拟器采用了Anders Brownworth的区块链演示，请点击这里获取详情：https://anders.com/blockchain/blockchain.html

2.4 向区块链追加区块

比特币网络中的所有矿工都在互相竞争，找到合法的区块追加到区块链，并获得奖励。为一个合法的区块找到nonce是低概率的事件，但是因为矿工的数量众多，所以由某个矿工找到一个区块的概率就非常高了。第一个提交合法区块的矿工可以将自己的区块追加到区块链，并获得比特币的奖励。但是如果两个或多个矿工同时提交区块会怎么样呢？

解决冲突

如果有两个矿工同时挖到了一个区块，那么网络中就会出现两个不同的区块链，那么我们需要等下个区块出现才能解决这个冲突。一些矿工决定继续开采区块链1，而其他人则选择区块链2。第一个找到新区块的矿工可以解决这个冲突。如果新区块是在区块链1上开采出来的，那么区块链2就失效了，所以前一个区块的奖励就要给区块链1中的矿工，且区块链2中的交易不会被加入区块链，所以它必须返回交易池与区块链1同步，然后继续下个区块。简而言之，如果区块链中有冲突，那么最长的那个链说了算。



Resolving conflicts - The longest chain wins

图片：解决冲突——最长的链说了算

2.5 区块链与双花

在本节中，我们将介绍区块链上应对双花攻击的最流行的做法，以及用户可以采用的防范措施。

速度攻击

一个攻击者以非常快的速度将同一个货币发到两个地址。为了阻止这种攻击，我们推荐至少等到下一个区块被确认再接受付款。

Finney攻击

一个攻击者预先挖掘出一个交易的区块，并在公布这个区块之前，在另一个交易中花掉同一个比特币。这种情况下，第二个交易将会是无效的。为了阻止这种攻击，我们推荐至少等6个区块被确认后再接受付款。

主要攻击（也被称之为51%攻击）

在这种攻击中，攻击者必须拥有网络中51%的计算力。这个攻击者可以创建一笔交易，并广播到整个网络中，然后在一个秘密的区块链上双花前一笔交易的货币，然后在此区块链上开采。由于该攻击者拥有主要的计算力，所以他肯定可以在某一时刻拥有比“诚实”的网络更长的链。之后，他可以用自己更长的区块链代替“诚实”的区块链，并取消原始的交易记录。因为这种攻击在比特币等区块链中非常昂贵，所以不大可能会出现。

3. 用Python实现区块链

在本节中，我们将用Python实现基本的区块链以及区块链的客户端。我们的区块链将拥有下列特征：

- 可以向区块链添加多个节点
- 工作量证明（Proof of Work：PoW）
- 两个节点冲突的简易解决方案
- 使用RSA加密交易

我们的区块链客户端将拥有下列特征：

- 基于RSA算法，使用公钥/私钥加密算法生成钱包
- 用RSA加密算法生成交易

我们还将实现两个dashboard：

- 面向矿工的“区块链前端”
- 面向用户的“区块链客户端”，提供生成钱包和发送货币的服务

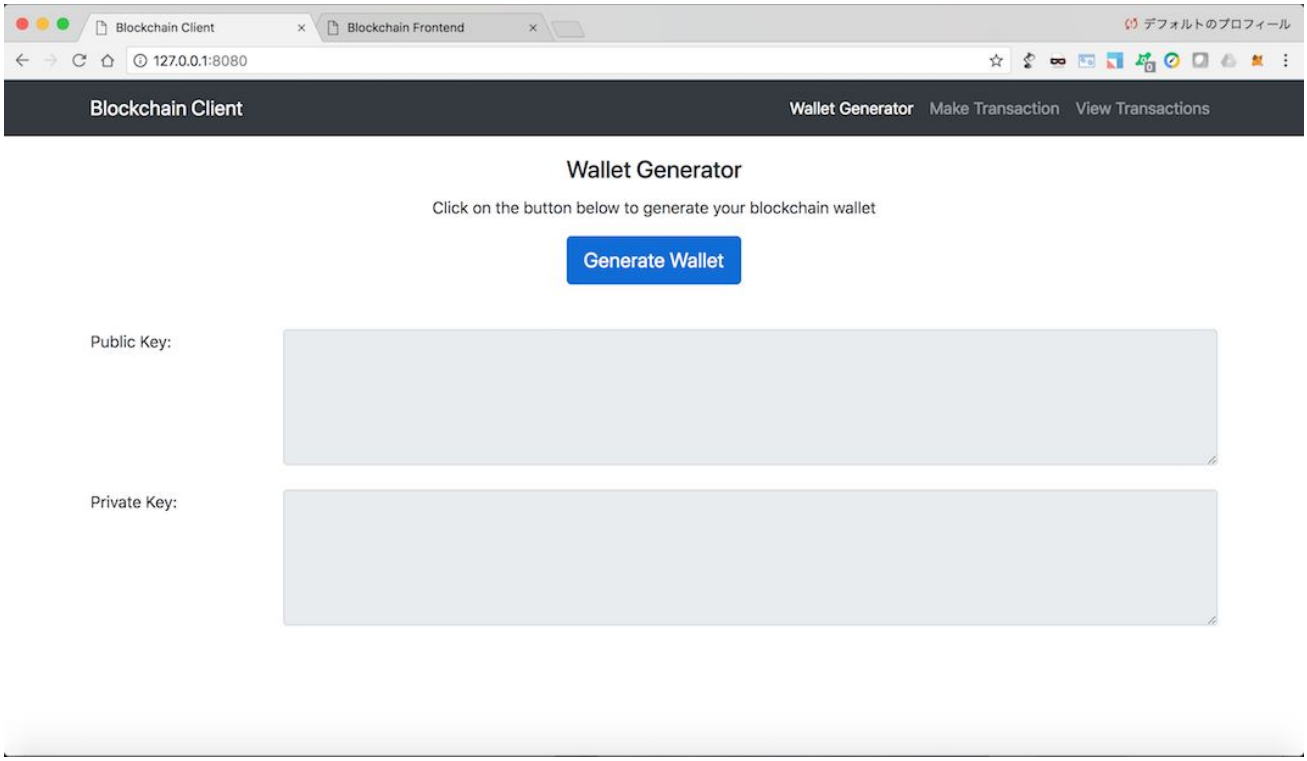
区块链的实现大部分基于GitHub上的这个项目：https://github.com/dvf/blockchain。为了在交易中使用RSA加密，我对原始代码做了一些改变。生成钱包和交易编码源于这个Jupyternotebook：https://github.com/julienr/ipynb_playground/blob/master/bitcoin/dumbcoin/dumbcoin.ipynb。而两个dashboard是用HTML/CSS/JS从零编写的。

你可以从此处下载所有的代码：https://github.com/adilmoujahid/blockchain-python-tutorial.

请注意本文中的实现方法仅用于教学目的，安全性上有所欠缺，没有良好的可扩展性，且省略了很多重要的功能，所以请勿用于产品开发。

3.1 实现区块链客户端

你可以使用终端，前往blockchain_client目录，然后输入pythonblockchain_client.py，启动区块链客户端。在浏览器中输入http://localhost:8080，就可以看到如下的dashboard。



Dashboard的导航条上有3个页签：

- Wallet Generator（钱包生成器）：使用RSA加密算法生成钱包（一对公钥/私钥）
- Make Transaction（创建交易）：生成交易并发送到区块链的节点
- View Transasctions（查看交易）：查看区块链上的交易

至少有一个运行的区块链节点，才可以创建或查看交易。关于如何运行节点，请参照下节内容。

下面是对blockchain_client.py代码中最重要部分的介绍。

我们定义一个名为Transaction的python类，它有4个属性：sender_address, sender_private_key，recipient_address, value。代表了创建交易时所必须的4种信息。

to_dict()方法以Python字典格式（不包含发送者的私钥）返回交易信息。sign_transaction()方法接收交易信息（不包含发送者的私钥），并使用发送者的私钥签名。

```
class Transaction:

    def __init__(self, sender_address, sender_private_key, recipient_address, value):
        self.sender_address = sender_address
        self.sender_private_key = sender_private_key
        self.recipient_address = recipient_address
        self.value = value

    def __getattr__(self, attr):
        return self.data[attr]

    def to_dict(self):
        return OrderedDict({'sender_address': self.sender_address,
                            'recipient_address': self.recipient_address,
                            'value': self.value})

    def sign_transaction(self):
        """
        Sign transaction with private key
        """
        private_key = RSA.importKey(binascii.unhexlify(self.sender_private_key))
        signer = PKCS1_v1_5.new(private_key)
        h = SHA.new(str(self.to_dict()).encode('utf8'))
        return binascii.hexlify(signer.sign(h)).decode('ascii')
```

下面这行代码对Python Flask进行了初始化，我们可以使用这个应用创建不同的API，与区块链和客户端交互。

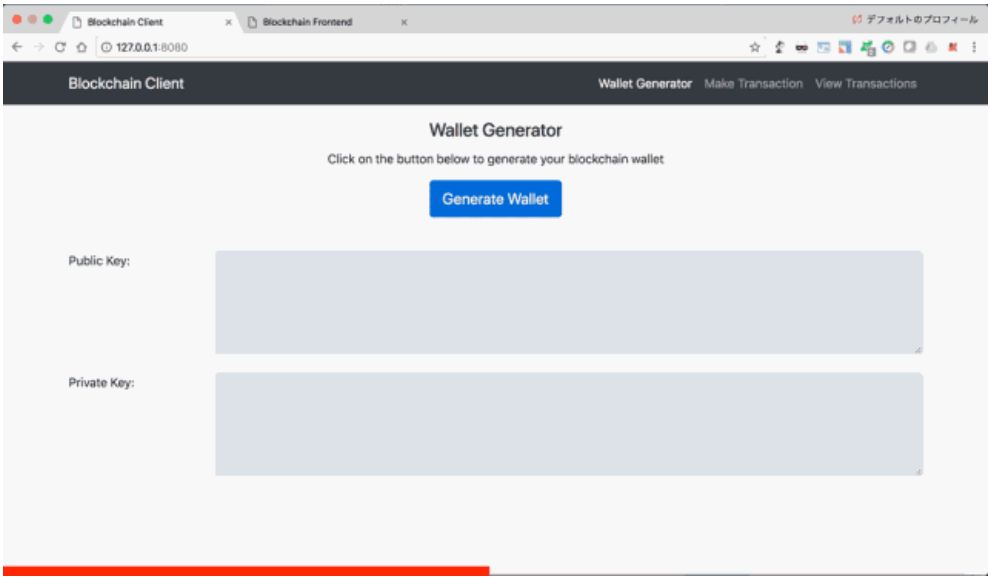
```
app = Flask(__name__)
```

如下我们定义了3个Flask路由，用以返回html页面。每个页签一个html页面。

```
@app.route('/')
def index():
  return render_template('./index.html')

@app.route('/make/transaction')
def make_transaction():
    return render_template('./make_transaction.html')

@app.route('/view/transactions')
def view_transaction():
    return render_template('./view_transactions.html')
```

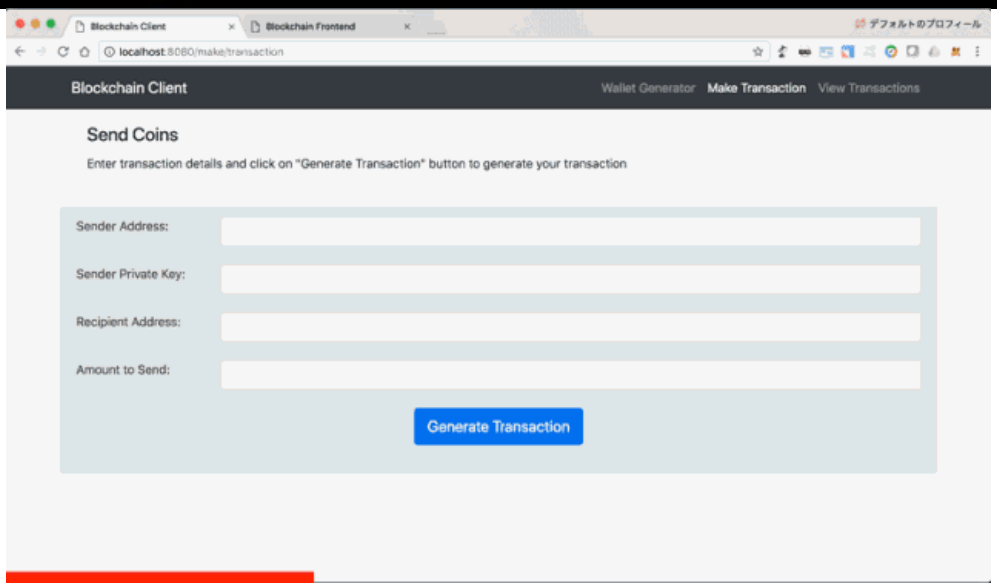
如下我们定义了生成钱包（一对公钥/私钥）的API。

```
@app.route('/wallet/new', methods=['GET'])
def new_wallet():
  random_gen = Crypto.Random.new().read
  private_key = RSA.generate(1024, random_gen)
  public_key = private_key.publickey()
  response = {
    'private_key': binascii.hexlify(private_key.exportKey(format='DER')).decode('ascii'),
    'public_key': binascii.hexlify(public_key.exportKey(format='DER')).decode('ascii')
  }

  return jsonify(response), 200
```



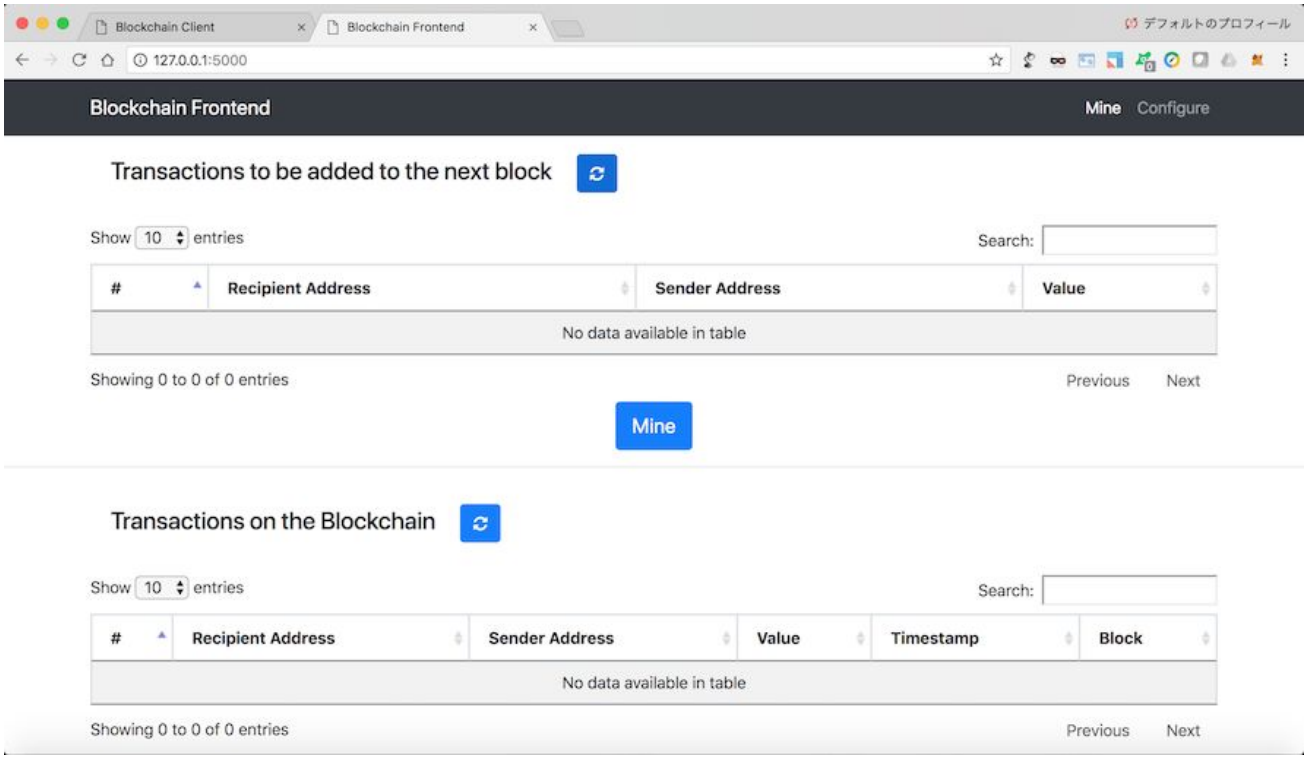
如下我们定义的API接收sender_address, sender_private_key,recipient_address, value作为输入，然后返回交易（不包含私钥）和签名。

```
@app.route('/generate/transaction', methods=['POST'])
def generate_transaction():

  sender_address = request.form['sender_address']
  sender_private_key = request.form['sender_private_key']
  recipient_address = request.form['recipient_address']
  value = request.form['amount']

  transaction = Transaction(sender_address, sender_private_key, recipient_address, value)

  response = {'transaction': transaction.to_dict(), 'signature': transaction.sign_transaction()}

  return jsonify(response), 200
```



3.2 实现区块链

你可以使用终端，前往blockchain目录，然后输入python blockchain_client.py或python blockchain_client.py -p <PORT NUMBER>，启动区块链的一个节点。如果不指定端口，默认会使用端口5000。在浏览器中输入http://localhost:<端口号>，就可以看到区块链前端的dashboard。



Dashboard的导航条上有2个页签：

- Mine（挖矿）：查看交易和区块链数据，并开采新的交易块。
- Configure（配置）：配置不同区块链节点之间的连接。

下面是对blockchain.py代码中最重要部分的介绍。

我们定义了一个Blockchain类，它有如下属性：

- transactions：可以加入到下个区块的交易列表。
- chain：实际的区块链，包含一组区块。
- nodes：一组节点的url。区块链使用这些节点获取其他节点的区块链数据，并在不同步的时候更新自己的区块链。
- node_id：识别区块链节点的随机字符串。

这个Blockchain类还实现了如下方法：

- register_node(node_url): 向节点列表添加新区块链节点。
- verify_transaction_signature(sender_address,signature, transaction): 检查所提供的签名是否与公钥（sender_address）生成的交易签名相符。
- submit_transaction(sender_address,recipient_address, value, signature)：如果签名验证成功，将交易添加到交易列表。
- create_block(nonce, previous_hash): 向区块链添加交易区块。
- hash(block)：创建块的SHA-256 hash值。
- proof_of_work()：工作量证明算法。寻找满足挖矿条件的nonce。
- valid_proof(transactions, last_hash,nonce, difficulty=MINING_DIFFICULTY): 检查hash值是否满足挖矿条件。这个函数用于proof_of_work()函数的内部。
- valid_chain(chain)：检查区块链是否合法。
- resolve_conflicts()：用网络中最长的链替换一个链，从而解决两个区块链节点之间的冲突。

```
class Blockchain:

    def __init__(self):

        self.transactions = []
        self.chain = []
        self.nodes = set()
        #Generate random number to be used as node_id
        self.node_id = str(uuid4()).replace('-', '')
        #Create genesis block
        self.create_block(0, '00')

    def register_node(self, node_url):
        """
        Add a new node to the list of nodes
        """
        ...

    def verify_transaction_signature(self, sender_address, signature, transaction):
        """
        Check that the provided signature corresponds to transaction
        signed by the public key (sender_address)
        """
        ...

    def submit_transaction(self, sender_address, recipient_address, value, signature):
        """
        Add a transaction to transactions array if the signature verified
        """
        ...

    def create_block(self, nonce, previous_hash):
        """
        Add a block of transactions to the blockchain
        """
        ...
```

下面这行代码对Python Flask进行了初始化，我们可以使用这个应用创建不同的API，与区块链交互。

```
app = Flask(__name__)
CORS(app)
```

下面我们初始化区块链的实例。

```
blockchain = Blockchain()
```

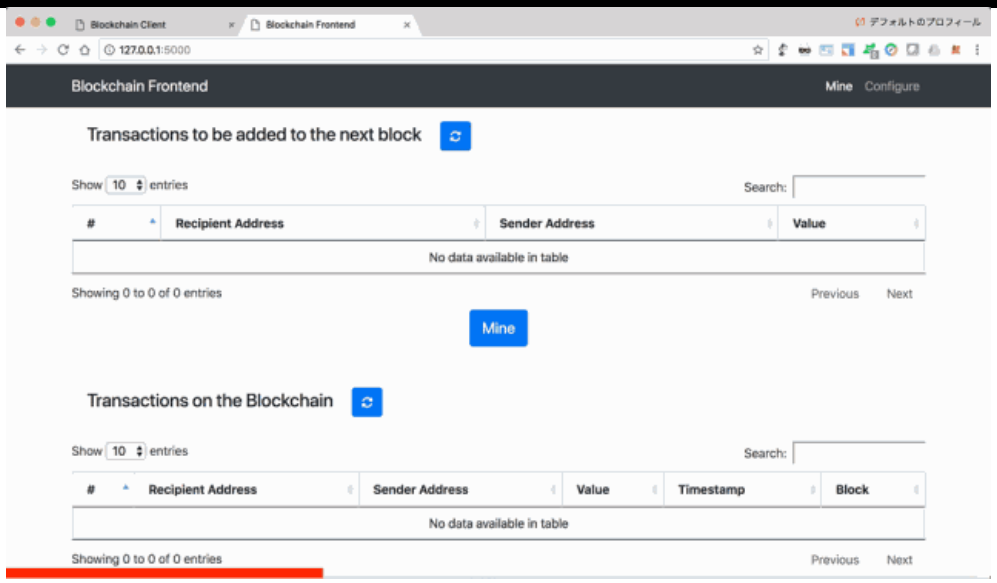
如下我们定义了2个Flask路由，用以返回区块链前端dashboard的html页面

```
@app.route('/')
def index():
    return render_template('./index.html')

@app.route('/configure')
def configure():
    return render_template('./configure.html')
```

如下我们定义了Flask的API，用以管理交易以及区块的挖矿。

- '/transactions/new'：这个API接受'sender_address', 'recipient_address','amount' and 'signature'作为输入，将交易添加到交易列表，如果签名合法，这个交易会被加入到下个区块。
- '/transactions/get'：这个API返回所有的准备添加到下个区块的备选交易。
- '/chain'：这个API返回所有区块链的数据。
- '/mine'：这个API运行工作量证明的算法，并将新的交易区块添加到区块链。

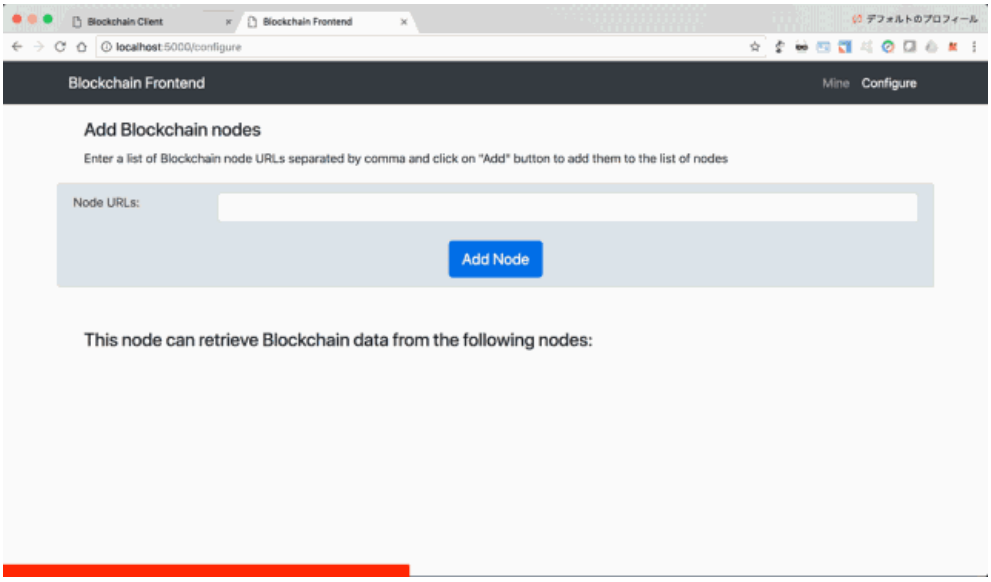
```python
@app.route('/transactions/new', methods=['POST'])
def new_transaction():
    values = request.form

    # Check that the required fields are in the POST'ed data
    required = ['sender_address', 'recipient_address', 'amount', 'signature']
    if not all(k in values for k in required):
        return 'Missing values', 400
    # Create a new Transaction
    transaction_result = blockchain.submit_transaction(values['sender_address'], values['recipient_address'], values['amo
unt'], values['signature'])

    if transaction_result == False:
        response = {'message': 'Invalid Transaction!'}
        return jsonify(response), 406
    else:
        response = {'message': 'Transaction will be added to Block '+ str(transaction_result)}
        return jsonify(response), 201

@app.route('/transactions/get', methods=['GET'])
def get_transactions():
    #Get transactions from transactions pool
    transactions = blockchain.transactions

    response = {'transactions': transactions}
    return jsonify(response), 200

@app.route('/chain', methods=['GET'])
def full_chain():
    response = {
        'chain': blockchain.chain,
        'length': len(blockchain.chain),
    }
    return jsonify(response), 200
```



如下我们定义了Flask的API，用以管理区块链的节点。

- '/nodes/register': 这个API以节点列表的url作为输入，将它们添加到节点列表。
- '/nodes/resolve': 这个API通过用网络中最长的链替换一个本地的链，解决两个区块链节点之间的冲突。
- '/nodes/get':这个API返回节点列表。

```python
@app.route('/nodes/register', methods=['POST'])
def register_nodes():
    values = request.form
    nodes = values.get('nodes').replace(" ", "").split(',')

    if nodes is None:
        return "Error: Please supply a valid list of nodes", 400

    for node in nodes:
        blockchain.register_node(node)

    response = {
        'message': 'New nodes have been added',
        'total_nodes': [node for node in blockchain.nodes],
    }
    return jsonify(response), 201


@app.route('/nodes/resolve', methods=['GET'])
def consensus():
    replaced = blockchain.resolve_conflicts()

    if replaced:
        response = {
            'message': 'Our chain was replaced',
            'new_chain': blockchain.chain
        }
    else:
        response = {
            'message': 'Our chain is authoritative',
            'chain': blockchain.chain
        }
    return jsonify(response), 200


@app.route('/nodes/get', methods=['GET'])
def get_nodes():
    nodes = list(blockchain.nodes)
    response = {'nodes': nodes}
    return jsonify(response), 200
```



总结

在这篇文章中，我们介绍了区块链背后的核心概念，并学习了用Python实现区块链的方法。为了简单起见，我并没有涉及太多技术细节，比如钱包地址以及Merkel树等。如果你想了解更多这方面的知识，我建议你阅读原版的比特币白皮书，并订阅比特币维基百科（https://en.bitcoin.it/wiki/Main_Page），以及参照AndreasAntonopoulos的著作：精通比特币第二版（https://download.csdn.net/download/weixin_36839098/10212083）。

链接：http://adilmoujahid.com/posts/2018/03/intro-blockchain-bitcoin-python/

作者：Adil Moujahid

译者：马晶慧

责编：琥珀

——————— 推荐阅读 ———————

点击图片即可阅读