教程获取方式见文章底部。

虽然有些人认为区块链是一个早晚会出现问题的解决方案，但是毫无疑问，这个创新技术是一个计算机技术上的奇迹。那么，究竟什么是区块链呢？

区块链

> 以比特币（Bitcoin）或其它加密货币按时间顺序公开地记录交易的数字账本。

更通俗的说，它是一个公开的数据库，新的数据存储在被称之为区块（block）的容器中，并被添加到一个不可变的链（chain）中（因此被称为区块链（blockchain）），之前添加的数据也在该链中。对于比特币或其它加密货币来说，这些数据就是一组组交易，不过，也可以是其它任何类型的数据。

区块链技术带来了全新的、完全数字化的货币，如比特币和莱特币（Litecoin），它们并不由任何中心机构管理。这给那些认为当今的银行系统是骗局并将最终走向失败的人带来了自由。区块链也革命性地改变了分布式计算的技术形式，如以太坊（Ethereum）就引入了一种有趣的概念：智能合约（smart contract）。

在这篇文章中，我将用不到 50 行的 Python 2.x 代码实现一个简单的区块链，我把它叫做 SnakeCoin。

不到 50 行代码的区块链

我们首先将从定义我们的区块是什么开始。在区块链中，每个区块随同时间戳及可选的索引一同存储。在 SnakeCoin 中，我们会存储这两者。为了确保整个区块链的完整性，每个区块都会有一个自识别的哈希值。如在比特币中，每个区块的哈希是该块的索引、时间戳、数据和前一个区块的哈希值等数据的加密哈希值。这里提及的“数据”可以是任何你想要的数据。

```
import hashlib as hasher
```

```
class Block:
  def __init__(self, index, timestamp, data, previous_hash):
    self.index = index
    self.timestamp = timestamp
    self.data = data
    self.previous_hash = previous_hash
    self.hash = self.hash_block()

  def hash_block(self):
    sha = hasher.sha256()
    sha.update(str(self.index) +
           str(self.timestamp) +
           str(self.data) +
           str(self.previous_hash))
    return sha.hexdigest()
```

真棒，现在我们有了区块的结构了，不过我们需要创建的是一个区块链。我们需要把区块添加到一个实际的链中。如我们之前提到过的，每个区块都需要前一个区块的信息。但问题是，该区块链中的第一个区块在哪里？好吧，这个第一个区块，也称之为创世区块，是一个特别的区块。在很多情况下，它是手工添加的，或通过独特的逻辑添加的。

我们将创建一个函数来简单地返回一个创世区块解决这个问题。这个区块的索引为 0，其包含一些任意的数据值，其“前一哈希值”参数也是任意值。

```
import datetime as date



def create_genesis_block():

  # Manually construct a block with

  # index zero and arbitrary previous hash

  return Block(0, date.datetime.now(), "Genesis Block", "0")
```

现在我们可以创建创世区块了，我们需要一个函数来生成该区块链中的后继区块。该函数将获取链中的前一个区块作为参数，为要生成的区块创建数据，并用相应的数据返回新的区块。新的区块的哈希值来自于之前的区块，这样每个新的区块都提升了该区块链的完整性。如果我们不这样做，外部参与者就很容易“改变过去”，把我们的链替换为他们的新链了。这个哈希链起到了加密的证明作用，并有助于确保一旦一个区块被添加到链中，就不能被替换或移除。

```
def next_block(last_block):

  this_index = last_block.index + 1

  this_timestamp = date.datetime.now()

  this_data = "Hey! I'm block " + str(this_index)

  this_hash = last_block.hash

  return Block(this_index, this_timestamp, this_data, this_hash)
```

这就是主要的部分。

现在我们能创建自己的区块链了！在这里，这个区块链是一个简单的 Python 列表。其第一个的元素是我们的创世区块，我们会添加后继区块。因为 SnakeCoin 是一个极小的区块链，我们仅仅添加了 20 个区块。我们通过循环来完成它。

```
    # Create the blockchain and add the genesis block
    blockchain = [create_genesis_block()]
    previous_block = blockchain[0]


    # How many blocks should we add to the chain
    # after the genesis block
    num_of_blocks_to_add = 20


    # Add blocks to the chain
    for i in range(0, num_of_blocks_to_add):
      block_to_add = next_block(previous_block)
      blockchain.append(block_to_add)
      previous_block = block_to_add
      # Tell everyone about it!
      print "Block #{} has been added to the blockchain!".format(block_to_add.index)
      print "Hash: {}n".format(block_to_add.hash)
```

让我们看看我们的成果：

```
Block #1 has been added to the blockchain!
Hash: 6311e5906c3fcbdec077aeb4e3f15aba1a398ffbffe74cafb033163e83c65cb2

Block #2 has been added to the blockchain!
Hash: 45c714818a556cde8ddec533f903daa92acd4ea0a03518a30ae1c3522d0e81ae

Block #3 has been added to the blockchain!
Hash: b58f7644cd153a910eee8a3fd20e2a453bacd13d253e54cadace8d615d80bbae

Block #4 has been added to the blockchain!
Hash: 5048e4dc66538c6baf45c7a7f7a211b23b89612e9afb8a2be485a2849df58005

Block #5 has been added to the blockchain!
Hash: a3f2caa493bfacca51796c63ca7ab4214d6e4403d70751ba526846094adc0160

Block #6 has been added to the blockchain!
Hash: 78384b46b91f52616794edc297997b6317e961130424fb3d5787f2627d3f1308

Block #7 has been added to the blockchain!
Hash: 0ffe13c9e1ce084455bb5247bb850e82bb0ad93d3cf3b6d011e23e4f64b81f13

Block #8 has been added to the blockchain!
Hash: c52392c8a89cc20714c10d0811bfbba373fc16582df3d9a49bf2834a8160f6e7

Block #9 has been added to the blockchain!
Hash: be740eab1b04382df0a980f3e804fa46115fd5aa5587f631f8662a010f31af4f

Block #10 has been added to the blockchain!
Hash: 0a71c411f6a68561abb7899c82e9fa0b81566c7ba00a7384cf50359cd9783041

Block #11 has been added to the blockchain!
Hash: 298ba050170e8e9a0ffe5048b53c7543966ca38610d45ca8cd8e8266987b4b5e

Block #12 has been added to the blockchain!
Hash: 67716d6d2cbab68ea77e08d130dbefbf7ad61a352e97002ba759b1208f633cfc

Block #13 has been added to the blockchain!
Hash: 5825a74a67777913683834 28f8f6997d2af2b51dad9188cadc35c40a340d0941

Block #14 has been added to the blockchain!
Hash: dcc4a9b103117ce1e95ef4f1eb4569a2562d85b1af20b44853fb4851bd65b6a9

Block #15 has been added to the blockchain!
Hash: 963d86d669d39b6cd02e84b8f70da8549c7f110fccebfd43d0d870204cdfe90f
```

别担心，它将一直添加到 20 个区块

很好，我们的区块链可以工作了。如果你想要在主控台查看更多的信息，你可以编辑其完整的源代码并输出每个区块的时间戳或数据。

这就是 SnakeCoin 所具有的功能。要使 SnakeCoin 达到现今的产品级的区块链的高度，我们需要添加更多的功能，如服务器层，以在多台机器上跟踪链的改变，并通过工作量证明算法（POW）来限制给定时间周期内可以添加的区块数量。

如果你想了解更多技术细节，你可以在这里查看最初的比特币白皮书。

让这个极小区块链稍微变大些

这个极小的区块链及其简单，自然也相对容易完成。但是因其简单也带来了一些缺陷。首先，SnakeCoin 仅能运行在单一的一台机器上，所以它相距分布式甚远，更别提去中心化了。其次，区块添加到区块链中的速度同在主机上创建一个 Python 对象并添加到列表中一样快。在我们的这个简单的区块链中，这不是问题，但是如果我们想让 SnakeCoin 成为一个实际的加密货币，我们就需要控制在给定时间内能创建的区块（和币）的数量。

从现在开始，SnakeCoin 中的“数据”将是交易数据，每个区块的“数据”字段都将是一些交易信息的列表。接着我们来定义“交易”。每个“交易”是一个 JSON 对象，其记录了币的发送者、接收者和转移的 SnakeCoin 数量。注：交易信息是 JSON 格式，原因我很快就会说明。

```
{
  "from": "71238uqirbfh894-random-public-key-a-alkjdflakjfewn204ij",
  "to": "93j4ivnqiopvh43-random-public-key-b-qjrgvnoeirbnferinfo",
  "amount": 3
}
```

现在我们知道了交易信息看起来的样子了，我们需要一个办法来将其加到我们的区块链网络中的一台计算机（称之为节点）中。要做这个事情，我们会创建一个简单的 HTTP 服务器，以便每个用户都可以让我们的节点知道发生了新的交易。节点可以接受 POST 请求，请求数据为如上的交易信息。这就是为什么交易信息是 JSON 格式的：我们需要它们可以放在请求信息中传递给服务器。

```
$ pip install flask # 首先安装 Web 服务器框架
```

```
from flask import Flask

from flask import request

node = Flask(__name__)

# Store the transactions that

# this node has in a list

this_nodes_transactions = []

@node.route('/txion', methods=['POST'])

def transaction():

 if request.method == 'POST':

   # On each new POST request,

   # we extract the transaction data

   new_txion = request.get_json()

   # Then we add the transaction to our list

   this_nodes_transactions.append(new_txion)

   # Because the transaction was successfully

   # submitted, we log it to our console

   print "New transaction"

   print "FROM: {}".format(new_txion['from'])

   print "TO: {}".format(new_txion['to'])
```

```
        print "AMOUNT: {}\n".format(new_txion['amount'])

        # Then we let the client know it worked out

        return "Transaction submission successful\n"

    node.run()
```

真棒！现在我们有了一种保存用户彼此发送 SnakeCoin 的记录的方式。这就是为什么人们将区块链称之为公共的、分布式账本：所有的交易信息存储给所有人看，并被存储在该网络的每个节点上。

但是，有个问题：人们从哪里得到 SnakeCoin 呢？现在还没有办法得到，还没有一个称之为 SnakeCoin 这样的东西，因为我们还没有创建和分发任何一个币。要创建新的币，人们需要“挖”一个新的 SnakeCoin 区块。当他们成功地挖到了新区块，就会创建出一个新的 SnakeCoin ，并奖励给挖出该区块的人（矿工）。一旦挖矿的矿工将 SnakeCoin 发送给别人，这个币就流通起来了。

我们不想让挖新的 SnakeCoin 区块太容易，因为这将导致 SnakeCoin 太多了，其价值就变低了；同样，我们也不想让它变得太难，因为如果没有足够的币供每个人使用，它们对于我们来说就太昂贵了。为了控制挖新的 SnakeCoin 区块的难度，我们会实现一个工作量证明（Proof-of-Work）（PoW）算法。工作量证明基本上就是一个生成某个项目比较难，但是容易验证（其正确性）的算法。这个项目被称之为“证明” ，听起来就像是它证明了计算机执行了特定的工作量。

在 SnakeCoin 中，我们创建了一个简单的 PoW 算法。要创建一个新区块，矿工的计算机需要递增一个数字，当该数字能被 9 （ “SnakeCoin” 这个单词的字母数）整除时，这就是最后这个区块的证明数字，就会挖出一个新的 SnakeCoin 区块，而该矿工就会得到一个新的 SnakeCoin。

```
    # ...blockchain

    # ...Block class definition

    miner_address = "q3nf394hjg-random-miner-address-34nf3i4nflkn3oi"
```

```
def proof_of_work(last_proof):
  # Create a variable that we will use to find
  # our next proof of work
  incrementor = last_proof + 1
  # Keep incrementing the incrementor until
  # it's equal to a number divisible by 9
  # and the proof of work of the previous
  # block in the chain
  while not (incrementor % 9 == 0 and incrementor % last_proof == 0):
    incrementor += 1
  # Once that number is found,
  # we can return it as a proof
  # of our work
  return incrementor
@node.route('/mine', methods = ['GET'])
def mine():
  # Get the last proof of work
  last_block = blockchain[len(blockchain) - 1]
  last_proof = last_block.data['proof-of-work']
```

```
# Find the proof of work for
# the current block being mined
# Note: The program will hang here until a new
#       proof of work is found
proof = proof_of_work(last_proof)
# Once we find a valid proof of work,
# we know we can mine a block so
# we reward the miner by adding a transaction
this_nodes_transactions.append(
  { "from": "network", "to": miner_address, "amount": 1 }
)
# Now we can gather the data needed
# to create the new block
new_block_data = {
  "proof-of-work": proof,
  "transactions": list(this_nodes_transactions)
}
new_block_index = last_block.index + 1
new_block_timestamp = this_timestamp = date.datetime.now()
```

```
last_block_hash = last_block.hash

# Empty transaction list

this_nodes_transactions[:] = []

# Now create the

# new block!

mined_block = Block(

  new_block_index,

  new_block_timestamp,

  new_block_data,

  last_block_hash

)

blockchain.append(mined_block)

# Let the client know we mined a block

return json.dumps({

    "index": new_block_index,

    "timestamp": str(new_block_timestamp),

    "data": new_block_data,

    "hash": last_block_hash

}) + "\n"
```

现在，我们能控制特定的时间段内挖到的区块数量，并且我们给了网络中的人新的币，让他们彼此发送。但是如我们说的，我们只是在一台计算机上做的。如果区块链是去中心化的，我们怎样才能确保每个节点都有相同的链呢？要做到这一点，我们会使每个节点都广播其（保存的）链的版本，并允许它们接受其它节点的链。然后，每个节点会校验其它节点的链，以便网络中每个节点都能够达成最终的链的共识。这称之为共识算法（consensus algorithm）。

我们的共识算法很简单：如果一个节点的链与其它的节点的不同（例如有冲突），那么最长的链保留，更短的链会被删除。如果我们网络上的链没有了冲突，那么就可以继续了。

```
@node.route('/blocks', methods=['GET'])
def get_blocks():
  chain_to_send = blockchain
  # Convert our blocks into dictionaries
  # so we can send them as json objects later
  for block in chain_to_send:
    block_index = str(block.index)
    block_timestamp = str(block.timestamp)
    block_data = str(block.data)
    block_hash = block.hash
    block = {
      "index": block_index,
```

```
        "timestamp": block_timestamp,
        "data": block_data,
        "hash": block_hash
    }
    # Send our chain to whomever requested it
    chain_to_send = json.dumps(chain_to_send)
    return chain_to_send
def find_new_chains():
    # Get the blockchains of every
    # other node
    other_chains = []
    for node_url in peer_nodes:
        # Get their chains using a GET request
        block = requests.get(node_url + "/blocks").content
        # Convert the JSON object to a Python dictionary
        block = json.loads(block)
        # Add it to our list
        other_chains.append(block)
    return other_chains
```

```
def consensus():
  # Get the blocks from other nodes
  other_chains = find_new_chains()
  # If our chain isn't longest,
  # then we store the longest chain
  longest_chain = blockchain
  for chain in other_chains:
    if len(longest_chain) < len(chain):
      longest_chain = chain
  # If the longest chain wasn't ours,
  # then we set our chain to the longest
  blockchain = longest_chain
```

我们差不多就要完成了。在运行了完整的 SnakeCoin 服务器代码之后，在你的终端可以运行如下代码。（假设你已经安装了 cCUL）。

1、创建交易

```
curl "localhost:5000/txion" \
     -H "Content-Type: application/json" \
     -d '{"from": "akjflw", "to":"fjlakdj", "amount": 3}'
```

2、挖一个新区块

curl localhost:5000/mine

3、 查看结果。从客户端窗口，我们可以看到。

对代码做下美化处理，我们看到挖矿后我们得到的新区块的信息：


```
{
  "index": 2,
  "data": {
    "transactions": [
      {
        "to": "fjlakdj",
        "amount": 3,
        "from": "akjflw"
      },
      {
        "to": "q3nf394hjg-random-miner-address-34nf3i4nflkn3oi",
        "amount": 1,
        "from": "network"
      }
    ],
```

```
    "proof-of-work": 36
  },
  "hash": "151edd3ef6af2e7eb8272245cb8ea91b4ecfc3e60af22d8518ef0bba8b4a6b18",
  "timestamp": "2017-07-23 11:23:10.140996"
}
```


大功告成！现在 SnakeCoin 可以运行在多个机器上，从而创建了一个网络，而且真实的 SnakeCoin 也能被挖到了。

你可以根据你的喜好去修改 SnakeCoin 服务器代码，并问各种问题了。

网罗Python学习所有优质资料

以下Pytho学习文档视频资料大部分为数十家专业机构
内部教程，仅限自己及同学朋友阅读观看，请勿商业用途。

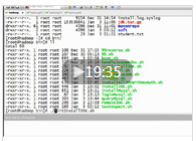
恭喜86马赫仔文库（www.86mhz.com）已经为超过30万IT学习者服务！

积累IT资料超过10000G+！

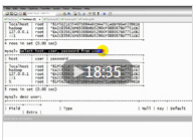
本站宗旨：我为人人，人人为我！

86马赫仔文库专注大数据等高端IT技术学习交流，分享各类资料教程，包括大数据，人工智能，Python，Java实战，Web前端从基础到实战，云计算实战，区块链入门等，所有资料教程全部免费下载！我们致力于帮助IT生态圈的工程师帮助别人，成就自己。大家也可以加群和众多志同道合的小伙伴们一起交流学习。

Python教程专区

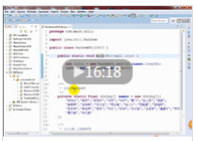


Python入门导学
167584人观看 免费下载



Python环境安装
190415人观看 免费下载

Python基本类型
204876人观看 免费下载



Python组的概念
193064人观看 免费下载

变量与运算符
176621人观看 免费下载

Python分支
186402人观看 免费下载

Python学习笔记专区

最新上传量：25943481 浏览量：1687765 学习交流群： 加入QQ群 加入QQ群 加入QQ群

Python网络爬虫快速入门实战	评分：9.36	下载量：305418	Python安装包语句	评分：9.20	下载量：230244
python中的&&及||	评分：8.97	下载量：265187	Python自然语言处理中文版	评分：8.89	下载量：270125
Python 也可以	评分：8.67	下载量：271540	Python 正则表达式操作指南	评分：8.45	下载量：299690
Python学习笔记	评分：9.15	下载量：293342	Tornado源码解析	评分：8.67	下载量：301514
Python知识总结	评分：9.73	下载量：356147	《简明 Python 教程》 中文版	评分：8.09	下载量：284840
[O'Reilly Media] Learning R	评分：8.77	下载量：245813	《Python语言入门》	评分：8.87	下载量：265341
Python3中文教程	评分：8.94	下载量：280114	Python数据分析基础教程：NumPy学习指南（第2版）	评分：8.24	下载量：270321
可爱的 Python	评分：8.44	下载量：274045	Python学习手册（Learning Python）第四版英文版	评分：8.38	下载量：310254
零基础学Python	评分：9.04	下载量：294811	《Python基础教程》 第二版	评分：8.68	下载量：302215
深入 Python 3	评分：8.66	下载量：277224	《Python核心编程》 第二版	评分：9.63	下载量：290142
Python的神奇方法指南	评分：9.22	下载量：342501	小猪的Python学习之旅	评分：9.30	下载量：273612
Django搭建简易博客教程	评分：8.92	下载量：310553	Python利器——各种工具包汇总	评分：8.81	下载量：300214
Django 1.8.2 文档中文版	评分：9.07	下载量：293645	Python绘制各大城市职工工资分布地图	评分：8.92	下载量：254873
Django 最佳实践	评分：9.14	下载量：301548	Python3的这些新特性很方便	评分：9.11	下载量：264542
The Django Book 中文版	评分：9.01	下载量：321548	利用Python读取网络数据文件	评分：8.70	下载量：278096
Python的神奇方法指南	评分：9.34	下载量：326255	纯Python实现人工智能	评分：8.94	下载量：293024
Web.py Cookbook 简体中文版	评分：9.11	下载量：273515	Python下切分文本的一些常用规则	评分：8.04	下载量：243011
Jinja2 文档中文版	评分：8.90	下载量：286631	10个基于python的的BBS论坛的源码	评分：8.95	下载量：274486

更多Python免费学习资料请访问：http://86mhz.com/python.html