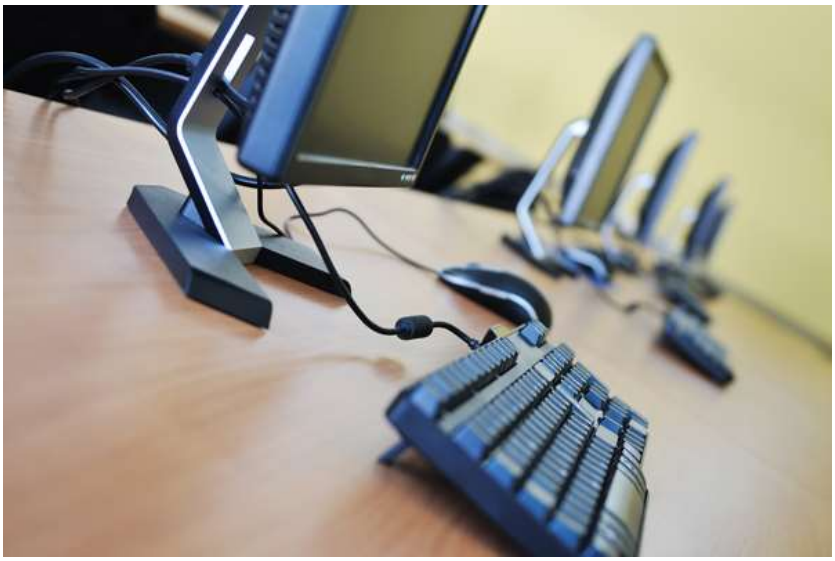
从零出发，用Python开发一个小型的区块链程序

本教程将向具有任何编程技能水平的Python开发人员介绍区块链。通过从零开始实现一个公有区块链并构建一个简单应用程序来利用它，您将了解区块链到底是多少。

作者：佚名 来源：今日头条 | 2018-06-15 11:08 收藏 分享



本教程将向具有任何编程技能水平的 Python 开发人员介绍区块链。通过从零开始实现一个公有区块链并构建一个简单应用程序来利用它，您将了解区块链到底是什么。

您将能够使用 Flask 微框架为区块链的不同功能创建端点，比如添加事务，然后在多个机器上运行脚本来创建一个去中心化网络。您还将了解如何构建一个简单的用户界面，以便与区块链进行交互，并存储任何用例的信息，比如对等支付、聊天或电子商务。

Python 是一种容易理解的编程语言，这是我在本教程中选择它的原因。通过学习本教程，您将实现一个公有区块链并了解它的实际应用。GitHub 上提供了一个完整的样本应用程序代码，该应用程序完全是用 Python 编写的。

背景

猜您喜欢 换一换

SAP HANA一体机性能哪家强？官方B4H测试...
佚名 3天前

数据中心管理需要具备哪五大技能？
圈圈 2019-03-07

联通信号变好后，到底移动和电信哪个更担忧？
通信人家园 2019-03-07

别光看无线路由器天线 WiFi信号只与这参数有...
陈赫 2019-03-07

运营商如何做才能让降费实实在在、消费者明...
张海龙 2019-03-07

都2019年了，还不会Docker？10分钟带你从...
Java高级互联网架构 2019-03-08

5G好消息不断：资费不高于4G 换手机不必换号
梁睿瑶 李佳 李碧雯 5天前

相比IPv4，IPv6绝不仅仅是地址长度的增加
企业网D1Net 3天前

第四范式 / 智能推荐

编辑推荐

头条 如何使用区块链技术管理网络？

关注 大规模采用区块链技术的巨大障碍

头条 五大行区块链布局提速 跨境支付等领域优势凸显

热点 区块链技术融合成云计算未来趋势

头条 区块链技术将如何对法律界产生重大影响

2008 年，一个名叫 Satoshi Nakamoto 的人（或者可能是一个小组）发表了一篇名为《比特币：一种对等电子现金系统》的白皮书。该文章结合了密码学技术和对等网络，不需要依靠中心化权威机构（比如银行）就能在人们之间实现付款。比特币应运而生。除了比特币之外，这篇文章还介绍了一种存储数据的分布式系统（即现在广为人知的“区块链”），该系统的适用范围远不只是付款或加密货币。

从那时起，几乎每个行业都对区块链产生了浓厚的兴趣。无论是像比特币这样的完全数字化的货币、像以太坊这样的分布式计算技术，还是像IBM Blockchain Platform 所基于的 Hyperledger Fabric 这样的开源框架，现在都以区块链作为其背后的基础技术。

“区块链”是什么？

区块链是一种存储数字数据的方式。数据可以是任何内容。对于比特币，它是事务（在帐户之间转移比特币），它甚至可以是文件；这都无关紧要。数据是以区块形式进行存储的，区块使用哈希值链接在一起。因此得名“区块链”。

区块链的神奇之处是在其中添加和存储此类数据的方式，该方式造就了一些非常理想的特征：

- 历史记录无法更改
- 系统无法攻破
- 数据的持久保存
- 没有单点故障

那么区块链如何能够实现这些特征呢？我们将通过实现一个区块链来深入剖析它。让我们开始吧。

关于该应用程序

首先定义一下我们将要构建的应用程序的用途。我们的目的是构建一个允许用户共享信息的简单网站。因为内容将存储在区块链中，所以它无法更改且会永远存在。

我们将采用自下而上的实现方式。首先定义我们将存储在区块链中的数据的结构。 一篇帖子（任何用户在我们的应用程序上发布的一条消息）将由 3 个基本要素来标识：

1. 内容
2. 作者
3. 时间戳

1.将事务存储到区块中

我们将采用一种广泛使用的格式来将数据存储在区块链中：JSON。以下是一篇存储在区块链中的帖子的格式：

2019年科技发展预测：云、大数据、AI、物...
看完此文再不懂区块链算我输，用Python从...
全球最牛的四个区块链项目都在这里！
大规模采用区块链技术的巨大障碍
区块链与数据库有什么区别?
到底什么是区块链技术中的“不可能三角”？
为什么自由职业者应该率先采用区块链？

```
{
    "author": "some_author_name",
    "content": "Some thoughts that author wants to share",
    "timestamp": "The time at which the content was created"
}
```

术语“数据”通常在互联网上被“事务”一词所取代。所以，为了避免混淆并保持一致，我们将使用术语“事务”来表示在我们的示例应用程序中发布的数据。

事务被打包到区块中。一个区块可以包含一个或许多个事务。包含事务的区块频繁地生成并添加到区块链中。因为可能有多个区块，所以每个区块都应有一个唯一 ID：

```
class Block:
    def __init__(self, index, transactions, timestamp):
        self.index = []
        self.transactions = transactions
        self.timestamp = timestamp
```

2.让区块不可更改

我们希望检测出对区块内存储的数据的任何篡改。在区块链中，这是使用一个哈希函数来实现的。

哈希函数接受任何大小的数据并生成固定大小的数据，该结果通常用于识别输入。下面是 Python 中的一个使用 sha256 哈希函数的示例：

```
>>> from hashlib import sha256
>>> data = "Some variable length data"
>>> sha256(data).hexdigest()
'b919fbbcae38e2bdaebb6c04ed4098e5c70563d2dc51e085f784c058ff208516'
>>> sha256(data).hexdigest() # no matter how many times you run it, the
 result is going to be the same 256 character string
'b919fbbcae38e2bdaebb6c04ed4098e5c70563d2dc51e085f784c058ff208516'
```

一个理想的哈希函数包括以下特征：

- 它应该很容易计算。
- 哪怕只更改数据中的一个位，哈希值也应该完全发生变化。
- 应该无法根据输出哈希值猜出输入。

您现在知道哈希函数是什么了吧。我们将每个区块的哈希值都存储在 Block 对象内的一个字段中，其作用类似于它所包含的数据的数字指纹：

```
from hashlib import sha256
import json

def compute_hash(block):
    """
    A function that creates the hash of the block.
    """
    block_string = json.dumps(self.__dict__, sort_keys=True)
    return sha256(block_string.encode()).hexdigest()
```

备注：在大多数加密货币中，甚至对区块中的各个事务也进行了哈希运算，从而形成一棵哈希树（也称为二进制哈希树），这棵树的根可以用作区块的哈希值。它不是区块链正常运作的必要条件，所以我们将省略它，以保持代码简洁。

3.链接区块

我们已设置了区块。区块链应该是一个区块集合。我们可以将所有区块都存储在 Python 列表中（等效于数组）。但这还不够，因为如果有人故意替换了集合中的一个区块该怎么办？用修改过的事务创建一个新的区块，计算哈希值，然后替换任何旧区块，这在我们的当前实现中并不是什么难事，因为我们会保持区块的不可更改性和顺序。

我们需要采用某种途径来确保对过去的区块的任何更改都会造成整个链的失效。一种方法是通过哈希值将区块链接起来。谈到链接，我们指的是将前一个区块的哈希值包含在当前区块中。所以，如果任何以前的区块的内容发生更改，该区块的哈希值也会发生更改，导致与下一个区块中的 previous_hash 字段不匹配。

每个区块都通过 previous_hash 字段链接到前一个区块，但是第一个区块该如何处理？第一个区块称为创始区块，大多数情况下，它是手动生成或通过某种独特逻辑生成的。让我们将 previous_hash 字段添加到 Block 类中，并实现我们的 Blockchain 类的初始结构（参见清单 1）。

清单 1. 我们的 Blockchain 类的初始结构

```
from hashlib import sha256
import json
import time
class Block:
    def __init__(self, index, transactions, timestamp, previous_hash):
        self.index = index
        self.transactions = transactions
        self.timestamp = timestamp
        self.previous_hash = previous_hash

    def compute_hash(self):
        block_string = json.dumps(self.__dict__, sort_keys=True)
        return sha256(block_string.encode()).hexdigest()
```

这是我们的 Blockchain 类：

```
class Blockchain:

    def __init__(self):
        self.unconfirmed_transactions = [] # data yet to get into blockchain
        self.chain = []
        self.create_genesis_block()

    def create_genesis_block(self):
        """
        A function to generate genesis block and appends it to
        the chain.The block has index 0, previous_hash as 0, and
        a valid hash.
        """
        genesis_block = Block(0, [], time.time(), "0")
        genesis_block.hash = genesis_block.compute_hash()
        self.chain.append(genesis_block)
```

4.实现工作量证明算法

但这里存在一个问题。如果我们更改前一个区块，我们可以非常轻松地重新计算所有后续区块的哈希值，并创建一个不同的有效区块链。为了预防这种情况，我们必须让计算哈希值的任务变得困难和随机化。

以下是我们实现此操作的方式。我们不会接受任何区块哈希值，而是对它添加某种约束。让我们来添加一种约束：哈希值应以两个前导零开始。另外我们知道，除非更改区块的内容，否则哈希值不会发生更改。

所以我们将在区块中引入一个称为随机数的新字段。随机数会不断变化，直到我们获得满足约束条件的哈希值。前导零的数量（在我们的例子中为值 2）决定了工作量证明算法的“难度”。您可能还注意到，我们的工作量证明很难计算，但在我们确定随机数后很容易验证（对于验证，您只需要再次运行哈希函数即可）：

```
class Blockchain:
    # difficulty of PoW algorithm
    difficulty = 2

    """
    Previous code contd..
    """

    def proof_of_work(self, block):
        """
        Function that tries different values of nonce to get a hash
        that satisfies our difficulty criteria.
        """
        block.nonce = 0

        computed_hash = block.compute_hash()
        while not computed_hash.startswith('0' * Blockchain.difficulty):
            block.nonce += 1
            computed_hash = block.compute_hash()

        return computed_hash
```

请注意，没有明确的逻辑来快速确定随机数；只能通过暴力破解。

5.将区块添加到链中

要将区块添加到链中，首先需要验证所提供的工作量证明是否正确，以及要添加的区块的previous_hash 字段是否指向链中的最新区块的哈希值。

让我们看看将区块添加到链中的代码：

```python
class Blockchain:
    """
    Previous code contd..
    """
    def add_block(self, block, proof):
        """
        A function that adds the block to the chain after verification.
        """
        previous_hash = self.last_block.hash

        if previous_hash != block.previous_hash:
            return False

        if not self.is_valid_proof(block, proof):
            return False

        block.hash = proof
        self.chain.append(block)
        return True

    def is_valid_proof(self, block, block_hash):
        """
        Check if block_hash is valid hash of block and satisfies
        the difficulty criteria.
        """
        return (block_hash.startswith('0' * Blockchain.difficulty) and
                block_hash == block.compute_hash())
```

挖矿

事务最初存储在一个未确认事务池中。将未确认事务放入区块中并计算工作量证明的过程被称为区块挖矿。一旦找到满足我们的约束条件的随机数，我们就可以说挖到了一个区块，这个区块就会放入区块链中。

在大多数加密货币（包括比特币）中，作为对耗费算力来计算工作量证明的奖励，矿工可以获得一些加密货币。以下是我们的挖矿函数的格式：

```python
class Blockchain:
    """
    Previous code contd...
    """

    def add_new_transaction(self, transaction):
            self.unconfirmed_transactions.append(transaction)

    def mine(self):
        """
        This function serves as an interface to add the pending
        transactions to the blockchain by adding them to the block
        and figuring out Proof of Work.
        """
        if not self.unconfirmed_transactions:
            return False

        last_block = self.last_block

        new_block = Block(index=last_block.index + 1,
                          transactions=self.unconfirmed_transactions,
                          timestamp=time.time(),
                          previous_hash=last_block.hash)

        proof = self.proof_of_work(new_block)
        self.add_block(new_block, proof)
        self.unconfirmed_transactions = []
        return new_block.index
```

OK,基本工作已经完成了。

6.创建接口

现在为我们的节点创建接口，以便与其他对等节点以及我们将要构建的应用程序进行交互。我们将使用 Flask 创建一个 REST-API 来与我们的节点进行交互。以下是它的代码：

```
from flask import Flask, request
import requests

app =  Flask(__name__)

# the node's copy of blockchain
blockchain = Blockchain()
```

我们的应用程序需要一个端点来提交新事务。我们的应用程序将使用此端点将新数据（帖子）添加到区块链中：

```
@app.route('/new_transaction', methods=['POST'])
def new_transaction():
    tx_data = request.get_json()
    required_fields = ["author", "content"]

    for field in required_fields:
        if not tx_data.get(field):
            return "Invlaid transaction data", 404

    tx_data["timestamp"] = time.time()

    blockchain.add_new_transaction(tx_data)

    return "Success", 201
```

下面是返回节点的链副本的端点。我们的应用程序将使用此端点来查询要显示的所有帖子：

```
@app.route('/chain', methods=['GET'])
def get_chain():
    chain_data = []
    for block in blockchain.chain:
        chain_data.append(block.__dict__)
    return json.dumps({"length": len(chain_data),
                       "chain": chain_data})
```

下面是请求节点挖掘未确认事务（如果有）的端点。我们将使用此端点从我们的应用程序自身发起一个挖矿命令：

```python
@app.route('/mine', methods=['GET'])
def mine_unconfirmed_transactions():
    result = blockchain.mine()
    if not result:
        return "No transactions to mine"
    return "Block #{} is mined.".format(result)


# endpoint to query unconfirmed transactions
@app.route('/pending_tx')
def get_pending_tx():
    return json.dumps(blockchain.unconfirmed_transactions)


app.run(debug=True, port=8000)
```

现在，您可以体验一下我们的区块链，创建一些事务，然后使用诸如 cURL 或 Postman 之类的工具来挖掘它们。

7.建立共识和去中心化

目前为止，我们实现的代码只能在单个计算机上运行。即使通过哈希值链接了区块，我们仍然不能信任单个实体。我们需要多个节点来维护我们的区块链。 所以让我们创建一个端点，以便让一个节点了解网络中的其他对等节点：

```python
# the address to other participating members of the network
peers = set()

# endpoint to add new peers to the network.
@app.route('/add_nodes', methods=['POST'])
def register_new_peers():
    nodes = request.get_json()
    if not nodes:
        return "Invalid data", 400
    for node in nodes:
        peers.add(node)

    return "Success", 201
```

您可能已经认识到，在多节点方面存在一个问题。由于故意操纵或意外的原因，一些节点的链副本可能有所不同。在这种情况下，我们需要商定采用链的某个版本，以维持整个系统的完整性。我们需要达成共识。

一种简单的共识算法可能是，在网络中的不同参与者构成的链出现分歧时，商定采用最长的有效链。选择此方法的理由是，最长的链是对已完成的最多工作量的有效估算：

```python
def consensus():
    """
    Our simple consensus algorithm.如果找到一个更长的有效链，则用它替换我们的链。
    """
    global blockchain

    longest_chain = None
    current_len = len(blockchain)

    for node in peers:
        response = requests.get('http://{}/chain'.format(node))
        length = response.json()['length']
        chain = response.json()['chain']
        if length > current_len and blockchain.check_chain_validity(chain):
            current_len = length
            longest_chain = chain

    if longest_chain:
        blockchain = longest_chain
        return True

    return False
```

最后，我们需要开发一种方法，让任何节点向网络宣布它已经挖到一个区块，以便每个人都能更新他们的区块链，并继续挖掘其他事务。其他节点可以轻松地验证工作量证明，并将它添加到各自的链中：

```python
# endpoint to add a block mined by someone else to the node's chain.
@app.route('/add_block', methods=['POST'])
def validate_and_add_block():
    block_data = request.get_json()
    block = Block(block_data["index"], block_data["transactions"],
                  block_data["timestamp", block_data["previous_hash"]])

    proof = block_data['hash']
    added = blockchain.add_block(block, proof)

    if not added:
        return "The block was discarded by the node", 400

    return "Block added to the chain", 201

    def announce_new_block(block):
    for peer in peers:
        url = "http://{}/add_block".format(peer)
        requests.post(url, data=json.dumps(block.__dict__, sort_keys=True))
```

announce_new_block 方法应在节点挖到每个区块后调用，以便对等节点能将该区块添加到自己的链中。

8.构建应用程序

好了，后端都设置好了。

现在，是时候创建应用程序的接口了。我们使用了 Jinja2 模板来呈现网页和一些 CSS，让页面看起来美观一些。

我们的应用程序需要连接到区块链网络中的某个节点，以便抓取数据和提交新数据。也可能存在多个节点：

```python
import datetime
import json

import requests
from flask import render_template, redirect, request

from app import app

.
CONNECTED_NODE_ADDRESS = "http://127.0.0.1:8000"

posts = []
fetch_posts 函数从节点的 /chain 端点获取数据，解析该数据并将它们存储在本地。
def fetch_posts():
    get_chain_address = "{}/chain".format(CONNECTED_NODE_ADDRESS)
    response = requests.get(get_chain_address)
    if response.status_code == 200:
        content = []
        chain = json.loads(response.content)
        for block in chain["chain"]:
            for tx in block["transactions"]:
                tx["index"] = block["index"]
                tx["hash"] = block["previous_hash"]
                content.append(tx)

        global posts
        posts = sorted(content, key=lambda k: k['timestamp'],
                       reverse=True)
```

该应用程序有一个 HTML 表单，用于接受用户输入，然后向一个已连接的节点发出一个 POST 请求，以便将该事务添加到未确认事务池中。然后，通过网络对该事务进行挖掘，最后在我们刷新网站后抓取该事务：

```python
@app.route('/submit', methods=['POST'])
def submit_textarea():
    """
    Endpoint to create a new transaction via our application.
    """
    post_content = request.form["content"]
    author = request.form["author"]

    post_object = {
        'author': author,
        'content': post_content,
    }

    # Submit a transaction
    new_tx_address = "{}/new_transaction".format(CONNECTED_NODE_ADDRESS)

    requests.post(new_tx_address,
                  json=post_object,
                  headers={'Content-type': 'application/json'})

    return redirect('/')
```

9.运行应用程序

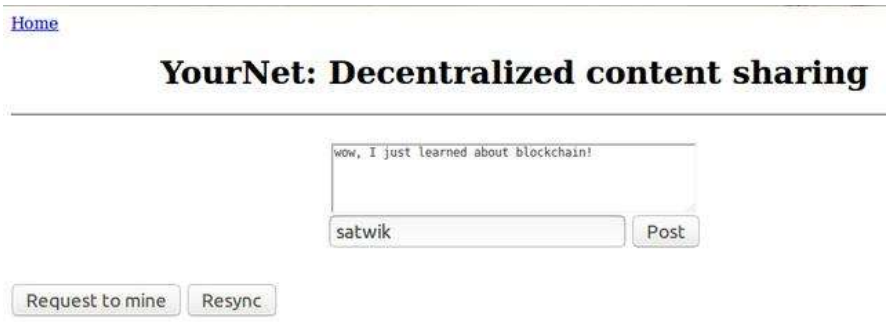
完工了！

要运行该应用程序：

1. 启动一个区块链节点服务器：
   python node_server.py
2. 运行该应用程序：
   python run_app.py

您应该能在 http://localhost:5000 上看到正在运行的应用程

尝试发布一些数据，您会看到类似下图的结果：



单击 **Request to mine** 按钮，您会看到类似下图的结果：

单击 **Resync** 按钮，您会看到应用程序与链重新同步：

验证事务

您可能注意到了应用程序中的一个缺陷：任何人都能更改任何名称和发布任何内容。解决此问题的一种方法是，使用公私密钥加密来创建帐户。每个新用户都需要一个公钥（类似于用户名）和私钥，才能在我们的应用程序中发布内容。这些密钥可以充当数字签名。公钥只能对使用相应私钥加密的内容进行解密。在将事务添加到任何区块之前，会使用作者的公钥对其进行验证。这样，我们就知道是谁写的这条消息。

结束

本教程介绍了公有区块链的基础知识。如果您一直在跟随操作，那么您已经从零开始实现了一个区块链，并构建了一个简单应用程序来允许用户在该区块链上共享信息。

原文链接：https://wwww.ibm.com/developerworks/cn/cloud/library/cl-develop-

1. 区块链入门教程第一期：区块链
2. 区块链入门教程第二期：挖矿？
3. 区块链如何更好地改进云计算解决方案
4. 区块链的信任输入、信任输出到底来自于哪里？
5. 区块链在金融领域的市场应用

【责任编辑：庞桂玉 TEL：（010）68476606】