



安装、配置和管理。

用户手册

1.0测试版

2020年7月。

MiniBlocklyChain是OpenQbit Inc.的注册商标，采用免费使用和商业许可。使用条款和条件见：www.OpenQbit.com

内容

1. 介绍 :	4
2. 什么是区块链方案中的公网或私网 ?	5
3. 什么是Blockly编程 ?	5
4. 什么是Termux ?	6
5. 什么是Mini BlocklyChain ?	6
6. 迷你BlocklyChain的流程架构	11
7. BlocklyChain功能图 (Mini BlocklyChain) 。	14
8. 什么是迷你BlocklyCode ?	15
9. 迷你BlocklyChain通信网络安装	16
10. 在系统节点 (手机) 分、秒同步。	38
11. Termux内的存储配置。	47
12. "Tor"网络安装和"Syncthing"安装。	48
13. 安装"Redis"数据库和SSH (安全壳) 服务器。	49
14. 在手机 (智能手机) 上配置SSH服务器。	50
15. Tor"网络配置与SSH (安全壳) 服务。	57
16. 点对点系统配置与手动同步。	60
17. Ambientes Blockly (App Inventor, AppyBuilder和Thunkable) 。	72
18. 什么是量子证明 (PQu) ?	74
19. 迷你BlocklyChain中区块的定义和用法	78
20. 在SQLite数据库中使用块 (MiniSQLite版本) 。	106
21. 担保块的定义和使用 ;	111
22. 在Mini BlocklyChain中设置安全参数。	122
23. 附件"创建KeyStore和PublicKeys数据库"。	126
24. 附件"RESTful SQLite GET/POST命令"。	141

25. 附件"Java代码SQLite-Redis连接器"。	148
26. 附件"开发者的迷你BlocklyChain"。	152
27. 附件"BlocklyCode智能合约"。	164
28. 附件"OpenQbit量子计算"。	171
29. 附件"SQLite数据库的扩展块"。	175
30. 附件"Mini BlocklyChain系统创建实例"。	175
31. 附件"与Ethereum和Bitcoin环境的整合"。	203
32. 软件的许可和使用；	222

1. 介绍：

区块链通常与比特币和其他加密货币相关联，但这些只是冰山一角，因为它不仅用于数字货币，而且可以用于任何可能对用户和/或公司有价值的信息。这项技术起源于1991年，当时Stuart Haber和W.Scott Stornetta描述了第一个关于密码学安全区块链的工作，直到2008年，随着比特币的到来，这项技术才被人们所关注。但目前其在其他商业应用中的使用需求，预计在中期的未来，在一些市场，如金融机构或物联网等行业中，其使用量会有所增长。

区块链这个词比较好理解，它是指分布在网络中多个节点（PC、智能手机、平板电脑等电子设备）上的单一的、约定俗成的记录。就加密货币而言，我们可以把它看作是记录每一笔交易的会计账本。

如果我们去了解它的内部实现细节，它的操作可能会很复杂，但基本思路很简单。

它被存储在每个块中。

1.- 有效记录或交易的数量；

2. 有关该区块的资料；

因此，**每个区块在链中都有特定的、不可移动的位置**，因为每个区块都包含前一个区块的哈希信息。整条链存储在组成区块链的每个网络节点上，所以**所有网络参与者身上都存储着链的精确副本**。

随着新记录的创建，它们首先会被网络节点检查和验证，然后被添加到一个新的区块中，该区块与链相连。

现在想象一下，这个网络的设备，彼此之间的通信，有能力进行交互，而不需要一个人的干预，即区块链的最大优势是它可以做出自主决定，这有利于在响应时间的

服务用户，一天 24 小时可用，最大限度地减少业务成本和首先有一个安全水平已经测试，这和其他原因已成为如此受欢迎的使用在不同的公共和私营部门。

2. 什么是区块链方案中的公网或私网？

公共网络。- 它是一个由计算机或移动设备组成的网络，相互之间进行通信并保持匿名性，不知道谁在这个网络中以正式的方式进行交易，在这种网络中，任何个人或公司都可以随时进行交互和连接，因为你不需要权限就可以连接，一个例子就是比特币的区块链，任何人都可以进入购买或出售。通常这类网络是面向或指向数字货币资产的买卖或其同义词"密码单据"，例。DogCoin、Ethereum、LiteCoin、BitCoin、Waves 等。

私人网络。- 它是由计算机或移动设备组成的网络，相互之间进行通信。但是，与公共网络不同的是，私有网络需要事先得到某些实体（公司或个人）的授权，才能连接并成为这种网络的一部分。通常情况下，私有区块链网络用于公司或企业进行各种不同类型的交易或操作，这些信息可以通过区块链应用和监督的文件、流程、授权和/或商业决策的形式具有有形价值，例如：金融部门、保险部门、政府等。

3. 什么是Blockly编程？

Blockly是一种**可视化的编程语言**，由一组简单的命令组成，我们可以将它们组合起来，就像拼图的碎片一样。对于那些想要以直观和简单的方式**学习如何编程的人**来说，或者对于那些已经知道如何编程并想要看到这种类型的编程潜力的人来说，这是一个非常有用的工具。

Blockly是一种不需要任何计算机语言背景的编程形式，这是因为它只是将图形块连接起来，就像我们玩乐高或拼图一样，你只需要具备一些逻辑性，就可以了！

任何人都可以为手机（智能手机）创建程序，不用再去搞那些难懂的编程语言，只需用图形化的方式把积木组合在一起，以一种简单、方便、快捷的方式来创建。

4. 什么是Termux？

Termux是一款安卓终端模拟器，也是一款Linux环境下的应用，不需要路由和配置就可以直接工作。自动安装一个最小的基础系统。

我们将使用Termux，因为它的稳定性和易于安装和管理，但是，你可以使用Ubuntu Linux for Android的安装环境。

在这个Linux环境下，你将拥有MiniBlocklyChain的通信过程的"核心"。

5. 什么是Mini BlocklyChain？

迷你BlocklyChain是一种全功能的区块链，是一种为手机开发的技术，其操作系统(Operating System)为Android，将成为发送和接收交易的节点。我们首创了区块链技术，通过Blockly编程，以"模块化"的方式架构，任何人只要具备最基本的知识，在不懂编程的情况下，都可以创建和开发手机程序，并以公网或私网模式创建自己的区块链。如果你想创建自己的数字货币，你可以这样做，或者在公司使用的解决方案，可能性是基于每个实际案例的需求和用户根据他的要求采用或创建的商业逻辑。

迷你BlocklyChain是第一个模块化的手机区块链，交易系统可以在短时间内实现。

在进入"模块"区块的定义和使用之前，我们需要掌握Mini BlocklyChain组件的基本概念，以便根据我们要实现的实际案例，知道何时、如何、在哪里应用它们。

下面的概念强调的是针对什么类型的用户，所以对于没有编程能力的人来说，完全理解开发用户的概念并不重要。

基本概念：

节点：- 每一台采用Android操作系统的移动设备（手机、平板电脑等），只要是Mini BlocklyChain公网或私网的一部分，都被命名为这个网络的节点。

哈什。- 它是与一组数据、一串字符、文件或某种数字信息相关联的数字签名，这种数字签名是唯一的、不可重复的，它有助于发送或接收信息，使发送的信息的初始内容不能被更改。

活跃。- 任何类型的数字数据或信息，都可以为人或公司赋予某种有形或无形的价值（文件、数字货币、音乐、视频、图像、数字授权等）。

交易。- 占有某种资产的节点之间的信息交流。

UXTO事务--这是一种将一个或几个节点的事务打包的事务类型，每个节点的所有未花费的事务（UXTO：*Unspent Transaction Outputs*）都可以作为新事务的输入进行花费。这些事务被归入一个队列中，每隔一定的时间进行处理，可以根据每个信息流的要求进行调节。

交易（输入）。- 它是一种基于UXTO交易的交易类型，当一个UXTO交易队列进入后，通过**输入交易**进行处理，将交易分类为存款或费用，从而能够对每个用户的最终结果有一个余额。

来源地址：- 这是在SQLite主队列中生成或发送要处理的事务的人的地址。它是一个64个字符的字母数字编号，由系统创建并验证。

目的地地址：-这是收到交易的人的地址，并将其添加到他的余额中或储存发送的资产。它是一个64个字符的字母数字编号，由系统创建并验证。

SQLite主。- API REST数据库，接收事务，并根据业务需要，每隔一定的变量或固定时间建立一个队列进行处理。

DataBase事务。- 是反映在SQLite数据库中保留或存储Mini BlocklyChain事务信息的事务

◦

高级加密标准- 它是一个安全过程，对存储在Mini BlocklyChain的SQLite数据库中的数据进行加密。

平衡。- 在迷你BlocklyChain中进行任何交易过程后，都会得到一个有形价值（加密货币）或无形价值（人或公司之间的业务流程）的操作余额。

业务流程。- 它是任何涉及某种信息流的过程，以获得一个结果给最终用户，最终用户可以是来自私营或公共部门的各种部门的个人或公司。

公钥和私钥。- 它是一种信息加密方式，即生成两个相互关联的字母数字密钥，用于通过公共或私人网络发送敏感信息。私钥是用来加密信息的，当通过网络发送信息时，它不能被篡改，也不能一目了然，公钥是共享的，是任何一个人或公司都能看到的，这个公钥将有助于验证发送的信息，也只有有效的收件人才能读取。

数字签名；- 这是一个可以用私钥创建的签名，通过这个签名，接收者可以确保信息没有被篡改，并确认信息对接收者有效。

数字地址（Mini BlocklyChain地址）。- 这是一个由每个Mini BlocklyChain用户独有的字母数字字符组成的地址，这个地址用于执行用户之间的交易，无论是公网还是私网。

魔法数字。- 这是一个由业务规则为每个运行中的UXTO交易定义的随机数，其作用是授权节点成为执行UXTO交易的候选者，并创建一个新的区块Mini BlocklyChain。

创建一个新的区块。- Mini BlocklyChain中的新区块是由出来作为中标候选人处理UXTO交易的节点创建并附加的哈希值。

PoW(工作证明)共识协议。- 这是一个执行所有节点的测试，第一个成功完成测试的节点就是选择执行UXTO事务的节点。它是一种由数学计算组成的算法，根据Mini BlocklyChain执行的每一笔交易中给出的规则获得一个结果(哈希)，是基于计算机信息处理资源的磨损或需求，在Mini BlocklyChain的情况下，已经被结构化和修改，以获

得一个"神奇的数字", 这是一个能够拥有大多数节点的授权或共识的数字, 是一个可以执行UXTO交易的数字。由于PoW只用于获得"魔数", 所以最高难度为5级。

PoQu共识协议(量子测试)- 它是一个运行所有节点的测试, 最初由PoW组成获得"魔数", 之后执行基于QRNG (量子随机数生成器) 一个量子随机数生成器的概率算法, 这个过程保证了所有节点的概率平等, 从而选择哪个节点来执行UXTO交易和新区块的创建。

Merkle树。- 这是一种安全方法, 以保证Mini BlocklyChain的交易是有效的或没有被任何外部实体修改。哈希梅克尔树是一种二进制或非二进制的数据树结构, 其中每个不是表的节点都被标记为其子节点的标签或值的连接的哈希值。它们是哈希列表和哈希字符串的泛化。

Redis DB。- 实时交易数据库用于处理并向节点发送已被请求的新交易。

SQLite DB。- 存储迷你BlocklyChain的余额和新区块的数据库, 以确保网络的完整性。

哨兵。- Redis和SQLite之间的安全和数据完整性连接器。这个连接器或程序负责审查、处理、验证、分发和翻译交易到节点。

OpenSSH。- 安全连接器, 在Android操作系统内执行任务。

Termux壳牌终端。- 在这个程序中, 你可以找到第三方的依赖关系来处理Mini BlocklyChain的交易, 在这个1.0版本中, 它是用来方便安装的, 但是在未来的版本中, 它将被目前正在开发的BlocklyShell系统所取代。

钱包。- 它是数字存储库, 在任何交易中, 有两个基本数据被保存在这里; 公钥和私钥将分别作为任何交易执行的源地址和数字签名, 以及发送或接收的交易余额。它是一个保存Mini BlocklyChain地址以及UXTO交易结果(余额)的仓库。

应用开发人员或程序员。

面向对象的程序设计(Java)- Mini BlocklyChain是用Java制作的。

BlocklyCode。- 是指在Mini BlocklyChain环境下可以执行的智能合约， BlocklyCode是用java编程创建的。

OpenJDK for Android。- 它是Android的JDK和JRE套件， BlocklyCodes就是在这里创建和执行的。

QRNG（量子随机数发生器）。- 它是一个量子随机数生成器， Mini BlocklyChain目前有两个API用于生成这些。

rsync。- 它是一个适用于Unix和微软Windows类型系统的免费应用程序， 它提供了增量数据的有效传输， 也可以对压缩和加密的数据进行操作。

ffsend。- 它是一个从命令行轻松安全地共享文件的应用程序。

NTP：- 网络时间协议， 是在网络中通过可变延迟的数据包路由来同步计算机系统时钟的互联网协议。

Termux（发展）；- 具有广泛的开源应用程序和库的Shell终端。

块状模块（数据）。- 开发者可以整合模块来增强Mini BlocklyChain的功能。

点对点。- 这个词指的是节点之间的直接通信， 也就是说， 信息更新并不依赖于中央服务器， 而是每个节点作为一个中央服务器，在拥有相同信息的所有节点之间进行通信， 这有助于避免故障点。

同步进行。- 用于在两个设备之间使用"点对点"通信类型同步数据或文件的工具。

红托。- 这是一个覆盖在互联网上的低延迟分布式通信网络， 用户之间交换信息的路由不会暴露其身份， 即IP地址（全网匿名）。

移动根。- 在手机上安装外部软件的过程中以系统管理员的身份进入，在Linux操作系统（Android）中管理员用户被称为"root"， 能够旋转你的手机就能进入任何进程。需要注意的是， 有些手机厂商（智能手机）表示， 由于手机的任何故障， 都会失去保修。

6. 迷你BlocklyChain的流程架构

Mini BlocklyChain由三个流程构成其架构，第一是业务流程，第二是通信流程，第三是补充模块和/或创建BlockyCode"智能合约"的开发环境。

业务流程是指在公网或私网中形成一系列例程来创建交易的块，这类流程是对业务的规划，将如何、何时、何事、何人、何地等更多属性进行排序、规划和分配，从而实现每笔交易发送、接收、存储和/或拒绝的主要功能。第一道工序由以下几类区块组成，分为四类。

1. 数据输入块
2. 数据处理模块
3. 安全区。
4. 模块化数据块（开发人员）
5. 通信机器人。

数据录入区块是指接收交易的区块，至少有四个输入参数（**源地址、目的地址、活动、数据**），根据变量"数据"的不同可以有更多的输入参数，这可以是第三方开发的区块，也可以是空值（NULL）。

数据流程块对每笔交易的物流、计算、数据规范化、逻辑决策和流程检查进行开发。这些类型的区块用于给业务流程提供智能，如其名称所示，主要基于处理所有类型的信息，以及从不同类型的数据中转换。

安全区块用于验证信息，确保交易从起源到目的地没有被改变，它们总是用区块链技术中使用的各种安全算法来处理，其中使用最多的工具有（哈希签名、数字签名、AES数据加密、数字地址的创建和验证等）。

模块化数据区块，这些区块是由第三方开发的，请记住，Mini BlocklyChain是以模块化的方式创建的，可以根据每个部门的需求，不管是公共的还是私人的，都可以通过新的区块来丰富。要了解更多关于如何创建模块的信息，请查看开发者附件。

正如我们之前评论的迷你BlocklyChain的架构在其第二个组件是通信的进程，这些进程是那些负责通过TCP和套接字的通信渠道，以提供发送和接收交易的灵活性，无论是通过不同的手段，目前使用最多的是。移动互联网、Wifi目前的工作包括通信媒介Bluetooth。

通信块基本上是基于数据的交换，并在传输通道中实现了安全，这是基于通过SSH（安全壳）协议的通信，发送或接收的交易要经过不同的阶段。

通信部分是最基本的，因为这些过程具有通过TCP和称为"点对点"的连接更新所有节点信息的功能，介入通信的块是基于节点之间的信息交换，没有中间服务器的干预，所以每个节点使其独立，能够创建一个节点网络，其中故障点最小或几乎为零。同样，每个节点的这种独立性也有助于他们根据业务的需要单独或整体做出决策。

"Peer to Peer"架构由三部分组成，构成了你决定创建的公共网络或私有网络，在这两种情况下，通信信道从节点到节点都是加密的。

移动设备(智能手机)或wifi之间通信的第一个组件是为节点或设备提供一个在世界任何地方都可以找到的网络，MiniBlocklyChain安装的通信网络是"**Tor**"网络。

什么是Tor网络？ - (<https://www.torproject.org>)

"**Tor**(西班牙语Te Onion Router的首字母缩写)是一个项目，其主要目标是开发一个叠加在互联网上的低延迟分布式通信网络，在这个网络中，用户之间交换的信息的路由不会暴露其身份，即其IP地址(网络一级的匿名性)，而且，它还能保持通过它传播的信息的完整性和保密性"。

第二个组成部分，也是同样重要的任务，就是让MiniBlocklyChain中的所有节点在任何时候都拥有相同的数据，或者让他们的数据库和文件同步，在节点之间执行这个任务将实现"**syncing**"。

什么是同步网络？ - (<https://syncing.net>)

Syncthing是一个免费的开源点对点文件同步应用程序，适用于Windows, Mac, Linux, Android, Solaris, Darwin和BSD。您可以在本地网络上的设备之间或通过互联网在远程设备之间同步文件。

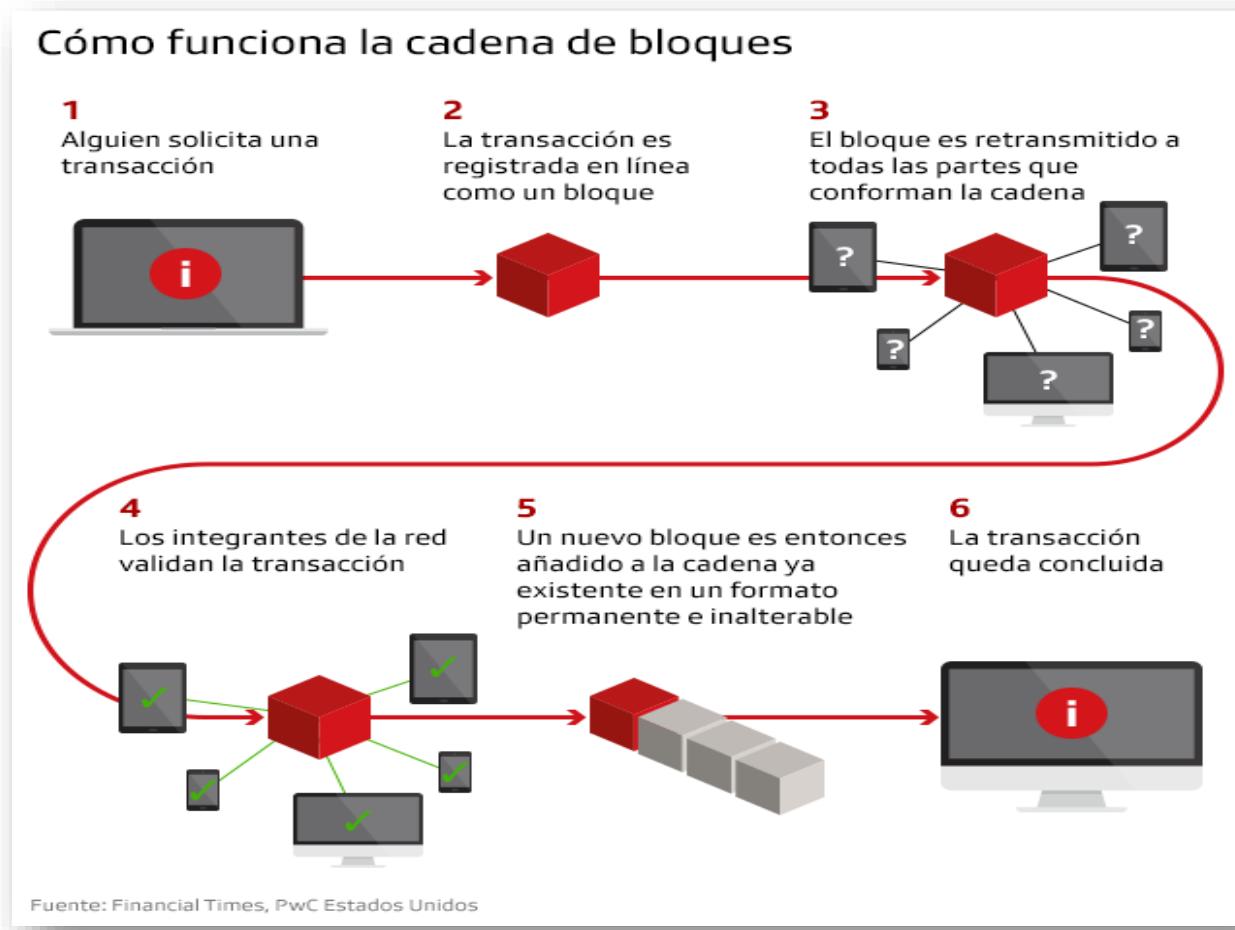
基于前面的两个通信组件，我们可以开始开发节点之间的信任网络，并与数据同步的水平，这将是业务流程的骨干或"核心"，以满足用户或公司的每一个需求。

通信网络中的第三个组件是Redis数据库，它将实时通知所有节点收到的新事务和发送的新事务。

什么是Redis DB网络？ - (<https://redis.io>)

Redis是一个内存数据库引擎，基于哈希表(key/value)的存储，但也可以选择作为持久性或持久性数据库使用。

7. BlocklyChain功能图（Mini BlocklyChain）。



8. 什么是迷你BlocklyCode？

迷你BlocklyCode是用Java语言创建的程序，存储在独立于节点的仓库中，它们通常被命名为智能合约，这些程序被设计成具有逻辑条件，当这些条件被满足时，它们会自动执行，而不需要依赖任何授权或外部人工干预。

"智能合同是一种计算机程序，它能促进、保障、执行和实施两方或多方（如个人或组织）之间的登记协议。因此，他们将协助他们谈判和确定这些协议，这些协议将导致在满足一些具体条件后发生某些行动。

智能合约是一个程序，它生活在一个不受任何一方或其代理人控制的系统中，它执行一个自动合约，它的工作原理就像其他计算机程序的if-then句子一样。与不同的是，它是以与真实资产互动的方式进行的。当触发一个不需要人为判断的预设条件时，智能合约就会执行相应的合约条款。

它们的目的是提供比传统合同法更大的保障，并降低与订立合同有关的交易成本。通过一个不需要信任的系统(如比特币)进行数字价值的转移，为可以利用智能合约的新应用打开了大门。“

Mini BlocklyCode是针对有java编程经验的开发者。在Mini BlocklyChain中，为了同一系统的安全，一些类型的库是被限制的，但是，可以创建交易的库来发送或接收两方或多方创造或约定的一些合同价值。

每条迷你BlocklyChain都符合以下前提。

- ✓ 任何迷你BlocklyChain的创建都是基于系统上只运行消息而不执行，然而，这些消息可能包含授权、物理合同批准或物理动作。
- ✓ 每一个Mini BlocklyChain的创建都要经过"审计师"通讯区块，这个区块负责监控恶意代码或违禁代码，对系统中的句子或未授权订单进行审核。
- ✓ 所有Mini BlocklyChain只在源节点的JVM上执行，程序不在全局执行，以保护系统。

更多细节请参见"Mini BlocklyChain for Developers"附录。

9. 迷你BlockyChain通信网络安装

1. 安装和配置Mini SQLSync网络

Mini SQLSync网络负责接收来自节点的事务，以便处理这些事务，这个网络是由一个主网络和一个名为Mini Sentinel RESTful的备份网络组成的，主网络是通过节点之间的"Peer to Peer"来实现的。

我们将从安装RESTful Mini Sentinel备份网络开始，这个服务是将接收来自节点的事务，这个服务是基于将事务存储在确定的时间内，之后通过连接器将信息转换，将所有加密的事务队列注入Redis服务。之后Redis数据库服务将实时执行加密交易队列的发布，将其发布到集成Mini BlocklyChain网络的所有节点。

RESTful类型的服务或设计是一种简单方便的方式，通过URL或互联网地址向数据库中查询、修改、删除和/或插入信息，在我们的案例中，我们将使用SQLite数据库，因为它的特点是所有信息都在一个文件中。对于需要大规模扩展的需求，我们可以使用其他类型的数据库，如MySQL、Oracle、DB2或其他商业数据库，只需将SQLite的Mini BlocklyChain连接器（免费版）换成Mini BlocklyChain DB others的连接器（商业版）。

重要提示：RESTful Mini Sentinel的配置在任何时候都不会处理任何类型的事务，它只是将"信息"格式化的服务，以便节点能够正确地、有计划地、同步地执行所有节点的事务处理。

RESTful服务的安装是基于SQLite数据库的，为了实用，这个安装将在Windows环境下完成，但是，这个模式可以很容易地复制到Linux类型的操作系统中。

如果想检讨SQLite查询和插入的性能和支持，请参考这些性能测试。

<https://www.sami-lehtinen.net/blog/sqlite3-performance-testing>

<https://sqlite.org/src4/doc/trunk/www/lsmperf.wiki>

RESTful Mini Sentinel备份网络安装。我们将在Windows 10电脑中安装和配置这个服务，不过，在Ubuntu 18.09 Linux环境下，这个过程也很容易安装。

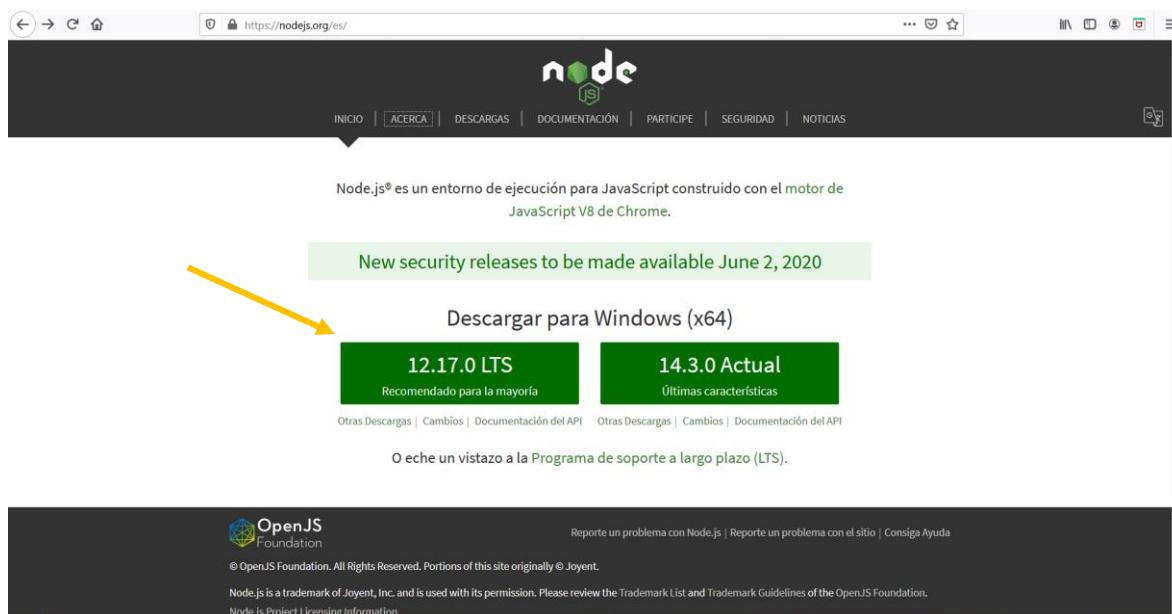
所需经费：

- ✓ RESTful模式的SQLite数据库服务器(<https://github.com/olsonpm/sqlite-to-rest>)。
- ✓ SQLite-Redis Sentinel连接器
- ✓ Redis数据库服务器在主从模式下 (<https://redis.io/topics/replication>)

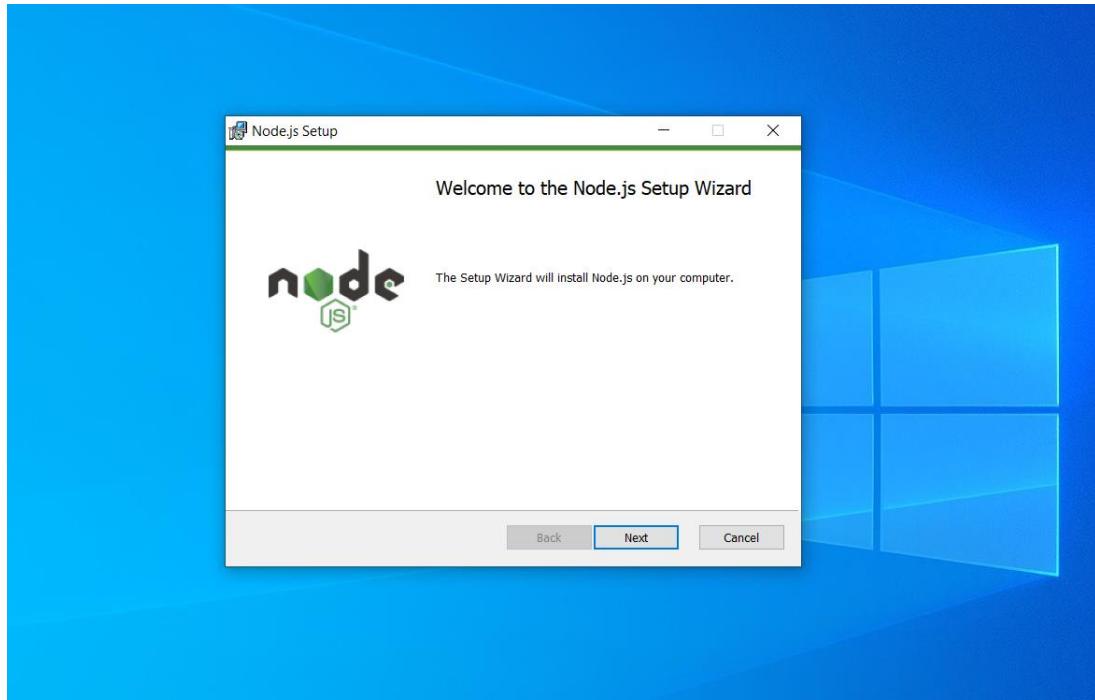
在RESTful模式下安装SQLite数据库服务器。

安装时，我们需要从以下软件包开始，Nodejs、sqlite3和Git Bash for Windows。

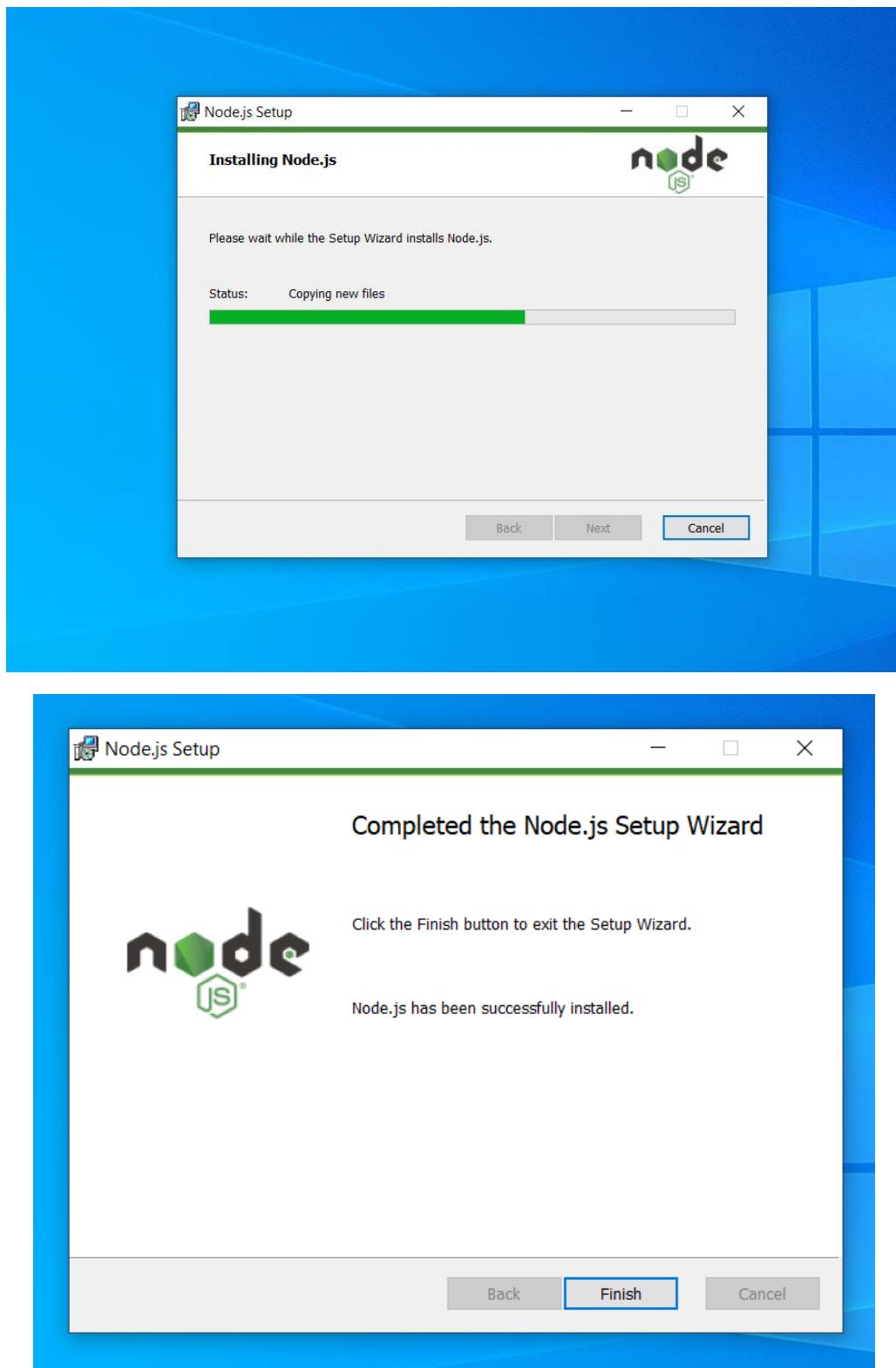
为了获得Nodejs，我们咨询了他们的官方网站<https://nodejs.org/es/>，选择了"推荐给大多数人"的版本。



下载到扩展名为.msi的文件后，我们双击安装。

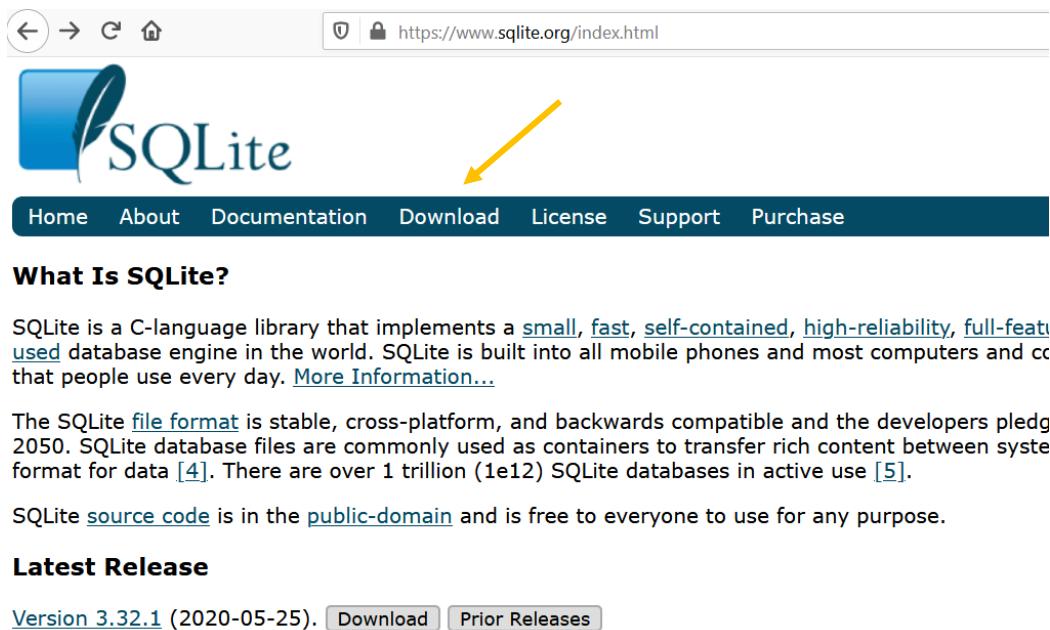


我们默认执行安装，只需点击"下一步"，无需选择您要求的任何额外选项。



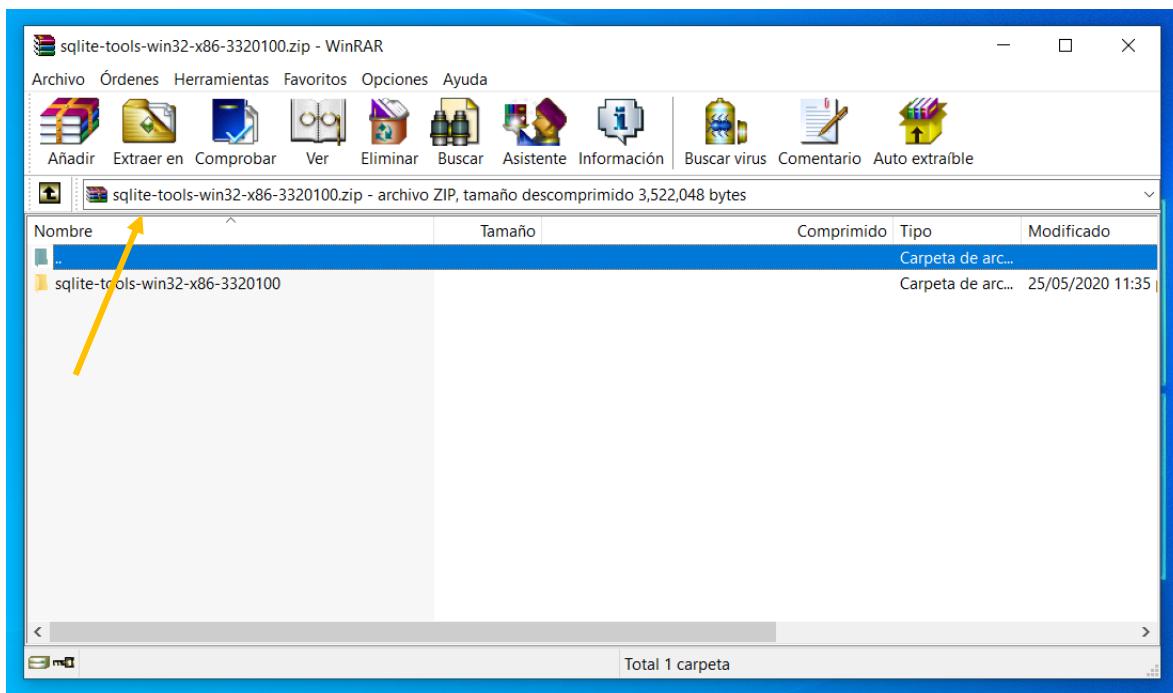
既然我们已经完成了Nodejs的安装， 我们继续安装SQLite数据库。

我们进入他们的官方网站, <https://www.sqlite.org/index.html>, 点击"下载"的部分。



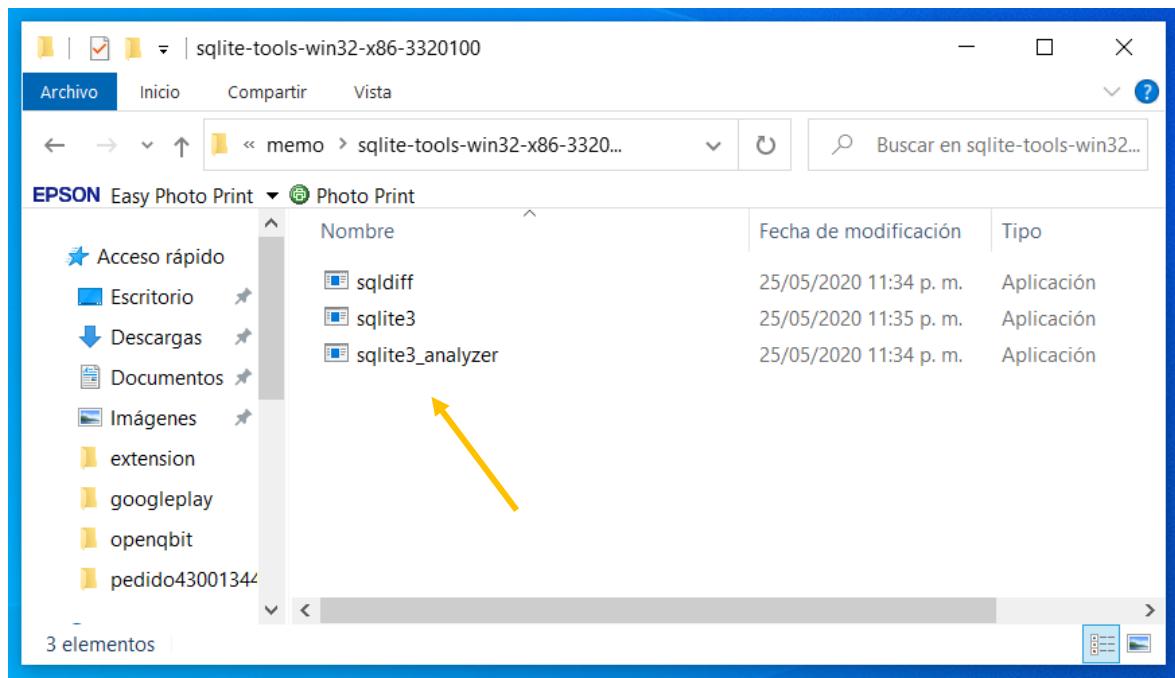
The screenshot shows the official SQLite website at <https://www.sqlite.org/index.html>. A yellow arrow points to the 'Download' link in the top navigation bar. The page content includes sections like 'What Is SQLite?' and 'Latest Release', along with download links for version 3.32.1.

接下来, 我们选择选项, 其中写着"Precompiled Binaries for Windows", 我们选择并点击软件版本"sqlite-tools-win32-x86-3320100.zip", 当我们完成下载到我们的计算机时,



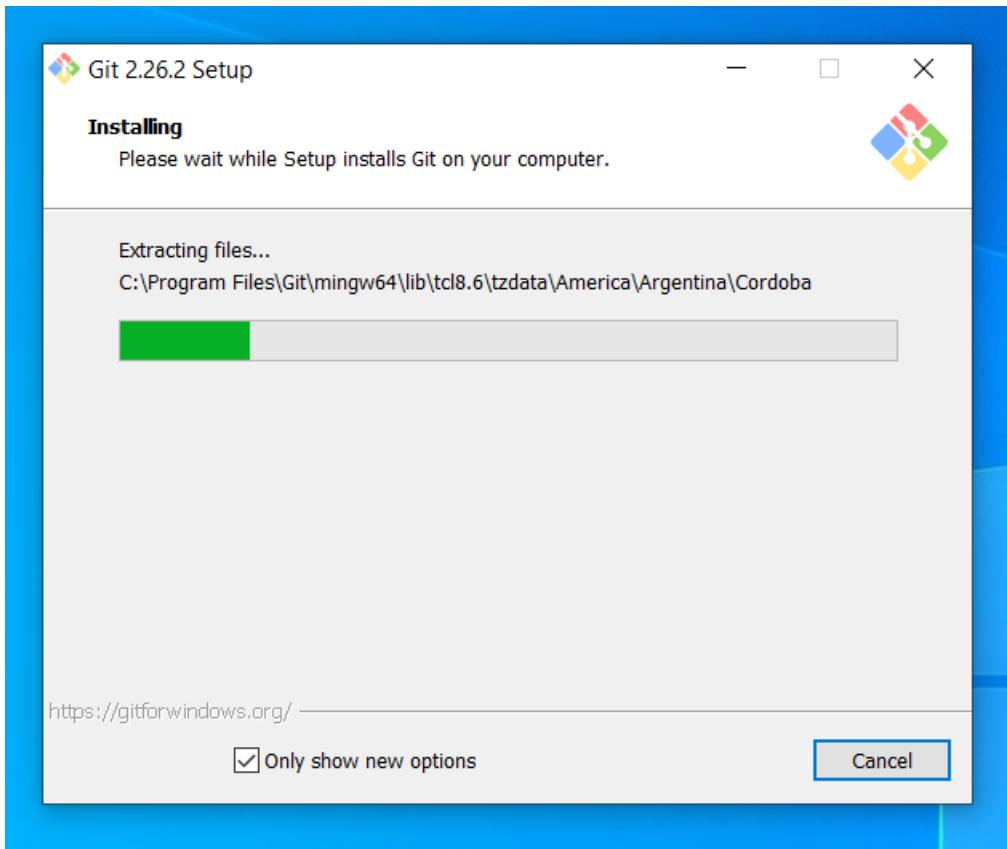
我们继续解压文件，这我们以一种简单的方式，我们打开文件下载的文件夹，我们给双击文件解压。

解压后我们会有三个文件，这三个文件是我们创建SQLite数据库的环境，我们以后会用到。



我们将从安装Git Bash开始，这是一个类似Linux的终端模拟器，它将帮助我们正确运行RESTful备份服务。

我们将从他们的官方网站<https://gitforwindows.org/> 下载，然后用它所标明的默认选项进行安装。



现在我们已经在Windows 10机器上安装了nodejs、sqlite和Git Bash，我们继续安装环境，以支持RESTful模式的SQLite数据库。

在下载软件的时候，将RESTful的功能赋予SQLite。为此我们必须进入以下网站，<https://github.com/olsonpm/sqlite-to-rest>，在这里我们将点击绿色的右键"克隆或下载"，并选择"下载ZIP"选项，这将下载软件在我们的计算机。

No description or website provided.

automatic-api

Branch: dev ▾ New pull request

Find file

Clone or download ▾

Clone with HTTPS

Use Git or checkout with SVN using the web URL.

<https://github.com/olsonpm/sqlite-to-rest>

Open in Desktop

Download ZIP

9 months ago

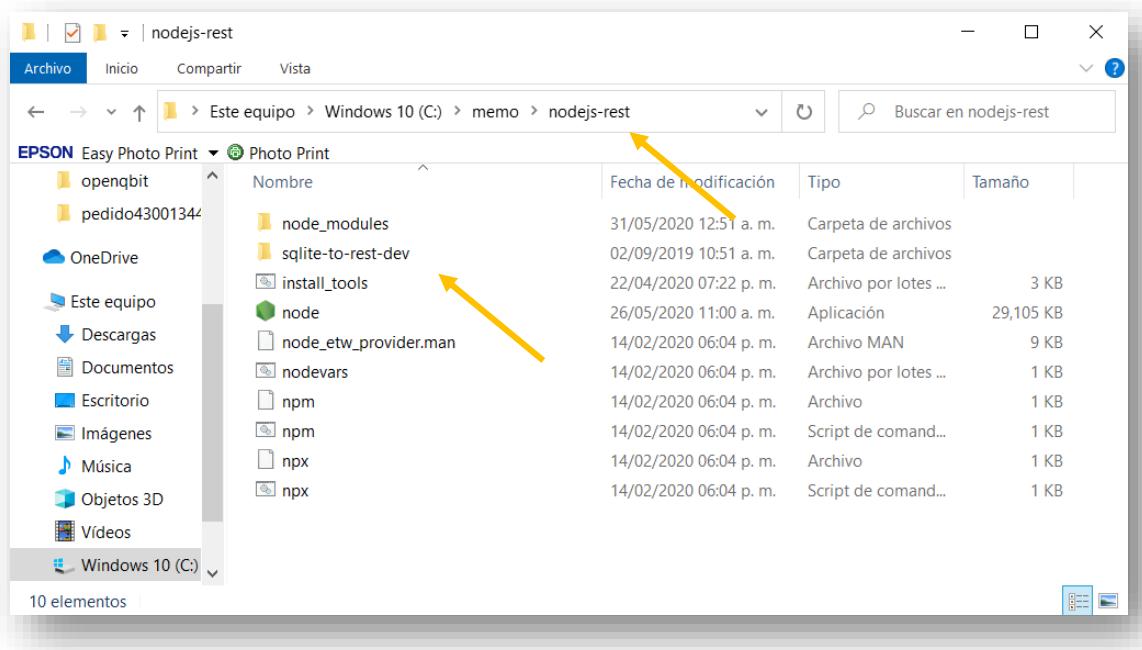
9 months ago

9 months ago

File	Description	Last Commit
bin	add prettier and eslint	9 months ago
cli/commands	add prettier and eslint	9 months ago
docs	add prettier and eslint	9 months ago
lib	add prettier and eslint	9 months ago
tests	add prettier and eslint	9 months ago
.gitignore	add prettier and eslint	9 months ago

下载到电脑后，我们继续将其解压到我们安装nodejs程序的目录或文件夹里面，我们将会有一个名为"sqlite-to-rest-dev"的目录。

注意：重要的是，sqlite-to-rest-dev目录必须在安装nodejs的目录内。



在安装好一切后，我们继续配置SQLite数据库，我们将使用它来存储RESTful备份环境中的节点的事务。

表的设计和数据结构。在CMD命令列表中在线执行的命令。

```
sqlite3 op.sqlite3 "CREATE TABLE trans (id integer primary key, addro, addrd, value);"
sqlite3 op.sqlite3 "CREATE TABLE sign (id integer primary key, trans_id references trans (id), sign, hash);"
```

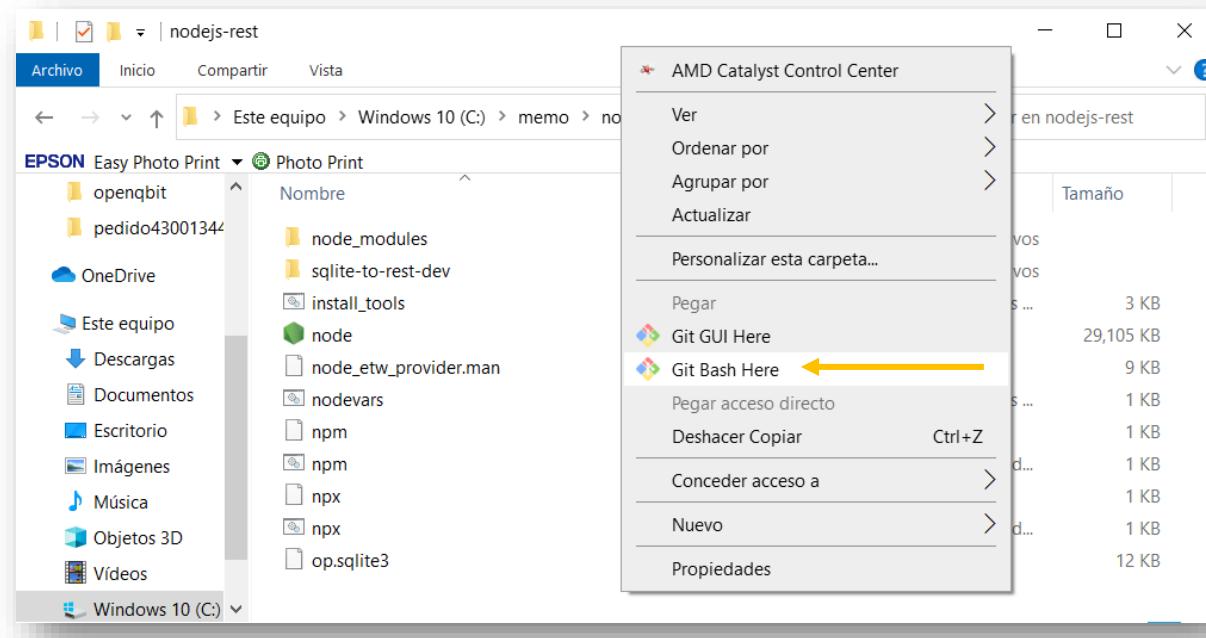
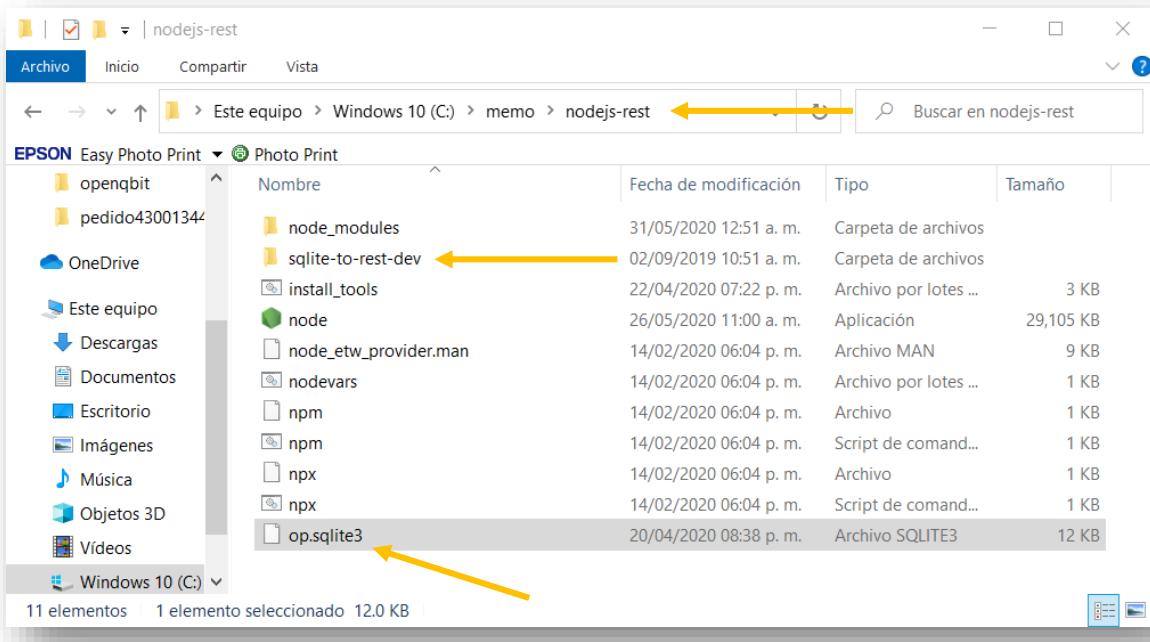
```
C:\memo\sqlite-tools-win32-x86-3320100>sqlite3 op.sqlite3 "CREATE TABLE trans(id integer primary key, addro, addrd, value);"
C:\memo\sqlite-tools-win32-x86-3320100>sqlite3 op.sqlite3 "CREATE TABLE sign(id integer primary key, trans_id references trans (id), sign, hash);"
C:\memo\sqlite-tools-win32-x86-3320100>dir
El volumen de la unidad C es Windows 10
El n m o de serie del volumen es: E019-5C05

Directorio de C:\memo\sqlite-tools-win32-x86-3320100

31/05/2020 02:29 a. m.    <DIR>        .
31/05/2020 02:29 a. m.    <DIR>        ..
31/05/2020 02:29 a. m.        12,288 op.sqlite3
25/05/2020 11:34 p. m.      516,608 sqldiff.exe
25/05/2020 11:35 p. m.      972,800 sqlite3.exe
25/05/2020 11:34 p. m.      2,032,640 sqlite3_analyzer.exe
                           4 archivos       3,534,336 bytes
                           2 dirs   137,272,078,336 bytes libres

C:\memo\sqlite-tools-win32-x86-3320100>
```

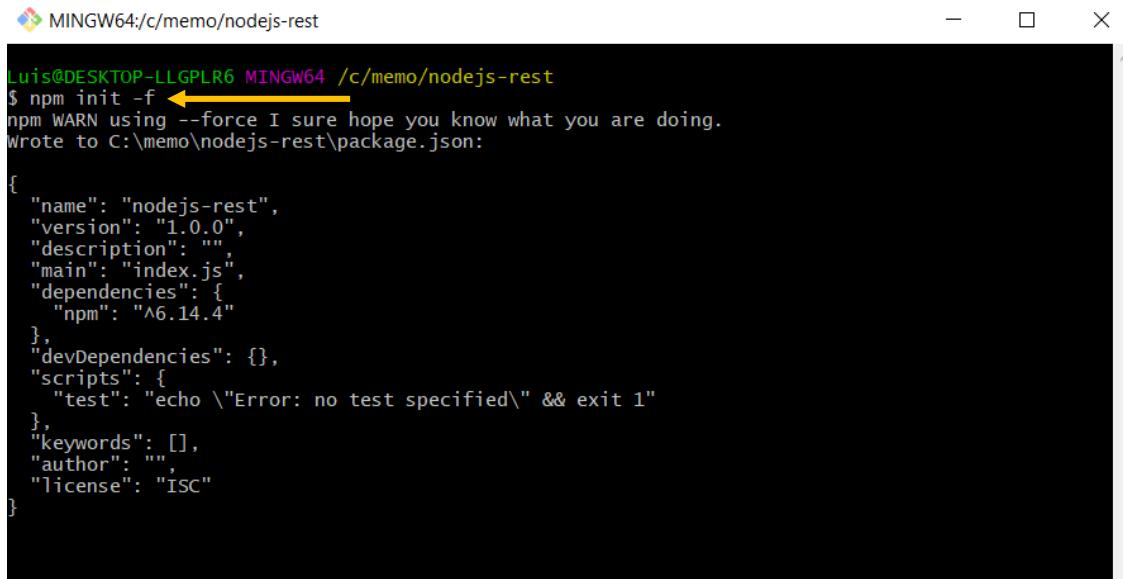
在创建数据库op.sqlite3后，我们必须在安装nodejs的目录下复制一份。在nodejs目录下还必须有"sqlite-to-rest-dev"的副本。我们把自己放在nodejs安装的文件夹里，



"sqlite-to-rest-dev"软件也应该在这里。用指针指向文件夹，右击显示菜单，并选择写着"Git Bash"的地方打开终端。

在新打开的Git Bash终端中，我们执行以下命令。

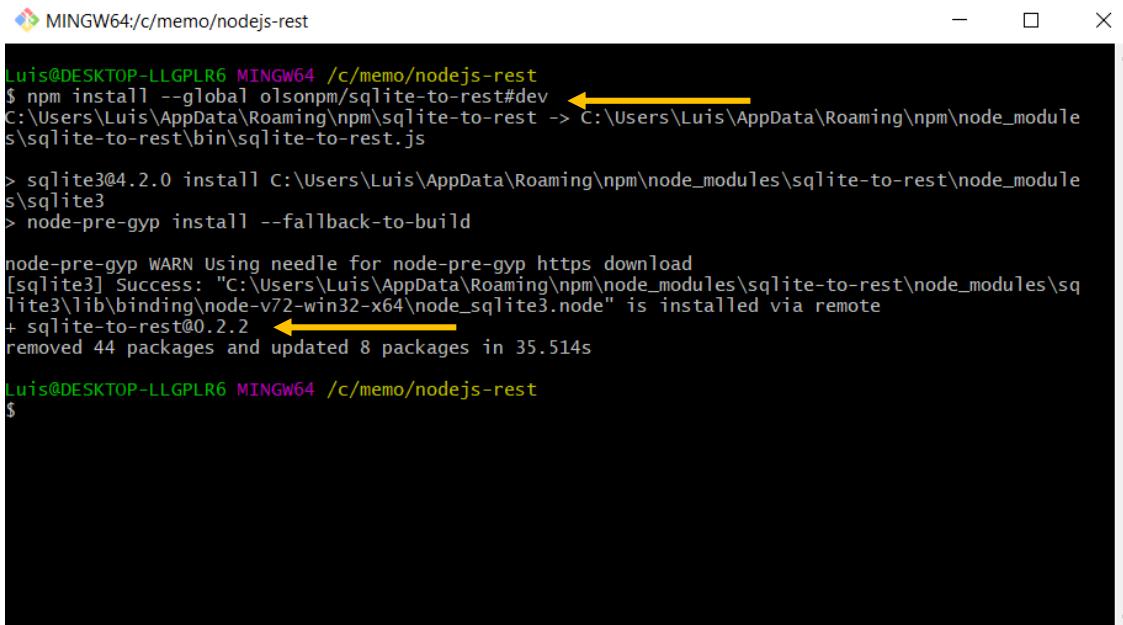
```
$ npm init -f
```



```
Luis@DESKTOP-LLGPLR6 MINGW64 /c/memo/nodejs-rest
$ npm init -f ←
npm WARN using --force I sure hope you know what you are doing.
Wrote to C:\memo\nodejs-rest\package.json:

{
  "name": "nodejs-rest",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "dependencies": {
    "npm": "^6.14.4"
  },
  "devDependencies": {},
  "scripts": {
    "test": "echo \\"Error: no test specified\\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC"
}
```

```
$ npm install --global olsonpm/sqlite-to-rest#dev。
```



```
Luis@DESKTOP-LLGPLR6 MINGW64 /c/memo/nodejs-rest
$ npm install --global olsonpm/sqlite-to-rest#dev ←
C:\Users\Luis\AppData\Roaming\npm\sqlite-to-rest -> C:\Users\Luis\AppData\Roaming\npm\node_modules\sqlite-to-rest\bin\sqlite-to-rest.js

> sqlite3@4.2.0 install C:\Users\Luis\AppData\Roaming\npm\node_modules\sqlite-to-rest\node_modules\sqlite3
> node-pre-gyp install --fallback-to-build

node-pre-gyp WARN Using needle for node-pre-gyp https download
[sqlite3] Success: "C:\Users\Luis\AppData\Roaming\npm\node_modules\sqlite-to-rest\node_modules\sqlite3\lib\binding\node-v72-win32-x64\node_sqlite3.node" is installed via remote
+ sqlite-to-rest@0.2.2 ←
removed 44 packages and updated 8 packages in 35.514s

Luis@DESKTOP-LLGPLR6 MINGW64 /c/memo/nodejs-rest
$
```

执行命令后，会出现"sqlite-to-rest"包的安装。

我们用之前创建的op.sqlite3基础为SQLite生成RESTful环境。

我们执行命令：`$ sqlite-to-rest generate-skeleton --db-path ./op.sqlite3`。



```
uis@DESKTOP-LLGPLR6 MINGW64 /c/memo/nodejs-rest
sqlite-to-rest generate-skeleton --db-path ./op.sqlite3
package.json found in working directory. ←
installing dependencies
writing the skeleton server to: skeleton.js
finished!
uis@DESKTOP-LLGPLR6 MINGW64 /c/memo/nodejs-rest
```

我们在默认端口8085上启动RESTful SQLite服务。我们运行以下命令：

`$ node skeleton.js`



```
uis@DESKTOP-LLGPLR6 MINGW64 /c/memo/nodejs-rest
node skeleton.js ←
listening on port: 8085
```

我们通过添加数据和查询数据来测试RESTful服务。

要测试SQLite RESTful服务，我们使用以下命令。

要在数据库内的表trans中插入数据op.sqlite3。

```
$ curl -s -H "Content-Type: application/json" -d '{"addr": "QWERTY1234",  
"addrd": "ASDFG4567", "value": "999"}' http://localhost:8085/trans。
```

要对跨表的所有数据进行查询。

The screenshot shows a terminal window titled 'MINGW64/c/memo/nodejs-rest'. It contains two command-line entries. The first entry is a POST request to insert a new row into the 'trans' table with id=6, addr='QWERTY1234', addrd='ASDFG4567', and value='999'. The second entry is a GET request to retrieve all data from the 'trans' table. Two yellow arrows point to the URL 'http://localhost:8085/trans' in both commands.

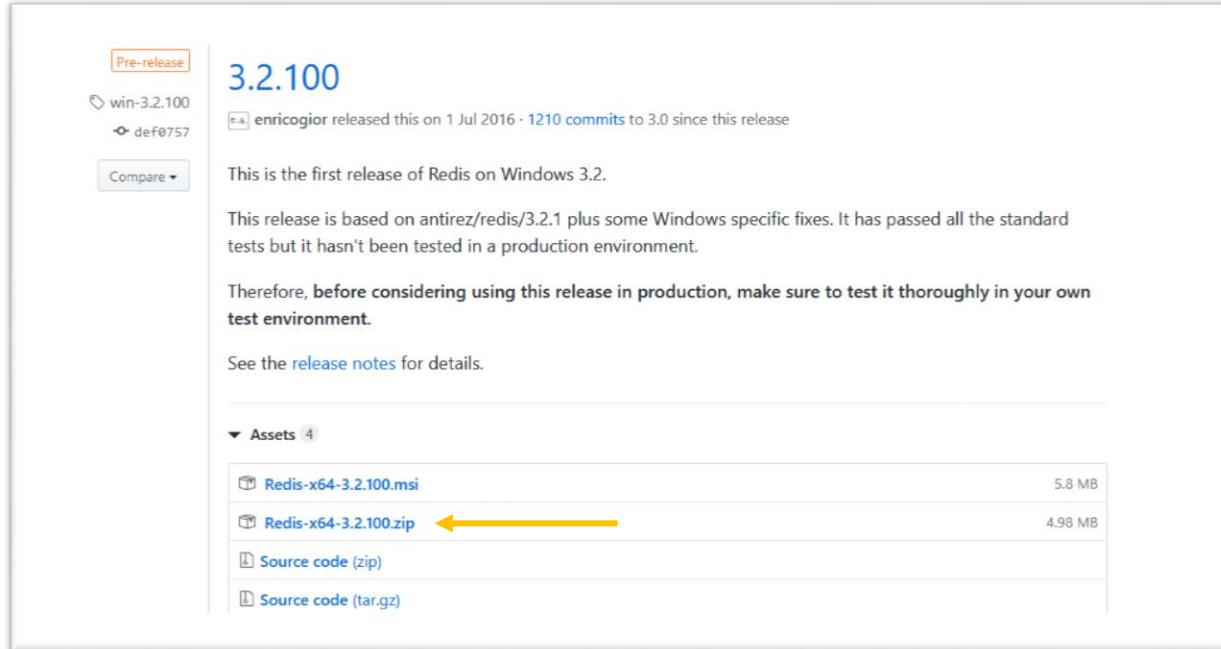
```
uis@DESKTOP-LLGPLR6 MINGW64 /c/memo/nodejs-rest  
curl -s -H "Content-Type: application/json" -d '{"addr": "QWERTY1234", "addrd": "ASDFG4567", "value": "999"}' http://localhost:8085/trans  
"id":6,"addr": "QWERTY1234", "addrd": "ASDFG4567", "value": "999"}  
uis@DESKTOP-LLGPLR6 MINGW64 /c/memo/nodejs-rest  
curl -s http://localhost:8085/trans  
  
"id":1,"addr": "CO", "addrd": "Boulder", "value": "Avery"}  
"id":2,"addr": "WI", "addrd": "New Glarus", "value": "New Glarus"}  
"id":3,"addr": "WI", "addrd": "Madison", "value": "One Barrel"}  
"id":4,"addr": "WI", "addrd": "Madison", "value": "One Barrel"}  
"id":5,"addr": "WI", "addrd": "Madison", "value": "One Barrel"}  
"id":6,"addr": "QWERTY1234", "addrd": "ASDFG4567", "value": "999"}
```

```
$ curl -s -H 'range: rows=0-2' http://localhost:8085/trans。
```

要查看如何详细使用所有RESTful启用的服务（查询、插入、更新和/或删除数据），请查看附录"Restful SQLite GET/POST命令"。

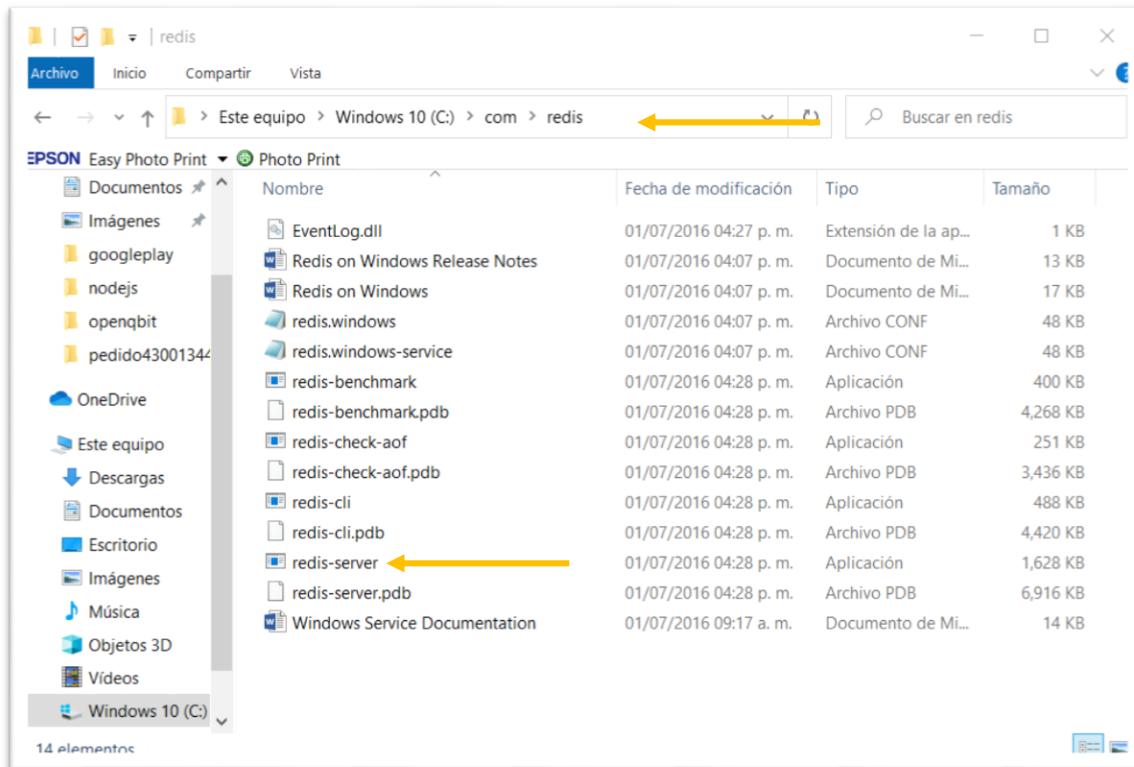
注意：在之前的安装中，所有的查询都是在URL中以"localhost"的地址进行的，但当服务器以公有IP（互联网）或私有IP（Wifi）暴露时，它将没有任何问题。我们在做SQLite RESTful服务和构成Mini BlocklyChain的通信网络节点之间的通信测试时，会进行测试。

Redis数据库安装为备份网络服务，要下载Windows的redis软件我们要到网站，<https://github.com/microsoftarchive/redis/releases/tag/win-3.2.100>，选择ZIP包。

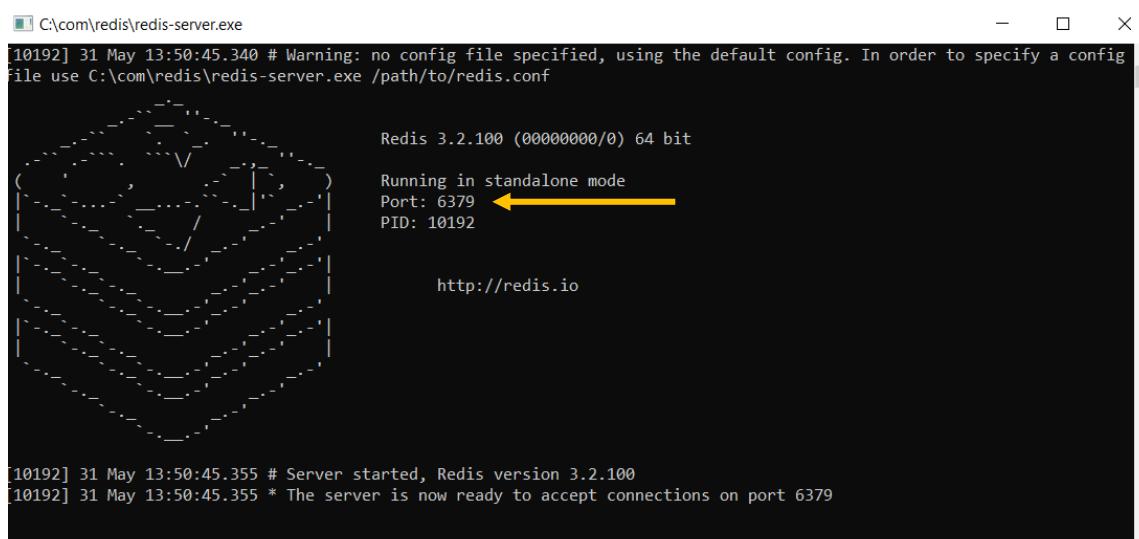


要找到安装的位置，我们将在Windows内创建一个名为"redis"的目录，并下载一个ZIP扩展名的文件，我们将其解压在之前创建的目录中，就已经完成了redis的安装。

我们通过双击"redis-server"命令运行服务器来测试安装。



执行"redis-server"命令后，我们会看到服务器在Windows CMD命令终端中运行， 默认端口为6379。



在这次测试中，我们有一个Windows 10的Redis 3.2.100版本，在Linux操作系统的情况下，我们有6.0.4版本（2020年5月发布），然而对于我们的Mini BlocklyChain配置来说，Windows版本有足够的功能特点，我们需要。

停止执行我们进行组合键Ctrl+C. 我们继续配置Windows 10的Redis 3.2.100数据库，Redis和Mini SQLSync通信网络节点之间的配置，类型为**Master(主站)-Slave(从站)**分布如下。

Master是我们配置的Windows 10的Redis服务器，这将实时复制数据，并将相同的信息同步到所有节点（手机）。这些节点将在Slave模式下安装一个Redis服务器。

这时我们开始看到我们安装和配置的备份网络是如何工作的。这种配置将服务于我们与所有节点的实时通信，它将帮助我们在有新的事务队列需要节点处理时，向网络的所有节点进行通信，在平等的信息传递给所有节点，使任何节点都有相同的概率能够处理"事务队列"信息。

我们开始配置SQLite-Redis Sentinel连接器。

这个连接器是一个用Java语言开发的程序，正如它的名字一样，它连接了SQLite和Redis (**Master**) 数据库。

连接器的功能是将信息（事务）从SQLite传输到Redis (**Master**)，并将"事务队列"发送到节点 (**Slaves**)。

关于SQLite-Redis Sentinel连接器的Java代码的更多细节, 请参见附件"SQLite-Redis Java代码连接器"。

配置Windows 10的Redis (主) 数据库redis.conf文件。

在文件中添加以下更改或指令, 保存更改并启动Redis服务器。

先找到tcp-keepalive设置, 按照评论中的建议设置为60秒。这将有助于Redis检测网络或服务问题。

tcp-keepalive 60

找到requirepass指令, 用强密码进行配置。虽然你的Redis流量必须受到第三方的保护, 但这提供了对Redis的认证。由于Redis的速度很快, 而且不会对边界的口令尝试进行评级, 所以要选择一个强大的、复杂的口令来防止蛮力尝试。

requirepass type_your_network_master_password。

例如 :

requirepass FPqwedsLMdf76ass7asddfd2g45vBN8ty99

最后, 根据你的使用场景, 你可能需要调整一些可选的设置。

如果你不希望Redis填满后自动放入最老和最少使用的密钥, 你可以禁用自动删除密钥。

记忆力-政策性免责声明

为了增强耐用性保证, 你可以启用附加的文件持久性。这将有助于最大限度地减少系统故障时的数据损失, 但代价是文件较大, 性能稍慢。

是

附录文件名 "redis-staging-ao.aof"

继续保存更改并重新启动Windows 10的Redis服务，用Ctrl + C键停止，再次运行Windows CMD命令行。

C:\Redis_redis_directory redis_server

在我们的例子中，我们可以看到我们有一个（Slave）节点连接。

```
:\\com\\redis> redis-server redis.conf
1:C 31 May 2020 23:44:56.633 # o000o000o000 Redis is starting o000o000o000
1:C 31 May 2020 23:44:56.634 # Redis version=5.0.7, bits=64, commit=00000000, modified=0,
id=51, just started
1:C 31 May 2020 23:44:56.634 # Configuration loaded
1:M 31 May 2020 23:44:56.635 * Increased maximum number of open files to 10032 (it was originally set to 1024).

Redis 3.2.100 (00000000/0) 64 bit
Running in standalone mode
Port: 6379
PID: 51

http://redis.io

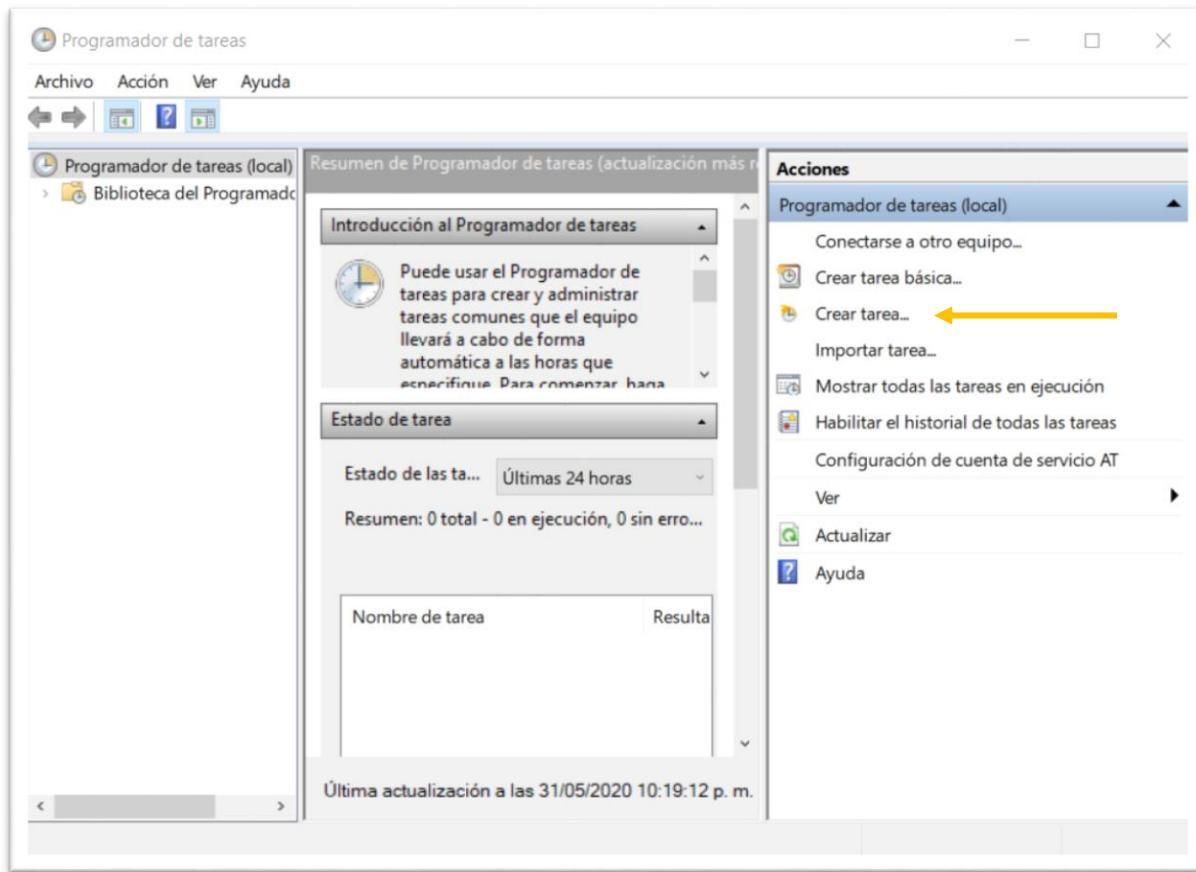
1:M 31 May 2020 23:44:56.648 # WARNING: The TCP backlog setting of 511 cannot be enforced because /proc/sys/net/core/somaxconn is set to the lower value of 128.51:May 2020 23:44:56.648 # Server initialized
1:M 31 May 2020 23:44:56.649 # WARNING overcommit_memory is set to 0! Background save may fail under low memory condition. To fix this issue add 'vm.overcommit_memory = 1' to /etc/sysctl.conf and then reboot or run the command 'sysctl vm.overcommit_memory=1' for this to take effect.
1:M 31 May 2020 23:44:56.669 * DB loaded from append only file: 0.020 seconds
1:M 31 May 2020 23:44:56.669 * Ready to accept connections
1:M 31 May 2020 23:49:57.013 * 10 changes in 300 seconds. Saving...
1:M 31 May 2020 23:49:57.031 * Background saving started by pid 82
2:C 31 May 2020 23:49:57.052 * DB saved on disk
1:M 31 May 2020 23:49:57.133 * Background saving terminated with success
1:M 31 May 2020 23:50:24.600 * Replica 192.168.1.68:6379 asks for synchronization
1:M 31 May 2020 23:50:24.602 * Full resync requested by replica 192.168.1.68:6379
1:M 31 May 2020 23:50:24.602 * Starting BGSAVE for SYNC with target: disk
1:M 31 May 2020 23:50:24.619 * Background saving started by pid 83
3:C 31 May 2020 23:50:24.642 * DB saved on disk
1:M 31 May 2020 23:50:24.670 * Background saving terminated with success
1:M 31 May 2020 23:50:24.689 * Synchronization with replica 192.168.1.68:6379 succeeded
```

我们可以看到，我们已经同步了一个IP为192.168.1.68的节点，默认端口为6379。

现在我们需要在Window 10操作系统中安排自动执行SQLite-Redis Sentinel连接器的任务。我们用Windows 10工具是我们从左下角输入"任务调度"来执行。



我们将创建一个新的任务"创建任务", 在这里, 我们将包括连接器的执行。

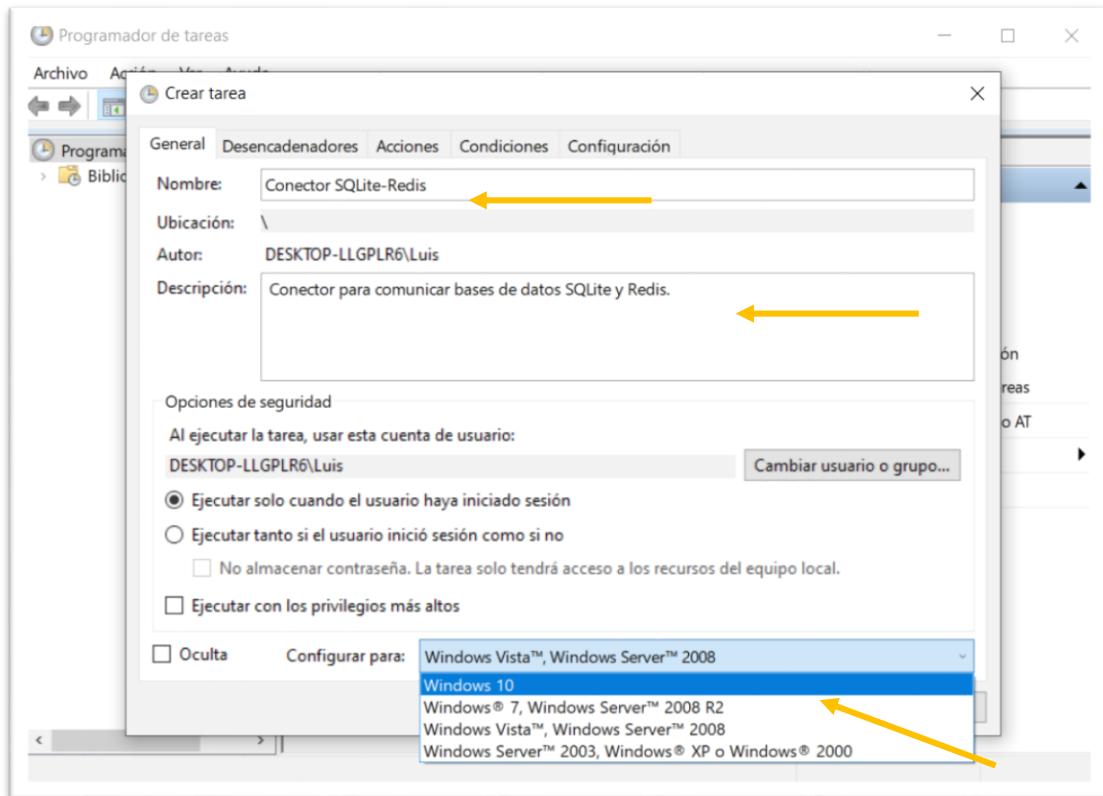


点击"创建任务"将打开一个额外的窗口, 我们必须在"常规"选项卡中给出以下参数。

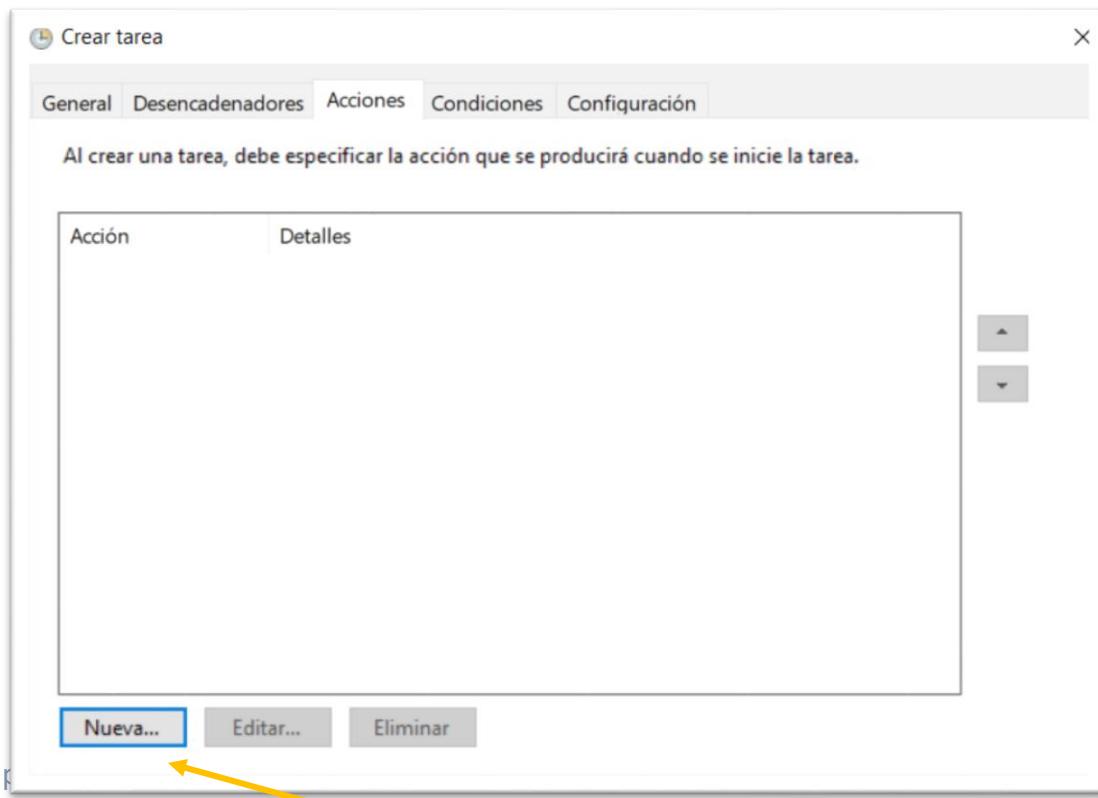
名称: task_name

说明 : 可选

配置为 : 选择 _操作系统_windows_ 版本。



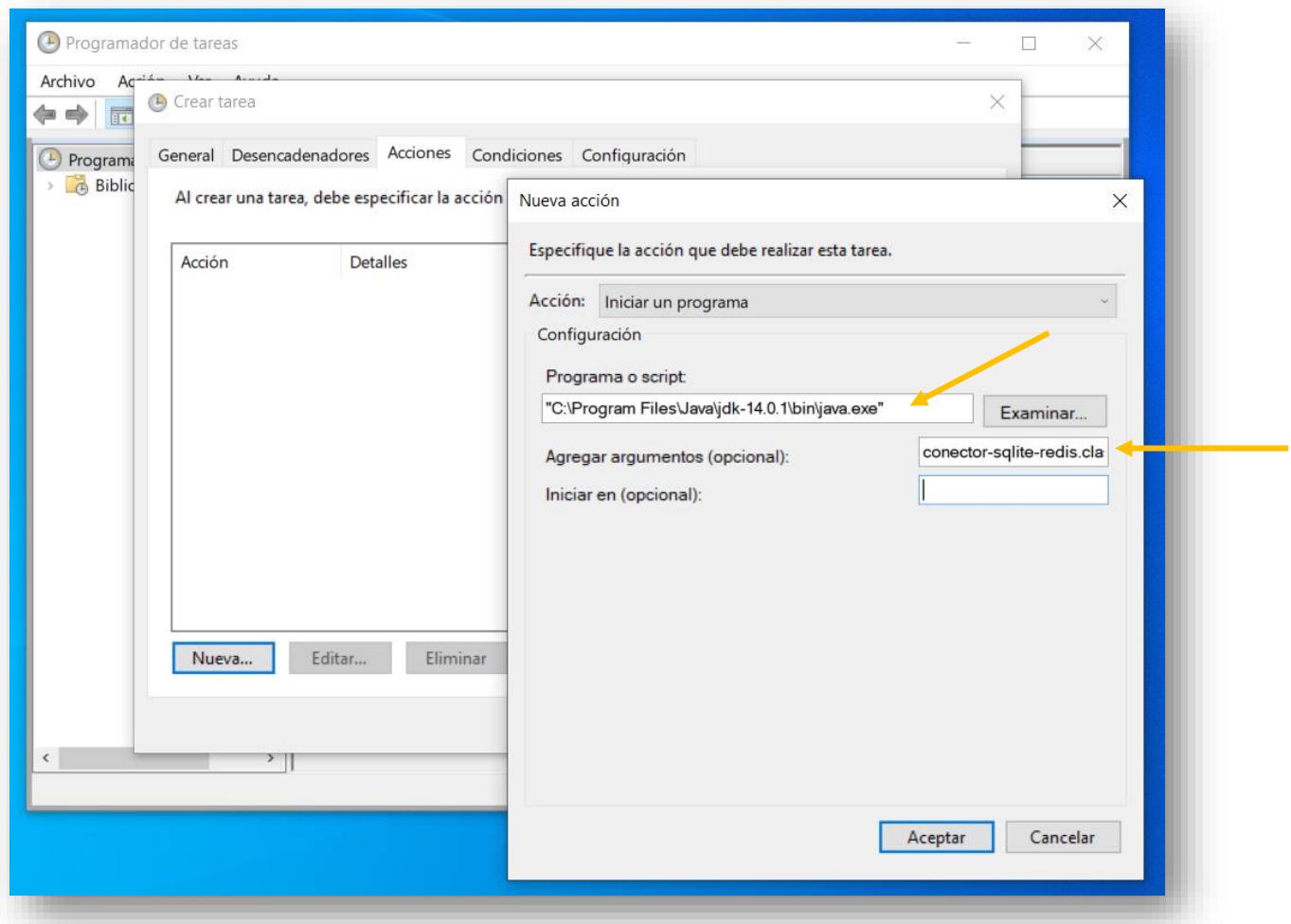
点击"行动"选项卡，再点击"新建"按钮进行更改。



我们给出以下参数：

程序或脚本: connector_path

添加参数：连接器名称



上述参数可能会根据连接器的位置而有所不同。主要思路是像平时在命令行中一样执行**连接器-sqlite-redis-v1.class**程序。

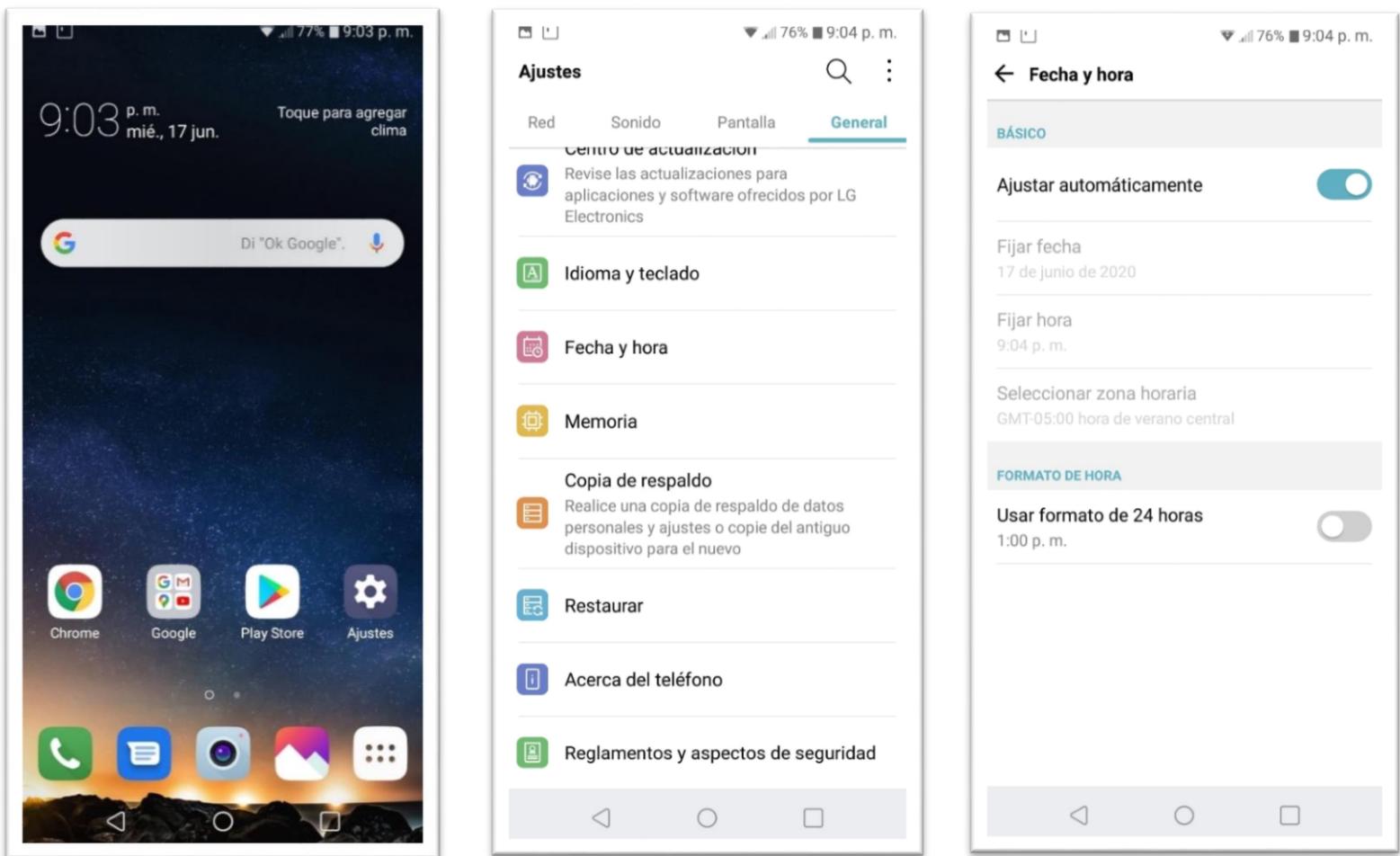
C:\java connector-sqlite-redis-v1。

最后，我们必须选择"触发器"选项卡，在这个选项卡中，我们将给出我们希望任务执行的时间（天、小时、分钟）的参数，这些参数是基于Mini BlocklyChain系统将不得不创建的业务规则。

10. 在系统节点（手机）分、秒同步。

非常重要的是，所有节点的同步主要体现在分、秒部分。由于在确定的交易队列时间内完成发送或发布时，所有的节点都应该同步，因为这将取决于在本地系统中建立的任务管理器与CRON工具在所有节点中同时执行，这将使所有的节点都有相同的概率获得被选择的权利，能够处理交易队列，并能够生成新的块，将其添加到系统的区块链中。为了同步节点，我们有两个选择。

同步节点的第一个选项，我们就可以用简单的方法来实现。在我们的设备是通过内部选项，包括安卓系统，我们将不得不去的部分设置>日期和时间>自动调整。



有了前面的配置，我们将以分秒为单位同步了系统的所有节点无论在世界哪个国家都已经同步了是基于每个地理区域可能变化的是时间但根据分秒应该同步这就足够了我们在所有节点中用cron工具按计划运行任务的过程，以分钟为单位每隔一定的时间运行一次，即我们可以在crontab中创建一个任务，根据每个系统的设计每10分钟或30分钟运行一次。

这在使用"点对点"通信网络时将会很有用，在使用备份网络的情况下，这个过程将不适用，因为事务队列的分配是以客户端-服务器的模式进行的，服务器是控制cron工具的。

参考资料：<https://appinventor.mit.edu/explore/blogs/karen/2016/08.html>

第二个选择是使用外部API，我们将通过扩展（ConnectorSSHClient）执行Curl命令。

我们将使用NTP（网络时间协议）的对外服务的地方是。

<http://worldtimeapi.org/>

现在我们来看看如何从分布在全球的NTP服务器上获取时间，这将帮助我们让所有的节点在同一日期和时间的某个时间有一个查询。

带扩展功能的查询示例（ConnectorSSHClient）。

```
$ curl "http://worldtimeapi.org/api/timezone/America/Mexico_City"
```

我们进行了与Termux终端的连接。



我们执行Curl命令。



我们必须考虑到，Curl命令的结果将是JSON格式，类似于下面的结果。

```
abbreviation: "CDT"
client_ip: "200.77.16.151"
datetime: "2020-06-18T14:16:57.750466-05:00"
day_of_week: 4
day_of_year: 170
dst: true
dst_from: "2020-04-05T08:00:00+00:00"
dst_offset: 3600
dst_until: "2020-10-25T07:00:00+00:00"
raw_offset: -21600
timezone: "America/Mexico_City"
unixtime: 1592507817
utc_datetime: "2020-06-18T19:16:57.750466+00:00"
utc_offset: "-05:00"
week_number: 25
```

另外，结果也可以是没有数据格式化的JSON格式，或者是线性形式的JSON，如下图所示。

```
{"abbreviation": "CDT", "client_ip": "200.77.16.151", "datetime": "2020-06-18T14:16:57.750466-05:00", "day_of_week": 4, "day_of_year": 170, "dst": true, "dst_from": "2020-04-05T08:00:00+00:00", "dst_offset": 3600, "dst_until": "2020-10-25T07:00:00+00:00", "raw_offset": -21600, "timezone": "America/Mexico_City", "unixtime": 1592507817, "utc_datetime": "2020-06-18T19:16:57.750466+00:00", "utc_offset": "-05:00", "week_number": 25}。
```

前面两种方式中的任何一种，我们都要通过现有的JSON扩展，比如"JSONTOOLS"来过滤信息，或者在App Inventor中使用过滤器进行文本处理，根据各个系统的需要，只获取时、日、秒。处理完结果后，可以进行逻辑比较，根据比较结果，我们可以执行一个已经用"cron"服务编程好的任务，稍后我们会看到它在每个节点的配置。

JSONTOOLS扩展参考。

<https://thunkableblocks.blogspot.com/2017/07/jsontools-extension.html>

我们现在已经回顾了两种同步节点时间的选择。我们将继续在每个节点上配置"cron"服务。

配置Android系统（系统节点）上使用CRON服务执行的自动任务调度。

首先我们需要了解自动调度器的工作原理。

cron服务通常所有的系统都预装了这个，所有要自动执行的任务都安排在一个叫 crontab的文件中。

我们用\$ crontab -e编辑crontab文件，我们会使用vi编辑器，里面我们使用格式。

m h dom mon dow 命令

其中：

- m对应脚本执行的分钟，数值从0到59。
- h准确的时间，格式是24小时，数值从0到23，是0午夜12点。
- dom指的是每月的哪一天，例如，如果你想每15日运行一次，你可以指定为15
- dow是指一周中的某一天，它可以是数字（0到7，其中0和7是周日），也可以是英文中一天的前3个字母：mon, tue, wed, thu, fri, sat, sun。
- user定义了执行命令的用户，可以是root，也可以是其他用户，只要他有执行脚本的权限。
- 命令指的是要执行的脚本的命令或绝对路径，例如：

/home/user/scripts/update.sh，如果你调用一个脚本，它必须是可执行的。

为了让大家明白，解释几个cron任务的例子。

15 10 * * * 命令 /home/user/scripts/update.sh

您将在每天上午10:15运行update.sh脚本。

15 22 * * * 命令 /home/user/scripts/update.sh

您将在每天晚上10:15运行update.sh脚本。

00 10 * * 0 root apt-get - ~~并使用根目录~~

您将在每周日上午10点进行更新。

45 10 * * sun root apt-get - ~~并更新~~

根用户将在每周日（太阳）上午10:45运行更新。

我们在编辑器中保存了改动，就这样我们完成了cron服务的配置。如果你的系统中没有安装cron，你可以用下面的命令来完成。

\$ apt install cron

2. 网络节点的安装和配置--移动电话。

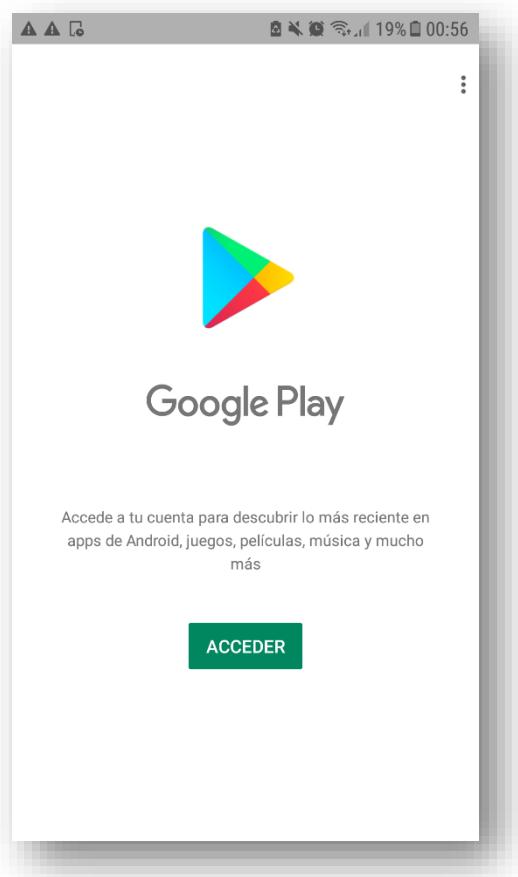
让我们从将使用Mini BlocklyChain的节点的通信网络开始。

首先我们需要一个Linux环境，因为每个Android系统都是基于Linux的，以保证工具的安全性和灵活性，我们将使用"Termux"终端，包含该环境，我们将安装通信网络。

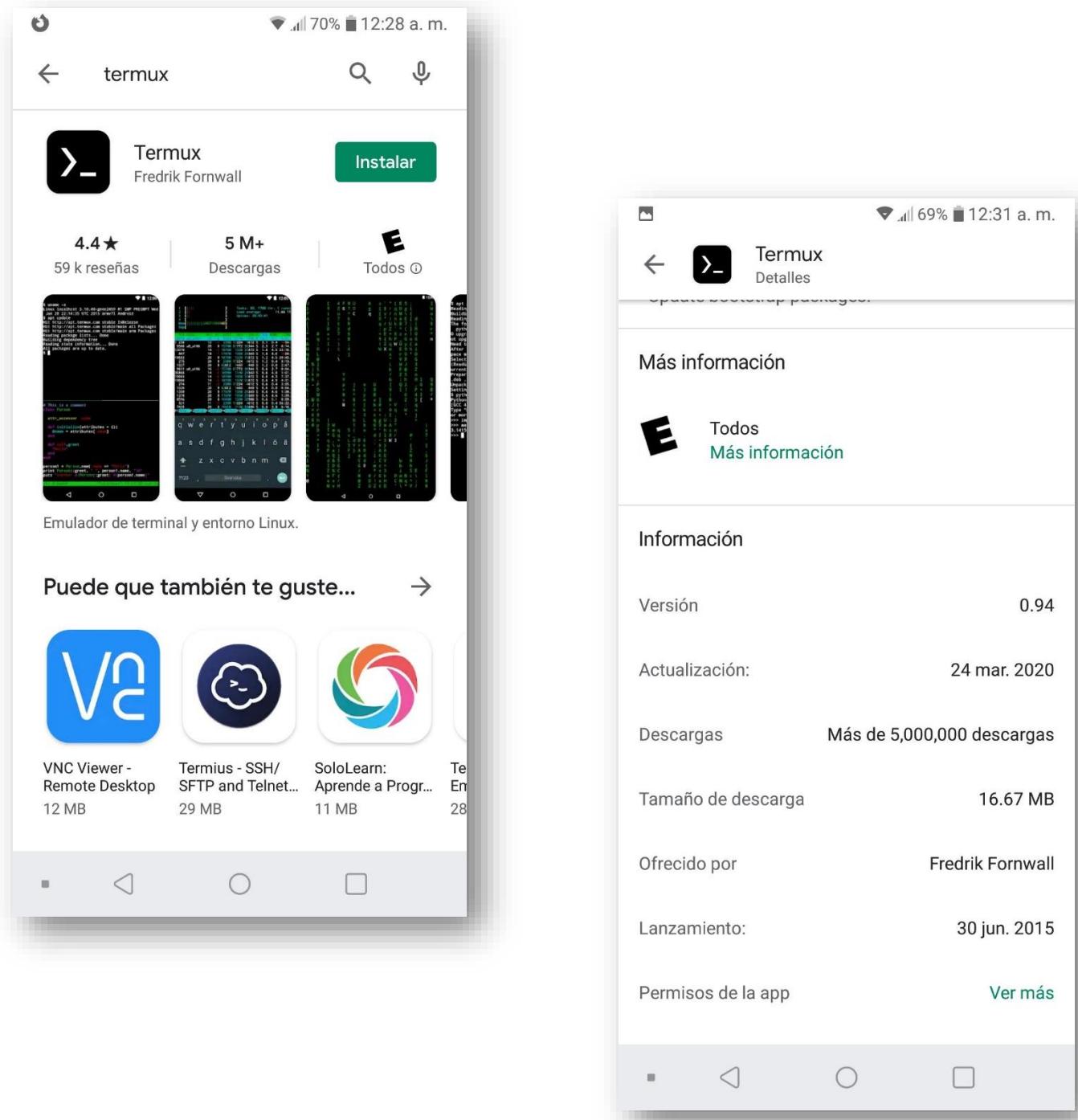
Termux是一个Linux模拟器，我们将在其中安装必要的包来创建我们的节点之间的通信网络。

使用Termux的主要优势之一是，您可以在不需要"旋转"手机（智能手机）的情况下安装程序。这确保了不会因为这种安装而失去制造商的保修。

Termux的安装。在手机上，进入Google Play图标应用 (play.google.com)。



通过应用程序"Termux"搜索，选择它并开始安装过程。



启动Termux应用程序。

启动后我们要执行以下两个命令来执行Linux操作系统模拟器的更新。

```
$ apt update
```

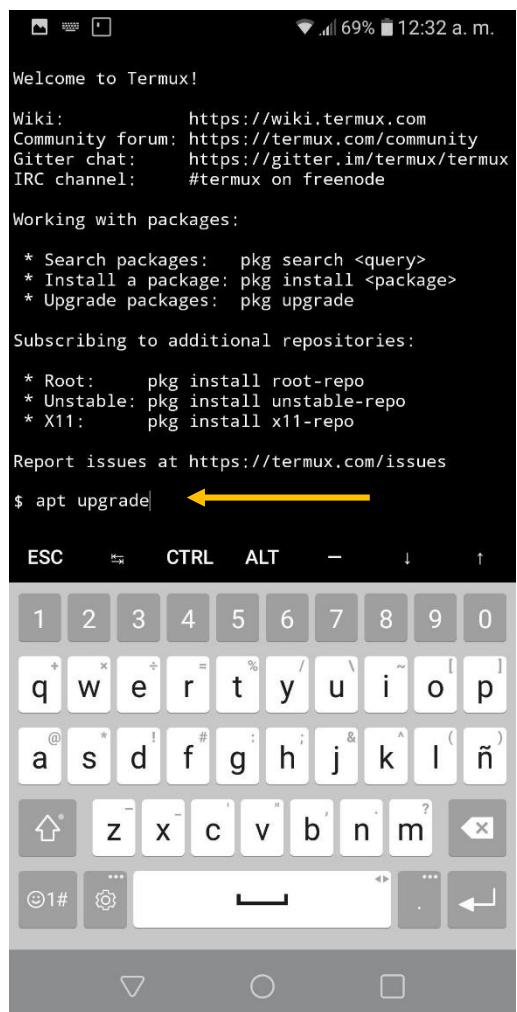
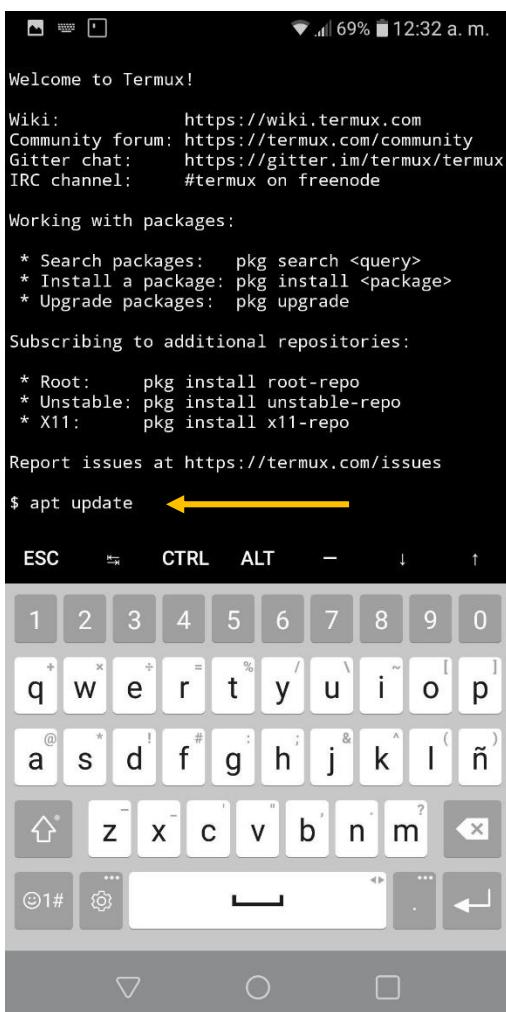
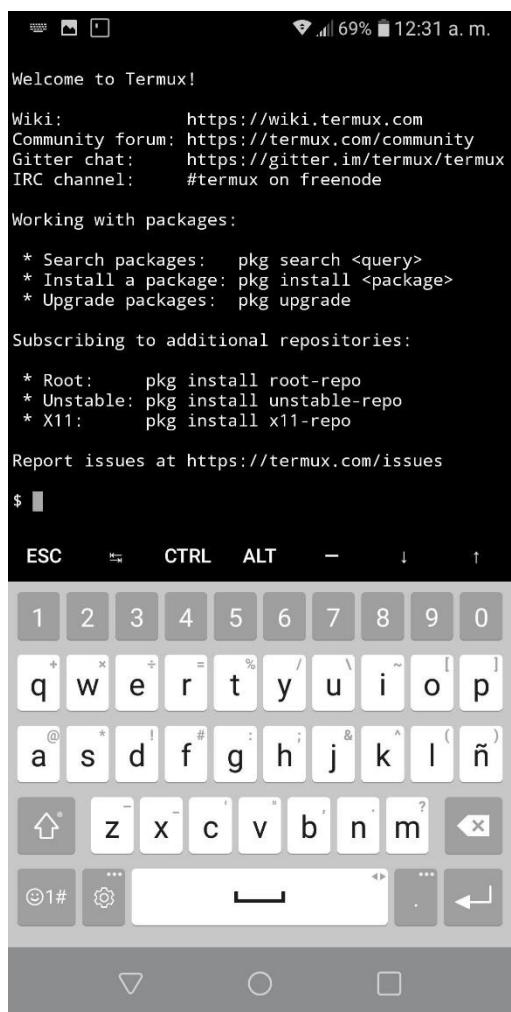
\$ apt升级

确认所有选项 Y(是)...

Termux

```
$ apt update
```

```
$ apt upgrade
```



11. Termux内的存储配置。

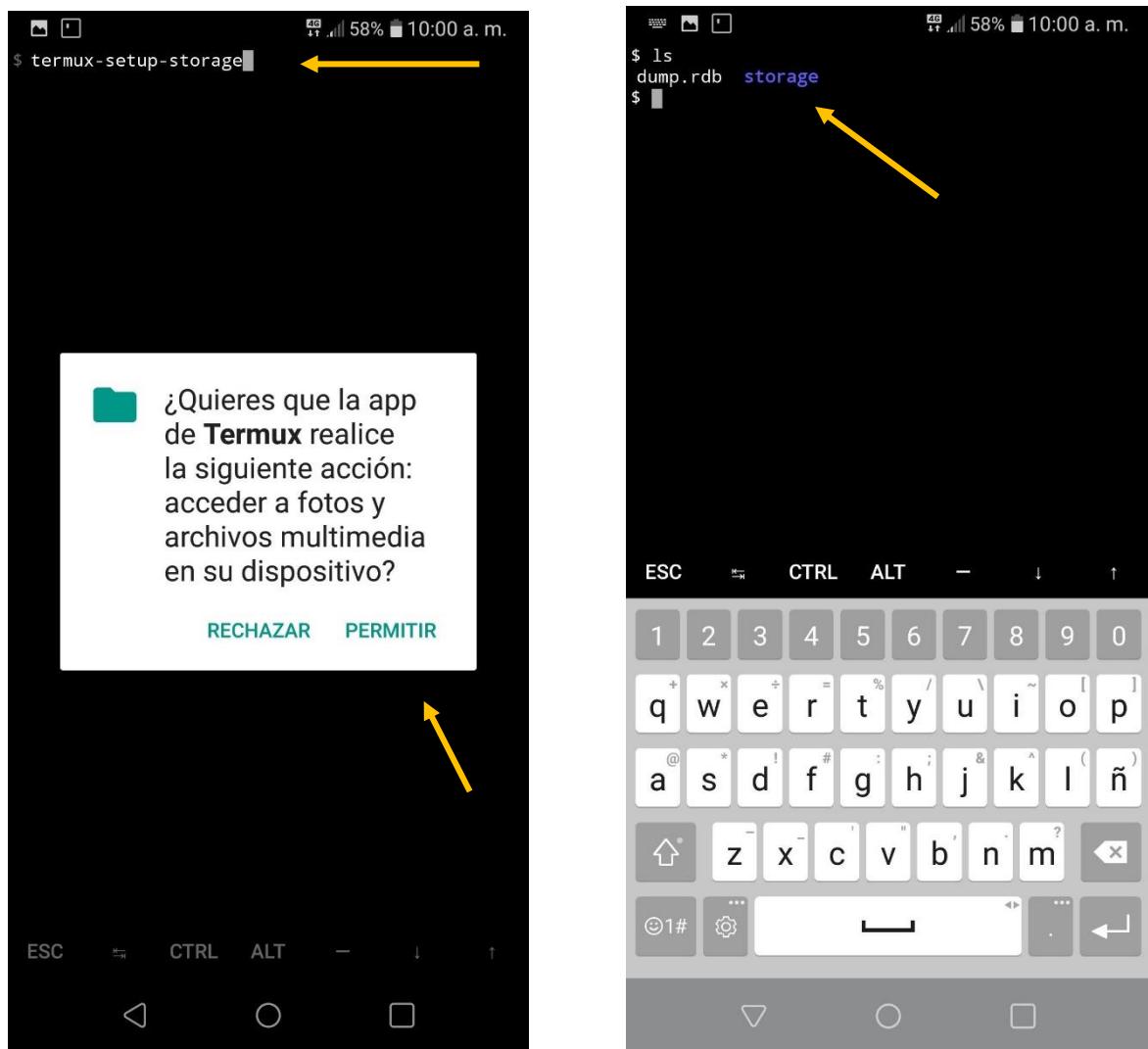
在您更新升级了Termux系统之后，我们将开始配置如何在Termux系统中查看手机的内部存储，这将帮助您能够在Termux和我们的手机信息之间进行信息交换。

可以通过在Termux终端上运行以下命令来简单快速地完成。

```
$ termux-setup-storage
```

当你执行上一条命令时，会出现一个窗口，要求你确认在Termux中创建一个虚拟存储（目录）。我们通过下达命令来验证。

```
$ ls
```



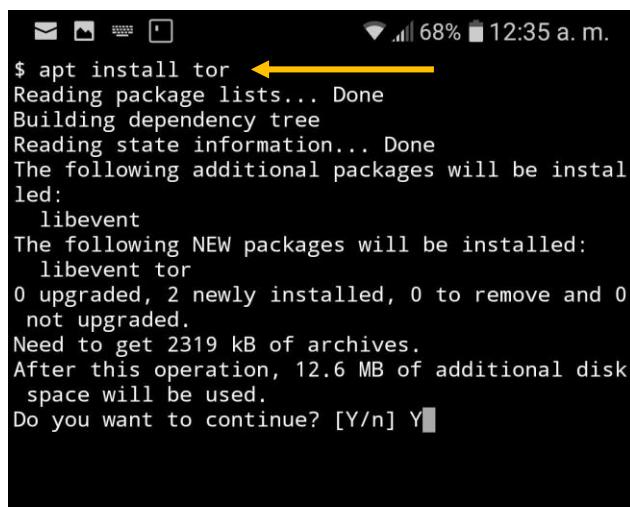
12. "Tor"网络安装和"Syncthing"安装。

```
$ apt install tor
```

```
$ apt install syncthing
```

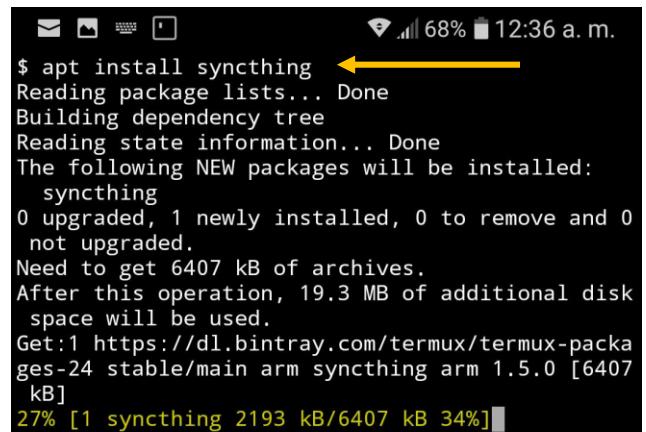
如果要求，在这两种情况下输入大写的Y接受安装...

```
$ apt install tor
```

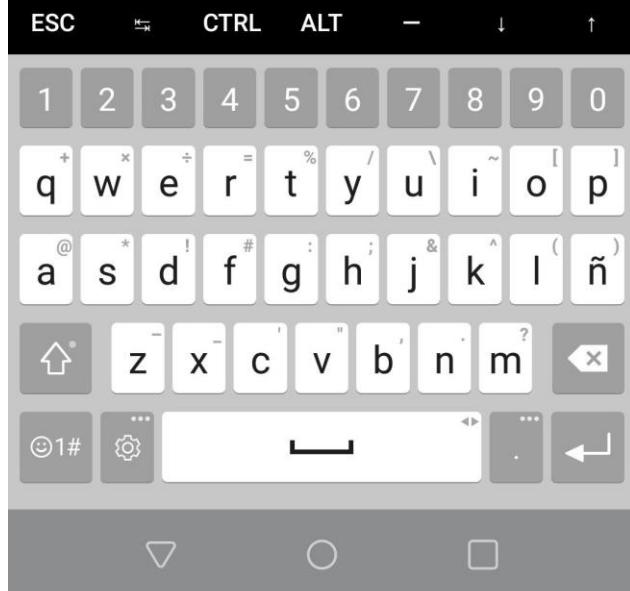


```
$ apt install tor
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
  libevent
The following NEW packages will be installed:
  libevent tor
0 upgraded, 2 newly installed, 0 to remove and 0
not upgraded.
Need to get 2319 kB of archives.
After this operation, 12.6 MB of additional disk
space will be used.
Do you want to continue? [Y/n] Y
```

```
$ apt install syncthing
```



```
$ apt install syncthing
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following NEW packages will be installed:
  syncthing
0 upgraded, 1 newly installed, 0 to remove and 0
not upgraded.
Need to get 6407 kB of archives.
After this operation, 19.3 MB of additional disk
space will be used.
Get:1 https://dl.bintray.com/termux/termux-pac
ges-24 stable/main arm syncthing arm 1.5.0 [6407
kB]
27% [1 syncthing 2193 kB/6407 kB 34%]
```



13. 安装"Redis"数据库和SSH（安全壳）服务器。

```
$ apt install redis
```

```
$ apt install openssh
```

```
$ apt install sshpass
```

\$ apt install redis

```
$ apt install redis
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following NEW packages will be installed:
  redis
0 upgraded, 1 newly installed, 0 to remove and 0
not upgraded.
Need to get 528 kB of archives.
After this operation, 3056 kB of additional disk
space will be used.
Get:1 https://dl.bintray.com/termux/termux-pac
ges-24 stable/main arm redis arm 6.0.1 [528 kB]
Fetched 528 kB in 1s (296 kB/s)
Selecting previously unselected package redis.
(Reading database ... 3265 files and directories
currently installed.)
Preparing to unpack .../archives/redis_6.0.1_arm
.deb ...
Unpacking redis (6.0.1) ...
Setting up redis (6.0.1) ...
$
```



\$ apt install openssh

```
$ apt install openssh
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be instal
led:
  krb5_ldns libdb libedit termux-auth
The following NEW packages will be installed:
  krb5_ldns libdb libedit openssh termux-auth
0 upgraded, 6 newly installed, 0 to remove and 0
not upgraded.
Need to get 2255 kB of archives.
After this operation, 11.9 MB of additional disk
space will be used.
Do you want to continue? [Y/n] Y
Get:1 https://dl.bintray.com/termux/termux-pac
ges-24 stable/main arm libdb arm 18.1.32-4 [465
kB]
Get:2 https://dl.bintray.com/termux/termux-pac
ges-24 stable/main arm krb5 arm 1.18.1 [839 kB]
24% [2 krb5 131 kB/839 kB 16%]
```



\$ apt install sshpass

```
$ apt install sshpass
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following NEW packages will be installed:
  sshpass
0 upgraded, 1 newly installed, 0 to remove and 0
not upgraded.
Need to get 7158 B of archives.
After this operation, 57.3 kB of additional disk
space will be used.
0% [Working]
```



通信网络的安装已经完成，我们继续配置包：Tor、Syncthing和Redis DB。

14. 在手机（智能手机）上配置SSH服务器。

我们将使手机中的SSH服务器能够从我们的PC连接到手机上，并且能够以更快、更舒适的方式工作，同时它也将起到检查手机中SSH服务器的服务是否正常工作的作用，因为我们将在Mini BlocklyChain的通信网络中使用它。

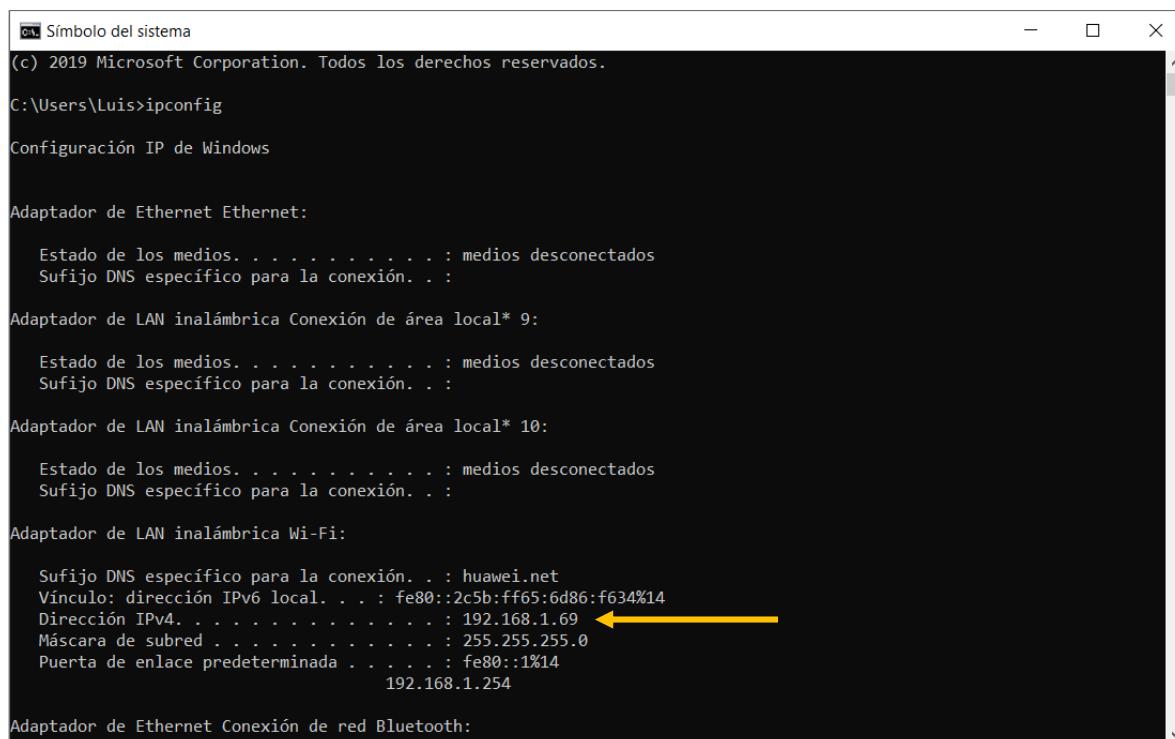
首先我们要做的是将手机和电脑连接到同一个WiFi网络，这样就可以看到对方了。IP或地址必须类似于192.168.XXX.XXX的XXX值是每个计算机中随机分配的可变数字。

这个例子是在LG Q6手机和装有Windows 10 Home的PC上测试的。

检查电脑连接到WiFi的IP或地址，我们必须在Windows中打开一个终端。

在底部面板搜索放大镜的地方写上cmd，按回车键。一个终端将打开，我们在其中写下命令。

C:\User_Name> ipconfig



```
Símbolo del sistema
(c) 2019 Microsoft Corporation. Todos los derechos reservados.

C:\Users\Luis>ipconfig

Configuración IP de Windows

Adaptador de Ethernet Ethernet:
  Estado de los medios. . . . . : medios desconectados
  Sufijo DNS específico para la conexión. . . :

Adaptador de LAN inalámbrica Conexión de área local* 9:
  Estado de los medios. . . . . : medios desconectados
  Sufijo DNS específico para la conexión. . . :

Adaptador de LAN inalámbrica Conexión de área local* 10:
  Estado de los medios. . . . . : medios desconectados
  Sufijo DNS específico para la conexión. . . :

Adaptador de LAN inalámbrica Wi-Fi:
  Sufijo DNS específico para la conexión. . . : huawei.net
  Vínculo: dirección IPv6 local. . . : fe80::2c5b:ff65:6d86:f634%14
  Dirección IPv4. . . . . : 192.168.1.69
  Máscara de subred . . . . . : 255.255.255.0
  Puerta de enlace predeterminada . . . . : fe80::1%14
                                         192.168.1.254

Adaptador de Ethernet Conexión de red Bluetooth:
```

它将显示我们分配给PC的IP是192.168.1.69，但这很可能在每个情况下是不同的。

注意：应该取写有"IPv4地址"的地址，不要和网关混淆。

现在在Termux终端的手机的情况下，我们必须键入下面的命令来知道我们的用户名，我们将用来连接到拥有我们手机的SSH服务器，我们执行下面的命令。

现在在Termux终端的手机的情况下，我们必须键入下面的命令来知道我们的用户名，我们将用来连接到拥有我们手机的SSH服务器，我们执行下面的命令。

\$ Whoami

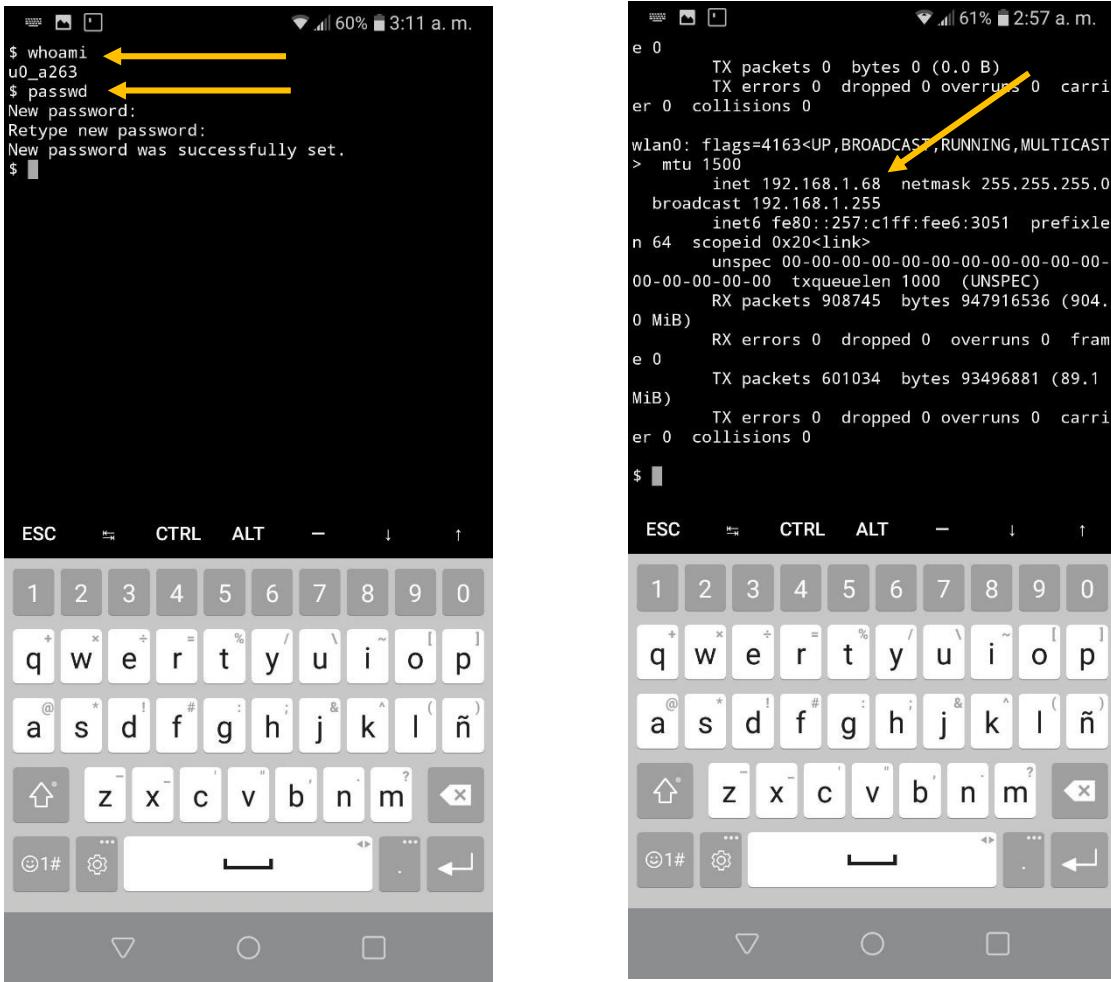
之后我们必须给这个用户一个密码，所以我们要执行下面的命令。

\$ passwd

它会要求我们输入密码并按回车键，再次要求我们输入密码我们确认后按回车键，如果已经**成功了"新密码设置成功"**在标记错误的情况下有可能是密码没有输入正确。再次执行该程序。

然后要知道我们在Termux中的IP是什么，我们输入以下命令，IP在"inet"后面。

\$ ifconfig -a



现在是时候启动手机上的SSH服务器服务了，这样你就可以从电脑上接收会话了。我们在Termux终端执行以下命令，这个命令没有任何结果。

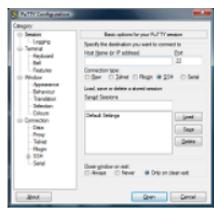
```
$ sshd
```



现在我们要在电脑上安装一个程序，这个程序可以从电脑上与手机的SSH服务器进行通信。

我们要去<https://www.putty.org>

选择"您可以在这里下载PuTTY"的链接所在。

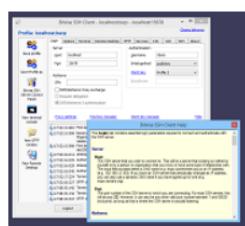


Download PuTTY

PuTTY is an SSH and telnet client, developed originally by Simon Tatham for the with source code and is developed and supported by a group of volunteers.

You can download PuTTY [here](#).

Below suggestions are independent of the authors of PuTTY. They are *not* to be seen as



Bitvise SSH Client

Bitvise SSH Client is an SSH and SFTP client for Windows. It is developed and supported prof supports all features supported by PuTTY, as well as the following:

- graphical SFTP file transfer;
- single-click Remote Desktop tunneling;
- auto-reconnecting capability;
- dynamic port forwarding through an integrated proxy;
- an FTP-to-SFTP protocol bridge.

Bitvise SSH Client is **free to use**. You can [download it here](#).

选择32位的版本，不管你的系统是不是64位的都可以。

Download PuTTY: latest release

[Home](#) | [FAQ](#) | [Feedback](#) | [Licence](#) | [Updates](#) | [Mirror](#)
Download: **Stable** · [Snapshot](#) | [Docs](#) | [Changelog](#)

This page contains download links for the latest released version of PuTTY. Currently this is 0.73, released on 2019-09-29.

When new releases come out, this page will update to contain the latest, so this is a good page to bookmark or link to. Alternative Release versions of PuTTY are versions we think are reasonably likely to work well. However, they are often not the most up-to-date, so check the [development snapshots](#), to see if the problem has already been fixed in those versions.

Package files

You probably want one of these. They include versions of all the PuTTY utilities.

(Not sure whether you want the 32-bit or the 64-bit version? Read the [FAQ entry](#).)

MSI ('Windows Installer')

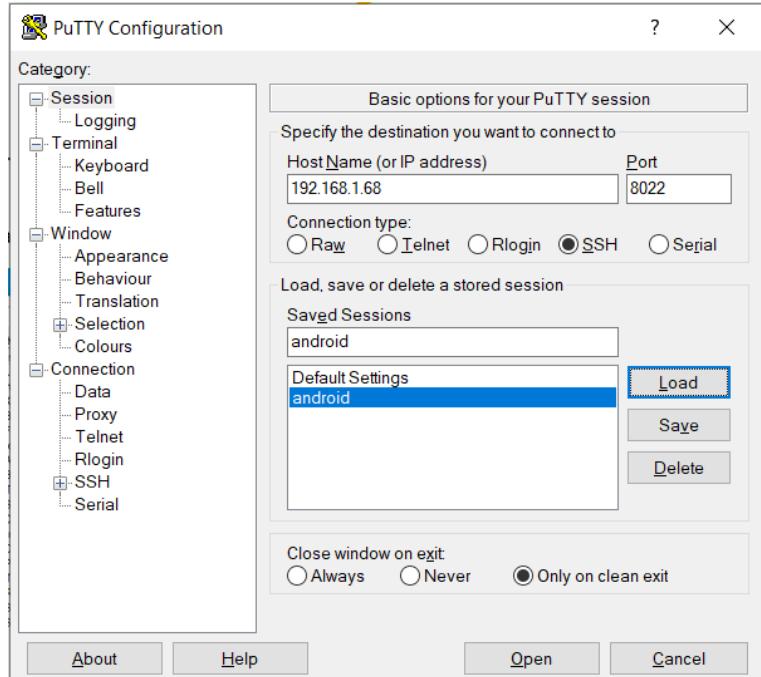
32-bit:	putty-0.73-installer.msi	(or by FTP)	(signature)
64-bit:	putty-64bit-0.73-installer.msi	(or by FTP)	(signature)

Unix source archive

.tar.gz:	putty-0.73.tar.gz	(or by FTP)	(signature)
----------	-----------------------------------	------------------------------	-----------------------------

下载到电脑后，运行它并以默认选项进行安装。然后启动PuTTY应用程序。

在这个环节中，我们将从我们安装在手机中的Openssh服务器中输入数据。



输入手机的IP。

主机名或IP地址。

192.168.1.68 (IP例子)

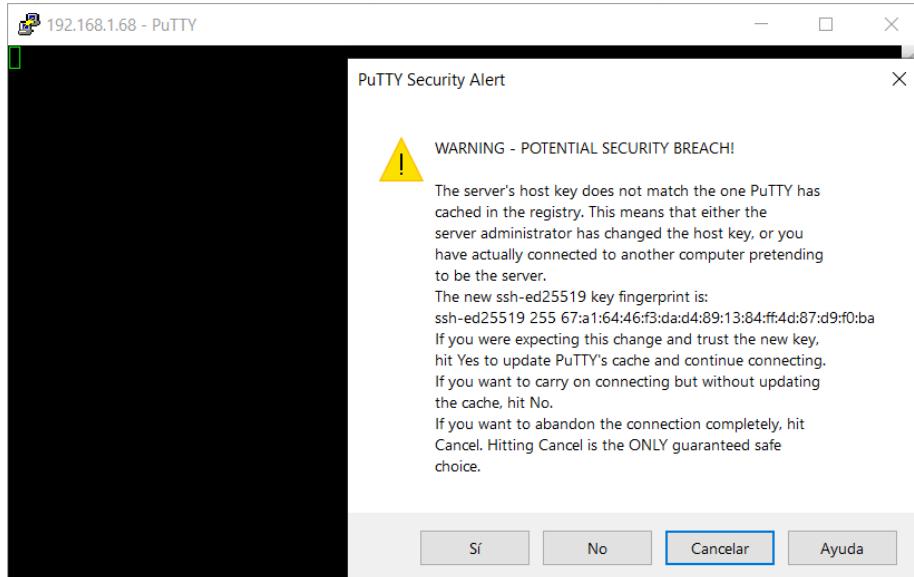
港口：

8022 (移动SSH服务器的默认端口)。

我们可以在"保存的会话"中给会话起个名字，然后点击保存按钮。

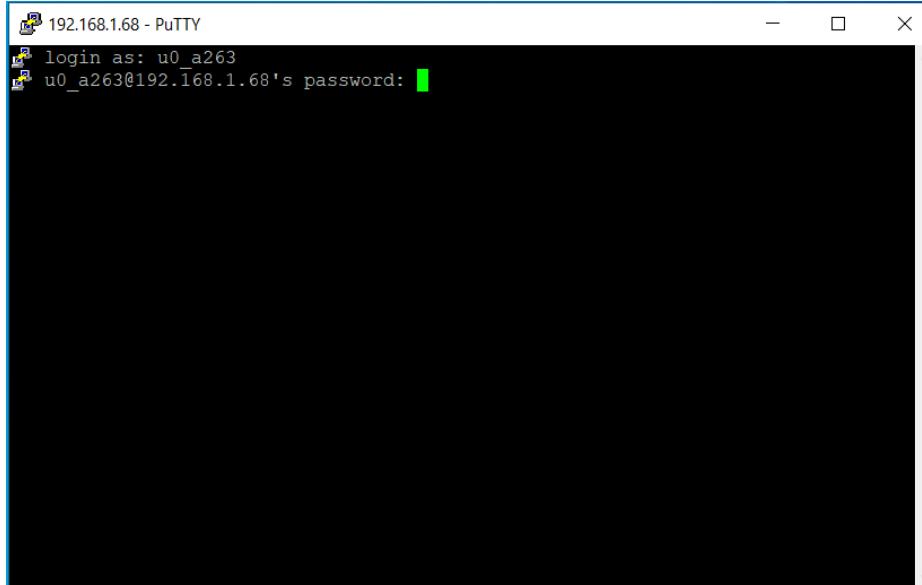
稍后在下半部分我们按下打开连接服务器的按钮，给出"Open"。

在PC上，当您第一次连接时，您会被要求确认信息加密密钥，点击"是"按钮。

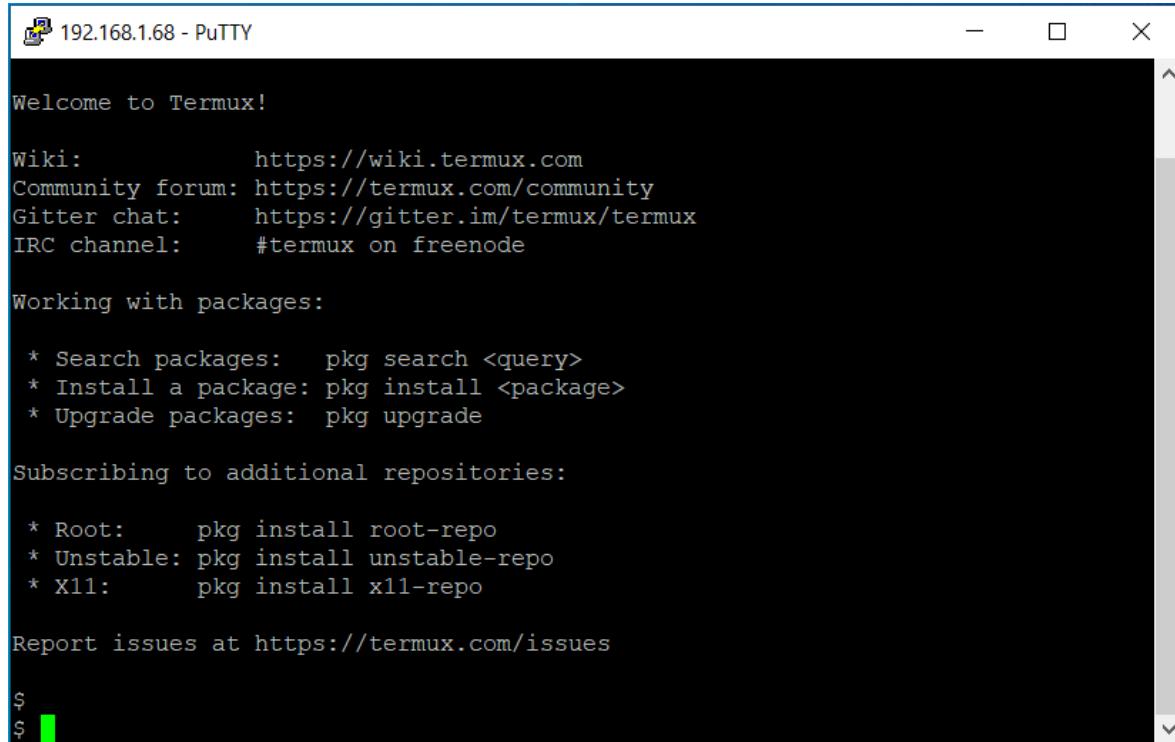


稍后会问我们要连接的用户。我们将使用之前获得的信息（用户名和密码）。

在登录为：我们必须输入我们的用户名，并给回车，然后我们会要求密码再次给回车按钮。



如果数据正确，我们将在从PC（客户端）在手机（SSH服务器）上执行的SSH（安全壳）会话中。



```
Welcome to Termux!

Wiki: https://wiki.termux.com
Community forum: https://termux.com/community
Gitter chat: https://gitter.im/termux/termux
IRC channel: #termux on freenode

Working with packages:

* Search packages: pkg search <query>
* Install a package: pkg install <package>
* Upgrade packages: pkg upgrade

Subscribing to additional repositories:

* Root: pkg install root-repo
* Unstable: pkg install unstable-repo
* X11: pkg install x11-repo

Report issues at https://termux.com/issues

$
```

重要提示：请记住，PC的IP（地址）和连接在同一WiFi上的手机的IP（地址）可能会在每次断开和重新连接时发生变化，所以我们必须仔细检查每个设备的地址，这将确保设备之间通过手机的SSH服务器和PC（客户端）连接成功。

到目前为止，我们只能连接到同一个WiFi网络，但如果我们将手机移动到与PC相同的网络之外，我们将无法连接，因为通过其他更复杂的通信设备涉及到不同的网络。当我们建立"Tor"网络时，我们将解决这个问题。

请记住，这个连接只是为了验证我们安装在手机上的服务器的服务，通过PC与手机的远程会话，可以有一个更舒适的工作环境。

15. Tor"网络配置与SSH（安全壳）服务。

通过从PC上的远程会话，我们将开始配置启用SSH服务的"Tor"网络。

拥有"Tor"网络的重要意义在于，让设备具备了在世界任何地方都能通过互联网进行通信的属性，而不在同一个WiFi网络中，无论我们身处何地，都能在节点之间进行连接，形成"Peer to Peer"的网络，这也是Mini BlocklyChain架构的一个重要功能。

Tor"网络在配置上有很大的灵活性，因为它的配置文件"torrc"中有几个参数，这个文件在路径中(`$PREFIX/etc/tor/torrc`)，在我们的例子中，我们将通过输入下面的命令来了解Termux会话。

```
$ echo $PREFIX
```

```
/data/data/com.termux/files/usr。
```

因此，我们需要编辑的文件将在路径中。

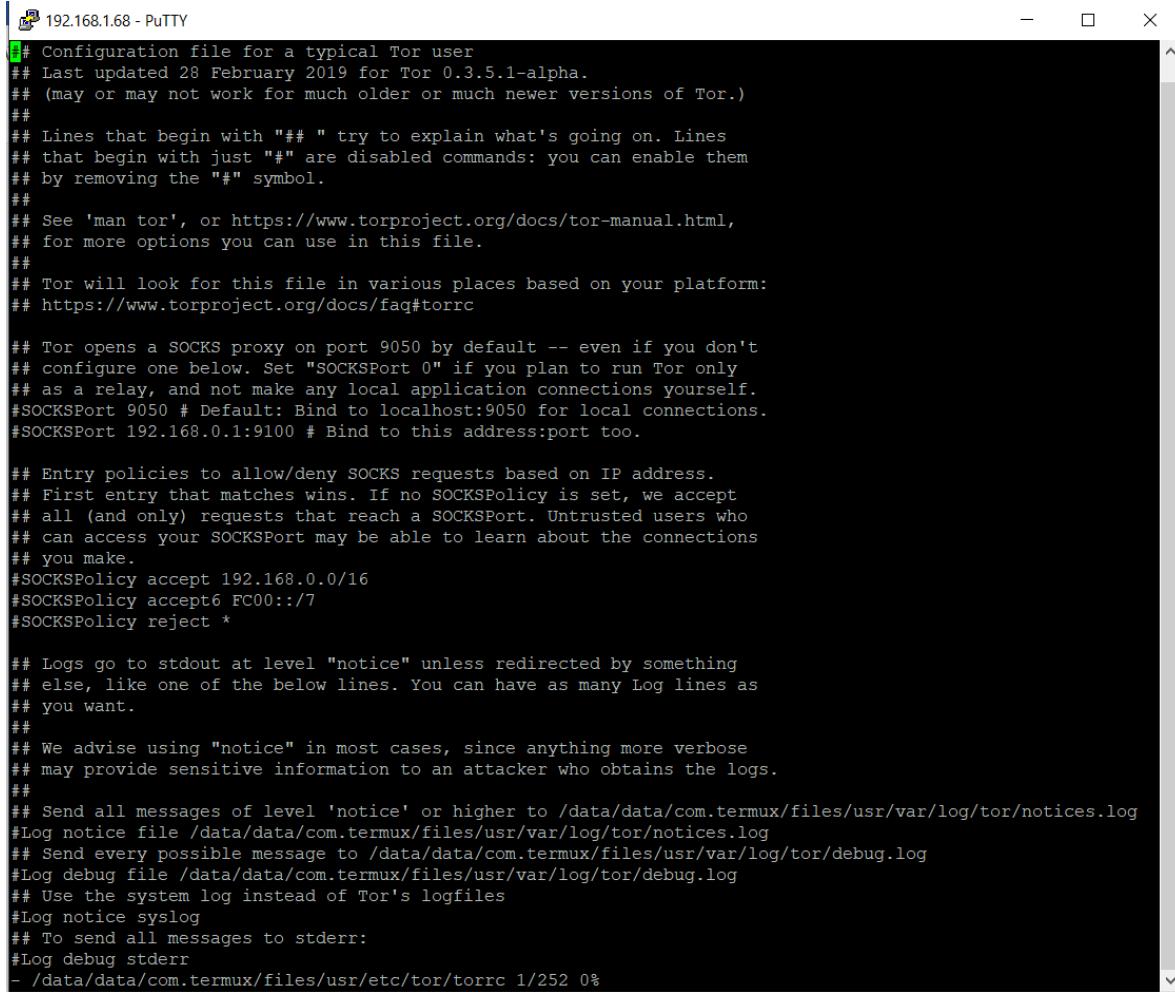
`PREFIX/etc/tor/torrc`，等于`/data/data/com.termux/files/usr/etc/tor/torrc`。

我们继续使用命令行编辑器"vi"编辑配置文件，执行以下命令。

```
vi /data/data/com.termux/files/usr/etc/tor/torrc
```

他将为我们编辑该文件，作为一个例子。编辑了"torrc"文件。





```
# Configuration file for a typical Tor user
## Last updated 28 February 2019 for Tor 0.3.5.1-alpha.
## (may or may not work for much older or much newer versions of Tor.)
##
## Lines that begin with "## " try to explain what's going on. Lines
## that begin with just "#" are disabled commands: you can enable them
## by removing the "#" symbol.
##
## See 'man tor', or https://www.torproject.org/docs/tor-manual.html,
## for more options you can use in this file.
##
## Tor will look for this file in various places based on your platform:
## https://www.torproject.org/docs/faq#torrc

## Tor opens a SOCKS proxy on port 9050 by default -- even if you don't
## configure one below. Set "SOCKSPort 0" if you plan to run Tor only
## as a relay, and not make any local application connections yourself.
#SOCKSPort 9050 # Default: Bind to localhost:9050 for local connections.
#SOCKSPort 192.168.0.1:9100 # Bind to this address:port too.

## Entry policies to allow/deny SOCKS requests based on IP address.
## First entry that matches wins. If no SOCKSPolicy is set, we accept
## all (and only) requests that reach a SOCKSPort. Untrusted users who
## can access your SOCKSPort may be able to learn about the connections
## you make.
#SOCKSPolicy accept 192.168.0.0/16
#SOCKSPolicy accept6 FC00::/7
#SOCKSPolicy reject *

## Logs go to stdout at level "notice" unless redirected by something
## else, like one of the below lines. You can have as many Log lines as
## you want.
##
## We advise using "notice" in most cases, since anything more verbose
## may provide sensitive information to an attacker who obtains the logs.
##
## Send all messages of level 'notice' or higher to /data/data/com.termux/files/usr/var/log/tor/notices.log
#Log notice file /data/data/com.termux/files/usr/var/log/tor/notices.log
## Send every possible message to /data/data/com.termux/files/usr/var/log/tor/debug.log
#Log debug file /data/data/com.termux/files/usr/var/log/tor/debug.log
## Use the system log instead of Tor's logfiles
#Log notice syslog
## To send all messages to stderr:
#Log debug stderr
- /data/data/com.termux/files/usr/etc/tor/torrc 1/252 0%
```

在这个"torrc"文件中，我们要增加或使用该文件所拥有的行，要做如下修改，三行是以下内容。

语法：SOCKSPort <应用端口号>。

例如：SOCKSPort 9050

SOCKSPort变量告诉我们，这个通过TCP-IP协议的通信套接字默认使用移动设备（电话），9050端口将被打开或使用。

语法：HiddenServiceDir <将保存应用程序配置的目录>。

例如：HiddenServiceDir /data/data/com.termux/files/。

HiddenServiceDir变量告诉我们，这将是存储通过Tor网络使用的服务配置的目录，在这个目录中，你会发现服务的配置和安全文件，在这个目录中，你会发现一个名为**hostname**的文件。

我们可以看到Tor网络为创建的特定服务分配的地址，在我们的案例中是创建一个SSH服务，将在Tor网络上实现，我们命名为**hidden_ssh**的目录将包含**主机名**文件。这个目录不需要创建，因为当我们启动Tor网络服务时，它会自动创建。

要想看到Tor网络提供的地址，我们可以使用"more"命令，在使用这个命令之前，我们必须完成"torrc"文件的配置，并注册Tor网络服务，这样**hidden_ssh**目录和里面的文件就会被创建，就像**hostname**文件一样。

更多 /data/data/com.termux/files/。

上面的命令将产生一个带有字母数字字符串的地址，其扩展名为.onion，类似于：

uqwthf6ojdmoybyzvudoq3x2weq7k7rjitblhba3pjbausk2nvjmx3wer.onion。

最后，我们需要在"torrc"配置文件中添加**HiddenServicePort**变量，这个参数告诉Tor网络，我们要注册的应用将在Tor网络上使用哪个端口。

语法：HiddenServicePort <出发端口> <本地或公共IP> : <出发端口>。

O

HiddenServicePort <出发端口>。

例如：

HiddenServicePort 22 127.0.0.1:8022

O

隐藏服务端口 8022

通过以上的介绍，我们已经完成了对Tor网络的通用服务和SSH（Secure Shell）服务的配置，这个服务将用于SQLite数据库的更新通信，我们将在这里保存Mini BlocklyChain系统的验证过程。

我们继续进行迷你BlocklyChain通信网络的配置。

16. 点对点系统配置与手动同步。

在区块链技术系统中，“点对点”架构是最基本的，因为这种架构在系统通信过程中给出了两个中心点，一个是无论节点是在私网（wifi）还是公网（互联网），或者是这两者的混合体，都能在任何时候在所有节点中提供相同的更新（同步）信息，这种架构的第二点是它们不依赖中介（服务器）在节点之间传输、更新或咨询信息。这都是因为通信是直接在节点之间完成的，在我们的案例中Mini BlocklyChain的通信是直接在组成网络的手机之间完成的，没有中间人。

对于这个任务，我们将使用开源的Syncthing工具，它基于“点对点”架构工作。

设备（手机）的Syncthing配置将被手动和自动查看。手动形式最多适用于5个节点的同步，在拥有大量节点的情况下，自动配置是最佳的，这个过程中重点是注册我们要与之进行同步的节点，选择信息。

开始的要求是：



1. 已经启动了Tor网络的服务。
2. (可选--推荐)安装了一个可以读取二维码的应用程序，我们建议从Google Play上安装App inventor Android应用程序，该应用程序简单易行，稍后在本手册中，我们将在"迷你区块链中区块的定义和使用"一节中使用它。
3. 启动了新星的服务。

我们展示了App Inventor应用程序，我们将用于读取二维码，将来服务于我们在Syncthing的节点注册。

下面的例子是在两个移动设备(手机)之间使用以下型号进行的。

移动设备1：LG Q6型

移动设备2：Samsung模型

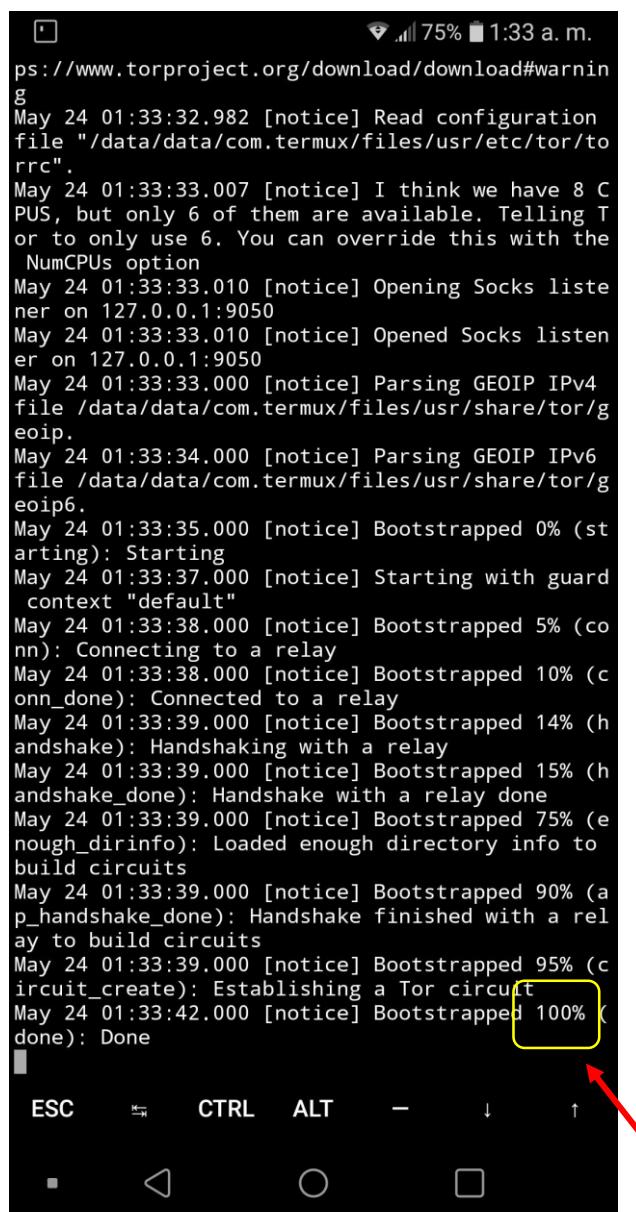
我们先用以下命令启动Tor网络服务和同步服务，在Termux里面打开两个终端，分别运行每个命令。

\$ tor

在你输入Tor网络程序启动命令后，你应该检查执行是否成功，通过检查是否100%完成。

在Termux终端上运行Tor程序，我们验证它是否以100%的速度运行。

\$ tor



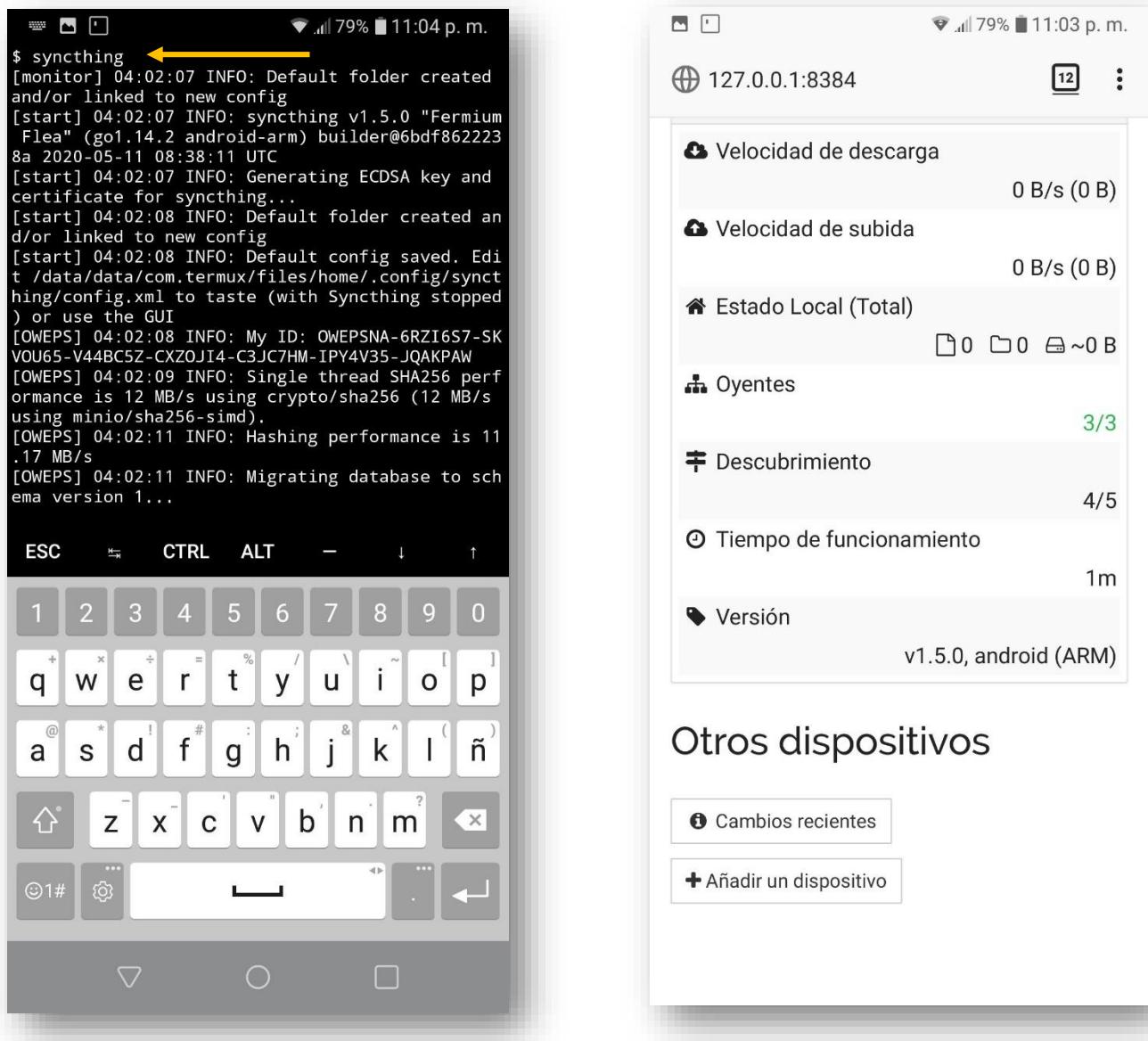
然后我们执行syncing命令。

\$同步

执行这个命令后，它会在我们手机的浏览器中打开一个管理页面，如果它没有自动打开，我们可以进入我们安装的任何一个浏览器，在我们平时上网浏览的地方，我们可以把下面的。

<http://127.0.0.1:8384>

它将打开该工具的管理界面，帮助我们在系统的所有节点（电话）之间同步信息。



由于我们已经打开了"同步"管理界面，我们继续注册我们想要同步信息的节点。此时，我们将占据读取二维码的程序。

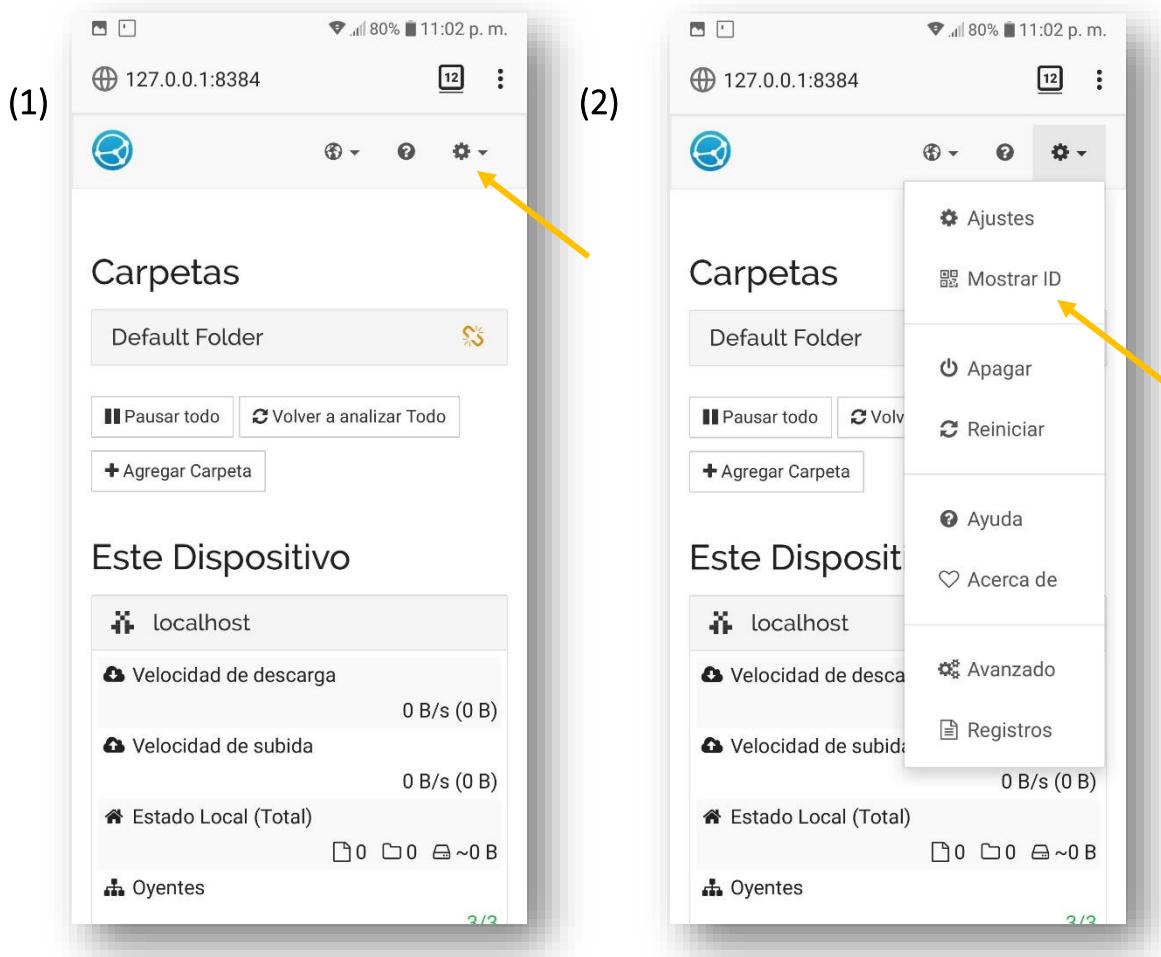
同步程序在第一次启动时，会创建一个手机的唯一标识符，这个标识符由八组大写字母数字字符组成，这个标识符（ID）是我们在我们要同步信息的节点中注册的。

在我们的案例中，LG Q6的手机ID必须要注册到Samsung手机上，而Samsung手机ID必须要注册到LG Q6上。它们必须同时存在于两部手机中才能正常工作。

我们将在LG Q6手机上进行Samsung手机注册的步骤。

首先(1)在Samsung手机的管理界面(互联网浏览器)上面，带同步的我们会点击右上方的菜单标签。

第二 (2) 步我们会看到一个菜单，在这里面我们点击"显示ID"的Samsung。

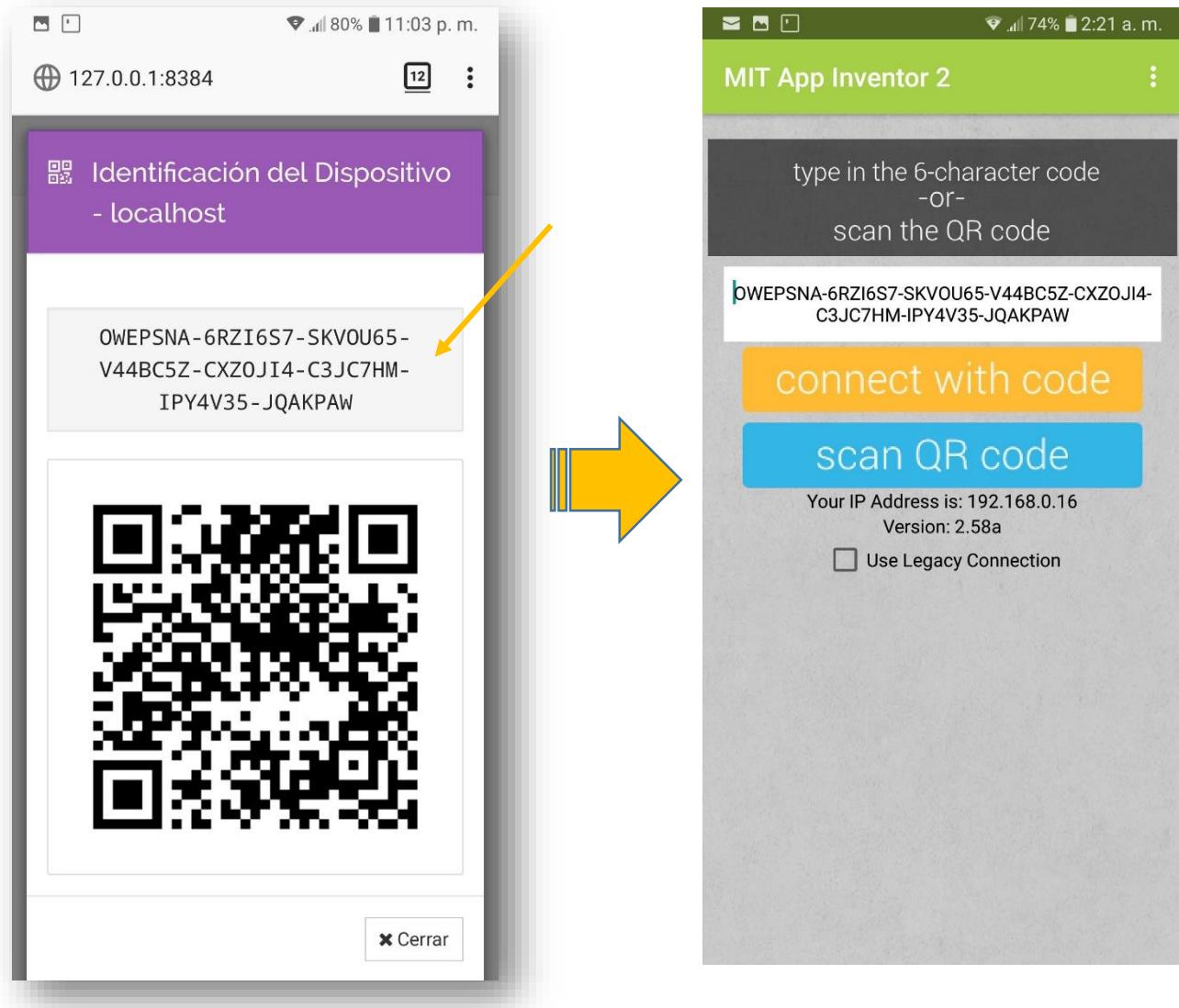


当你点击"显示ID"时，会出现以下界面，这是一个Samsung手机的二维码，在我们的案例中，识别手机的ID为

OWEPSNA-6RZI6S7-SKVOU65-V44BC5Z-CXZOJI4-C3JC7HM-IPY4V35-JQAKPAW

那么在LG Q6手机中，帮助我们捕捉这个Samsung的二维码的程序就是我们建议从App Inventor应用中获取的程序（可选），虽然在我们没有这个程序的情况下，我们也可以在LG Q6开始注册的时候直接手动引入，不过为了避免出现错误，我们建议使用它。

利用安装在LG Q6手机中的App程序发明者，采集我们要同步的远程三胞手机的二维码。



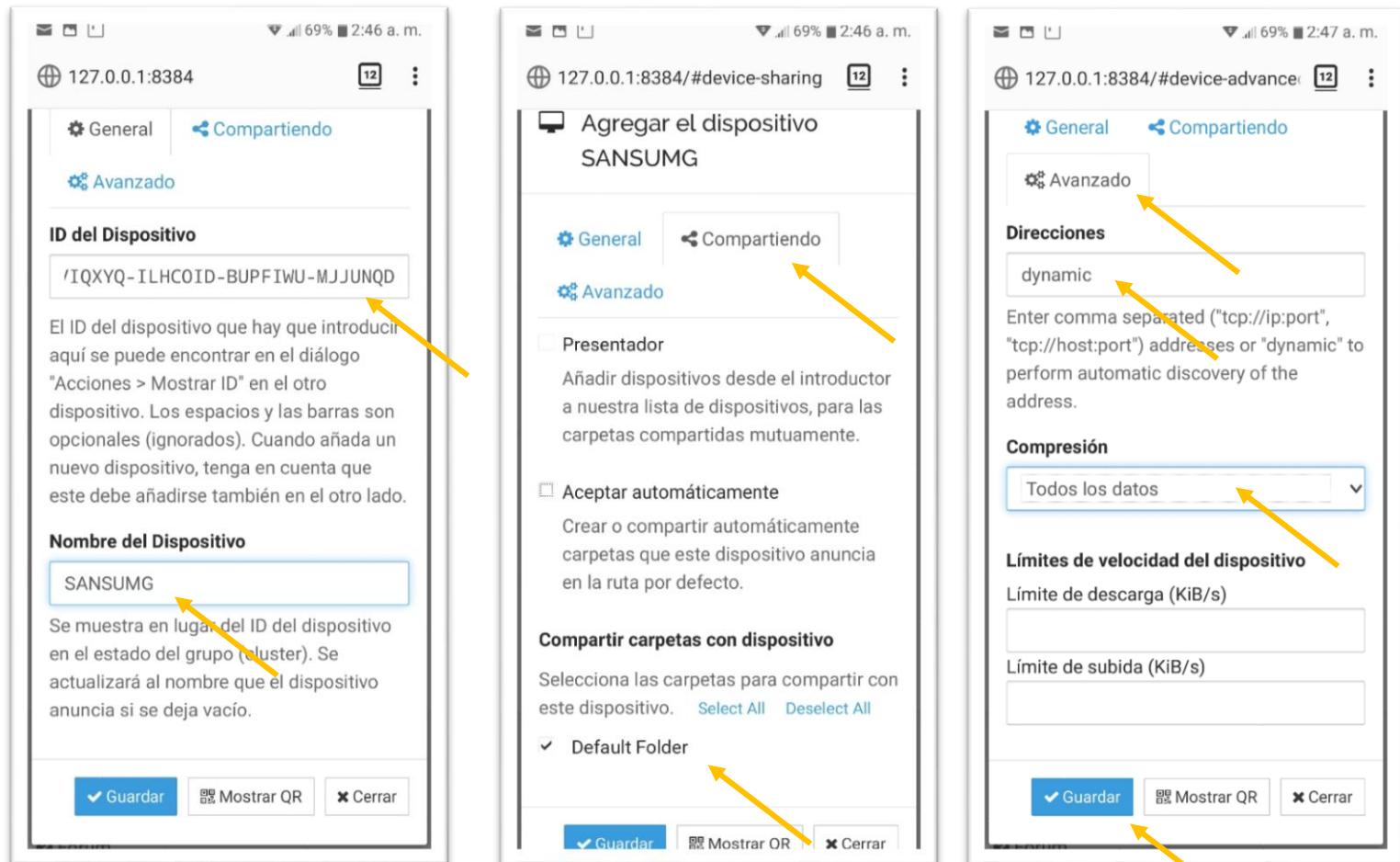
然后我们在App inventor程序中点击捕捉到的二维码复制到剪贴板，我们选择"SELECT ALL"→"COPY"。

有了这些信息，我们就开始在LG Q6上注册Samsung。在管理界面中，我们移动到下方写着"其他设备"的地方，点击"添加设备"按钮，我们介绍一下Samsung的ID，并给它起一个注册名称。

然后在上面的标签"共享"，我们点击，并在此我们标记在文件夹"默认"的较低的选项，我们将共享一个目录，由默认创建的同步在我们的设备。

然后在顶部我们点击"高级"标签，选择"所有数据"下的数据压缩选项。

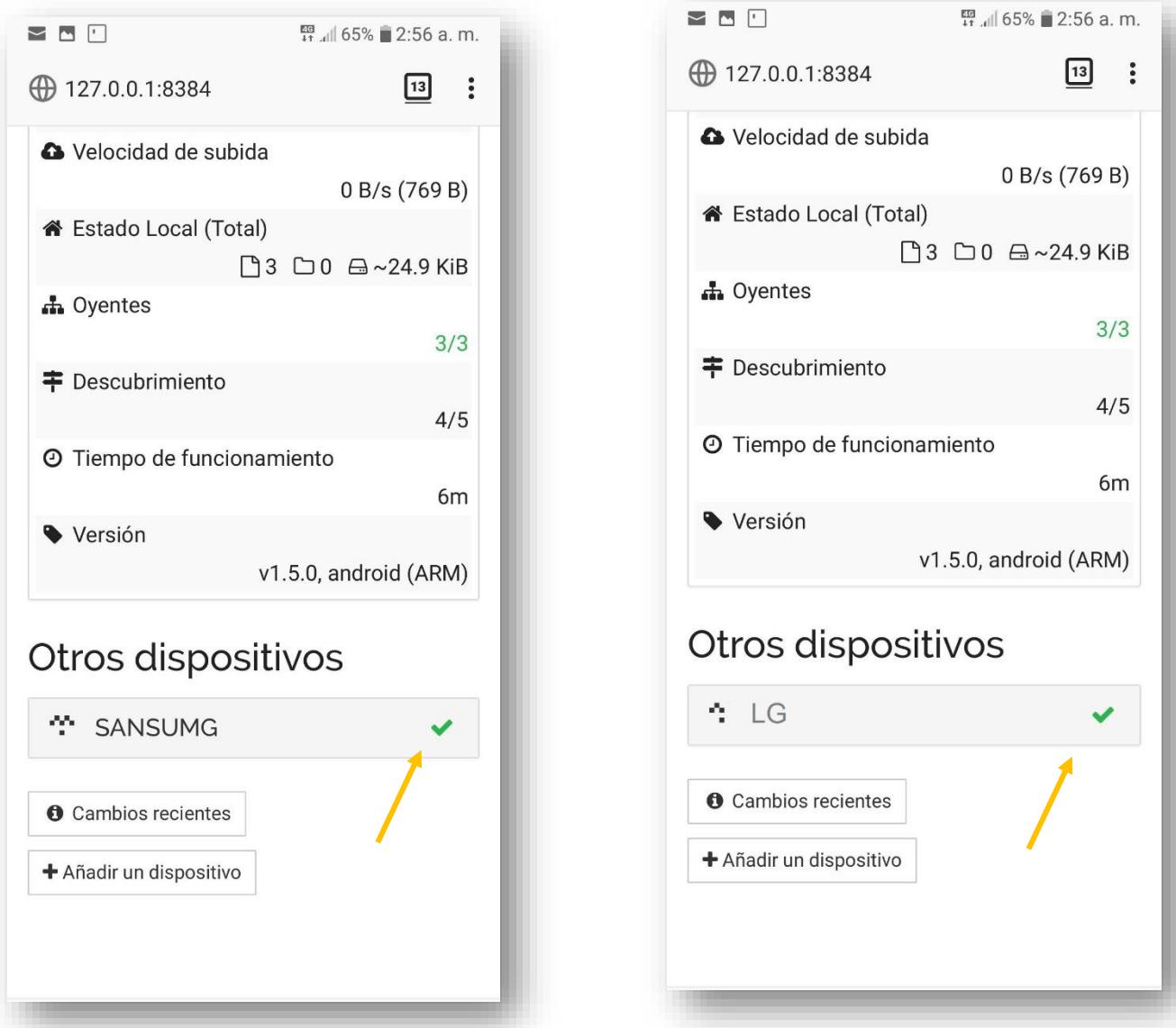
最后我们点击下方的保存按钮，就完成了一个节点中的注册，这个过程同样必须在我们要同步的远程手机或手机中完成。



当在两部手机中保存和注册时，我们会在设备所在的地方等待最长30秒的时间，设备之间的连接将显示为良好，并给出绿色的确认。

设备#1 LG Q6

设备#2 Sansumg



位于路径 /data/data/com.termux/file/home/Sync 目录下的所有信息都将被同步，任何变化都将被复制并同步。

Peer to Peer 系统配置与 Syncthing 自动在 Mini BlocklyChain 中使用。

现在我们将以自动的方式进行配置，之前我们以手动的方式进行配置，如果我们处理的节点数量最少的话，这是很有用的，但是，在节点数量很多的情况下，效率就

会很低，所以我们后面会有SyncthingManager工具，以自动的方式使用自动在线命令进行配置。

(Slave)模式节点的Redis数据库配置。

Android手机用Redis (Slave) 数据库的 /data/data/com.termux/files/usr/etc/redis.conf 文件的设置。

在文件中添加以下更改或指令，保存更改并启动Redis服务器。

首先，找到并删除 slaveof 行中的#字符。这条指令取你用来安全地联系Redis主服务器的IP地址和端口，用空格隔开。默认情况下，Redis服务器在本地接口的6379上监听，但每一种网络安全方法都会以某种方式改变其他人的默认。

您使用的值将取决于您用来保护网络流量的方法。

隔离网络：使用隔离网络的IP地址和主服务器的Redis端口(6379)(例如，简单 IP_address 6379的slave)。

stunnel或spiped：使用本地接口(127.0.0.1)和配置的端口来隧道流量。

PeerVPN：使用主服务器的IP VPN地址和正常的Redis端口。

将军会改变它。

ip _contact_server的slave 主站_contact_port。

例如： slaveof 192.168.1.69 6379。

masterauth 您的网络主密码。

例如： masterauth sdfssdfs12WqE34Rfghtdfd。

requirepass your_network_slave_password。

例如： requirepass asdsjdsh34sds67sdFGbbnh。

用以下命令从Termux终端保存并运行服务器。

```
$ redis-server redis.conf
```

执行后我们可以看到它与Windows 10服务器（Master）的同步情况。

配置文件redis.conf \$ redis-server redis.conf

安装节点同步管理工具。我们继续安装SyncThingManager。

首先，我们安装SyncThingManager正常工作所需要的东西。

```
$ apt install Python
```

```
$ pip3 安装 -upgrade pip
```

```
$ npm install syncthingmanager
```

SyncThingManager工具将帮助我们自动同步"点对点"，而不是手动同步新节点和现有节点。

```
$ apt install python
```

```
$ pip3 install --upgrade pip
```

```
$ pip3 install syncthingmanager
```

17. Ambientes Blockly (App Inventor, AppyBuilder和Thunkable)。

App Inventor是谷歌实验室创建的软件开发环境，用于构建Android操作系统的应用程序。用户可以，直观地，从一组基本工具，链接一系列的块来创建应用程序。该系统是免费的，可以方便地从网上下载。使用App Inventor创建的应用程序非常容易创建，因为不需要任何编程语言知识。

目前所有使用Blockly技术的环境，如AppyBuilder和Thunkable等都有他们的免费版本，他们的使用方式可以通过互联网在他们不同的站点，也可以在家里安装。

组成Mini BloclyChain架构的块已经在App inventor和AppyBuilder中进行了测试，但由于它们的代码优化，应该可以在其他平台上使用。

在线版本。

应用发明家。

<https://appinventor.mit.edu/>

AppyBuilder。

<http://appybuilder.com/>

可通关。

<https://thunkable.com/>

要安装在您的计算机（PC）上的版本。

<https://sites.google.com/site/aprendeappinventor/instala-app-inventor>

为Blockly区块的开发者提供环境。

<https://editor.appybuilder.com/login.php>

18. 什么是量子证明 (PQu) ?

PoQu。- "量子证明"是为Mini BlocklyChain开发的一个共识算法，这个测试是工作测试 (PoW) 的一个变种，工作原理如下。

启动时的"量子测试" (PoQu) 与"工作测试" (PoW) 的算法相同，是基于将设备 (PC、服务器、平板电脑或手机) 的处理器投入工作，以获得一串被称为"哈希"的数学难题的字符。

请记住，"哈希"是一种算法或数学过程，当引入一个短语或某种类型的数字信息，如文本文件，程序，图像，视频，声音或其他不同类型的数字信息，给我们作为一个字母数字字符，代表数字签名，代表它在一个独特的和不可重复的方式的数据。哈希算法是单向的，这意味着当你输入一个数据获得它的签名"哈希"时，它的逆向过程无法进行，有了签名"哈希"我们就不能知道获得了什么信息这个特性给我们处理我们在互联网上发送的信息带来了安全优势。想象一下，通过非安全渠道发送任何一种信息，并附上各自的"源哈希"，接收者在接收信息时，可以得到所接收信息的"哈希"，我们称之为"目的哈希"，并与"源哈希"进行核对，如果两个"哈希"相同，我们就可以确认信息在发送的渠道中没有被改变，这只是目前这种信息安全过程中使用的一个例子。

目前，有不同类型的算法或哈希过程，其安全级别不同。最常用或已知的有：MD5、SHA256和SHA512。

SHA256的例子。

我们有如下的连锁或句子。"Mini BlocklyChain是模块化的。

如果我们对前一个字符串应用SHA256哈希，就会得到下一个哈希。

f41af7e61c3b02fdd5e5c612302b62a2dd52fcb38f9of97cb2afd827e8804db8

上面的字母数字字符串是代表上面例子中的句子的签名。

更多的例子我们可以用互联网上的网站。

<https://emn178.github.io/online-tools/sha256.html>

在"测试工作"(PoW)算法的情况下， 它的工作原理是利用计算能力来获得一个预定义的哈希值。

让我们想象一下， 我们之前的"哈希"是从"迷你BlocklyChain是模块化的"链中提取的。

f41af7e61c3b02fdd5e5c612302b62a2dd52fcb38f9of97cb2afd827e8804db8

对这个"哈希"， 在它的开头， 我们把难度的参数， 就是简单地把零"0"的开头， 也就是说， 如果我们说难度是4， 它将有"**0000**"+"哈希"， 我们将它称为"种子哈希"

0000 f41af7e61c3b02fdd5e5c612302b62a2dd52fcb38f9de97cb2afd827e8804db8

现在考虑到我们知道的输入信息是字符串："Mini BlocklyChain是模块化的"， 我们在字符串的末尾添加一个从0开始的数字"0"， 然后我们取出它的哈希值， 我们称它为"哈希nonce"。

f41af7e61c3b02fdd5e5c612302b62a2dd52fcb38f9de97cb2afd827e8804db80

我们得到了哈希nonce。

7529f3ad273fc8a9eff12183f8d6f886821900750bb6b59c1504924dfd85a7c8。

然后， 我们对新的"哈希nonce"和"哈希种子"进行比较， 如果它们相等， 首先发现相等的节点将赢得处理当前事务的执行。我们可以看到， 这个过程是基于设备的概率和计算强度的， 它给"工作证明"测试提供了所有节点的共识公平。

如果"种子哈希"与"哈希非奇"不一致， 则难度增加1， 再去掉"哈希非奇"， 被增加的数字称为"非奇"数， 它与"种子哈希"进行比较， 直到它们一致或相同。

我们可以看到， 数字"nonce"或增加是将有助于获得平等的"哈希"。

在"工作测试"(PoW)算法的基础上，量子测试(PoQu)算法的基础是像PoW那样获得数字"nonce"，并使用1到5的最低级别难度，这服务于移动设备获得成为候选人的权利，以赢得共识。

量子测试(PoQu)，当手机完成最低PoW，并赢得通关，在QRNG系统中获得一个概率数时，就会激活。

QRNG(Quantum Random Number Generator)是量子随机数生成器，这个基于量子力学生成真实随机数的系统是目前最安全的生成此类数字的系统。详见附件"OpenQbit的量子计算"。

Mini BlocklyChain可以实现最小PoW和PoQu两种特许类型。

PoQu测试的基础是获得数字"nonce"这个数字在PoQu测试中被称为"Magic Number"有了这个数字，"Peer to Peer"系统会确认这个数字是否正确，然后用QRNG服务器池获得一个随机数。这个随机数将被登记在所有节点中，将创建一个包含(**(节点总和/2)+1的列表**)，并从这个列表中选择概率最高的一个成为共识(PoQu)的赢家候选人，这一个将执行当前的事务队列。

PoQu算法还使用了NIST（美国国家标准与技术研究所）的测试来保证QRNG中的随机数是真正的随机数。

<https://nvlpubs.nist.gov/nistpubs/legacy/sp/nistspecialpublication800-22r1a.pdf>

在Mini BlocklyChain中，我们实现了一个PoW的块和一个PoQu的块。

这些区块使用一种类型的哈希：免费使用的SHA256，商业使用的你有一个SHA512和其他类型的哈希根据需要。

关于HASH概念的更多细节请见：

https://es.wikipedia.org/wiki/Funcion_hash

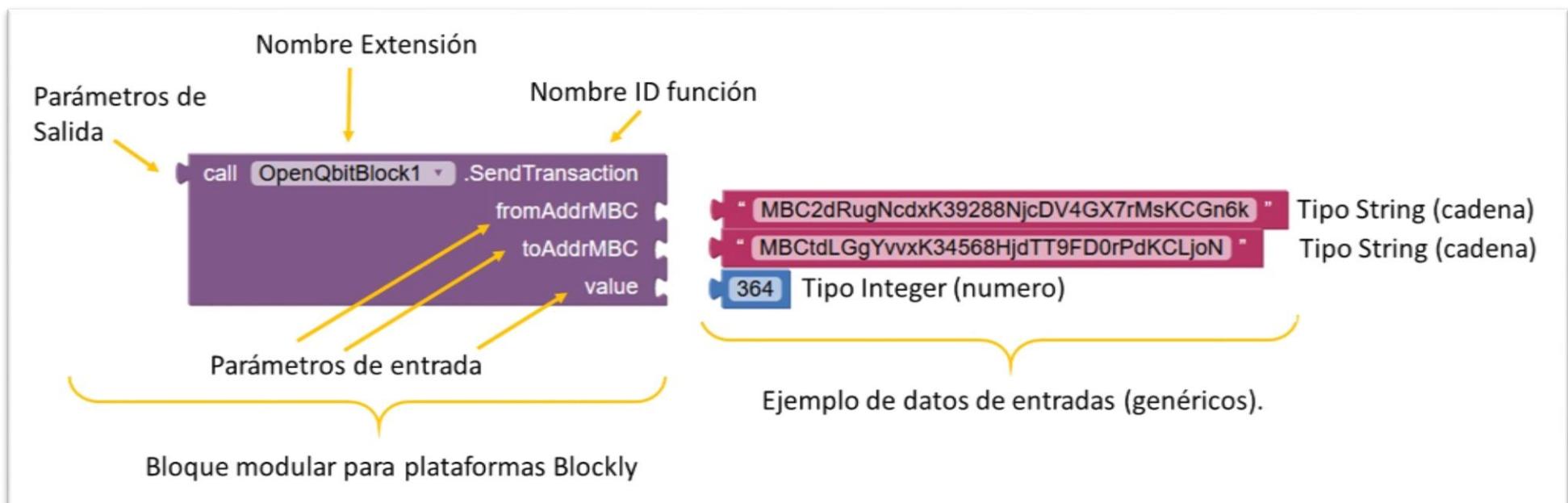
注意：手机中使用的工作测试（PoW）只能使用最高难度5，因为这些设备的数学处理不像服务器或PC那样是专用的。我们只使用PoW算法来获得您进入量子随机数生成器（QRNG）系统的机会或权限，并与之执行量子随机数生成器（PoQu）算法。

在手机上不要使用最高难度5，因为系统可能会锁定，无法正常响应。

19. 迷你BlocklyChain中区块的定义和用法

我们先来解释一下所有区块会有的数据分布，它们的使用语法和配置。

在下面的例子中，我们可以看到一个模块块及其输入和输出参数，以及输入数据的类型，这些数据的类型可以是String（字符串）或Integer（整数或十进制）。我们将展示如何使用它，并配置它以使其正常运行。



每一个模块块都会有它的描述，并会被命名，如果它有任何强制性或可选性的依赖其他模块作为输入参数，集成过程将被公布。

块在预定义数组中创建一个临时字符串列表，内部称为"链"-- (AddHash) 。

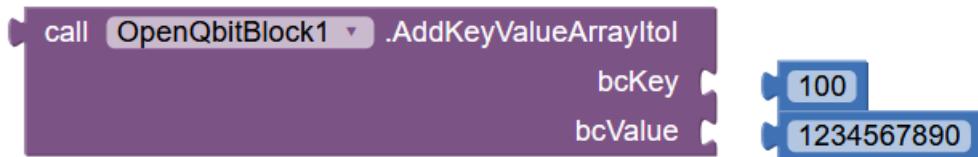


输入参数：块<字符串>

输出参数：不适用。

描述：用于在内部称为"链"的预定义数组中存储一个临时的哈希或字符串数组的块。

创建键值数组的块（整数-整数）-- (AddKeyValueArrayItol) 。

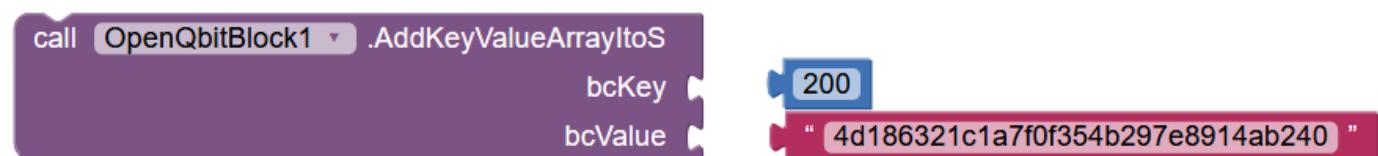


输入参数：bcKey <Integer>, bcValue <Integer>。

输出参数：返回输入时输入的数值。

说明：这是一个临时的键值排列，它的内部名称为"itol_UTXO"，这对处理UTXO交易很有用。

创建键值数组的块(Integer-String) - (AddKeyValueArrayItos)

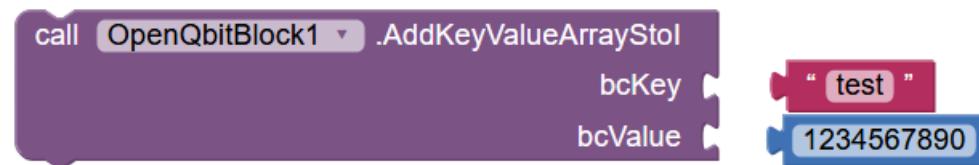


输入参数：bcKey <Integer>, bcValue <String>。

输出参数：返回输入时输入的数值。

说明：这是一个临时的键值排列，它的内部名称为"ltoS_UTXO"，这对处理UTXO交易很有用。

创建键值数组的块(String-Integer) - (AddKeyValueArrayItoi)



输入参数：bcKey <String>, bcValue <Integer>。

输出参数：返回输入时输入的数值。

说明：它是一个临时的键值排列，它被预定义为内部名称"Stoi_UTXO"，这对处理UTXO交易很有用。

创建键值数组的块(String-String) - (AddKeyValueArrayStoS)



输入参数：bcKey <String>, bcValue <String>。

输出参数：返回输入时输入的数值。

说明：它是一个临时的键值排列，它被预定义为内部名称"StoS_UTXO"，这对处理UTXO交易很有用。

用于生成十进制随机量子数的块 - (ApiGetQRNGdecimal)



输入参数: qty < Integer>

输出参数 : 给出输入的随机量子小数的数量"qty", 输入的数字在0和1的范围内, 以 JSON格式。

例如 :

```
qty = 5; output: {"result": [0.5843012986202495, 0.7746497687824652, 0.05951126805960929, 0.1986079055812694, 0.03689783439899279]}
```

描述 : 量子随机数生成器 (QRNG) API

用于生成十进制随机量子数的块 - (ApiGetQRNGdecimal)



输入参数 : 数量<整数>, 最小<整数>, 最大<最大>。

输出参数 : 给出输入的随机量子整数的数量"qty", 这些数字在最小和最大的范围内, 采用JSON格式。

例如 :

```
qty = 8, min = 1, max = 100; output: {"result": [3, 53, 11, 2, 66, 44, 9, 78]}
```

描述 : 量子随机数生成器 (QRNG) API

字符串的散列块"SHA256"。 (ApplySha256)。



输入参数：输入<字符串>。

输出参数：计算字符串的"SHA256"哈希值。

例如：

输入="Mini BlocklyChain是模块化的"

产出：F41af7e61c3b02fdd5e5c612302b62a2dd52fcb38f9de97cb2af827e8804db8

说明：删除哈希值"SHA256"的功能。Sha或SHA指的是：安全哈希算法，是美国国家安全部局设计的一套哈希函数。SHA256采用256位算法。

块来清理预定义的内部数组"链"（ClearBlockList）。



输入和输出参数：不适用

说明：删除所有具有内部时空排列"链"的元素。

块清理预定义的内部数组（Integer-Integer）--（ClearItol）。



输入和输出参数：不适用

说明：删除所有内部临时排列为"Itol_UTXO"的元素。

清理预定义内部数组的块（Integer-String）-（ClearItoS）。



输入和输出参数：不适用

说明：删除所有内部临时排列为"ItoS_UTXO"的元素。

块来清理预定义的内部数组（String-Integer） - （ClearStol）。

call OpenQbitBlock1 .ClearStol

输入和输出参数：不适用

说明：删除所有内部临时排列方式为"Stol_UTXO"的元素。

块来清理预定义的内部数组（String-String） - （ClearStoS）。

call OpenQbitBlock1 .ClearStoS

输入和输出参数：不适用

说明：删除所有内部临时排列方式为"StoS_UTXO"的元素。

块将字符串解码为Base10。（DecodeBase58）

call OpenQbitBlock1 .DecodeBase58

input

“oBRN8Aj67yJsbRGHfNSF9PTdZYGyVWrwMW7mTX”

输入参数：输入<字符串>。

输出参数：它提供的是块中使用的原始字符串（EncodeBase58）。

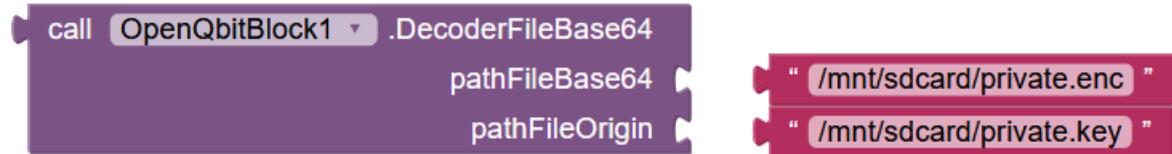
例如：

输入=oBRN8Aj67andJsbRGHfNSF9PTdZYGandVWrwMW7mTX。

输出："Mini BlocklyChain是模块化的"。

描述：将Base58字符串转换为块中给出的原始文本(EncodeBase58)

用Base64算法（DecoderFileBase64）对文件进行解码的块。



输入参数：pathFileBase64 <字符串>, pathFileOrigin <字符串>。

输出参数：输入块中的源文件（EncoderFileBase64）。

描述：Base64文件被转换为插入块中的原始文件（EncoderFileBase64）。

了解预定义内部数组"链"是否为空的块。(EmptyBlockList)



输入参数：不适用。

输出参数：如果为空则返回"True", 如果有数据则返回"False"。

描述：用于询问预定义临时内部数组"chain"是否有元素的块。

块将字符串编码为Base58。(EncodeBase58)。



输入参数：输入<字符串>。

输出参数：它提供的是块中使用的原始字符串（EncodeBase58）。

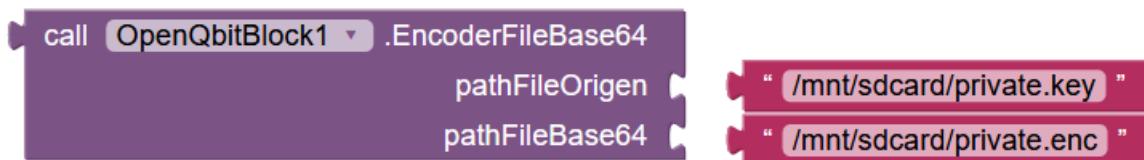
例如：

输入="Mini BlocklyChain是模块化的"。

Output: oBRN8Aj67yJsbRGhfNSF9PTdZYGyVWrwMW7mTX

说明：在Base58中把乳链转换为链子。Base58算法是一组用于将大整数表示为字母数字文本的二进制转文本编码方案，由中本聪推出，用于比特币。

用Base64算法（EncoderFileBase64）对文件进行编码的块。



输入参数：pathFileOrigin <String>, pathFileBase64 <String>。

输出参数：Base64编码文件。

描述：将任何格式的源文件转换为Base64文件。文件名可以是任意的，由用户选择。

用户地址的区块生成器（GenerateAddrBitcoin）。



强制性单位：块（ApiGetQRNGinteger）。

依赖项（可选）：OpenQbitFileEncryption扩展，OpenQbitFileDecryption扩展和OpenQbitSQLite扩展。

输入参数：qrng <强制依赖性>。

输出参数：34个字符的字母数字交易地址和keyStore使用地址。

说明：为用户和私钥生成器（用于发送交易的数字签名）和公钥（用于进行交易的公共地址）创建一个新的比特币通用交易地址的块。这个密钥生成器基本上是用于数字钱包或俗称"钱包"的地址生成器。

该块与量子随机数生成器（QRNG）块配合使用，两个输出参数必须插入KeyStore。

KeyStore是一个存储十六进制格式私钥的数据库。

存储在KeyStore中的十六进制地址的例子。

024C8E05018319ARD4BB04E184C307BFF115976A05F974C7D945B5151E490ERD。

这个基础只由本地用户使用，并对信息进行加密，这个过程是通过使用OpenQbitSQLite扩展和OpenQbitAESSEncryption和OpenQbitAESDecryption信息加密扩展。

要创建一个KeyStore，请参见"创建一个KeyStore"附录。生成的地址使用相同的比特币地址算法，初始比特币地址标识符为"1"。

区块（GenerateAddrBitcoin）生成的地址是34个字母数字字符，由33个字母数字字符和1个比特币标识符组成，具体如下。

12dRugNcdxK39288NjcDV4GX7rMsKCGn6k.

用户地址生成块（GenerateAddrEthereum）。



强制性依赖性：在以下网站获取令牌：<https://accounts.blockcypher.com/signup>。

依赖项（可选）：OpenQbitFileEncription扩展，OpenQbitFileDecryption扩展和OpenQbitSQLite扩展。

输入参数：token<强制性依赖性 - 字符串>。

输出参数：40个字母数字字符的交易地址不包括初始Ethereum指标"0x"，这将给我们提供42个字符的通用地址。它还返回公钥和私钥。

例如：

{

"**私人**"："227ac59f480131272003c2d723a7795ebd3580acaab62b5c537989e2ce4e08ef"。

"**公共**"：

"04e2d55ebccd32a7384e096df559cc36b856c64a16e5b402e10585dc3ea055672aafa84df8a859531570a650a8ab1e7a22949100efa1aa5f072c035551cac1ce"。

"**地址**"："14e150399b0399f787b4d6fe30d8b251375f0d66"}。

描述：为用户和私钥生成器（发送交易的数字签名）和公钥（进行交易的公共地址）创建一个新的交易地址的块。这个密钥生成器是一个外部的API，你必须要有Wifi或者手机连接，它基本上是地址生成器，可以在数字货币钱包或者俗称"钱包"中使用。

您可以创建一个KeyStore，是一个以十六进制格式存储私钥的数据库，参见附录"KeyStore创建"。

存储在KeyStore中的十六进制地址的例子。

0x14e150399b0399f787b4d6fe30d8b251375f0d66

私钥只由本地用户使用，并对信息进行加密，这个过程是通过使用OpenQbitSQLite扩展和OpenQbitAESEncryption和OpenQbitAESDecryption信息加密扩展。

用户地址生成块（GenerateAddrMiniBlocklyChain）。

call OpenQbitBlock1 .GenerateAddrMiniBlocklyChain
qrng

强制性单 位：块（
ApiGetQRNGinteger）。

依赖项（可选）：OpenQbitFileEncription扩展，OpenQbitFileDecryption扩展和
OpenQbitSQLite扩展。

输入参数：qrng <强制依赖性>。

输出参数：36个字母数字字符交易地址和keyStore使用地址。复核块法（PairKeysMBC
）。

描述：为用户和私钥生成器（发送交易的数字签名）和公钥（进行交易的公共地址
）创建一个新的交易地址的块。这个密钥生成器基本上是地址生成器，可以在数字钱
包或者俗称“钱包”中使用。

该块与量子随机数生成器（QRNG）块配合使用，两个输出参数必须插入KeyStore。

KeyStore是一个存储十六进制格式私钥的数据库。

存储在KeyStore中的十六进制地址的例子。

024C8E05018319ARD4BB04E184C307BFF115976A05F974C7D945B5151E490ERD。

这个基础只由本地用户使用，并对信息进行加密，这个过程是通过使用OpenQbitSQLite扩展和OpenQbitAESSEncryption和OpenQbitAESDecryption信息加密扩展。

要创建一个KeyStore，请参见“创建一个KeyStore”附录。生成的地址使用相同的比特币地址算法，唯一不同的是将原来是“1”或“3”的比特币地址标识符改为Mini BlocklyChain标识符“MBC”。

块（GenerateAddrMiniBlocklyChain）生成的地址为36个字母数字字符，由33个字母数字字符和3个Mini BlocklyChain标识符大写字母“MBC”组成，具体如下。

Mbc2dRugNcdxK39288NjcDV4GX7rMsKCGn6k。

块方法（PairKeysMBC）是块输出（GenerateAddrMiniBlocklyChain）。



输入参数：不适用。

输出参数：addressMBC <String>, privateKeyHex <String>。

说明：交付迷你BlocklyChain格式的公共用户地址和十六进制格式的私钥。

例如：

地址MBC：MBC2dRugNcdxK39288NjcDV4GX7rMsKCGn6k。

privateKeyHex:

024C8E05018319ARD4BB04E184C307BFF115976A05F974C7D945B5151E490ERD。

用户地址的块生成器（GenerateKeyPair）。

```
call OpenQbitBlock1 .GenerateKeyPair  
qrng
```

强制性单位：块（ApiGetQRNGinteger）。

依赖项（可选）：OpenQbitFileEncryption扩展，OpenQbitFileDecryption扩展和OpenQbitSQLite扩展。

输入参数：qrng <强制依赖性>。

输出参数：提供本地用户的公钥和主密钥。

描述：使用ECC算法⁽¹⁾和十六进制格式生成公钥和主钥。

⁽¹⁾ 椭圆曲线密码学（ECC）是基于椭圆曲线数学的非对称或公钥密码学的一种变体。

块来签署交易（GenerateSignature）。

```
call OpenQbitBlock1 .GenerateSignature
```

所需依赖：Block（GenerateKeyPais）、Block（SenderLoadKeyPair）、Block（RecipientLoadKeyPair）。在使用前，先为这些区块作序。

输入参数：<强制检查点依赖性>。

输出参数：无输出。

说明：它从发送方生成一个数字签名，应用于交易中发送给接收方的资产，这个签名将在交易被网络的某个节点处理时得到确认，有了这个签名，Mini BlocklyChain系统就能确保发送的资产没有被修改，并且它符合发送方和接收方的关系，没有被转移或应用到另一个数字地址。

块来获取用户的资产总余额 (GetBalance) 。

```
call OpenQbitBlock1 .GetBalance  
fromAddrMBC " MBC2dRugNcdxK39288NjcDV4GX7rMsKCGn6k "
```

强制性依赖 : Block (ConnectorTransactionTail) 。

输入参数 : fromTransTail <数组字符串>, <强制性板块依赖性>。

输出参数 : 核对每笔交易的资产支出入库和出库情况。

说明 : 检查余额, 以批准用户是否有资产发送, 或者他收到的资产是否被添加到他的余额中。

块从手机上获取q22。(GetDeviceID)

```
call OpenQbitBlock1 .GetDeviceID
```

输入参数 : 不适用

出站参数 : 它传递的是手机的内部标识符。

说明 : 提供每个设备唯一的IMEI手机标识符。

从字符串中删除RIPEMD-160哈希值的块。(Ripemd-160)

```
call OpenQbitBlock1 .GetHashRipemd160  
str " Mini BlocklyChain es modular "
```

输入参数 : str <字符串>

输出参数 : 计算字符串的"RIPEMD-160"哈希值。

例如：

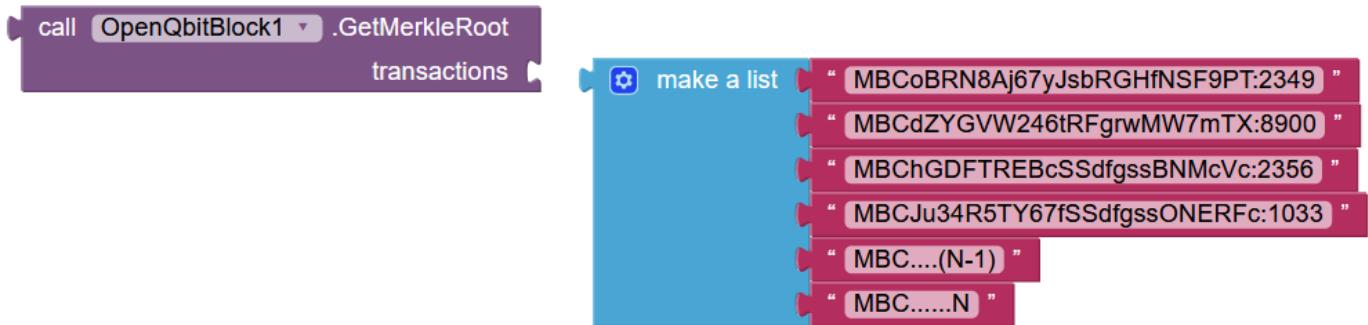
输入="Mini BlocklyChain是模块化的"

输出：ae29436e4b8ea8ed6143f3f92380dfa2f4f47336。

说明：获得"RIPEMD-160"散列。这个哈希值用于生成比特币和迷你BlocklyChain的有效地址的过程中。

RIPEMD-160 (RACE Integrity Primitives Evaluation Message Digest的缩写) 是一种160位的消息摘要算法 (和加密哈希函数) 。

块来计算梅克勒的树。(GetMerkleRoot)



输入参数：交易 <数组字符串>

输出参数：它提供的哈希类型为"SHA256"。

例如：

输入=事务队列，是所有待处理事务的排列。

产出：b4a44c42b6070825f763cd118d6ab49a8e80bbb7cdc0225064f8e042b94196bd。

描述：使用Merkle树算法获得"SHA256"哈希值。

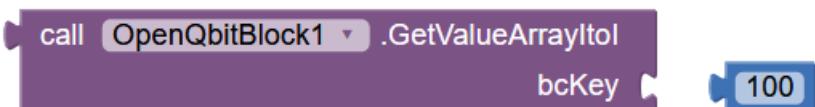
Merkle哈希树是一种二进制或非二进制的数据树结构，其中每个不是叶子的节点都被标记为其子节点的标签或值（对叶子节点而言）的连接的哈希值。这些都是哈希列表和哈希字符串的概括。

它允许将大量的独立数据链接到一个哈希值，即树根节点的哈希值。这为验证大型数据结构的内容提供了一种安全有效的方法。在其实际应用中，根节点的哈希值通常经过签名的，以保证其完整性，验证是完全可靠的。证明一个叶子节点是给定哈希树的一部分，需要的数据量与树中节点数量的对数成正比。

目前，Merkle树的主要用途是对点对点网络中从其他对等体接收到的数据块进行安全处理，保证接收到的数据块不被破坏，不被更改。

它用于验证拥有待处理交易的队列是否被修改，并确保其完整性，以便分散在MiniblocklyChain网络的所有节点中。所有的节点都必须执行这个算法，以确保每个将被应用的交易的完整性。

从预定义数组"!tol_UTXO"(GetValueArray!tol)中获取一个值的块。

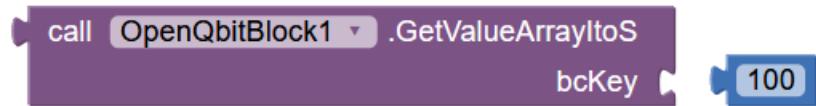


输入参数：`bcKey <Integer>`。

输出参数：返回以给定数字为输入的标签中存储的<整数>值。

说明：这是一个对临时键值排列的请求，它是用内部名称"!tol_UTXO"预定义的，这对处理UTXO交易很有用。

从预定义数组"ItoS_UTXO"(GetValueArrayItoS)中获取一个值的块。



输入参数：**bcKey** <Integer>。

输出参数：返回以给定数字为输入的标签中存储的<字符串>值。

说明：这是一个对临时键值排列的请求，它是用内部名称"ItoS_UTXO"预定义的，这对处理UTXO交易很有用。

从预定义数组"StoI_UTXO"(GetValueArrayStoI)中获取一个值的块。



输入参数：**bcKey** <字符串>。

输出参数：返回以给定名称为输入的标签中存储的<Integer>值。

说明：这是一个对临时键值安排的请求，它是用内部名称"StoI_UTXO"预定义的，这对处理UTXO交易很有用。

从预定义数组"StoS_UTXO"(GetValueArrayStoS)中获取一个值的块。



输入参数：**bcKey** <字符串>。

输出参数：返回以给定名称为输入的标签中存储的<String>值。

说明：这是一个对临时键值安排的请求，它是用内部名称"StoS_UTXO"预定义的，这对处理UTXO交易很有用。

区块链的区块验证器。(IsChainValid)

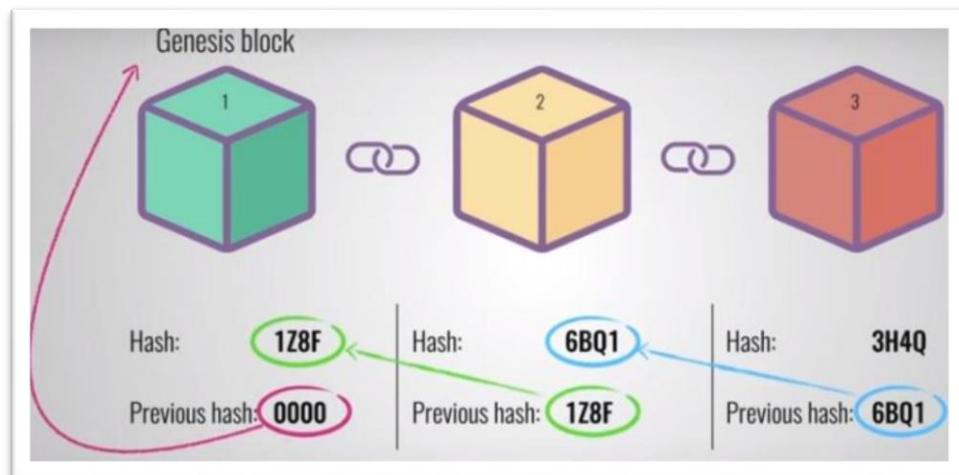
call OpenQbitBlock1 .IsChainValid

所需依赖：Block(GenerateKeyPairs)、Block(SenderLoadKeyPair)、
Block(RecipientLoadKeyPair)、Block(GenerateSignature)。在使用前，先为这些区块作序。

输入参数：不适用。

输出参数：如果区块字符串的验证正确，则交付"True"，如果验证失败，则交付"False"。

说明：它为我们提供了对之前插入区块链系统的组件的验证，这个验证是整个系统中最重要的。区块链的验证是如何进行的，或者说一般人都知道区块链（BlockChain）。它是每个系统的核心部分。



在上一张图中我们可以看到，存储系统中增加的每一个块都是通过哈希算法与前一个块相关联的。

例如，在创建一个新的区块添加时，取新信息区块的哈希值+之前的哈希值就可以得到代表新信息的哈希值。有了这种类型的链接，区块链存储中任何最小的修改都会被检测到，这使得数据的安全性非常高和有效。

下面是SQLite数据库中存储一串块的例子，我们会看到之前的哈希如何与插入的最后一个块（最后一个处理的事务队列）的新哈希相连。"prevhash"和"newhash"两列中的每个哈希值都代表了当时处理的交易队列的信息。

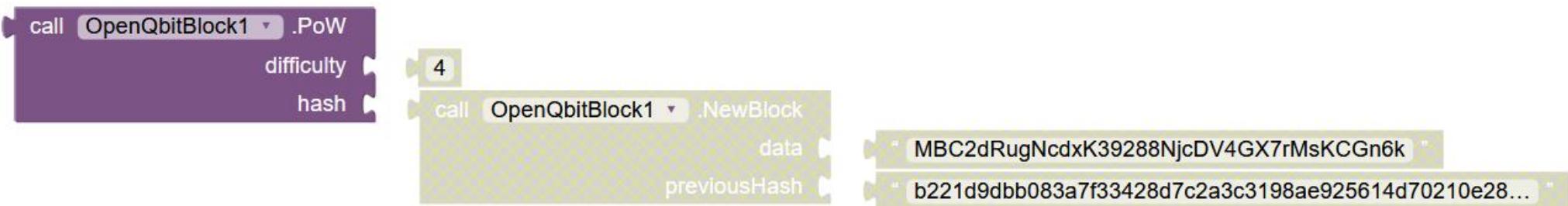
The screenshot shows the SQLite Expert Personal 5.3 interface. The database is named 'miniblock' and the table is 'nblock'. The table has columns: rowid, id, prevhash, newhash, opera, trans, and balan. The data is as follows:

rowid	id	prevhash	newhash	opera	trans	balan
1	1	0767c864cef0334f27473902eb9868e7	bdc9065d20a4cd037bb1a7538486403e	2	0	0
2	2	bdc9065d20a4cd037bb1a7538486403e	6619f4809d73a267a4b9ac554bb4523a	1	5	5
3	3	6619f4809d73a267a4b9ac554bb4523a	4d186321c1a7f0f354b297e8914ab240	1	5	5

Two yellow arrows point from the 'prevhash' column of row 2 to the 'newhash' column of row 3, illustrating the chain connection.

前一个哈希与新的哈希有关。

块进行PoW工作测试并开采新的区块。(Pow)



强制性依赖：NewBlock。在使用本块之前，先为其作序。

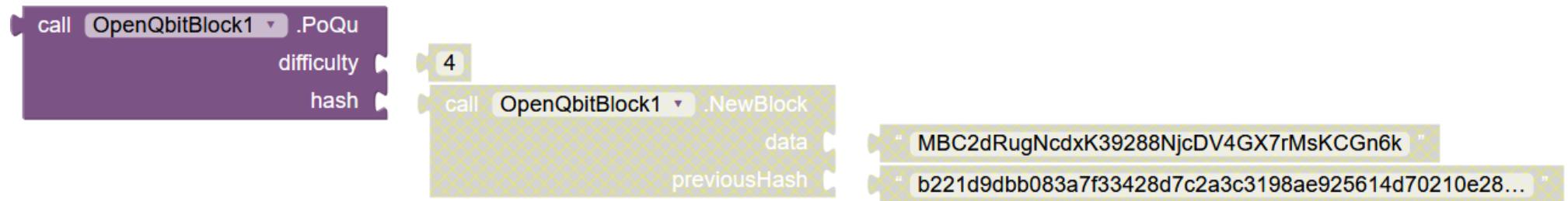
输入参数：难度<整数>， 哈希<强制依赖性>。

输出参数：给出一个由输入参数难度中给定的#数字"0"组成的哈希值"SHA256"。

说明：这个区块执行的活动叫做"挖矿"一个新的区块就是我们之前所说的PoW (Proof of Work) 的过程，请看什么是量子证明 (PQu) 一节。

挖矿或运行PoW是一个概念，节点被赋予一个任务来解决一个数学难题。在Mini BlocklyChain的情况下，谁先解决了它，谁就有机会继续进行这个过程，以便被选中成为能够处理等待处理的交易队列的赢家。

块进行PoW工作测试并开采新的区块。(PoQu)



强制性依赖：**NewBlock**。在使用本块之前，先为其作序。

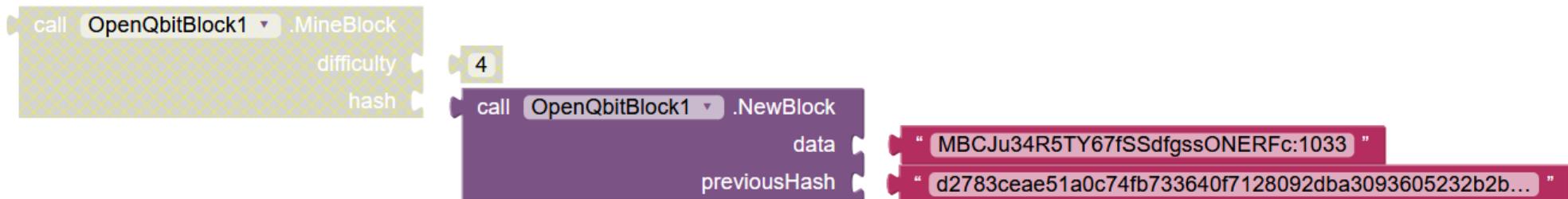
输入参数：难度<整数>，哈希<强制依赖性>。

输出参数：结果为数字"Magic Number"和随机量子数，在0和1的范围内的16位十进制类型，例如（0. 5843012986202495）和哈希"SHA256"组成的#数字"0"的输入参数难度。

说明：这个区块执行"挖掘"一个新区块的过程就是我们之前所说的PoW (Proof of Work) 的过程，但这个过程调用随机量子数的函数，计算后生成的数字"nonce"适合于满足难度级别，可以在最小1，最大5的范围内。请看章节什么是量子证明 (PQu-Proof Quantum) ？

挖矿或运行PoW或PoQu是一个概念，其中节点被赋予一个任务来解决一个数学难题。在任何区块链系统的情况下，首先解决它的节点获得了继续处理的机会，以便被选中成为能够处理等待处理的交易队列的赢家。

用于创建新的块（NewBlock）。



输入参数：data <String>, previousHash <String>。

输出参数：它提供的哈希值计算为：SHA256（数据（输入参数）+ previousHash（输入参数）+IMEI（内部参数）+MerkleRoot（内部参数））。

说明：该块执行创建一个新的待处理块。它作为输出给出一个由输入参数串组成的"SHA256"哈希值，在数据因每个系统设计不同而不同的情况下，前一个哈希值是基于前一个已经处理过的交易串的哈希值，并存储在Mini BlocklyChain区块串系统中。这两个是可变参数，有两个系统内部参数是固定的，保证了信息和系统的完整性，这两个内部参数是手机的唯一ID和正在处理的交易队列的梅克雷特树的哈希值。

节点本地 总交易量 的区块(NumberTransactions)

```
call OpenQbitBlock1 .NumberTransactions  
calculateNumTransactions
```

强制性依赖：块（AddKeyValueArrayStoS）。在使用本块之前，先为其作序。

输入参数：<强制依赖性>。

输出参数：返回该节点的本地交易总量。

说明：提供只适用于节点本地账户的交易总量。

读取二进制文件中收件人的地址。（RecipientLoadKeyValuePair）

```
call OpenQbitBlock1 .RecipientLoadKeyValuePair
```

输入参数：不适用。

输出参数：返回Base64格式的私钥和公钥。

描述：从二进制文件中获取收件人的私有地址和公共地址作为输入参数。

在内部临时排列的"链"中删除元素的块（RemoveHash）。

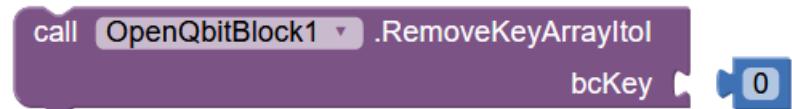
```
call OpenQbitBlock1 .RemoveHash  
block " test "
```

输入参数：块<字符串>。

输出参数：不适用。

描述：从二进制文件中获取收件人的私有地址和公共地址作为输入参数。

在内部临时排列中删除元素的块"ltoL_UTXO"(RemoveKeyArrayltol)

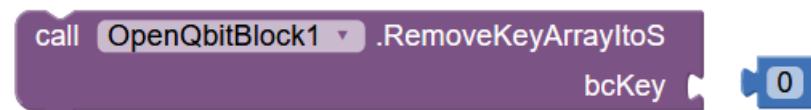


输入参数：bcKey < Integer>。

输出参数：不适用， 元素类型删除<整数>。

说明：删除与内部临时排列的数字标签"ltoL_UTXO"相关联的元素。

块删除内部临时排列"ltoS_UTXO"中的元素（RemoveKeyArrayltoS）。

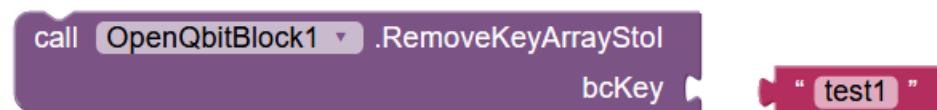


输入参数：bcKey < Integer>。

输出参数：不适用， 要删除的元素类型<字符串>。

说明：删除与内部临时排列的数字标签"ltoS_UTXO"相关联的元素。

用于删除内部临时安排"StoL_UTXO"中的元素的块(RemoveKeyArrayStoL)

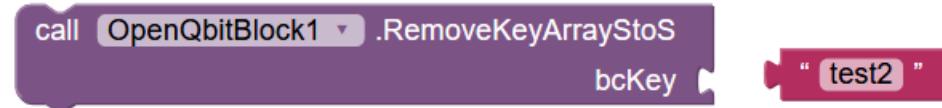


输入参数：bcKey <字符串>。

输出参数：不适用， 元素类型删除<整数>。

说明：删除与内部临时排列的数字标签"StoL_UTXO"相关联的元素。

用于删除内部临时安排"StoS_UTXO"中的元素的块(RemoveKeyArrayStoS)

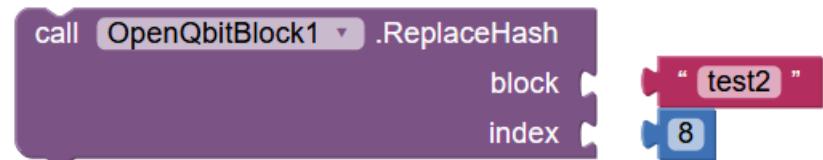


输入参数：bcKey <字符串>。

输出参数：不适用，要删除的元素类型<字符串>。

说明：删除内部临时安排标签"StoS_UTXO"的元素。

块来替换内部临时安排"链"的一个值（ReplaceHash）。



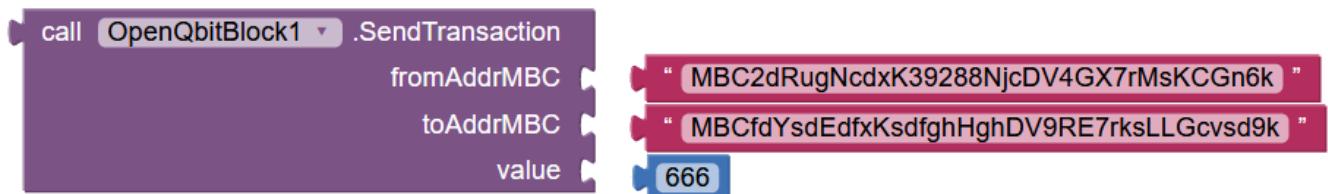
输入参数：块<字符串>, 索引<整数>。

输出参数：不适用。

说明：与索引"index"相关联的元素被替换为临时内部排列"chain"中的标签"block"的值

。

阻止开始（发送）新的交易（SendTrasaction）。



输入参数 : fromAddrMBC <字符串>, toAddrMBC <字符串>, value <Integer>。

输出参数 : 如果交易发送成功则返回值为"True", 如果没有发送成功则返回值为"False"。

描述 : 将发件人和收件人的地址转换为二进制格式的块, 并附加到待处理的交易队列中。

读取二进制文件中发件人地址的块。(SenderLoadKeyPair)

call OpenQbitBlock1 .SenderLoadKeyPair

输入参数 : 不适用。

输出参数 : 返回Base64格式的私钥和公钥。

描述 : 从二进制文件中获取发件人的私有地址和公共地址作为输入参数。

块获取临时排列"链"的元素编号 (SizeBlockList) 。

call OpenQbitBlock1 .SizeBlockList

输入参数 : 不适用。

输出参数 : 返回内部数组"链"的元素编号。

说明 : 获取内部数组"链"的元素编号。

获取临时安排"ltoL_UTXO"(Sizeltol)的元素编号的块。

call OpenQbitBlock1 .Sizeltol

输入参数 : 不适用。

输出参数：返回内部数组"ltoI_UTXO"的元素编号。

说明：获取内部数组"ltoI_UTXO"的元素编号。

获取临时安排"ltoS_UTXO"(SizeItoS)的元素编号的块。

call OpenQbitBlock1 .SizeItoS

输入参数：**不适用**。

输出参数：返回内部数组"ltoS_UTXO"的元素编号。

说明：获取内部数组"ltoS_UTXO"的元素编号。

获取临时安排"StoI_UTXO"(SizeStoI)元素编号的块。

call OpenQbitBlock1 .SizeStoI

输入参数：**不适用**。

输出参数：返回内部数组"StoI_UTXO"的元素编号。

Description: 获取内部数组 "StoI_UTXO" 的元素编号。

获取临时安排"StoS_UTXO"(SizeStoS)元素编号的块。

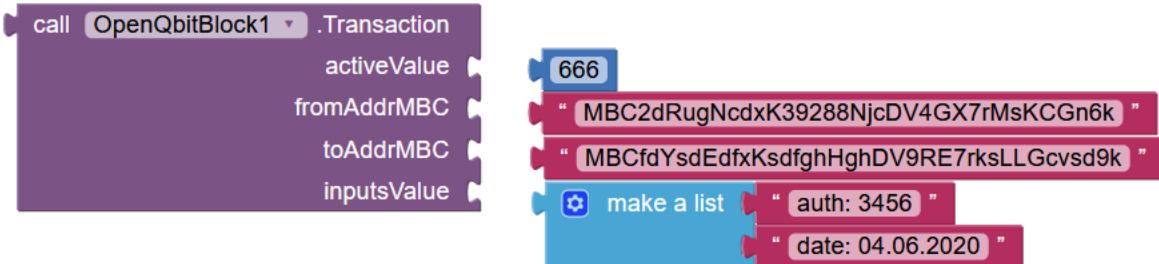
call OpenQbitBlock1 .SizeStoS

输入参数：**不适用**。

输出参数：返回内部数组"StoS_UTXO"的元素编号。

说明：获取内部数组"StoS_UTXO"的元素编号。

块创建附加值的交易（交易）。



输入参数：`activeValue < Integer>`, `fromAddrMBC < String>`, `toAddrMBC < String>`,
`inputValue < Array String>`。

输出参数：它给我们提供二进制格式的地址和`inpustValue`的值转换成一个字符串
"String", 并给我们`ActiveValue`值的哈希值"SHA256"。

说明：准备一个新的事务由节点处理。

迷你BlocklyChain地址验证块(ValidateAddMiniBlocklyChain)



输入参数：Mini BlocklyChain格式的用户地址。

输出参数：如果地址格式正确，返回"True"；如果地址无效，返回"False"。

说明：验证输入的Mini BlocklyChain地址是否正确，该块应用算法验证该地址是否是
使用Mini BlocklyChain系统中要使用的地址创建机制创建的。

比特币地址验证块。(ValidateAddBitcoin)

call OpenQbitBlock1 .ValidateAddressBitcoin

addr

" 12dRugNcdxK39288NjcDV4GX7rMsKCGn6k "

输入参数：**addr <字符串>**，比特币格式的用户地址（接受初始标识符为"1"的比特币地址，标识符为"3"的地址不适用）。

输出参数：如果地址格式正确，返回"True"；如果地址无效，返回"False"。

说明：验证输入的比特币地址是否正确，该块应用算法验证该地址是否是使用比特币系统中的地址创建机制创建的。

当前交易的数字签名验证块。（**VerifySignature**）。

call OpenQbitBlock1 .VerifySignature

输入参数：**不适用**。

输出参数：如果检查有效则返回"True"，如果检查无效则返回"False"。

说明：获取发送者在发送你要进行的交易过程中应该看到的数字签名的验证。在这个验证过程中，检查发送的源值在发送交易的通道中没有被改变，以及验证交易应适用的收件人。

20. 在SQLite数据库中使用块（MiniSQLite版本）。

在本节中，我们将看到如何使用区块来执行我们感兴趣的Mini BlocklyChain系统功能的两个主要操作，这就是将数据"INSERT"到区块链中并进行查询。

注意：SQLite数据库中的交易与迷你区块链系统中要应用的节点发送的交易不同。

数据库中的事务是基于CRUD(创建、读取、更新和删除)模型的，是指在SQLite数据库中只能用外部或内部数据执行的过程。在区块链系统中主要使用的是创建和读取的过程，为了安全被丢弃的更新或删除的过程和更多的地方，它是存储的区块链，使系统的整体安全。

另一方面，节点发送的交易指的是所有涉及使Mini BlocklyChain系统的成员（节点）之间发送某种类型的资产的动作的过程，这种类型的交易包括其应用的多样化的过程，这些过程可以是从数字地址的创建、数字签名、签名验证、SQLite数据库内的过程等。

我们先从SQLite数据库的块的定义和使用说起，MiniSQLite版这个版本只集成了8个块。你有一个完整的版本来操作SQLite数据库中的数据，然而，对于实际目的来说，Mini BlocklyChain系统与MiniSQLite版本已经足够了。

如果你想回顾所有的组件，它们的用途和描述请看附件"SQLite数据库的扩展块"。

MiniSQLite版本块。

在SQLite数据库中启动某种事务的块(BeginTransaction)

call OpenQbitQSQLite1 .BeginTransaction

法定单位：块(ImportDatabase)、块(OpenDatabase)。

输入参数：在<强制性依赖关系>之前使用。

输出参数：不适用，它在SQLite数据库中启动一个事务。

说明：在SQLite数据库中启动进程的块，必须先执行了数据库导入块（ImportDatabase）和数据库打开块（OpenDatabase）。

块在SQLite中进行Commit。(CommitTrasaction)

call OpenQbitQSQLite1 .CommitTransaction

所需的依赖关系：Block (BeginTransaction), Block (ImportDatabase), Block (OpenDatabase)

◦

输入参数：**在<强制性依赖关系>之前使用。**

输出参数：不适用，它在SQLite数据库中执行一个事务的提交。

说明：在SQLite数据库上启动**提交**进程的块，必须先执行了数据库导入块（**ImportDatabase**）和数据库打开块（**OpenDatabase**）。

用于关闭导入或导出的SQLite数据库的块(**CloseDatabase**)

call **OpenQbitQSQLite1** .CloseDatabase

法定单位：块(**ImportDatabase**)、块(**OpenDatabase**)。

输入参数：**在<强制性依赖关系>之前使用。**

输出参数：不适用，它关闭了一个SQLite数据库。

说明：关闭SQLite数据库的块，必须先执行了数据库导入块（**ImportDatabase**）和数据库打开块（**OpenDatabase**）。

块导出一个SQLite数据库（**ExportDatabase**）。

call **OpenQbitQSQLite1** .ExportDatabase
fileName " mbcExport.sqlite "

法定单位：块(**ImportDatabase**)、块(**OpenDatabase**)。

输入参数：**fileName <字符串>** 输入一个路径，在这个路径上可以找到已经用SQLite格式创建的数据库。

在<强制性依赖关系>之前使用。

输出参数：输出到SQLite数据库。

说明：导出SQLite数据库的块，必须先执行了数据库导入块（ImportDatabase）和数据库打开块（OpenDatabase）。

块导入SQLite数据库。(导入数据库)

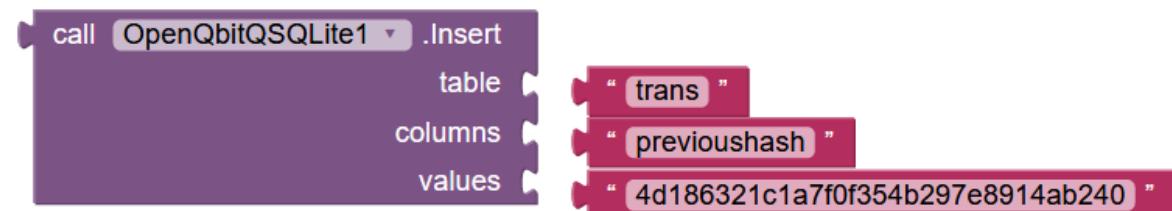


输入参数：fileName <字符串> 输入一个路径，在这个路径上可以找到已经用SQLite格式创建的数据库。

输出参数：启动SQLite数据库执行事务。

说明：在SQLite数据库中启动进程的块，必须先执行了数据库导入块（ImportDatabase）和数据库打开块（OpenDatabase）。

在SQLite数据库中插入数据的块。(插入)



法定单位：块(ImportDatabase)、块(OpenDatabase)。

输入参数：table < String> , columns < String> , values < String> , Use before < Mandatory dependence(s)>。

输出参数：在SQLite数据库中插入一个事务。

说明：将数据插入SQLite数据库的块，必须先执行了数据库导入块（ImportDatabase）和数据库打开块（OpenDatabase）。

块打开SQLite数据库（`OpenDatabase`）。

call `OpenQbitQSQLite1` .`OpenDatabase`

法定单位：块(进口数据库)。

输入参数：在<强制性依赖关系>之前使用。

输出参数：不适用，启动或打开SQLite数据库执行事务。

说明：启动SQLite数据库的块，前面必须有。

在SQLite中进行数据查询的块（`Execute`）。

call `OpenQbitQSQLite1` .`Execute`

sql

bindParams

法定单位：块(`ImportDatabase`)、块(`OpenDatabase`)。

输入参数：`sql < String >`, `bindParams < String >`, Use before < Mandatory Dependency(s)>。

输出参数：执行SQL语句，在SQLite数据库中创建一个事务。

说明：在SQLite数据库中执行SQL语句的块，必须先执行导入数据库的块（`ImportDatabase`）和打开数据库的块（`OpenDatabase`）。

21. 担保块的定义和使用；

在本次会议中，我们将回顾使用为我们提供安全级别的区块，以保存、验证和转移 Mini BlocklyChain 网络节点的交易。

安全区块基于以下扩展：

- I. OpenQbitAESEncryption 扩展。
- II. OpenQbitAES 解密扩展。
- III. OpenQbitAEToString 扩展。
- IV. OpenQbitEncDecData 扩展。
- V. OpenQbitFileHash 扩展。
- VI. OpenQbitRSA 扩展。
- VII. OpenQbitSSHClient 扩展(将需要)
- VIII. OpenQbitStringHash 扩展。

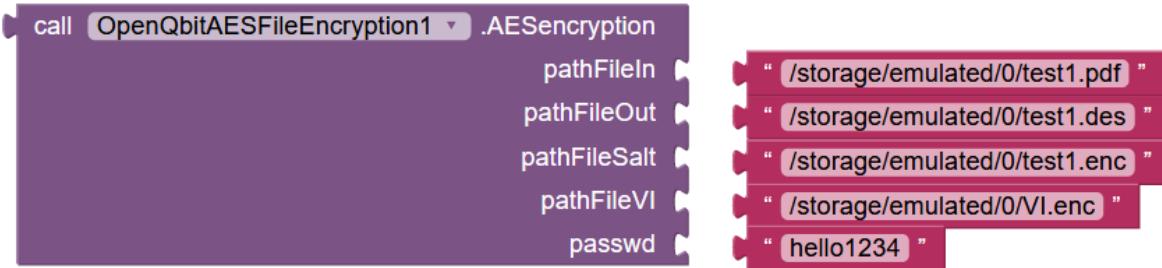
除了 OpenQbitSSHClient 扩展是强制性的，以上扩展在创建一个公共或私有的 Mini BlocklyChain 网络时是可选的。

然而，为了使您的交易系统安全，根据不同的业务情况，应酌情使用可选择的扩展。

例如，在使用哈希的情况下，我们可以使用一些算法选项，如 MD5、SHA1、SHA128、SHA256、SHA512 为一串字符，也可应用于任何类型的文件，这取决于每个用 Mini BlocklyChain 创建的系统的信息流。

OpenQbitAESEncryption 扩展。

阻止用AES安全加密文件。(AESEncryption)。



输入参数：`pathFileIn < String>`，`pathFileOut < String>`，`pathFile < String>`，`pathFileVI < String>` 和 `passwd < String>`。文件名是任意的，由各系统设计决定。

输出参数：输入参数`pathFileIn`中输入的AES加密文件。

说明：给我们三个输出文件的块，其中一个是已经被AES算法用256位密钥加密的源文件，另外两个文件是用来控制扩展名来解密和恢复原文件的。

OpenQbitAES解密扩展。

块来解密AES文件。(AESDecryption)。



输入参数：`pathFileIn < String>`，`pathFileOut < String>`，`pathFile < String>`，`pathFileVI < String>` 和 `passwd < String>`。文件的名称与区块（AESEncryption）中得到的结果相同。

输出参数：在输入参数`pathFileIn`中输入用AES解密的原始文件，在这种情况下，要解密文件就在`pathFileIn`中输入加密文件，在`pathFileOut`中就会得到原始文件（解密后）

◦

说明：在pathFileOut参数中给我们提供一个文件的块，这个文件将被AES算法用256位密钥解密后的输出。

OpenQbitAEToString扩展。

这个扩展是每个设备会话一次性使用的，也就是说，只有在设备没有重启（手机）的时候，它才会起作用，因为VI加密值是临时生成的。

注意：如果设备被重启，则无法恢复加密字符串。

字符串临时加密块(DecrypSecretText)



输入参数：secretText <字符串>

输出参数：用256位密钥的AES解密的原始字符串。

说明：给我们提供一串字符的块，是块中引入的输入参数（EncrypPlainText）。

字符串解码块(EncrypPlainText)



输入参数：plainText <字符串>

输出参数：使用256位密钥的AES加密字符串。

描述：提供一个使用256位数字密钥的AES加密的字母数字字符串的块。

OpenQbitEncDecData扩展。

通用数据库的专用加密块(**加密数据**)



输入参数：plainText<字符串>

输出参数：使用256位密钥的AES加密字符串。

描述：提供一个使用256位数字密钥的AES加密的字母数字字符串的块。

通用数据库的专用加密块(**解密数据**)



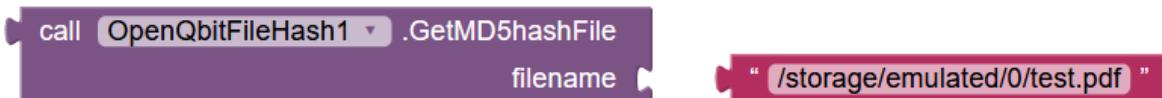
输入参数：plainText<字符串>

输出参数：使用256位密钥的AES加密字符串。

描述：提供一个使用256位数字密钥的AES加密的字母数字字符串的块。

OpenQbitFileHash扩展。

从文件中生成MD5的块（GetMD5hashFile）。

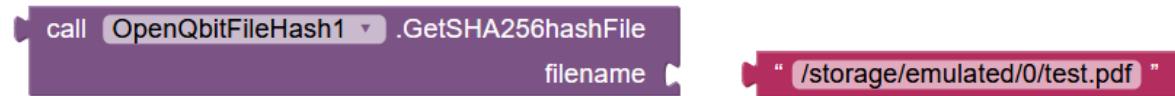


输入参数：文件名<字符串>。

输出参数：提供MD5哈希文件。

描述：用于创建输入参数中给出的文件的MD5哈希值的块。

从文件中生成SHA256的块(GetSHA256hashFile)

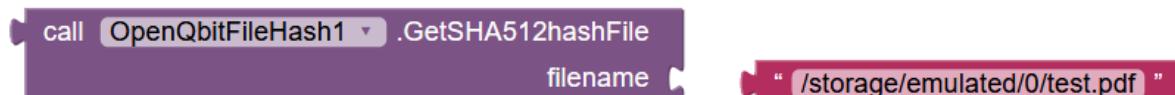


输入参数：**文件名**<字符串>。

输出参数：提供SHA256档案哈希值。

描述：用于创建输入参数中给出的文件的SHA256哈希值的块。

从文件中生成SHA512的块（GetSHA512hashFile）。

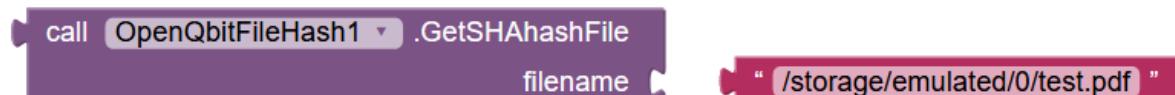


输入参数：**文件名**<字符串>。

输出参数：提供SHA256档案哈希值。

描述：用于创建输入参数中给出的文件的SHA256哈希值的块。

从文件中生成SHA1的块（GetSHA1hashFile）。



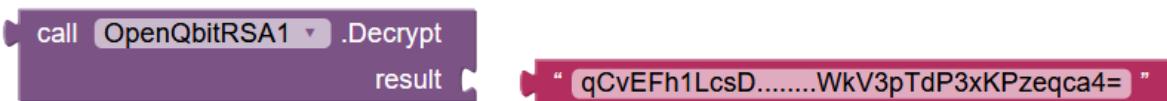
输入参数：**文件名**<字符串>。

输出参数：提供SHA1档案哈希值。

说明：用于创建输入参数中模子的SHA1哈希值的块。

OpenQbitRSA扩展。

用RSA解密字符串的块(解密)



需要的依赖关系：Block (Encrypt), Block (OpenFromDiskPrivateKey), Block (OpenFromDiskPublicKey).

输入参数：result <字符串>

输出参数：用RSA解码的字符串。

描述：给我们一个字母数字字符串的块，使用块中使用的大小的密钥 (GenKeyPair) 解密。

用RSA(加密)加密字符串的块。



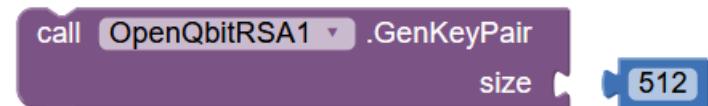
所需单位：Block (GenKeyPair), Block (SaveFromDiskPrivateKey), Block (SaveFromDiskPublicKey)

输入参数：普通<字符串>。

输出参数：RSA加密字符串。

描述：给我们一个字母数字字符串的块，使用块中使用的大小的密钥（GenKeyPair）解密。

用RSA解密字符串的块(解密)

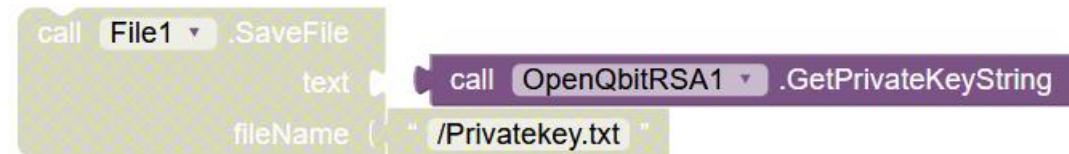


输入参数：size <Integer>

输出参数：不适用。

说明：根据选择的大小生成私钥和公钥的块。

获取私钥的区块(GetPrivateKeyString)



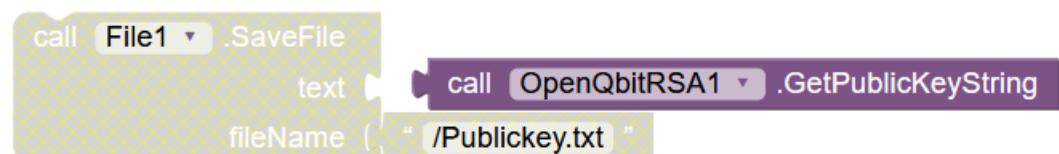
法定单位：区块(GenKeyPair), 区块(GenKeyPair), 区块(文件)

输入参数：不适用。

输出参数：带有RSA加密字符串的文件（私钥）。

描述：给我们提供一个字母数字字符串的块，代表使用块中使用的大小密钥（GenKeyPair）加密的私钥。

获取私钥的区块(GetPublicKeyString)



法定单位：区块(GenKeyValuePair), 区块(GenKeyValuePair), 区块(文件)

输入参数：**不适用。**

输出参数：带有RSA加密字符串的文件（公钥）。

描述：给我们一个字母数字字符串的块，代表使用块中使用的密钥大小加密的公钥（GenKeyValuePair）。

注意：在前面的块（GetPrivateKeyString）和（GetPublicKeyString）的依赖关系中，我们将使用"存储"托盘会话的App Inventor应用程序的通用块（文件）。

这种以块（文件）的方式存储私钥和公钥的方式，可以帮助我们对信息有更好的操作。

块从文件中读取私钥（OpenFromDiskPrivateKey）。

```
call OpenQbitRSA1 .OpenFromDiskPrivateKey  
path "/sdcard/PrivateKey.txt"
```

所需的依赖关系：Block (SaveFromDiskPrivateKey) 或 Block (GetPrivateKeyString)。

输入参数：**路径<字符串>**

输出参数：系统加载RSA私钥加密字符串。

描述：为我们提供一个存储在所提供的文件路径中的私钥的加密字母数字字符串的块。

从文件中读取公钥的块（OpenFromDiskPublicKey）。

```
call OpenQbitRSA1 .OpenFromDiskPublicKey  
path "/sdcard/PublicKey.txt"
```

强制性单位 : Block(SaveFromDiskPublicKey)或Block(GetPublicKeyString)。

输入参数 : **路径**<字符串>

输出参数 : 载入系统RSA公钥加密字符串。

描述 : 给我们提供一个存储在提供的文件路径中的公钥的加密字母数字字符串的块。

用于将私钥保存到文件的块 (SaveToDiskPrivateKey) 。

```
call OpenQbitRSA1 .SaveToDiskPrivateKey  
path " /sdcard/PrivateKey.txt "
```

法定单位 : 块(GenKeyValuePair)。

输入参数 : **普通**<字符串>。

输出参数 : 带有RSA加密字符串的文件。(私钥)

描述 : 给我们一个文件的字母数字字符串的块, 使用块中使用的大小的私钥 (GenKeyValuePair) 加密。

用于将私钥保存到文件的块 (SaveToDiskPublicKey) 。

```
call OpenQbitRSA1 .SaveToDiskPublicKey  
path " /sdcard/PublicKey.txt "
```

法定单位 : 块(GenKeyValuePair)。

输入参数 : **普通**<字符串>。

输出参数 : 带有RSA加密字符串的文件。(公钥)

描述 : 给我们提供一个文件的块, 其中有一个字母数字字符串, 使用块中使用的大小的公钥加密 (GenKeyValuePair) 。

OpenQbitSSHClient扩展。

连接器块SSH客户端(ConnectorMiniBlocklyChain)

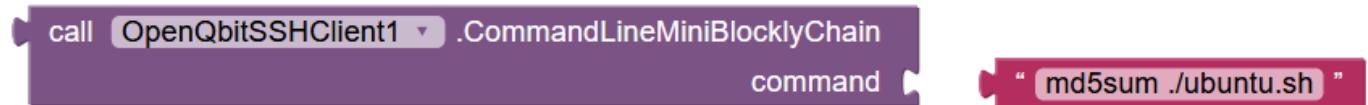


输入参数：**用户名**<string>, **密码**<string>, **主机**<string>, **端口**<整数>。

输出参数：如果与Termux终端的ssh服务器连接成功，则给我们一个消息；"连接SSH"，如果不成功，则给我们一个NULL消息。

描述：通过SSH（安全壳）通信协议，将Mini BlocklyChain连接到Termux终端。

在Termux Linux终端中执行命令的块（CommandLineMiniBlocklyChain）。



输入参数：**命令**<字符串>。

输出参数：变量数据，根据执行的命令或程序而定。

说明：Termux终端中命令执行块的前提条件是与块（ConnectorMiniBlocklyChain）进行连接，可以在线执行各种命令和/或从具有CLI（命令行接口）的脚本或程序中在线获取特定的执行数据。

DisconnectMiniBlocklyChainSSH。

call OpenQbitSSHClient1 .DisconnectMiniBlocklyChainSSH

输入和输出参数：不适用(无)

描述：关闭SSH会话的块。

OpenQbitStringHash扩展。

生成MD5字符串的块(MD5ThisString)

call OpenQbitStringHash1 .MD5ThisString

string

“ Mini BlocklyChain es modular ”

输入参数：字符串

输出参数：提供MD5哈希值。

描述：用于创建输入参数中字符串的MD5哈希值的块。

生成 SHA256 字符串的块 (SHA256ThisString) 。

call OpenQbitStringHash1 .SHA256ThisString

string

“ Mini BlocklyChain es modular ”

输入参数：字符串

输出参数：提供SHA256哈希值。

描述：用于创建输入参数中字符串的SHA256哈希值的块。

生成SHA512字符串的块(SHA512ThisString)

call OpenQbitStringHash1 .SHA512ThisString
string

" Mini BlocklyChain es modular "

输入参数：字符串

输出参数：提供SHA512哈希值。

描述：用于创建输入参数中给出的字符串的SHA512哈希值的块。

生成SHA1字符串（SHAThisString）的块。

call OpenQbitStringHash1 .SHAThisString
string

" Mini BlocklyChain es modular "

输入参数：字符串

输出参数：提供SHA1哈希值。

描述：用于创建输入参数中字符串的SHA1哈希值的块。

22. 在Mini BlocklyChain中设置安全参数。

任何系统的设计和应用，其安全参数都分为以下三个部分。

- a. Redis数据库(备份通信网)
 - b. 点对点同步系统
 - c. 整合安全性以保护SQLite数据库
 - d. 模块集成的开发者环境
-
- a. Redis数据库(备份通信网)

重命名危险的命令，Redis的额外内置安全功能涉及重命名或禁用一些被认为是危险的命令。

当未经授权的用户执行这些命令时，可以用来重新配置、销毁或删除他们的数据。和认证密码一样，重命名或禁用命令也是在`/etc/redis/redis.conf`文件的SECURITY部分配置的。

一些被认为是危险的命令：`FLUSHDB`、`FLUSHALL`、`KEYS`、`PEXPIRE`、`DEL`、`CONFIG`、`SHUTDOWN`、`BGREWRITEAOF`、`BGSAVE`、`SAVE`、`SPOP`、`SREM`、`RENAME`和`DEBUG`。这不是一个完整的列表，但重命名或禁用该列表中的所有命令是提高Redis服务器安全性的良好开端。

根据您或您网站的具体需求，您必须重新命名或禁用一个命令。如果你知道你永远不会使用可以操纵的命令，你可以禁用它。另一方面，你可能想重新命名。

要启用或禁用Redis命令，请重新打开配置文件。

```
$ vi /etc/redis/redis.conf
```

警告：以下禁用和重命名命令的步骤为示例。你应该只选择禁用或重命名适用于你的命令。您可以在redis.io/commands中查看完整的命令列表，并确定它们如何被滥用。

要禁用命令，只需将其重命名，使其成为一个空字符串（用一对引号来表示，中间没有字符），如下所示。

```
/etc/redis/redis.conf
```

```
# 也可以通过命令重命名  
# 一个空字符串  
#  
rename-command FLUSHDB ""  
rename-command FLUSHALL ""  
rename-command DEBUG ""
```

要重命名一个命令，请给它另一个名字，如下面的例子所示。重命名的命令应该让别人很难猜到，但你却很容易记住。

/etc/redis/redis.conf

```
# rename-command CONFIG ""  
rename-command SHUTDOWN SHUTDOWN_MENOT  
rename-command CONFIG ASC12_CONFIG
```

保存更改并关闭文件。

重命名一个命令后，通过重启Redis来应用更改。

- sudo systemctl restart redis.service

要测试新命令，请输入Redis命令行。

- Redis-cli

然后，进行认证。

- auth your_redis_password

产量
好的

如同前面的例子，假设你把CONFIG命令重命名为ASC12_CONFIG。首先，尝试使用原来的CONFIG命令。因为你重名了，所以应该不能用。

- 配置 get requirepass

产量
(错误 ERR 没有命令 config'。

但是，重命名命令可以成功调用。它不区分大小写。

- `asc12_config get requirepass`

产量

1) "requirepass"
2) "your_redis_password"

最后，你就可以结束重选了。

- 退出

需要注意的是，如果你已经使用Redis命令行并重新启动Redis，你将需要再次进行验证。否则，如果你输入一个命令，将显示这个错误。

产量

需要NOAUTH认证。

b. 点对点同步系统

在节点之间使用"点对点"系统时，该系统在通信网络中应用的内容有以下三点。

-私密性：除了您的电脑外，没有其他地方存储信息。没有中央服务器可以被入侵（合法或非法）。

-加密：所有的通信都是通过TLS（传输层安全）协议来保证的，这是一个加密协议，包括一个完美的序列，以防止任何人在您的信任之外访问您的信息。

-认证：每个节点都有一个强大的加密证书来识别。只有你明确允许的节点，才能连接到你的信息。

<https://docs.syncthing.net/users/security.html>

使用在线命令SyncthingManager。

<https://github.com/classicsc/syncthingmanager>

c. 整合安全性以保护SQLite数据库

当使用扩展(OpenQbitSQLite)或在它的情况下使用扩展(ConnectorSSHClient)来使用SQLite CLI时，这两个扩展可以与安全扩展AES(OpenQbitEncDecData)相结合。

参见附录"Mini BlocklyChain系统创建实例"。

d. 模块集成的开发者环境

要在开发环境中实现安全，可以通过两个过程来实现。

- 限制使用OpenJDK库。
- 只限于授权系统节点使用。

23.附件"创建KeyStore和PublicKeys数据库"。

KeyStore是安全证书的存储库，可以是授权证书，也可以是公钥证书、密码或通用安全密钥，如用户相应的私钥（地址），用于创建或处理交易。

它是一个加密的数据库，所以只有所有者才能使用它。通常情况下，它是一个本地类型，但是，在Mini BloclyChain系统中，它是一个安全的数据库，但它在所有节点中是共享和分布的，这基本上是由于一个简单的原因，所有节点必须知道所有用户的地址（公共地址），私人地址始终是本地的，并且是每个用户唯一和专属使用的，然而，当进行一个入口（存款）或出口（支出）交易时，每个交易在创建时总是至少有三个组成部分：起源地址，目的地地址和发送的资产或价值。

要创建我们的KeyStore，我们必须遵循以下步骤和要求。

第一个要求是要有一个存储类型，在我们的例子中，我们将使用SQLite数据库，我们将创建两个KeyStore，一个是用户的私钥，它将是本地的，它将是安全的"加密"信息，另一个数据库将存储公共密钥，这个基础将在网络的所有节点中共享，在网络中

共享的基础也将是加密的，但是这个将有一个密码，只有网络的成员（节点）能够共享。

第二个基本要求是信息加密的过程，这将通过在通用数据库中使用的名为（`OpenQbitEncDecData`）的专门扩展来完成，它使用AES算法。

扩展(`OpenQbitEncDecData`)的功能是用于验证区块链的单元元素，但是，在必须检查区块链的全部完整性的情况下，它的效率将不高，所以我们还将进行副本的加密，但在这种情况下，它将通过扩展。`(OpenQbitAESEncryption)`和`(OpenQbitAESDecryption)`，工作在一个文件上，而不是通过引用数据。

注意：SSH服务器必须在Termux终端上运行，`KeyStore`数据库才能正常运行。在终端中执行该命令。

`$ sshd`

基于AES算法的扩展可以用于任何类型的数据或文件，之所以选择这种算法，是因为它是唯一一个被证明可以抵御基于量子计算的攻击的算法。

Criptosistema	Categoría	Tamaño de clave	Parámetro de seguridad	Algoritmo cuántico estimado que rompa el criptosistema	Nº de qubits lógicos necesarios	Nº de qubits físicos necesarios	Tiempo necesario para romper el sistema	Estrategias de reemplazo cuántico-resilientes
AES-GCM	Cifrado simétrico	128 192 256	128 192 256	Algoritmo de Grover	2.953 4.449 6.681	$4,61 \times 10^6$ $1,68 \times 10^7$ $3,36 \times 10^7$	$2,61 \times 10^{12}$ años $1,97 \times 10^{22}$ años $2,29 \times 10^{32}$ años	
RSA	Cifrado asimétrico	1024 2048 4096	80 112 128	Algoritmo de Shor	2.290 4.338 8.434	$2,56 \times 10^6$ $6,2 \times 10^6$ $1,47 \times 10^7$	3,58 horas 28,63 horas 229 horas	Migrar a un algoritmo PQC seleccionado por el NIST
ECC Problema del logaritmo discreto	Cifrado asimétrico	256 386 512	128 192 256	Algoritmo de Shor	2.330 3.484 4.719	$3,21 \times 10^6$ $5,01 \times 10^6$ $7,81 \times 10^6$	10,5 horas 37,67 horas 95 horas	Migrar a un algoritmo PQC seleccionado por el NIST
SHA256	Minado de Bitcoin	N/A	72	Algoritmo de Grover	2.403	$2,23 \times 10^6$	$1,8 \times 10^4$ años	
PBKDF2 con 10.000 iteraciones	Hashing de contraseñas	N/A	66	Algoritmo de Grover	2.403	$2,23 \times 10^6$	$2,3 \times 10^7$ años	Abandonar la autenticación basada en contraseñas

上参考美国国家工程院研究报告

<https://www.nap.edu/catalog/25196/quantum-computing-progress-and-prospects>

说明

量子力学是描述极小粒子（量子）行为的物理学子领域，为新的计算范式提供了基础。量子计算领域最早是在20世纪80年代提出的，作为改进量子系统计算建模的一种方式，最近由于小规模器件的构建进展，引起了人们的极大关注。然而，在大规模实现实用量子计算机之前，还需要在技术上取得重大进展。

量子计算：进展与展望》介绍了该领域的情况，包括该技术的独特特点和局限性，并评估了创建能够解决现实世界问题的功能性量子计算机的可行性和意义。本报告考虑了硬件和软件要求、量子算法、量子计算和量子器件进展的驱动因素、与相关用例相关的基准、所需的时间和资源，以及如何评估成功的概率。在TERMUX终端安装SQLite数据库处理程序，并通过创建名为**keystore.db**的数据库来测试**sqlite3**命令的CLI（命令行）。

我们使用命令。

```
$ apt install sqlite  
$ sqlite3 keystore.db
```

```
$ apt install sqlite ←  
Reading package lists... Done  
Building dependency tree  
Reading state information... Done  
The following NEW packages will be installed:  
sqlite  
0 upgraded, 1 newly installed, 0 to remove and 0  
not upgraded.  
Need to get 459 kB of archives.  
After this operation, 811 kB of additional disk  
space will be used.  
Get:1 https://dl.bintray.com/termux/termux-pac  
ges-24 stable/main arm sqlite arm 3.32.2-3 [459  
kB]  
Fetched 459 kB in 1s (237 kB/s)  
Selecting previously unselected package sqlite.  
(Reading database ... 16937 files and directo  
ries currently installed.)  
Preparing to unpack .../sqlite_3.32.2-3_arm.deb  
...  
Unpacking sqlite (3.32.2-3) ...  
Setting up sqlite (3.32.2-3) ...  
$
```

```
$ sqlite3 keystore.db ←  
SQLite version 3.32.2 2020-06-04 12:58:43  
Enter ".help" for usage hints.  
sqlite> .databases ←  
main: /data/data/com.termux/files/home/mbr/keyst  
ore.db  
sqlite> .quit ←  
$ ls ←  
keystore.db  
$
```

然后

在**sqlite>处理程序**中，我们执行**.databases**句子来检查我们正在创建的数据库的路径
，然后退出并保存数据库，我们给出**.quit**的句子。

注意：在**sqlite>处理程序**的两个语句或命令中，你必须先**在语法中加上一个点"."**。

最后，我们通过下达命令来检查（空）数据库是否已经创建。

```
$ ls
```

我们继续创建一个表，主密钥将以三种不同的格式存储：十六进制、二进制和Mini BlockyChain用户参考地址，这是各自主密钥的公钥。

AES数据加密将只应用于十六进制和二进制格式。在用户的地址addrMBC和别名的情况下，不因为它是公钥，可以而且应该在整个网络上共享，以接收交易，以及别名执行该领域的搜索。

在SQLite中的数据库中创建了一个名为"privatekey"的表（keystore.db）。

```
CREATE TABLE privatekey (
    id整数主键
    又名      VARCHAR(50) NOT NULL
    addrHexVARCHAR (65) NOT NULL
    addrMBCVARCHAR (65) NOT NULL
    addrBinBLOB NOT NULL
);
```

让我们执行sqlite CLI中的句子来创建keystore.db数据库。

我们再使用sqlite3命令行，使用以下命令。

```
$ sqlite3
```

这将会把我们送到sqlite>数据库管理器里面，在这第一条里面，我们将打开已经创建的数据库，在管理器里面用下面的句子就可以对它进行操作。

```
Sqlite> .open keystore.db
```

然后我们将输入SQL语句"CREATE TABLE"来创建**私钥表**。

接下来，显示了两个选项来创建同一个表，一个是通过单行，其中包括所有集成它的元素的SQL语句。第二个例子是通过每个元素的引入，将结构进行分段整合。

```
$ sqlite3
```

SQLite版本3.32.2 2020-06-20 15:25:24

输入".help"以获得使用提示。

连接到一个短暂的内存数据库。

使用".open FILENAME"在持久性数据库上重新打开。

```
sqlite> .open keystore.db。
```

```
sqlite> CREATE TABLE privatekey (id integer primary key AUTOINCREMENT NOT NULL, aka  
VARCHAR(50) NOT NULL, addrHex VARCHAR(65) NOT NULL, addrMBC VARCHAR(65) NOT  
NULL, addrBin BLOB NOT NULL)。
```

```
sqlite> .quit
```

接下来展示了同样的**私钥表**的创建，不过是通过分段引入SQL语句的方式。

```
sqlite> CREATE TABLE privatekey (  
...> i d 整数建 AUTOINCREMENT  
...> 又名VARCHAR(50) NOT NULL。  
...> addrHex VARCHAR (65) NOT NULL,  
...> addrMBC VARCHAR (65) NOT NULL,  
...> addrBin BLOB NOT NULL。  
...> );  
sqlite> .tables  
私钥  
sqlite> .quit
```

以上两个例子给出了同样的结果。后面我们执行.tables句子来检查私钥表是否已经创建，最后我们会给出.quit句子，有了这个过程我们已经在SQLite keystore.db数据库里面创建了私钥表。

直到此刻，我们已经有了keystore.db数据库的结构和它的私钥表，私钥将以加密的方式存储在那里。

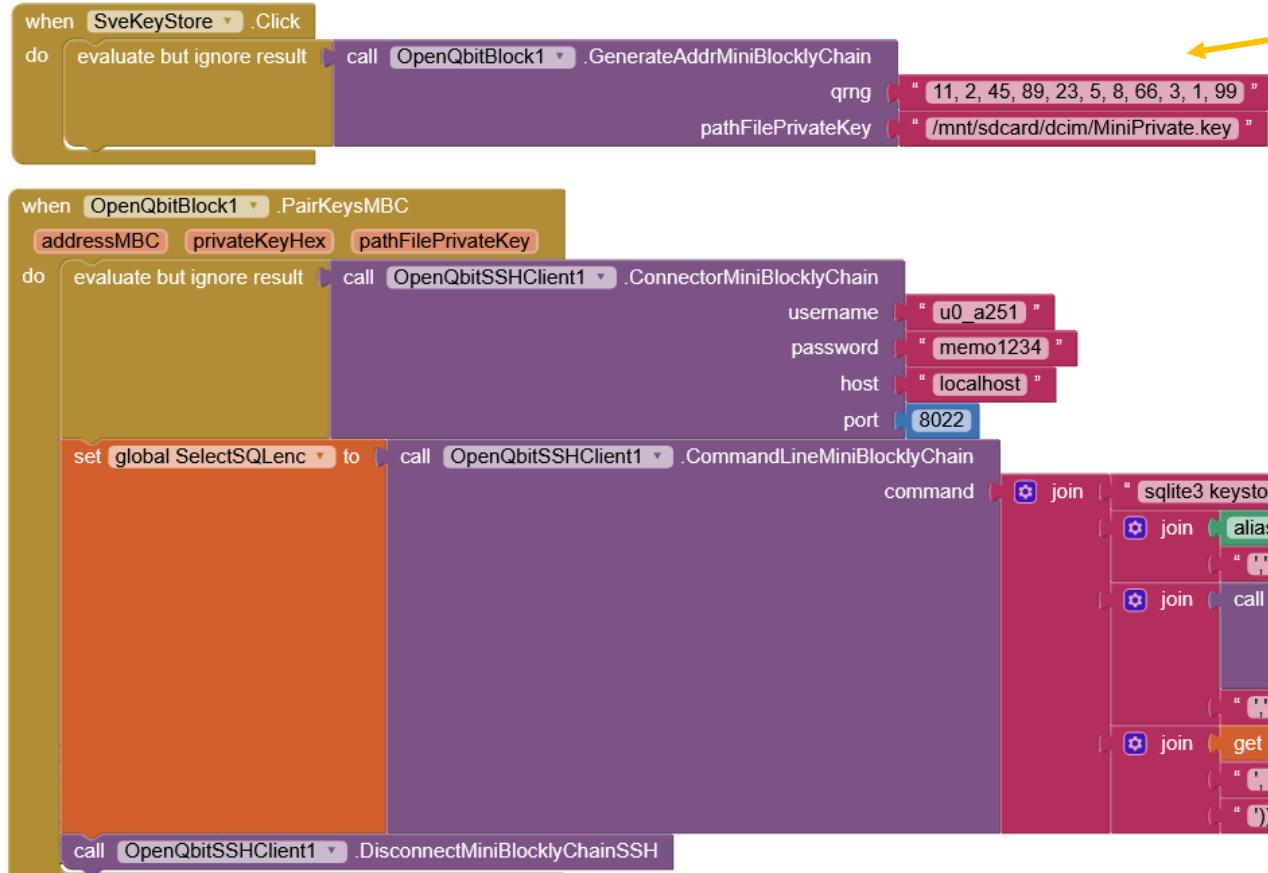
这在节点（手机）与TERMUX终端中应该会有类似的显示。

```
$ sqlite3
SQLite version 3.32.2 2020-06-04 12:58:43
Enter ".help" for usage hints.
Connected to a transient in-memory database.
Use ".open FILENAME" to reopen on a persistent database.
sqlite> .open keystore.db
sqlite> CREATE TABLE privatekey (
...>     id integer primary key AUTOINCREMENT,
...>     alias VARCHAR(50) NOT NULL,
...>     addrHex VARCHAR(65) NOT NULL,
...>     addrMBC VARCHAR(65) NOT NULL,
...>     addrBin BLOB NOT NULL
...> );
sqlite> .tables
privatekey
sqlite> .quit
$
```

现在我们将使用一些安全扩展来插入"私钥"数据，然后查阅这些数据。

我们将使用这个块来生成一个新的用户地址（`GenerateAddrMiniBlocklyChain`），这个块将给我们提供两种格式（十六进制和二进制）的私有地址，以及MBC通用地址格式的公共地址。我们还将使用扩展(`OpenQbitSSHClient`)和扩展(`OpenQbitEncDecData`)来查看更多细节，请查看"安全块的定义和使用"一节。在Termux终端中，你应该运行SSH服务。

将加密的数据插入keystore.db数据库中。

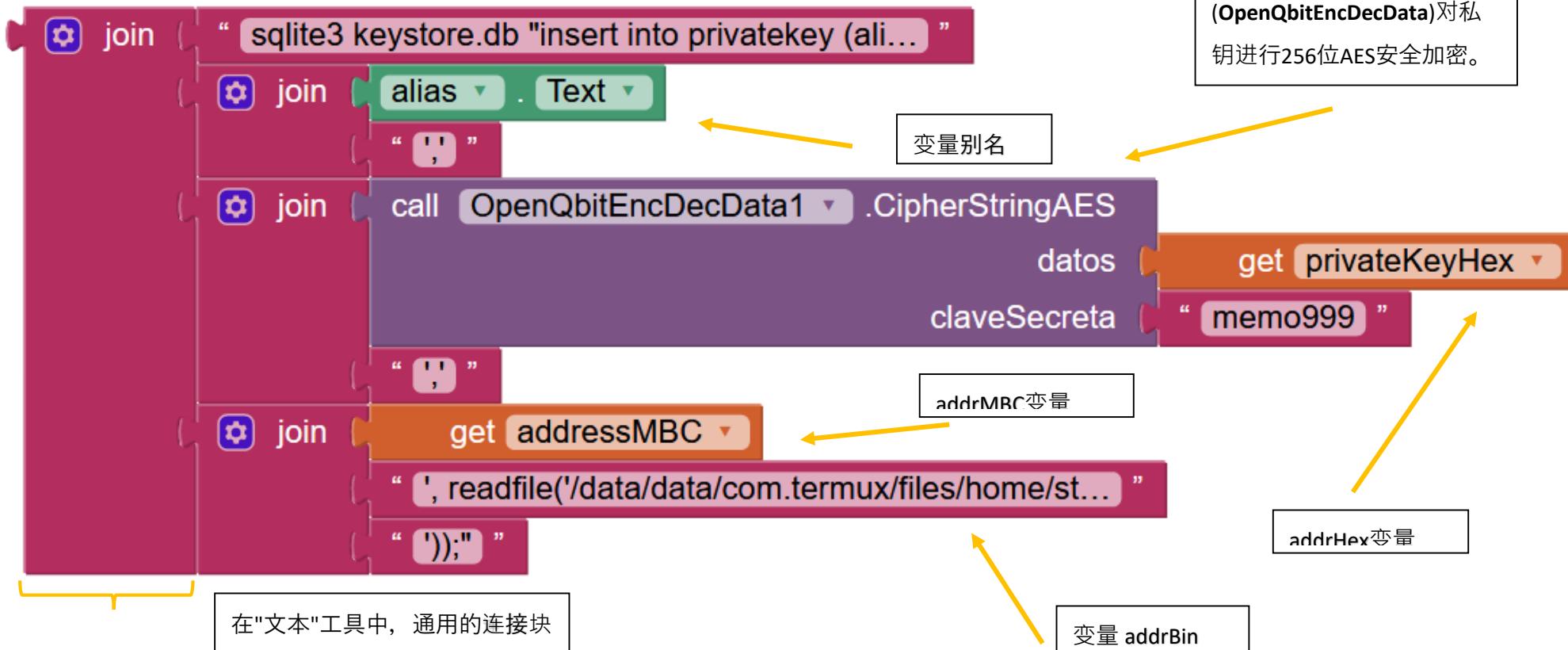


量子随机数系列生成的块(ApiGetQRNGInteger)
检查如何格式化JSON结果，因为块条目
(GenerateAddrMiniBlocklyChain)需要的一系列数
字只用逗号", "隔开。

命令的语法是非常重要的，我们必须在Blockly环境中以正确的格式介绍以下命令。

值的值将永远是变量，例。

```
sqlite3 keystore.db "insert into privatekey (alias, addrHex, addrMBC, addrBin) values ('memo', 'QWERTY', 'MBC12345',  
readfile('/data/data/com.termux/files/home/storage/dcim/MiniPrivate.key'));"
```



执行块后，我们会在keystore.db数据库的私钥表中得到一个类似于下面的结果。

我们在Termux终端输入sqlite处理程序，打开keystore.db库，执行这句话。

\$ sqlite3

SQLite版本3.32.2 2020-06-20 15:25:24

输入 ".help" 以获得使用提示。

连接到一个短暂的内存数据库。

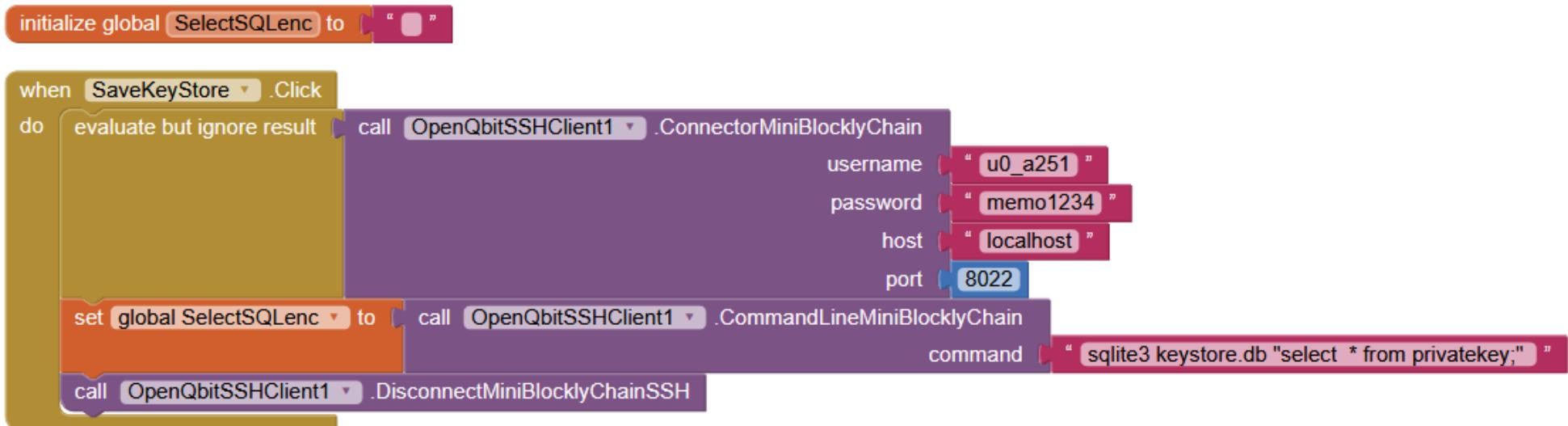
使用 ".open FILENAME" 在持久性数据库上重新打开。

sqlite> .open keystore.db。

sqlite> select * from privatekey;



查询SQLite keystore.db数据库中私钥表的所有数据。



查询SQLite keystore.db数据库中私钥表中数据的别名。

```
sqlite3 keystore.db "select alias from privatekey;"
```

查询在SQLite keystore.db数据库的私钥表中加密的addrHex列的所有数据。

```
sqlite3 keystore.db "select addrHex from privatekey;"
```

CONSULT检索SQLite keystore.db数据库的privatekey表中的addrBin私钥。

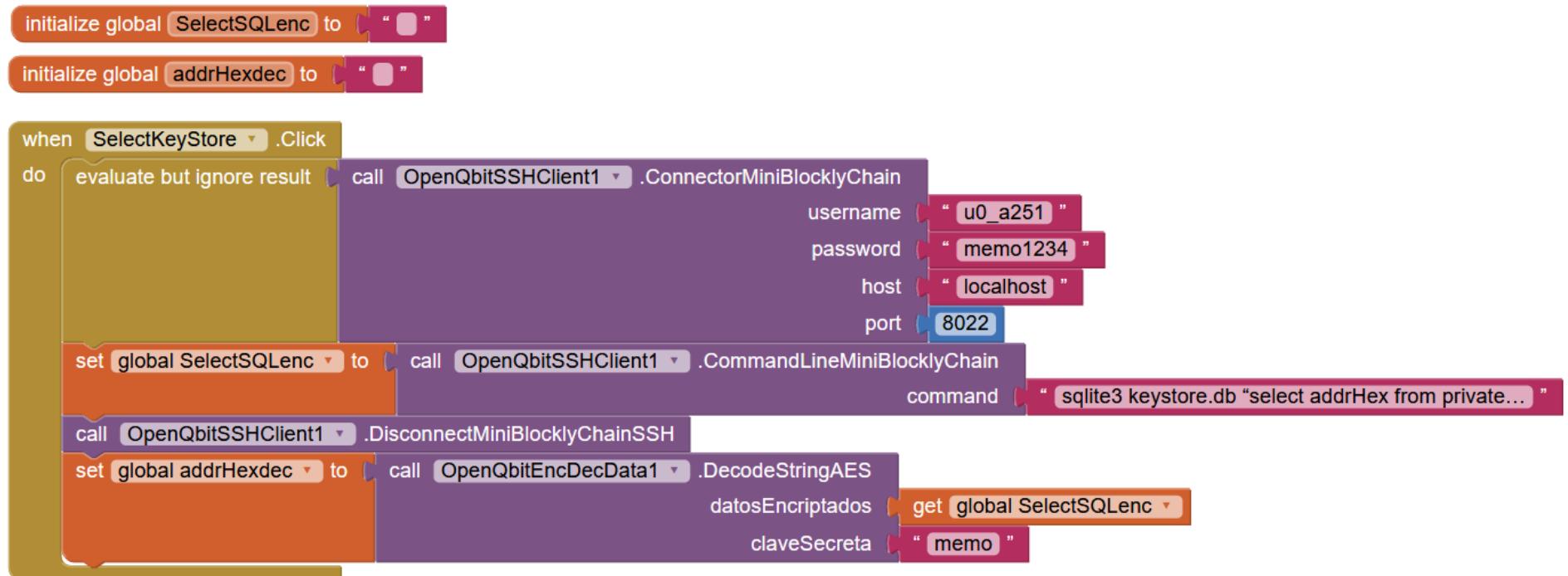
```
sqlite3 writefile('PrivateKey.key', addrBin) FROM privatekey WHERE alias='mexico'; (2)
```

(1) 这类查询将主要是查询私钥来执行交易的签名过程。

查询SQLite keystore.db数据库privatekey表中addrHex列的别名解密数据。

在这种情况下，我们将使用块（DecodeStringAES）来解码存储在"addrHex"列中的数据。

```
sqlite3 keystore.db "select addrHex from privatekey where='mexico';"
```



这个咨询给我们提供了十六进制格式的私钥，这是任何接收或发送交易的基本部分。再三建议对这个keystore.db数据库进行备份。

数据库设计publickeys.db， 表publicaddr:

\$ sqlite3

SQLite版本3.32.2 2020-06-20 15:25:24

输入 ".help" 以获得使用提示。

连接到一个短暂的内存数据库。

使用 ".open FILENAME" 在持久性数据库上重新打开。

sqlite> .open publickeys.db。

sqlite> CREATE TABLE publicaddr (id integer primary key AUTOINCREMENT NOT NULL, pubMBC VARCHAR NOT NULL, pubHex VARCHAR NOT NULL, pubBin BLOB NOT NULL)。

sqlite> .quit

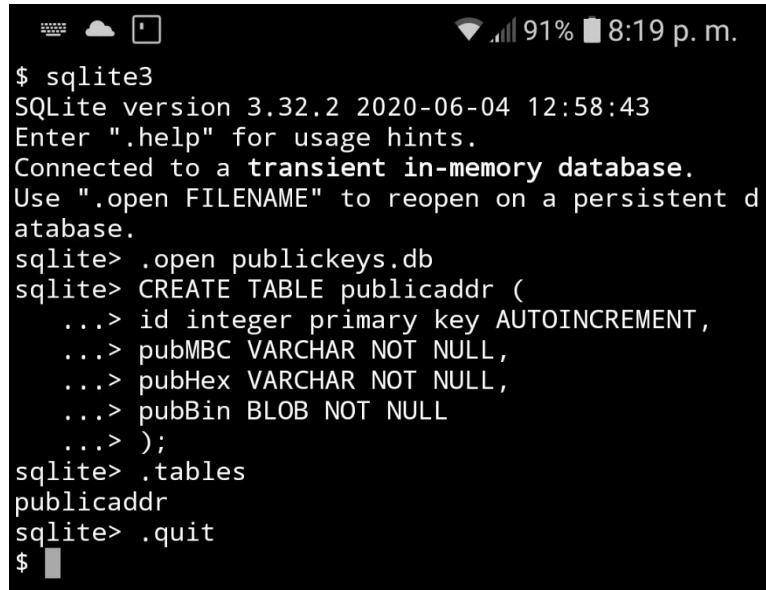
接下来， 创建**发布的**表， 用下面的SQL句式分段显示。

```
sqlite> CREATE TABLE publishedaddr (
...> i d 整数建 AUTOINCREMENT
...> pubMBC VARCHAR NOT NULL,
...> pubHex VARCHAR NOT NULL,
...> pubBin BLOB NOT NULL
...> );
```

sqlite> .tables

公布的地址

sqlite> .quit



```
$ sqlite3
SQLite version 3.32.2 2020-06-04 12:58:43
Enter ".help" for usage hints.
Connected to a transient in-memory database.
Use ".open FILENAME" to reopen on a persistent database.
sqlite> .open publickeys.db
sqlite> CREATE TABLE publicaddr (
    ...> id integer primary key AUTOINCREMENT,
    ...> pubMBC VARCHAR NOT NULL,
    ...> pubHex VARCHAR NOT NULL,
    ...> pubBin BLOB NOT NULL
    ...> );
sqlite> .tables
publicaddr
sqlite> .quit
$
```

24. 附件"RESTful SQLite GET/POST命令"。

接下来， 我们可以在备份网络的Restful SQLite中使用不同的命令。这些都是从GITHUB (<https://github.com/olsonpm/sqlite-to-rest>) 最初开发时就开始咨询的。

CRUD操作（创建、读取、更新和删除） RESTful。

下面是RESTful API以伪例子的形式提供的可用操作列表。我们将假设一个基础的 "op.sqlite" 和两个相关的 "trans" 表用于交易，另一个 "sign" 用于源地址、目的地址和资产价值，两个列 id INTEGER PRIMARY KEY。

正如在限制中所指出的那样，请注意，这些方法（DELETE和POST）一次只能影响一条记录。

获得这可以最大限度的变化。稍后我们将看到所有可用的查询操作符。

标题可以指定在URL的下面。

所有行的transRequests。

/trans?id=1其中
id=1。

/trans
range: rows=0-2
前三行。

/trans
range: rows=-5最后
五行。

/trans
范围 rows=0-

服务器能提供的行数，实际上是maxRange和总行数中较少的一个。

/trans
范围: rows=1-

从第1行开始，服务器能提供多少行，就提供多少行。

/trans

顺序:

姓名升序排列

/trans

顺序:姓名

降序姓生名降序排列。

/trans

顺序:姓名 降序 ID

有贡献的，但首先按名字降序排列，如果是平局，则按id升序排列。

/trans?id>1

其中id>1

/trans?id>=2&id<5。

其中id>=2且id<5

/trans?name_NOTNULL。

其中名称不是空的

/trans?name_ISNULL其中

name为空。

/trans?id!=5&name_LIKE'Spotted%'。

其中id !=5且名称为LIKE"Spotted%"(忽略%)

/trans?id>=1&id<10&name_LIKE'Avery%'。

顺序 name desc,id range: rows=2-4

为举例而贡献。

获取标识符在1到9（含）之间的交易，名称为"Avery%"，先按名称降序排列，再按标识符升序排列，获取结果的第三行到第五行。或者在SQL中。

DELETE 需要一个查询字符串，所有主键都设置为等于一个值。这就要求最大限度地单行删除。

/trans?id=1

删除id=1的trans。

如果事务反而有一个由id和name组成的主键。

/trans?id=1&name='艾利IPA'。

POST创建

你不能传递一个查询字符串。如果传递了一个查询字符串，则假定为POST更新。所有的POST请求必须传递头内容类型：application / json。

请注意，正文必须包含所有不可取消的PRIMARY KEY列，而不是INTEGRATE。否则，将发送400响应，说明哪些字段丢失。可空列将为空，INTEGER PRIMARY KEY列将按照sqlite3规范自动增加。

JSON数据将被指定在URL的下方。

/trans
{"id":1,"name":"Serendipity"}
创建一个id=1, name='Serendipity'的trans。

/trans
{"id":1}
创建一个id=1, name=NULL的trans。

/trans
{"name": "Serendipity"}。

创建一个事务，id设置为下面的值，由sqlite3规格增加INTEGER PRIMARY KEY。

/trans
{}

创建一个id增加的事务，名称设置为NULL。

POST更新

它必须包含一个查询字符串。如果没有查询字符串，则假设创建POST。与POST创建一样，需要头内容类型：application / json。

查询字符串必须包含所有的主键，以确保只更新一行。如果传递的值不正确，将返回一个400的违规键。

请求的主体必须包含一个非空对象，并且必须包含与列名相对应的有效键。

JSON数据将被指定在URL的下方。

```
/trans?id=1  
{ "id":2 }
```

将ID为1的事务更新为2。

```
/trans?id=1  
{ "name": "MCBza45Rt56cvbfdR2Swd788kj" }。
```

更新ID为1的交易，将其名称或值设置为"MCBza45Rt56cvbfdR2Swd788kj"。

如果交易台反而有一个由id和name组成的主键。

```
/trans?id=1&name=MCBza45Rt56cvbfdR2Swd788kj。  
{ "name": "MCB3ofFG5Hj678MNb09vLdfaasX" }。
```

更新交易，其中id为一，地址为MCBza45Rt56cvbfdR2Swd788kj，设置MCB3ofFG5Hj678MNb09vLdfaasX。

参考资料

isSqliteFile

只需检查文件的前16个字节，看看它是否等于'sqlite format 3'，后面是一个空字节。

isDirectory

返回fs.statsSync的结果，后面是.isDirectory。

是文件

返回fs.statsSync的结果，后面跟着.isFile。

获取咨询运营商

查询条件必须用连接符号标明，如：id > 5 & name = MCBza45Rt56cvbfdR2Swd788kj。

二进制运算符（要求后面有一个值）文。

```
=  
!=  
>=  
<=  
>  
<  
_LIKE
```

LIKE之所以特别，是因为它必须有简单的开头和结尾引号。否则，将产生一个400错误，显示分析无法完成的地方和预期。参见CRUD RESTful操作的例子。

单一运算符（必须跟在列名后面）。

IS NULL

非空

路由器配置对象

isLadenPlainObject

这个对象的目的是为sqlite路由器提供一个通用配置。允许以下属性：

prefix: isLadenString 传递给koa-router prefix builder选项的字符串。例如，骨架服务器不指定前缀，这样就可以直接从http域名的根部打啤酒API：// localhost : 8085 / trans。如果你把前缀设置为' / api'，那么你应该把请求发送到http: // localhost: 8085 / api / trans。

allTablesAndViews：一个表格配置对象。

此对象中指定的设置将应用于所有表和视图，可选择由 tablesAndViews 属性覆盖。

tablesAndViews: isLadenPlainObject 过去的对象必须有与数据库列或视图名称相匹配的键。否则，将发出友好的错误信息。每个表和视图的值必须是一个表格配置对象。

列表式配置对象

isLadenPlainObject 这个对象表示可以为视图或表设置的设置。它允许以下属性：

maxRange: isPositiveNumber(正数)

默认应用：1000

这是你的服务器允许请求的最大范围。如果GET请求到达时没有范围头，规范会假设你想要整个资源。如果GET产生的行数大于maxRange，则返回一个416状态，并带有自定义的max-range头。应用程序的默认值故意保守，希望作者根据自己的需要设置maxRange。

请注意，“Infinity”是一个有效的正数。

标志：isLadenArray

目前，唯一接受的指标是字符串'sendContentRangeInHEAD'。当设置时，HEAD请求将以content-range: * / <max-range>的形式返回可用内容的范围。它是可配置的，原因是计算最大范围可能会比它值得更多的工作，这取决于服务器负载和你的表的大小。

自定义页眉

应用

order：这个头只为GET定义，可以认为相当于sql ORDER BY。它必须包含一个以逗号分隔的列名，每个列名后面必须有一个空格和字符串'asc'或'desc'。如果发送的订单值不正确，400响应会提示是哪些。

答案

不一定都是定制的，但所有的使用都在规范之外，因此需要说明。

获取

max-range：当请求的行数超过配置的maxRange时，将返回这个头。请注意，该请求可能没有指定范围头，但该资源产生的行数仍然会被验证。

内容范围：RFC7233规定

只有状态码206（部分内容）和416（不满意范围）描述了内容范围的含义。

当sqlite-to-rest以状态码200响应时，内容范围头以<行开始>-<行结束>/<行计数>的格式206发送。

当发送请求时，如果没有范围头，结果行数超过maxRange，则返回400，内容范围设置为416格式，即*/<行数>。

注意，这个头可以在HEAD响应中返回。

accept-order：如果请求头的顺序语法不正确或指定的列名不正确，将被返回。更多详情请看下面的HEAD -> accept-order。

25.附件"Java代码SQLite-Redis连接器"。

SQLite-Redis两个数据库之间的通信链接代码如下所示。基本上，我们可以看到，连接器将SQLite格式改为Redis接收数据（事务）的通用字符串。

上述过程作为一个事务队列触发器，向所有连接的、可能成为处理当前事务队列候选者的节点发送。

当Redis数据库通过它所拥有的主从配置接收到事务队列时，它会将信息实时、平等地分配给所有节点。

另一个基本点是，所有节点的时钟必须同步，这将像我们之前看到的查询到服务器池NTP（网络时间协议）的功能一样。

该连接器还通过计算所有交易的Merkle根树来执行第一级数据安全审查。

SQLite-Redis连接器的基础代码。

```
导入java.sql.*。
import java.util.*;
import java.util.Arrays.Import java.util.Arrays;
import java.util.List.Inc;
导入 java.util.array。
import java.security.*;
导入java.security.MessageDigest。
import redis.client.jedis.Jedis.Jedis;
class RedisSqlite{
    public String previousHash;
    公用字符串 merkleRoot.Public String merkleRoot;
    public static ArrayList<String> transactions = new ArrayList<String>();
    public static ArrayList<String> treeLayer = new ArrayList<String>();
    公共静态字符串getMerkleRoot() {
        int count = transactions.size();
        int item = 1;
        奇数int=0。
        ArrayList<String> previousTreeLayer = transactions;
        while(count > 1) {
            treeLayer = new ArrayList<String>();
            for(int i=1; i <= count ; i=i+2) {

                if (!esPar(count) && i == count) odd = 1;
                treeLayer.add(applySha256(aforeTreeLayer.get(i-item)
previousTreeLayer.get(i-impar)))+

            }
        项目=1;
        奇数=0。
        count = treeLayer.size();
        previousTreeLayer = treeLayer;
    }
}
```

```
String merkleRoot = (treeLayer.size() == 1) ? treeLayer.get(0) : ""。
    返回 merkleRoot。
}

//定义Merkle树是偶数还是奇数。
static boolean enPar(int number) {
    if (numero%2==0) return true; else return false;
}

public static String applySha256(String input) {
    试试
        MessageDigest digest = MessageDigest.getInstance("SHA-256")。
        //将sha256应用于我们的输入。
        byte[] hash = digest.digest(input.getBytes("UTF-8"))。
        StringBuffer hexString = new StringBuffer(); // 这将包含十六进制的哈希值。
        for (int i = 0; i < hash.length; i++) {
            String hex = Integer.toHexString(0xff & hash[i]);
            if(hex.length() == 1) hexString.append('0')。
            hexString.append(hex)。
        }
        返回hexString.toString()。
    }
    catch(Exception e) {
        throw new RuntimeException(e);
    }
}

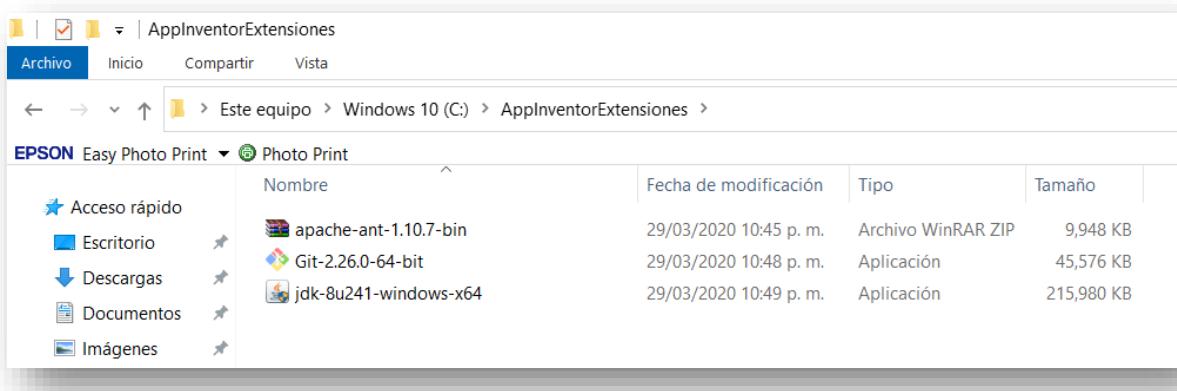
public static void main(String args[]) {
    尝试{
        Connection con=DriverManager.getConnection("jdbc:sqlite:C://memo/sqlite-tools-
win32-x86-3310100/trans.sqlite3")。
        //连接到本地主机上的Redis服务器。
        Jedis = new Jedis ("localhost")。
```

```
jedis.auth("moo1234")。
Statement stmt=con.createStatement()。
String sql = "SELECT * FROM brewery"。
ResultSet rs=stmt.executeQuery(sql)。
while(rs.next()) {
    transactions.add(rs.getString(2))。
    jedis.set("LATAM:"+String.valueOf(rs.getInt(1)), "["+"\""+rs.getString(2)+"\""+",
    +"\""+rs.getString(3)+"\", "+"\\""+rs.getString(4)+"\""+") )"
}
jedis.set("LATAM:merkleroot", getMerkleRoot())。
System.out.println("ArrayList元素数 : : "+treeLayer.size())。
与.close()。
}catch(Exception e){ System.out.println(e);}。
}
}
```

26. 附件"开发者的迷你BlocklyChain"。

在这个附件中，我们将回顾如何基于Java编程语言为Blockly环境生成外部模块的配置、安装和基本使用，我们将能够为每个业务案例创建专门的模块，并将功能插入到Mini BlocklyChain系统中，或者为我们的移动应用添加其他功能。

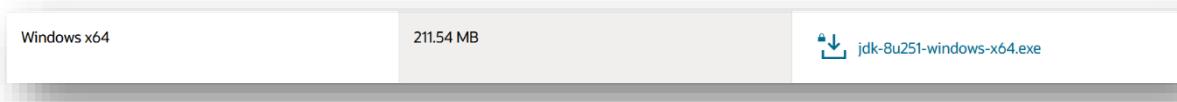
我们在Windows系统中创建了一个名为"ApplInventorExtensions"的目录，这个目录将下载以下公共软件包。



1.-我们将下载最新版本的JDK（Java开发工具包）。

例如：jdk-8u251-windows-x64.exe (211 MB)

<https://www.oracle.com/java/technologies/javase/javase-jdk8-downloads.html>



2.-使用JAVA构建应用程序的Apache Ant库，<http://ant.apache.org/bindownload.cgi>，在我的例子中，我已经下载了Ant 1.10.8（二进制分发）（apache-ant-1.10.8-bin.zip）。可能还有更高级的版本。

1.9.15 release - requires minimum of Java 5 at runtime

- 1.9.15 .zip archive: [apache-ant-1.9.15-bin.zip \[PGP\] \[SHA512\]](#)
- 1.9.15 .tar.gz archive: [apache-ant-1.9.15-bin.tar.gz \[PGP\] \[SHA512\]](#)
- 1.9.15 .tar.bz2 archive: [apache-ant-1.9.15-bin.tar.bz2 \[PGP\] \[SHA512\]](#)

1.10.8 release - requires minimum of Java 8 at runtime

- 1.10.8 .zip archive: [apache-ant-1.10.8-bin.zip \[PGP\] \[SHA512\]](#)
- 1.10.8 .tar.gz archive: [apache-ant-1.10.8-bin.tar.gz \[PGP\] \[SHA512\]](#)
- 1.10.8 .tar.bz2 archive: [apache-ant-1.10.8-bin.tar.bz2 \[PGP\] \[SHA512\]](#)
- 1.10.8 .tar.xz archive: [apache-ant-1.10.8-bin.tar.xz \[PGP\] \[SHA512\]](#)

Old Ant Releases

Older releases of Ant can be found [here](#). We highly recommend to not use those releases but upgrade to Ant's [latest](#) release.

3.-我们从您的网站上安装了Git Bash, <https://git-scm.com/download/win>

Downloading Git



You are downloading the latest (**2.27.0**) **64-bit** version of **Git for Windows**. This is the most recent [maintained build](#). It was released **9 days ago**, on 2020-06-01.

[Click here to download manually](#)

Other Git for Windows downloads

[Git for Windows Setup](#)

[32-bit Git for Windows Setup](#).

[64-bit Git for Windows Setup](#).

[Git for Windows Portable \("thumbdrive edition"\)](#)

[32-bit Git for Windows Portable](#).

[64-bit Git for Windows Portable](#).

The current source code release is version **2.27.0**. If you want the newer version, you can build it from [the source code](#).

4.-解压"Apache Ant."解压完成后，可以用双文件夹进行解压，比如。

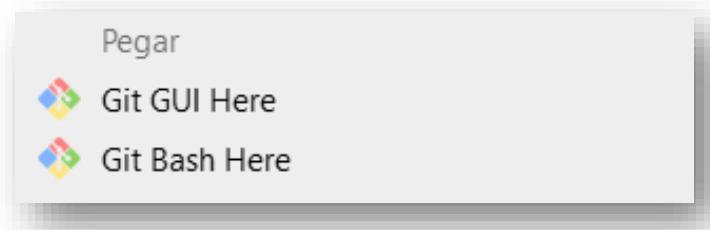
C: AppInventorExtensionsapache-ant-1.10.8-binapache-ant-1.10.8-bin。

- 将其改为C : Apache-ant-1.10.8-bin。

Este equipo > Windows 10 (C:) > AppInventorExtensiones > apache-ant-1.10.7 >					
Print ▾		Nombre	Fecha de modificación	Tipo	Tamaño
o		bin	01/09/2019 11:43 a.m.	Carpeta de archivos	
o		etc	01/09/2019 11:43 a.m.	Carpeta de archivos	
o		lib	01/09/2019 11:43 a.m.	Carpeta de archivos	
o		manual	01/09/2019 11:43 a.m.	Carpeta de archivos	
o		CONTRIBUTORS	01/09/2019 11:43 a.m.	Archivo	7 KB
		contributors	01/09/2019 11:43 a.m.	Documento XML	33 KB
		fetch	01/09/2019 11:43 a.m.	Documento XML	14 KB
		get-m2	01/09/2019 11:43 a.m.	Documento XML	5 KB
		INSTALL	01/09/2019 11:43 a.m.	Archivo	1 KB
		KEYS	01/09/2019 11:43 a.m.	Archivo	94 KB
		LICENSE	01/09/2019 11:43 a.m.	Archivo	15 KB
		NOTICE	01/09/2019 11:43 a.m.	Archivo	1 KB
		patch	01/09/2019 11:43 a.m.	Documento XML	2 KB
		README	01/09/2019 11:43 a.m.	Archivo	5 KB
		WHATSNEW	01/09/2019 11:43 a.m.	Archivo	250 KB

5.-我们安装了Git Bash。我们在安装过程中把所有的东西都留作默认。

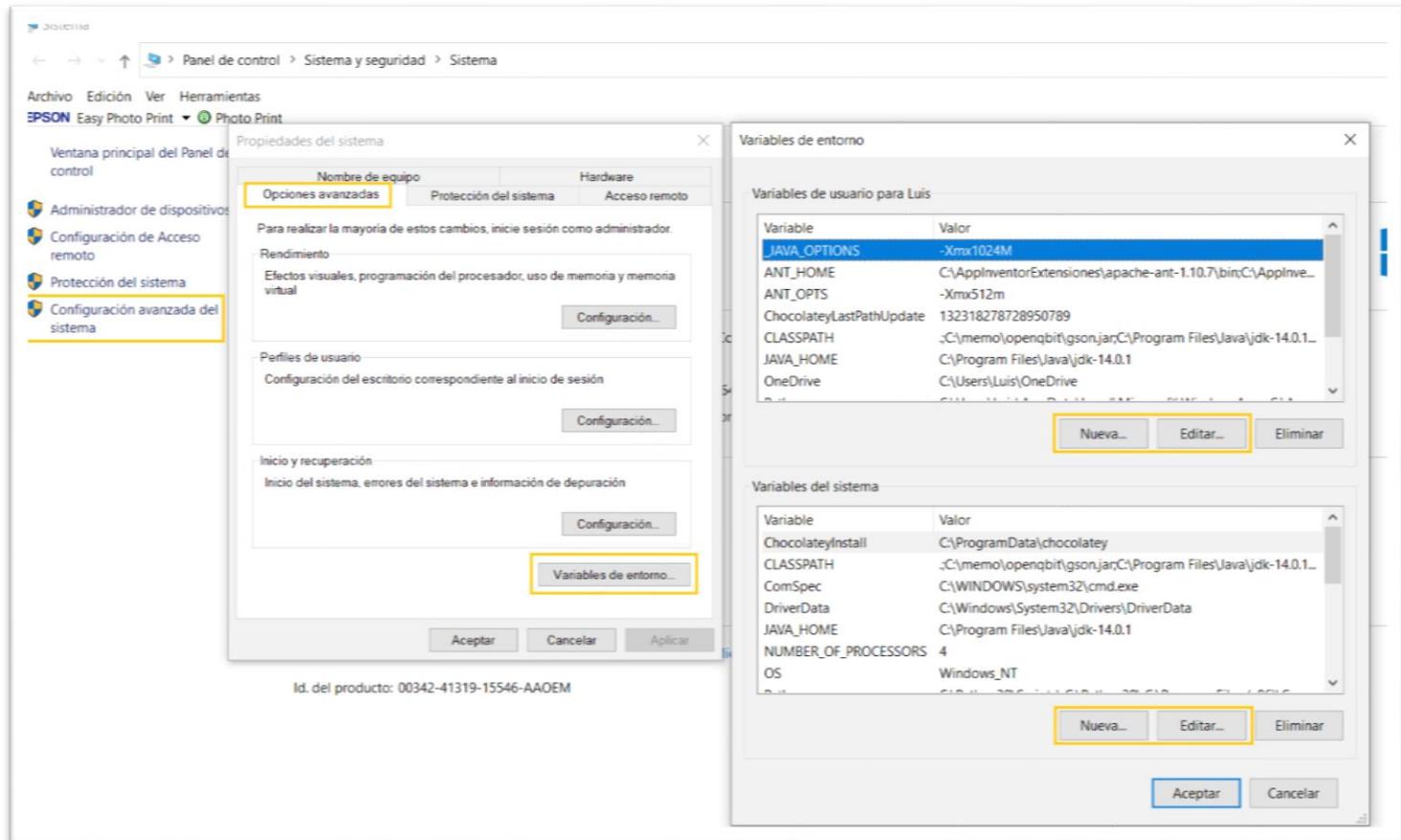
- 后面我们可以看到，在Windows的开始按钮中，通过点击文件浏览器中的左键，它有一个链接。



6.-我们安装了Java SE开发包。根据Windows的版本，你可能需要重新启动电脑。

Windows系统环境变量。我们将创建环境变量。为此，我们将：

控制面板->系统->高级系统设置->高级选项->环境变量。



根据我们是否要新建一个变量或编辑一个现有的变量，我们按"新建..."或"编辑..."按钮。

如果要在已经建立的地址上添加地址，我们用分号将它们分开。

在John的用户变量部分，我们设置了这些新...

JAVA_OPTIONS我们把你的值-Xmx1024m。

ANT_HOME, 我们把价值C:\AppInventorExtensions\apache-ant-1.10.8-bin[也就是我们解压apache-ant的文件夹]。

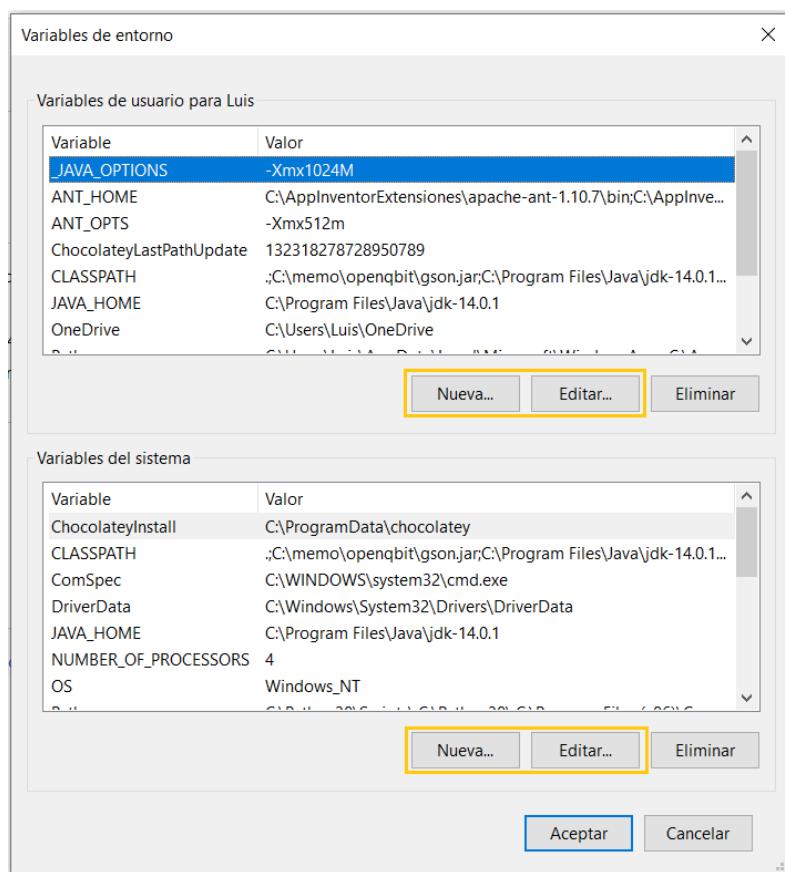
ANT_OPTS我们把价值-Xmx256M的

JAVA_HOME被设置为C:\Program Files\jdk1.8.0_131 [如果它有其他值, 请更改。注意是jdk而不是jdr]。

CLASSPATH我们把值%ANT_HOME%/lib;%JAVA_HOME%/lib的值放进去

在PATH中, 我们添加了;%ANT_HOME%/bin;%JAVA_HOME%/bin[注意;以分号开头;来添加到已有的内容中]。

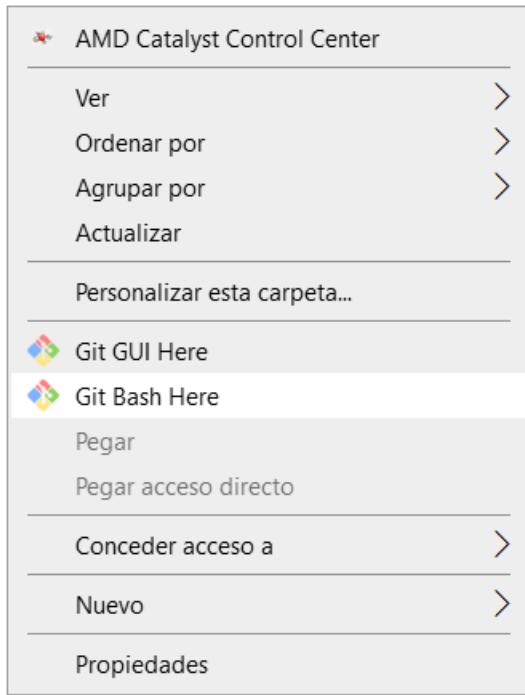
注意: 变量之间用分号隔开: 变量-1; 变量-n; 变量-n+1。



7. 在我们的电脑中创建App Inventor克隆。

我们将在我们的服务器（PC）上创建一个App Inventor的“克隆”副本，直接从互联网上下载并创建该副本。

要做到这一点，我们将使用**Git Bash**应用程序，并点击它打开一个终端。



我们在Git Bash终端上运行这个命令。

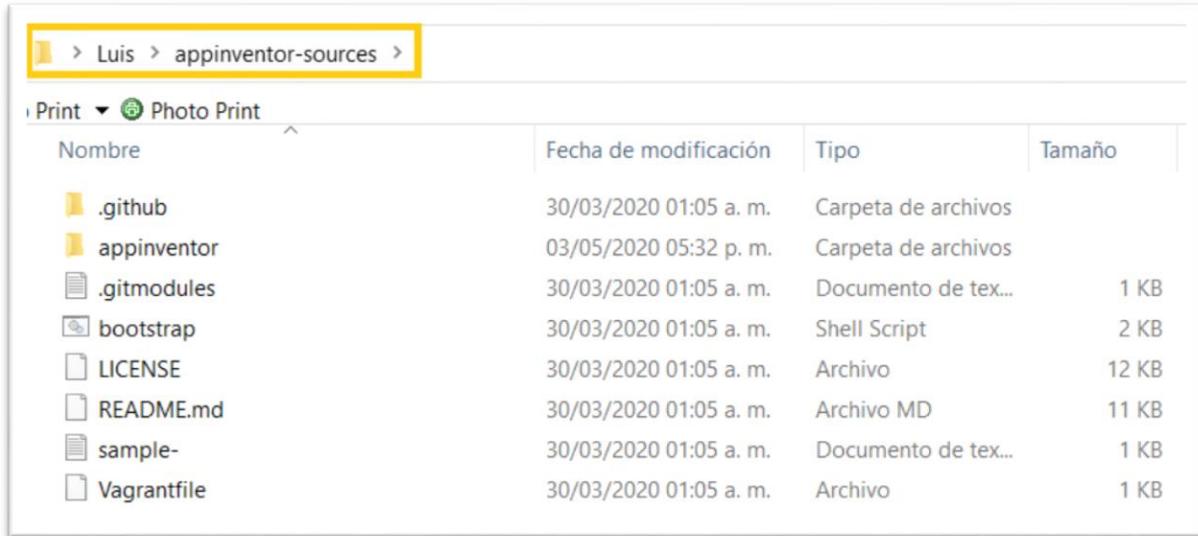
```
$ git clone https://github.com/mit-cml/appinventor-sources.git
```

A screenshot of a terminal window titled 'MINGW64 / C:/Users/Luis'. The command 'git clone https://github.com/mit-cml/appinventor-sources.git' is being run. The output shows the progress of cloning the repository, including object counting, receiving objects, and resolving deltas. The process is completed successfully with 41191 objects cloned.

储存库所在的网站。

<https://github.com/mit-cml/appinventor-sources/>

它将创建一个名为 appinventor-sources 的文件夹，其中包含 App Inventor 源码。

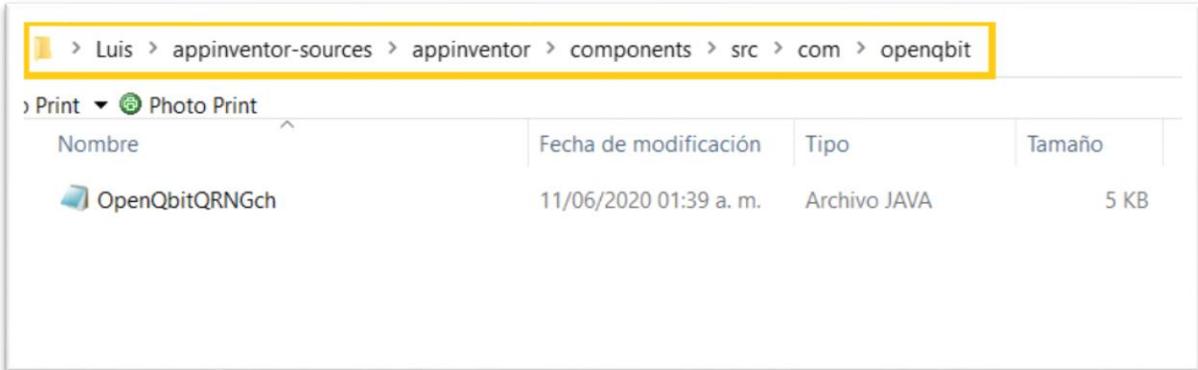


The screenshot shows a file explorer window with the following details:

Nombre	Fecha de modificación	Tipo	Tamaño
.github	30/03/2020 01:05 a. m.	Carpeta de archivos	
appinventor	03/05/2020 05:32 p. m.	Carpeta de archivos	
.gitmodules	30/03/2020 01:05 a. m.	Documento de tex...	1 KB
bootstrap	30/03/2020 01:05 a. m.	Shell Script	2 KB
LICENSE	30/03/2020 01:05 a. m.	Archivo	12 KB
README.md	30/03/2020 01:05 a. m.	Archivo MD	11 KB
sample-	30/03/2020 01:05 a. m.	Documento de tex...	1 KB
Vagrantfile	30/03/2020 01:05 a. m.	Archivo	1 KB

我们在路径C中创建了如下目录：用户-Appinventor-sourcesv-babkup-Appinventor-components-rc-openqbit。

在里面我们复制了我们下面的测试程序，名为OpenQbitQRNGch.java。



The screenshot shows a file explorer window with the following details:

Nombre	Fecha de modificación	Tipo	Tamaño
OpenQbitQRNGch	11/06/2020 01:39 a. m.	Archivo JAVA	5 KB

创立分机。

尊重大写字母和小写字母，这和你好我好大家好是不一样的。

不要加任何重音。我们已经准备好做第一个扩展，它将是量子随机数发生器。

我们使用文本编辑器Notepad++, 我们创建一个名为OpenQbitQRNGch.java的文件, 生成随机量子数, 代码如下。

```
// 该组件输出量子随机数生成器RNG瑞士API。

包 com.openqbit.OpenQbitQRNGch。
导入 com.google.appinventor.component.annotations.DesignerComponent。
导入 com.google.appinventor.component.annotations.DesignerProperty。
导入 com.google.appinventor.component.annotations.PropertyCategory。
import com.google.appinventor.component.annotations.SimpleEvent。
导入 com.google.appinventor.component.annotations.SimpleFunction。
导入 com.google.appinventor.component.annotations.SimpleObject。
import com.google.appinventor.component.annotations.SimpleProperty。
导入
com.google.appinventor.component.common.ComponentCategory.ComponentCatego
ry。
导入 com.google.appinventor.component.common.PropertyTypeConstants。
import com.google.appinventor.component.runtime.util.MediaUtil;
import com.google.appinventor.component.runtime.*。
import java.io.*;

@DesignerComponent(version = OpenQbitQRNGch.VERSION,
    description =
"API量子随机数生成器，量随数微的一种商务QRNGaaS，由瑞士公司开发Quantis量子随机数生成器的网络API - " +
"Guillermo Vidal - OpenQbit.com "。
   类别= ComponentCategory.EXTENSION
    nonVisible = true。
    iconName = "http://www.pinntar.com/logoQbit.png")
@SimpleObject(external = true)

public class OpenQbitQRNGch extends AndroidNonvisibleComponent implements
Component {

    public static final int VERSION = 1;
    public static final String DEFAULT_TEXT_1 = "";
    私有ComponentContainer容器
    private String text_1 = "";

    publicOpenQbitQRNGch(ComponentContainer container) {
        super(container.$form())。
this.container = container;
    }

    // 建议。
// @DesignerProperty(editorType =
.PropertyTypeConstants.PROPERTY_TYPE_STRING, defaultValue =
OpenQbitQRNGch.DEFAULT_TEXT_1 + "")
// @SimpleProperty(description = "API量子随机数生成器和之间随机数输入变量数量的数字生成")
```

```
@SimpleFunction(description = "API 独立于随机数生成器。和之闻随机数输入可变数 数量数字生成")
public APIGetQRNGdecimal(String qty) throws Exception {
    String url = "http://random.openqu.org/api/rand?size=" + qty;
    String[] command = { "curl", url };

    ProcessBuilder process = new ProcessBuilder(command);
    进程 p;
    p = process.start();
    BufferedReader reader = new BufferedReader(new
InputStreamReader(p.getInputStream()));
    StringBuilder builder = new StringBuilder();
    字符 line = null;
    而 (line = reader.readLine()) != null) {
        builder.append(line);
        builder.append(System.getProperty("行分隔符"));
    }
    String result = builder.toString();
    //System.out.print(result);
    返回结果;

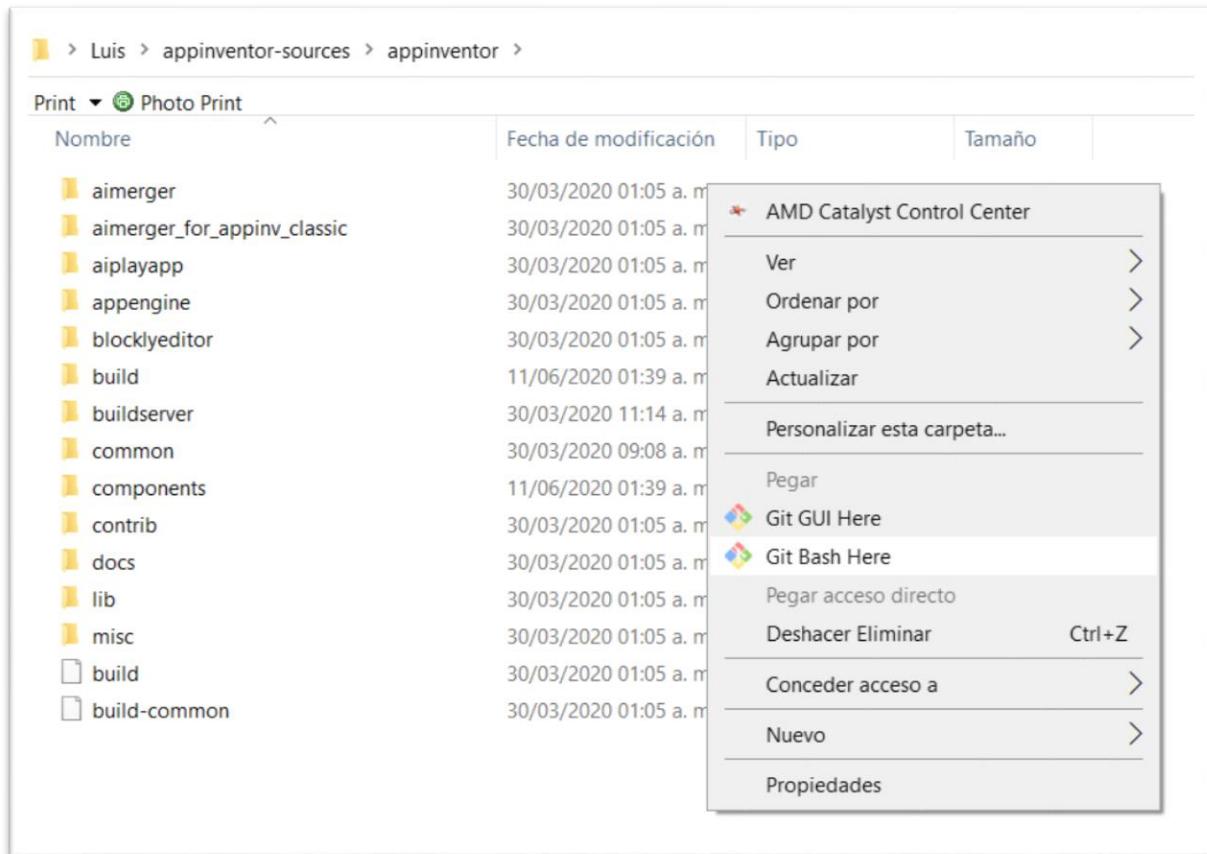
}

@SimpleFunction(description =
"API 独立于随机数生成器，在最小到最大的数字范围内随机数。输入可变数 数量 的数字来生成最小 最小 数和最大 最大 数的范围")
public String APIGetQRNGinteger(String qty, String min, String max)
throws Exception {
    String url = "http://random.openqu.org/api/randint?size=" + qty +
"&min=" + min + "&max=" + max;
    String[] command = { "curl", url };

    ProcessBuilder process = new ProcessBuilder(command);
    进程 p;
    p = process.start();
    BufferedReader reader = new BufferedReader(new
InputStreamReader(p.getInputStream()));
    StringBuilder builder = new StringBuilder();
    字符 line = null;
    而 (line = reader.readLine()) != null) {
        builder.append(line);
        builder.append(System.getProperty("行分隔符"));
    }
    String result = builder.toString();
    //System.out.print(result);
    返回结果;

}
}
```

我们执行Git Bash时，将自己定位在以下路径：C：Luis-appinventor-sources-appinventor。在这个目录下，我们点击鼠标左键，选择Git Bash终端。



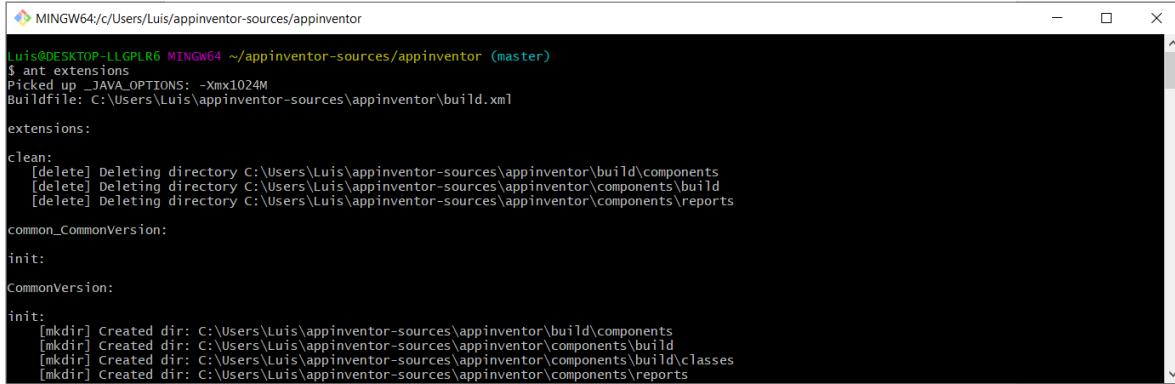
Git bash 终端将被放置在以下目录中。

C:Users/Luis/appinventor-sources/appinventor (master)

```
MINGW64:c/Users/Luis/appinventor-sources/appinventor
Luis@DESKTOP-LLGPLR6 MINGW64 ~/appinventor-sources/appinventor (master)
$ |
```

在Git Bash终端中，执行以下命令。

\$蚂蚁扩展



```
Luis@DESKTOP-LLGPLR6 MINGW64 ~/appinventor-sources/appinventor (master)
$ ant extensions
Picked up _JAVA_OPTIONS: -Xmx1024M
Buildfile: C:\Users\Luis\appinventor-sources\appinventor\build.xml

extensions:

clean:
[delete] Deleting directory C:\Users\Luis\appinventor-sources\appinventor\build\components
[delete] Deleting directory C:\Users\Luis\appinventor-sources\appinventor\components\build
[delete] Deleting directory C:\Users\Luis\appinventor-sources\appinventor\components\reports

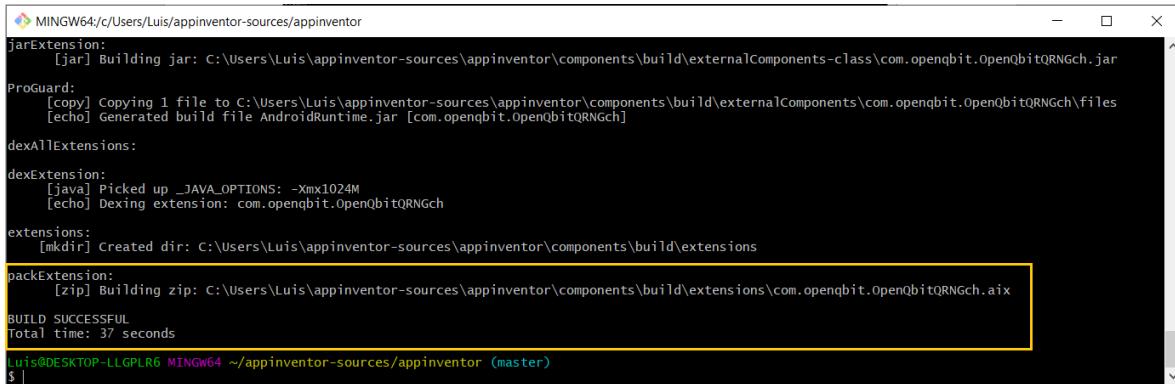
common_CommonVersion:
init:
CommonVersion:
init:
[mkdir] Created dir: C:\Users\Luis\appinventor-sources\appinventor\build\components
[mkdir] Created dir: C:\Users\Luis\appinventor-sources\appinventor\components\build
[mkdir] Created dir: C:\Users\Luis\appinventor-sources\appinventor\components\build\classes
[mkdir] Created dir: C:\Users\Luis\appinventor-sources\appinventor\components\reports
```

如果一切顺利，我们会得到它。成功地建立。

我们的分机已被创建在...

C:Users/Luis/appinventor-sources/appinventor/components/build/extensions

注意：每次我们做蚂蚁扩展时，扩展文件夹的内容都会被删除并重新创建。



```
Luis@DESKTOP-LLGPLR6 MINGW64 ~/appinventor-sources/appinventor (master)
jarExtension:
[jar] Building jar: C:\Users\Luis\appinventor-sources\appinventor\components\build\externalComponents-class\com.openqbit.OpenQbitQRNGch.jar
ProGuard:
[copy] Copying 1 file to C:\Users\Luis\appinventor-sources\appinventor\components\build\externalComponents\com.openqbit.OpenQbitQRNGch\files
[echo] Generated build file AndroidRuntime.jar [com.openqbit.OpenQbitQRNGch]
dexAllExtensions:
dexExtension:
[java] Picked up _JAVA_OPTIONS: -Xmx1024M
[echo] Dexing extension: com.openqbit.OpenQbitQRNGch
extensions:
[mkdir] Created dir: C:\Users\Luis\appinventor-sources\appinventor\components\build\extensions
packExtension:
[zip] Building zip: C:\Users\Luis\appinventor-sources\appinventor\components\build\extensions\com.openqbit.OpenQbitQRNGch.aix
BUILD SUCCESSFUL
Total time: 37 seconds
Luis@DESKTOP-LLGPLR6 MINGW64 ~/appinventor-sources/appinventor (master)
```

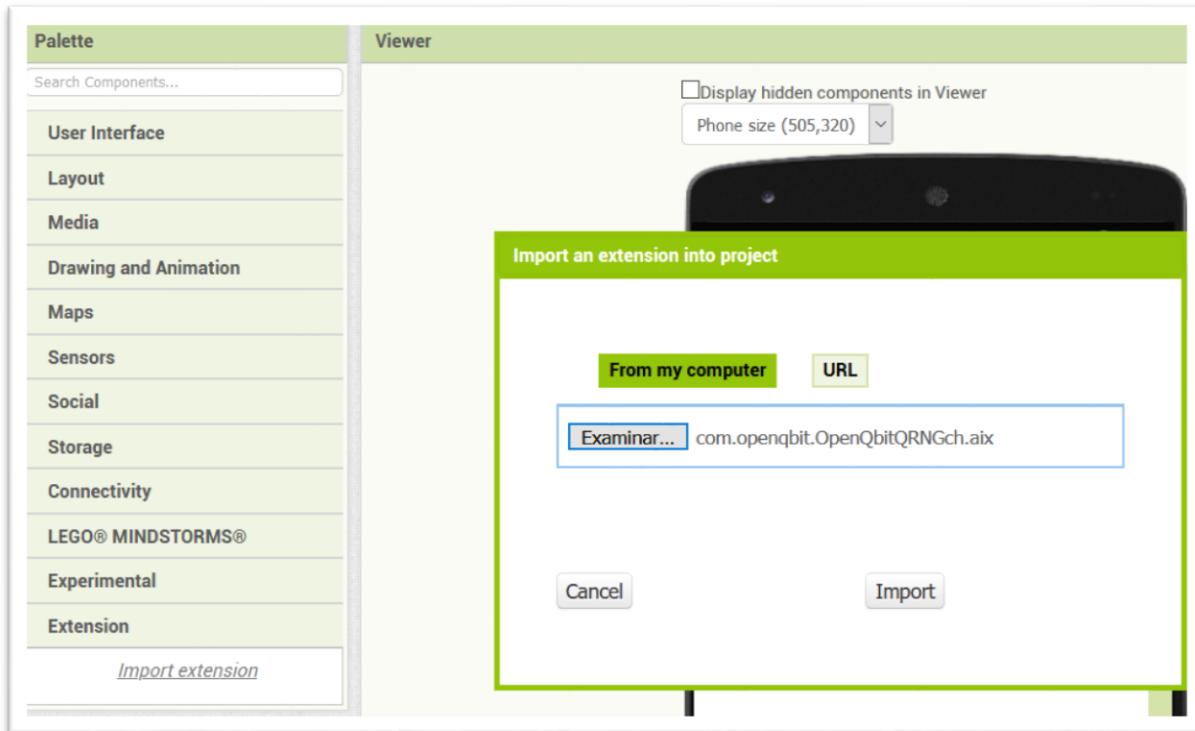
让我们去那个文件夹。

C:Users/Luis/appinventor-sources/appinventor/components/build/extensions

并复制文件com.openqbit.OpenQbitQRNGch.aix，这个文件是我们的扩展名。

在App Inventor或其他Blockly系统中已经有了账户，我们进入添加新的扩展并进行测试。

到最下面的扩展选项处，点击"导入扩展"。



我们点击"导入"按钮。现在我们有了要使用的块(APIGetQRNGdecimal)和(APIGetQRNGinteger)。



我们已经有了我们的第一个创建和功能扩展。

27.附件"BlocklyCode智能合约"。

BlocklyCodes是用java语言制作的程序。扩展创建这种类型的程序，俗称"智能合约"，是一种处理用户（公司或人）之间协议的方式。

这个区块（BlocklyCode），是在一个已经建立了参数的程序中实现的，根据协议的类型，可以由迷你BlocklyChain系统以自动的方式执行，当管理"智能合约"的前提是满足。

要考虑的参数建议或输入参数'输入'是指

失效日期

参考日期

过期时间

参考时间

参考资产

固定资产共计

可变资产

合同成员

确认数据(字符串)

确认的数据（文件名

可变事件

固定事件

审定文件

字符串验证

签字有效

定义的时间间隔(整数)

小数区间

既定的最低限度

既定上限

在开始使用区块(BlocklyCode)之前，我们必须首先安装系统，以执行"智能合约"，这是通过在终端安装Termux OpenJDK和OpenJRE完成的。

OpenJDK（开放的Java开发工具包）。- 它是用java开发程序的工具，它包含库、编译器和JVM（Java虚拟机）。

OpenJRE(Open Java Runtime Environment)。- 这是一个只用于运行java程序的工具。它包含JVM（Java虚拟机）。

我们继续在构成Mini BlocklyChain系统的节点的Termux终端上安装OpenJRE和OpenJDK

◦

我们安装的房间

\$ apt install - and wget

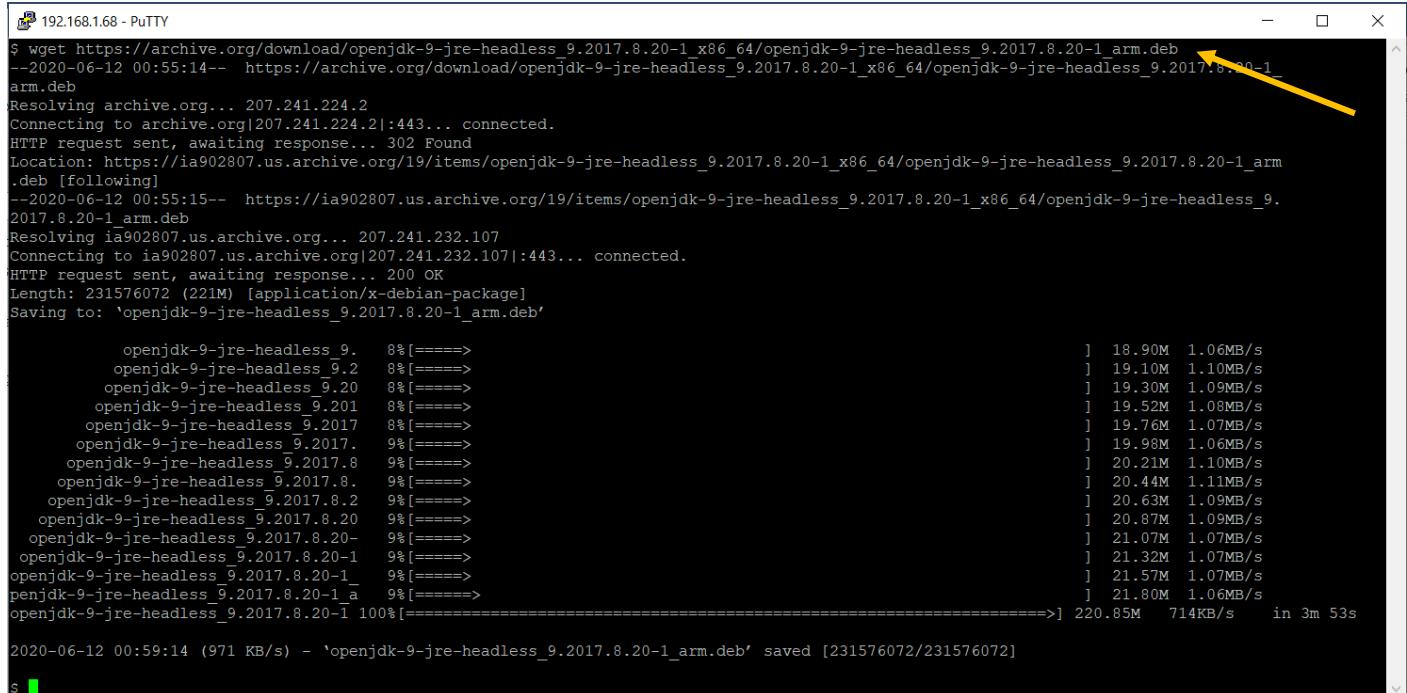


A screenshot of a Termux terminal window. The terminal shows the command \$ apt install wget being entered, with a yellow arrow pointing to the command. The output of the command follows, detailing the package installation process. Below the terminal is a standard QWERTY keyboard layout.

```
$ apt install wget
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following NEW packages will be installed:
  wget
0 upgraded, 1 newly installed, 0 to remove and 0
not upgraded.
Need to get 206 kB of archives.
After this operation, 430 kB of additional disk
space will be used.
Get:1 https://dl.bintray.com/termux/termux-packa
ges-24 stable/main arm wget arm 1.20.3-2 [206 kB]
Fetched 206 kB in 1s (128 kB/s)
Selecting previously unselected package wget.
(Reading database ... 16936 files and directorie
s currently installed.)
Preparing to unpack .../archives/wget_1.20.3-2_a
rm.deb ...
Unpacking wget (1.20.3-2) ...
Setting up wget (1.20.3-2) ...
$
```

我们下载了OpenJRE

\$ wget https://archive.org/download/openjdk-9-jre-headless_9.2017.8.20-1_x86_64/openjdk-9-jre-headless_9.2017.8.20-1_arm.deb



```
192.168.1.68 - PuTTY
$ wget https://archive.org/download/openjdk-9-jre-headless_9.2017.8.20-1_x86_64/openjdk-9-jre-headless_9.2017.8.20-1_arm.deb
--2020-06-12 00:55:14-- https://archive.org/download/openjdk-9-jre-headless_9.2017.8.20-1_x86_64/openjdk-9-jre-headless_9.2017.8.20-1_
arm.deb
Resolving archive.org... 207.241.224.2
Connecting to archive.org[207.241.224.2]:443... connected.
HTTP request sent, awaiting response... 302 Found
Location: https://ia902807.us.archive.org/19/items/openjdk-9-jre-headless_9.2017.8.20-1_x86_64/openjdk-9-jre-headless_9.2017.8.20-1_arm
.deb [following]
--2020-06-12 00:55:15-- https://ia902807.us.archive.org/19/items/openjdk-9-jre-headless_9.2017.8.20-1_x86_64/openjdk-9-jre-headless_9.
2017.8.20-1_arm.deb
Resolving ia902807.us.archive.org... 207.241.232.107
Connecting to ia902807.us.archive.org[207.241.232.107]:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 231576072 (221M) [application/x-debian-package]
Saving to: 'openjdk-9-jre-headless_9.2017.8.20-1_arm.deb'

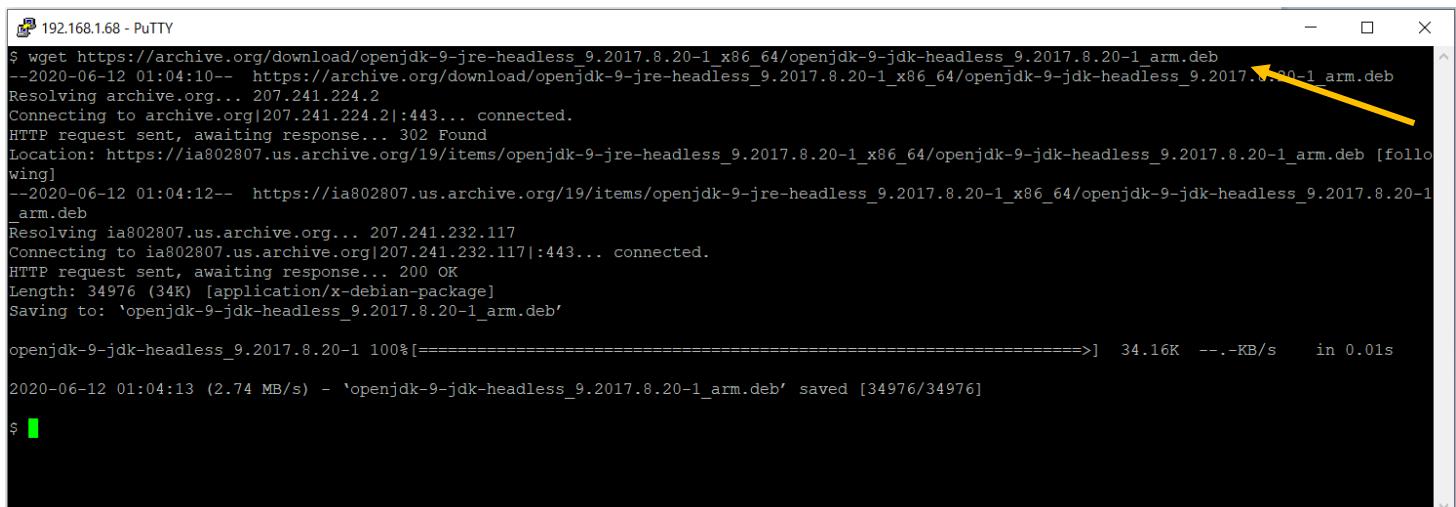
openjdk-9-jre-headless_9. 8%[=====] 18.90M 1.06MB/s
openjdk-9-jre-headless_9.2 8%[=====] 19.10M 1.10MB/s
openjdk-9-jre-headless_9.20 8%[=====] 19.30M 1.09MB/s
openjdk-9-jre-headless_9.201 8%[=====] 19.52M 1.08MB/s
openjdk-9-jre-headless_9.2017 8%[=====] 19.76M 1.07MB/s
openjdk-9-jre-headless_9.2017. 9%[=====] 19.98M 1.06MB/s
openjdk-9-jre-headless_9.2017.8 9%[=====] 20.21M 1.10MB/s
openjdk-9-jre-headless_9.2017.8. 9%[=====] 20.44M 1.11MB/s
openjdk-9-jre-headless_9.2017.8.2 9%[=====] 20.63M 1.09MB/s
openjdk-9-jre-headless_9.2017.8.20 9%[=====] 20.87M 1.09MB/s
openjdk-9-jre-headless_9.2017.8.20- 9%[=====] 21.07M 1.07MB/s
openjdk-9-jre-headless_9.2017.8.20-1 9%[=====] 21.32M 1.07MB/s
openjdk-9-jre-headless_9.2017.8.20-1_ 9%[=====] 21.57M 1.07MB/s
openjdk-9-jre-headless_9.2017.8.20-1_a 9%[=====] 21.80M 1.06MB/s
openjdk-9-jre-headless_9.2017.8.20-1 100%[=====] 220.85M 714KB/s in 3m 53s

2020-06-12 00:59:14 (971 KB/s) - 'openjdk-9-jre-headless_9.2017.8.20-1_arm.deb' saved [231576072/231576072]

$
```

我们下载了OpenJDK

\$ wget https://archive.org/download/openjdk-9-jdk-headless_9.2017.8.20-1_x86_64/openjdk-9-jdk-headless_9.2017.8.20-1_arm.deb



```
192.168.1.68 - PuTTY
$ wget https://archive.org/download/openjdk-9-jdk-headless_9.2017.8.20-1_x86_64/openjdk-9-jdk-headless_9.2017.8.20-1_arm.deb
--2020-06-12 01:04:10-- https://archive.org/download/openjdk-9-jdk-headless_9.2017.8.20-1_x86_64/openjdk-9-jdk-headless_9.2017.8.20-1_
arm.deb
Resolving archive.org... 207.241.224.2
Connecting to archive.org[207.241.224.2]:443... connected.
HTTP request sent, awaiting response... 302 Found
Location: https://ia802807.us.archive.org/19/items/openjdk-9-jdk-headless_9.2017.8.20-1_x86_64/openjdk-9-jdk-headless_9.2017.8.20-1_arm
.deb [following]
--2020-06-12 01:04:12-- https://ia802807.us.archive.org/19/items/openjdk-9-jdk-headless_9.2017.8.20-1_x86_64/openjdk-9-jdk-headless_9.2017.8.20-1
_arm.deb
Resolving ia802807.us.archive.org... 207.241.232.117
Connecting to ia802807.us.archive.org[207.241.232.117]:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 34976 (34K) [application/x-debian-package]
Saving to: 'openjdk-9-jdk-headless_9.2017.8.20-1_arm.deb'

openjdk-9-jdk-headless_9.2017.8.20-1 100%[=====] 34.16K --.-KB/s in 0.01s

2020-06-12 01:04:13 (2.74 MB/s) - 'openjdk-9-jdk-headless_9.2017.8.20-1_arm.deb' saved [34976/34976]

$
```

从Termux终端进行OpenJDK和OpenJRE的安装。

```
$ apt install - and ./openjdk-9-jre-headless_9.2017.8.20-1_arm.deb ./openjdk-9-jdk-headless_9.2017.8.20-1_arm.deb。
```



```
$ ls
openjdk-9-jdk-headless_9.2017.8.20-1_arm.deb
openjdk-9-jre-headless_9.2017.8.20-1_arm.deb
$ apt install -y ./openjdk-9-jdk-headless_9.2017.8.20-1_arm.deb ./openjdk-9-jre-headless_9.2017.8.20-1_arm.deb
Reading package lists... Done
Building dependency tree
Reading state information... Done
Note, selecting 'openjdk-9-jdk-headless' instead
of './openjdk-9-jdk-headless_9.2017.8.20-1_arm.
deb'
Note, selecting 'openjdk-9-jre-headless' instead
of './openjdk-9-jre-headless_9.2017.8.20-1_arm.
deb'
The following additional packages will be instal-
led:
  ca-certificates-java freetype libpng
The following NEW packages will be installed:
  ca-certificates-java freetype libpng
  openjdk-9-jdk-headless
  openjdk-9-jre-headless
0 upgraded, 5 newly installed, 0 to remove and 0
not upgraded.
Need to get 668 kB/232 MB of archives.
After this operation, 376 MB of additional disk
space will be used.
Get:1 https://dl.bintray.com/termux/termux-packa-
ges-24/stable/main arm ca-certificates-java all
20200101 [110 kB]
Get:2 https://dl.bintray.com/termux/termux-packa-
ges-24/stable/main arm libpng arm 1.6.37-2 [190
kB]
Get:3 https://dl.bintray.com/termux/termux-packa-
ges-24/stable/main arm freetype arm 2.10.2 [368
kB]
Get:4 /data/data/com.termux/files/home/openjdk/o-
penjdk-9-jre-headless_9.2017.8.20-1_arm.deb open-
jdk-9-jre-headless arm 9.2017.8.20-1 [232 MB]
Get:5 /data/data/com.termux/files/home/openjdk/o-
penjdk-9-jdk-headless_9.2017.8.20-1_arm.deb open-
jdk-9-jdk-headless arm 9.2017.8.20-1 [35.0 kB]
```

ESC ↵ CTRL ALT - ↓ ↑

▪ ◀ ○ □



```
Get:5 /data/data/com.termux/files/home/openjdk/o-
penjdk-9-jdk-headless_9.2017.8.20-1_arm.deb open-
jdk-9-jdk-headless arm 9.2017.8.20-1 [35.0 kB]
Fetched 668 kB in 23s (28.0 kB/s)
Selecting previously unselected package ca-certifi-
cates-java.
(Reading database ... 16939 files and directo-
ries currently installed.)
Preparing to unpack .../ca-certificates-java_202
00101_all.deb ...
Unpacking ca-certificates-java (20200101) ...
Selecting previously unselected package libpng.
Preparing to unpack .../libpng_1.6.37-2_arm.deb
...
Unpacking libpng (1.6.37-2) ...
Selecting previously unselected package freetype
.
Preparing to unpack .../freetype_2.10.2_arm.deb
...
Unpacking freetype (2.10.2) ...
Selecting previously unselected package openjdk-
9-jre-headless.
Preparing to unpack .../openjdk-9-jre-headless_9
.2017.8.20-1_arm.deb ...
Unpacking openjdk-9-jre-headless (9.2017.8.20-1)
...
Selecting previously unselected package openjdk-
9-jdk-headless.
Preparing to unpack .../openjdk-9-jdk-headless_9
.2017.8.20-1_arm.deb ...
Unpacking openjdk-9-jdk-headless (9.2017.8.20-1)
...
Setting up libpng (1.6.37-2) ...
Setting up freetype (2.10.2) ...
Setting up ca-certificates-java (20200101) ...
Setting up openjdk-9-jre-headless (9.2017.8.20-1)
...
Setting up openjdk-9-jdk-headless (9.2017.8.20-1)
...
$ █
```

ESC ↵ CTRL ALT - ↓ ↑

▪ ◀ ○ □

由于我们已经完成了OpenJDK和OpenJRE环境的安装。这些安装包含JVM(Java虚拟机),这将是运行BlocklyCode的环境。

现在我们将安装一个编辑器来在线创建文件, 我们将使用名为vi的编辑器执行以下命令。

```
$ apt install vim
```

现在让我们试着编译一个测试文件并运行它。我们在Termux终端中用vi编辑器创建并编写一个"Hello World"的java代码, 并将其保存在HelloWorld.java文件中。

```
public class HelloWorld {  
    public static void main(String[] args) {  
        // 打印你好世界到终端窗口。  
        System.out.println("Hello, World");  
    }  
}
```

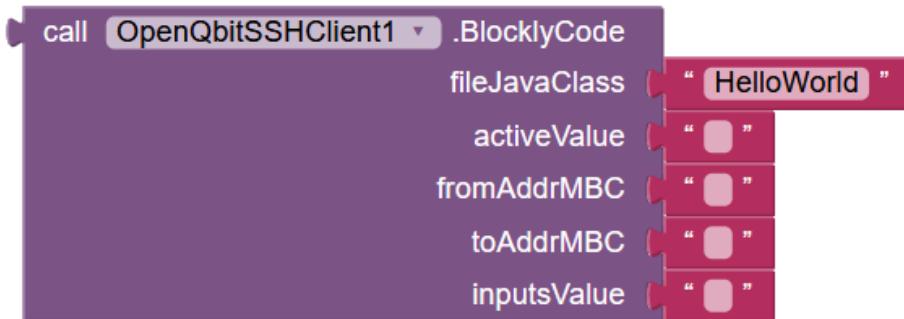
然后我们在Termux终端运行以下命令进行程序的编译, 然后执行程序。

```
$ javac HelloWorld.java (运行后, 会创建一个字节码的HelloWorld.class文件)
```

```
$ java HelloWorld (运行并打印广告后, Hello World)
```



使用块（**BlocklyCode**）这个块位于扩展（**ConnectorSSHClient**）中。



在前面的代码块中，你可以看到HelloWorld.class文件将如何执行，它已经被编译并定位在用户基础目录中。

它是一个区块，在这个区块中可以执行一个已经用java编译的程序，在开发环境中通常称为二进制形式的字节码文件。

这种类型的文件可以被Java虚拟机（JVM）执行。

创建字节码文件或扩展名为.CLASS的方式有两种，用户可以在自己的PC中舒适的创建，或者他的喜欢也可以在手机中创建，记得我们之前已经安装了编译JDK（Java开发工具包）的环境，虽然会因为键盘的限制而降低效率，也在选择Android环境的情况下要考虑到库被限制在已经安装的OpenJDK上。

输入参数在<inputsValue>中开放，其他固定的是activeValue、fromaddrMBC和toAddrMBC的参数。

在执行测试的情况下，它们可以没有内容而留有空白，只有字节码文件将被执行而不需要参数。

上一个区块就像下一个命令行正在执行。

\$ java HelloWorld

为BlocklyCode开发的程序将受制于每一个特定的设计环境，可以通过"cron"代理进行例行控制和执行，并与syncingmanager共享。

28. 附件"OpenQbit量子计算"。

量子计算是如何工作的？⁽²⁾

数字化转型给世界带来的变化比以往任何时候都要快，你会相信数字时代即将结束吗？**数字扫盲**已经被确定为一个领域，在这个领域，开放知识和学习技术的机会是迫切需要的，以解决社会和经济发展中的差距。随着另一个能够以惊人的速度和力量改变现有模式的新技术浪潮的即将到来，学习数字时代的关键概念将变得更加关键：**量子技术**。

在本文中，我们比较了传统计算和量子计算的基本概念；并开始探讨它们在其他相关领域的应用。

什么是量子技术？

纵观历史，人类通过科学了解自然界的运作方式，从而发展了科技。1900年至1930年间，对一些尚不十分清楚的物理现象的研究，产生了一种新的物理理论--**量子力学**。这一理论描述和解释了微观世界的功能，即分子、原子或电子的自然栖息地。由于这个理论，不仅可以解释这些现象，而且可以理解亚原子现实以一种完全反直觉的、近乎神奇的方式运作，在微观世界中发生了宏观世界中没有发生的事件。

这些**量子特性**包括量子叠加、量子纠缠和量子传送。

- **量子叠加**描述了一个粒子如何在同一时间处于不同的状态。
- **量子纠缠**描述了两个相距如意的粒子如何以这样的方式相关联，当与其中一个粒子相互作用时，另一个粒子会意识到它。
- **量子传送**利用量子纠缠将信息从一个地方传送到空间的另一个地方，而不需要穿越空间。

量子技术就是基于这些亚原子性质的量子特性。

在这种情况下，今天通过量子力学对微观世界的理解，使我们能够发明和设计能够改善人们生活的技术。使用量子现象的技术有很多，而且非常不同，其中一些技术，如激光或磁共振成像（MRI），已经陪伴我们半个多世纪了。然而，目前我们正在见证量子计算、量子信息、量子模拟、量子光学、量子计量、量子时钟或量子传感器等领域的技术革命。

什么是量子计算？首先，你要了解经典计算。

Carácter	Bits
7	111
A	01000001
\$	00100100
:)	0011101000101001

为了理解量子计算机的工作原理，我们可以方便地先解释一下我们日常使用的计算机（我们在本文中称之为数字计算机或经典计算机）的工作原理。这些和其余电子设备如平板电脑或手机一样，都是以比特为基本记忆单位。这意味着程序和应用程序是以比特为单位进行编码的，也就是以0和1的二进制语言进行编码。每当我们与这些设备进行交互时，例如按下键盘上的一个键，计算机内部就会产生、销毁和/或修改零和一的字符串。

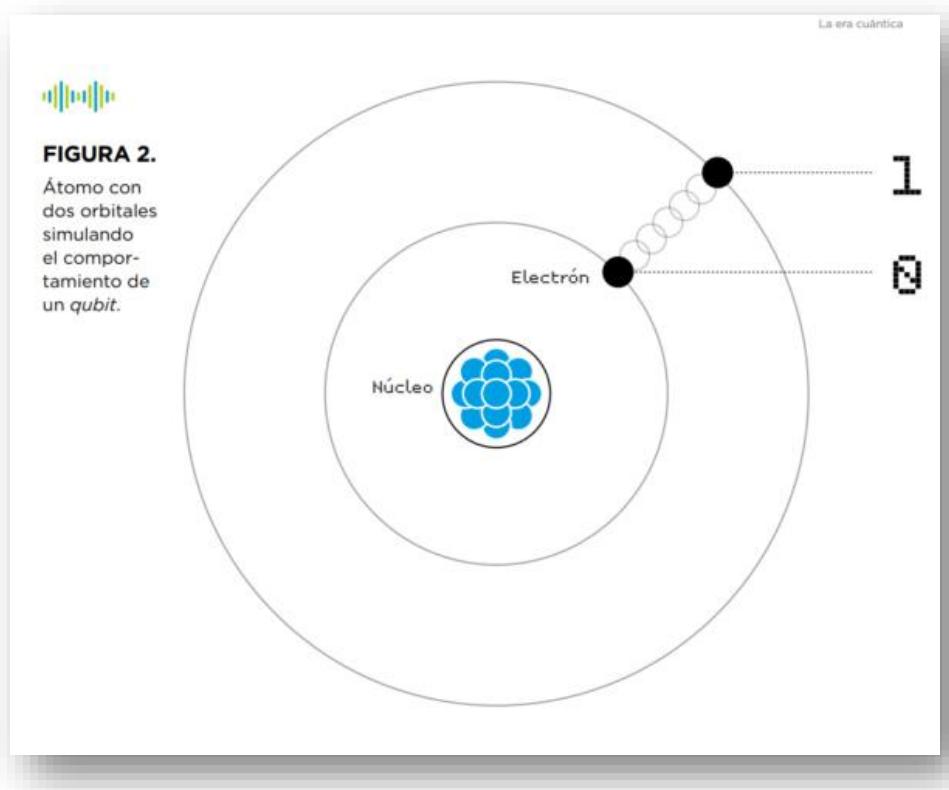
有趣的问题是，这些0和1在物理上是计算机内部的什么？零态和一态对应的是电流，这些电流通过称为晶体管的微观部件循环或不循环，晶体管作为开关。当没有电流流过时，晶体管为“关”，对应于位0；当有电流流过时，为“开”，对应于位1。

更简单地说，就好比0位和1位对应于空穴，所以空穴是0位，电子占据的空穴是1位。这就是为什么这些设备被称为电子器件的原因。举个例子，图1显示了一些字符的二

进制书写。现在我们对今天的计算机是如何工作的有了一个概念，让我们试着了解量子是如何工作的。

从比特到夸比特

量子计算中的基本信息单位是量子比特或qubit。根据定义，Qubits是两级量子系统--我们将在这里看到一些例子--它和比特一样，可以处于低级，对应于低激发状态或能量定义为0，或者处于高级，对应于高激发状态或定义为1。然而，与经典计算的根本区别就在于此，夸比特也可以处于0和1之间的任何一种无限的中间状态，比如一半0一半1的状态，或者是0的四分之三和1的四分之一。



量子算法，指数级的强大和高效计算。

量子计算机的目的是利用量子比特的这些

量子特性，

因为它们是量子系统，为了运行量子算法，使用重叠和交错的方式，提供比经典算法大得多的处理能力。重要的是要指出，范式的真正变化并不在于做与数字或经典计算机相同的事情--当前的计算机--但更快，正如在许多文章中可以读到的那样，而是

量子算法允许以一种完全不同的方式执行某些操作，在许多情况下，这种方式被证明是更有效的--也就是说，在更短的时间内或使用更少的计算资源--。

我们来看一个具体的例子。让我们想象一下，我们在波哥大，我们想知道在一百万个去利马的选择中，哪一条是去利马的最佳路线 ($N=1,000,000$)。为了利用计算机找到最优路线，我们需要将100万个选项数字化，这意味着对经典计算机来说，要将它们翻译成比特语言，对量子计算机来说，要将它们翻译成*qubits*。经典的计算机需要逐一分析所有的路径，直到找到所需的路径，而量子计算机则利用了被称为量子并行的过程，使其可以同时考虑所有的路径。这意味着，经典计算机需要 $N/2$ 步或迭代的顺序，即50万次尝试，而量子计算机只需要对注册表进行 \sqrt{N} 次操作，即1000次尝试，就能找到最优路径。

在前一种情况下，其优势是二次方的，但在其他情况下，其优势甚至是指数的，也就是说，在 n 个*qubits*的情况下，我们可以获得相当于 2^n 位的计算能力。为了说明这一点，人们通常会计算，在一台量子计算机中，我们可以拥有更多的基态--更多不同的和同时出现的字符串--比宇宙中的原子数量还要多，据估计，宇宙中的原子数量约为280个。另一个例子是，据估计，如果有一台2000到2500个基态的量子计算机，我们几乎可以破解今天使用的所有密码学（所谓的公钥密码学）。

为什么要了解量子技术？

我们正处在一个数字化转型的时刻，不同的新兴技术，如区块链、人工智能、无人机、物联网、虚拟现实、5G、3D打印机、机器人或汽车等，越来越多地出现在多个领域和行业。这些技术的目的是提高人类的生活质量，加速发展并产生社会影响，如今这些技术也在同步发展。只是我们很少看到公司开发利用其中两种或多种技术组合的产品，比如区块链和物联网或无人机和人工智能。虽然它们注定要融合，从而产生成倍的影响，但由于它们所处的发展初期阶段，以及具有技术专长的开发人员和人员的稀缺，意味着融合仍是一项有待完成的任务。

由于量子技术具有颠覆性的潜力，预计其不仅会与所有这些新技术相融合，而且会对几乎所有的新技术产生交叉影响。量子计算将威胁到数据的认证、交换和安全存储，对那些密码学有较多相关作用的技术有重大影响，如网络安全或区块链，负面影响不大，但也要考虑5G、物联网或无人机等技术。

你想练习量子计算吗？

网络上已经有几十个量子计算机模拟器，不同的编程语言已经在使用，如C、C++、Java、Matlab、Maxima、Python或Octave。另外，微软推出的Q#等新语言。你可以通过IBM和Rigetti等平台探索和玩转虚拟量子机。

Mini BlocklyChain是由OpenQbit.com公司创建的，该公司专注于为不同类型的私人和公共部门开发量子计算技术。

为什么Mini BlocklyChain与其他区块链不同，只是因为系统是为了模块化和包含量子计算而创建的。

- (2) <https://blogs.iadb.org/conocimiento-abierto/es/como-funciona-la-computacion-cuantica/>

29. 附件"SQLite数据库的扩展块"。

要查看更多的细节，正确和及时地使用每一个块，组成扩展OpenQbitSQLite检查原作者的扩展在下面的链接。

OpenQbitSQLite是对原设计的克隆，并做了一些小的修改，以用于AES安全级别。

<https://github.com/frdfsnlght/aix-SQLite>

30. 附件"Mini BlocklyChain系统创建实例"。

我们将制作一个测试系统，在这个系统中，我们可以验证创建Mini BlocklyChain系统的功能、优势和易用性。

我们将从设计和开发存储开始，信息将驻留在我们已经定义为区块链的地方。设计将基于SQLite数据库，根据每个组织或设计，这可以很容易地改变另一个数据库，如Microsoft SQL Server, MySQL, Maria DB, Oracle, DB2, PostgreSQL等。虽然同样由于事务处理和并发到SQLite数据库的过程中，可以在任何级别和企业或个人应用中使用。

在这个名为**minibc**的数据库创建中，将有一个存储块字符串的表。这个数据库将被嵌入到最终的程序代码中，安装在节点中，这个存储的地方叫做"assets packaged"是App Inventor等Blockly环境中的一个内部物种。

\$ sqlite3 minibc.db

SQLite版本3.32.2 2020-06-20 15:25:24

输入".help"以获得使用提示。

连接到一个短暂的内存数据库。

使用".open FILENAME"在持久性数据库上重新打开。

sqlite> .quit

设计**nblock**表的字段。

```
CREATE TABLE nblock (
    id整数主键 AUTOINCREMENT
    , prevhashVARCHAR NOT NULL
    , newhashVARCHAR NOT NULL
    ntransINTEGER NOT NULL
    nonceINTEGER NOT NULL
    ,qrng      INTEGER NOT NULL
);
```

我们创建了**nblock**表。

\$ sqlite3 minibc.db

SQLite版本3.32.2 2020-06-20 15:25:24

输入".help"以获得使用提示。

连接到一个短暂的内存数据库。

使用".open FILENAME"在持久性数据库上重新打开。

sqlite> .open minibc.db。

sqlite> CREATE TABLE nblock (id integer 主键 AUTOINCREMENT , prevhash VARCHAR NOT NULL , newhash VARCHAR NOT NULL, ntrans INTEGER NOT NULL ,nonce INTEGER NOT NULL ,qrng INTEGER NOT NULL).

我们会看到类似的东西。

A screenshot of an Android device's terminal window. The terminal shows the following SQLite session:

```
$ sqlite3 minibc.db
SQLite version 3.32.2 2020-06-04 12:58:43
Enter ".help" for usage hints.
sqlite> CREATE TABLE nblock (
...>     id integer primary key AUTOINCREMENT
...> , prehash VARCHAR NOT NULL
...> , newhash VARCHAR NOT NULL
...> , ntrans INTEGER NOT NULL
...> , nonce INTEGER NOT NULL
...> , qrng INTEGER NOT NULL);
sqlite> .tables
nblock
sqlite> .quit
$
```

The terminal includes a standard Android keyboard at the bottom.

创建nblock表的SQL语句。

接下来我们将创建系统的初始哈希，称为“创世”，这是基于创建一个新的哈希，我们将使用区块链的第一个区块（NewBlock）。然后我们会根据“创世”哈希创建第二个哈

希，我们称之为"一"，因为它必须与第一个哈希保持一致，因为初始区块链上的哈希验证测试必须始终符合：prevhash = newhash。

NewBlock.为了创建"创世"哈希，我们将使用输入参数。



输入参数：data <String>, previousHash <String>。

输出参数：一个SHA256哈希值。

在这种情况下，我们将使用"previousHash"初始等于"0"，在输入数据的情况下，"data"将是"Genensis"一词。

这将给我们一个计算出的两个数据组合的哈希值。

SHA256 (Genesis0) = eca6a17bbaeaafedb4db858843ad7a25479c2a99cb6b5adfbb09ba54e802e642。

上述字符串将被插入到nblock表中，这个字符串必须被加密，为此我们使用了扩展（OpenQbitEncDecData）。



我们将使用OpenQbitSSHClient扩展来插入minibc.db数据库中的数据，它将对minibc.db数据库中的nblock表进行INSERT语句。

```
sqlite3 keystore.db "insert into nblock (prevhash, newhash, ntrans, nonce, qrng) values ('0',
'eca6a17bbaeaafedb4db858843ad7a25479c2a99cb6b5adfb09ba54e802e642', '1', '0', '0');"
```

我们根据"Genensis"哈希值，用SQL语句创建了"一个"哈希值。

```
sqlite3 keystore.db "insert into nblock (prevhash, newhash, ntrans, nonce, qrng) values (
eca6a17bbaeaafedb4db858843ad7a25)db "插入到nblock中(prevhash, newhash, ntrans,
nonce, qrng)值('
```

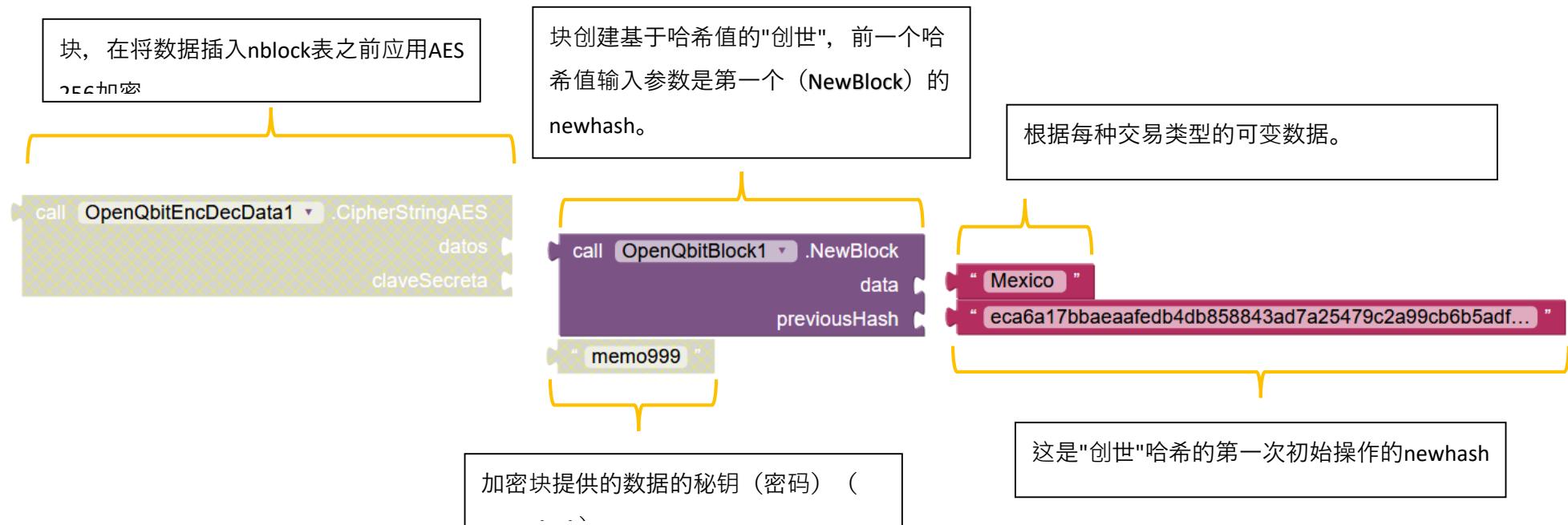
```
eca6a17bbaeaafedb4db858843ad7a25479c2a99cb6b5adfb09ba54e802e642'.'
```

```
f6a6b509376deebc8a5d70da9d0436943b76c3933f35c419ed48e78abab59a68','1','0','0')。"
```

哈希"一"newhash的计算方法为：。

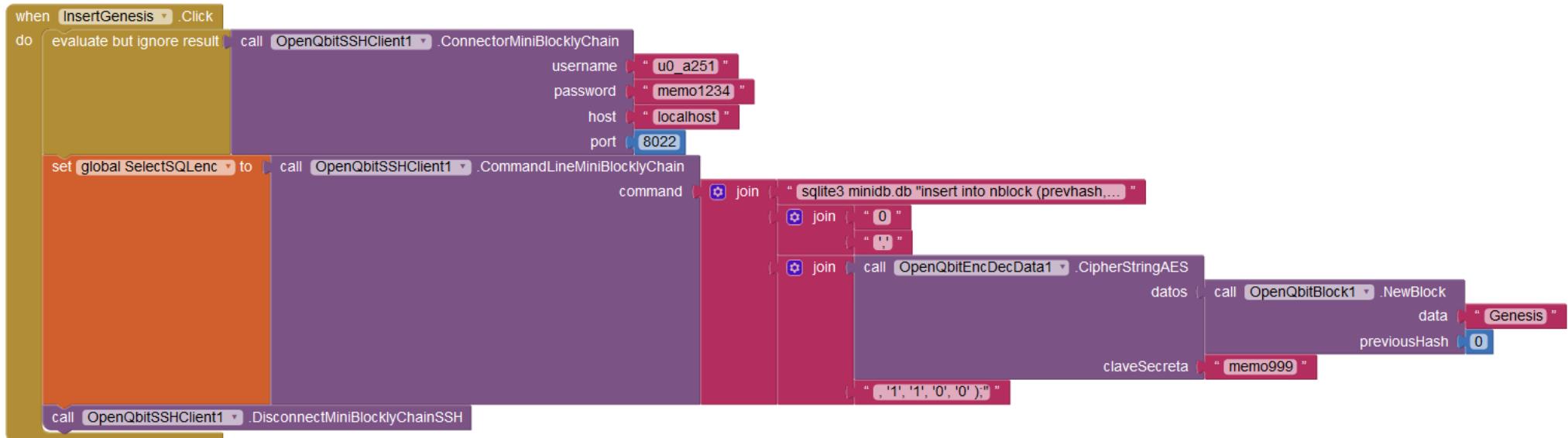
SHA256 (eca6a17bbaeaafedb4db858843ad7a25479c2a99cb6b5adfbb09ba54e802e642Mexico) =
f6a6b509376deebc8a5d70da9d0436943b76c3933f35c419ed48e78ab59a68。

第二个哈希基于第一个初始化哈希"创世"，我们必须在开始时做两个inser，因为区块链的任何初始验证都必须符合字段的平等性：prevhash（倒数第二个数据）=newhash（最后一个数据）。



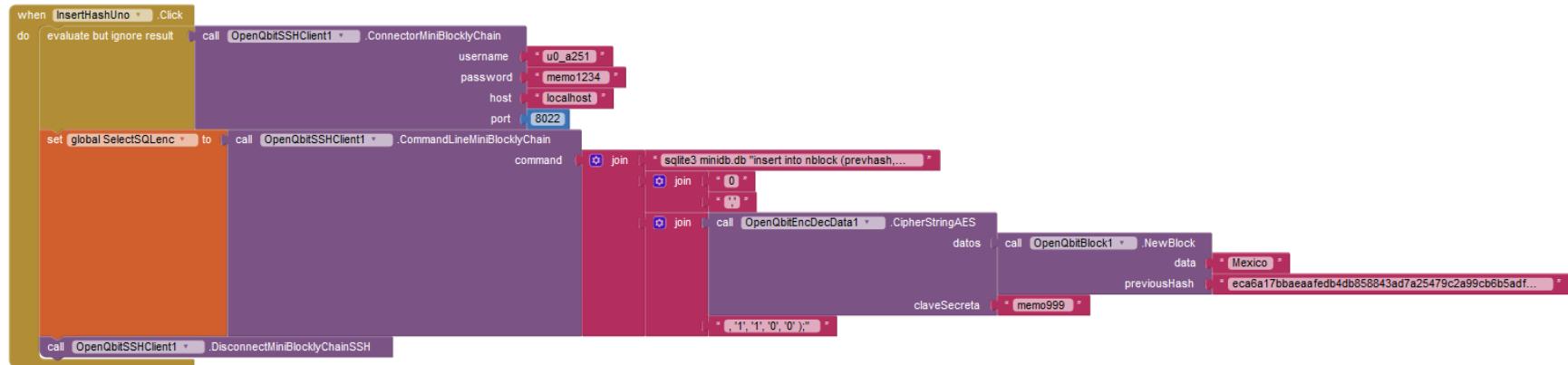
我们继续在App Inventor中创建执行上面的哈希；"创世"哈希和"开始"哈希。

现在，我们展示一下"创世"哈希的INSERT将如何整合到App Inventor系统中nblock表内的区块链初始结构中。



由于我们已经根据"创世"哈希值在nblock表中插入了第一个数据，所以我们继续进行引用"一"哈希值的第二个INSERT。

它将把"一"字哈希的INSERT整合到App Inventor系统中nblock表内的区块链初始结构中。



前面的两个编程结构（哈希"创世"和哈希"一"）应该是由Mini
minibc.db数据库及其所有的内部结构（表）将通过Mini BlocklyChain网络基于"Peer to Peer"架构进行复制。

BlocklyChain系统的初始节点整合而成的，重要的一点是

到目前为止，我们已经完成了区块链存储的基本和基础结构，它已经准备好接收未来系统中发起或请求的新区块。

我们已经将第一个哈希数据"Genesis"和哈希值"one"插入到minibc.db数据库中，现在我们将使用下面的SQL语句查询nblock表中的prevhash和newhash字段。

这一点将是最基本的，因为根据这两个字段的比较，我们将知道区块链是否是一致的，是否维护了交易的完整性和安全性。这只是一个审查区块链的起始机制，因为在未来，我们将审查使用区块来计算merkle树，这将是另一个必不可少的工具，以确保区块链在我们系统中的完整性和安全性也。

现在，我们将只审查我们必须使用的结构或SQL句子，就像我们之前使用扩展（OpenQbitSSHClient）一样，有了这些，我们就可以查阅prevhash和newhash字段。以后我们可以在每次要增加一个新的块时，对该检查进行简单的等值数据比较。

对于prevhash。

```
sqlite3 minibc.db "SELECT prevhash FROM nblock ORDER BY id DESC LIMIT 1;"
```

对于新哈什。

```
sqlite3 minibc.db "SELECT newhash FROM nblock ORDER BY id DESC LIMIT 1;"
```

现在我们将重点介绍如何提出请求，提交新的交易和/或交易，以便插入交易队列中。
。

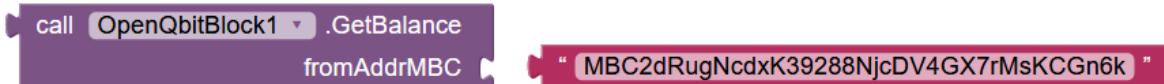
有两个步骤作为要求将一个事务发送到事务队列中。

第一个要求是我们的账户余额要有足够的"资产"来支付要寄的金额。请记住，"资产"不仅可以是一个数字或金额，我们可以根据每个要创建的系统创建任何形式的"资产"。
。

资产的例子：

- ✓ 内在量的"虚拟或加密货币"量的新起源。
- ✓ 参考数字数据的数据(所有类型的文件)
- ✓ 字符串或所有类型的文件的哈希认证签名。
- ✓ 数字信息或其代表的任何交易或转移；

使用块（GetBalance）获得每个节点的"资产"余额。



第二个要求是在发送交易时提供最小的参数，它们是

- ✓ 来源地址：- 是资产的发货地址。
- ✓ 目的地地址：- 是接收发送资产的用户的地址。
- ✓ 活跃。- 它是每个系统中给出的具有内在价值的数据，要在每次交易中发送。

以上地址（原点-目的地）对应的是每个用户账户创建时的公钥。记住，在创建用户账户时，会创建两种类型的公钥和私钥。

公钥是将在整个系统中共享的地址，系统的任何用户都可以看到并用于发送或接收交易。

私钥是每个节点在本地使用的地址，正如它的名字所示，它是供私人使用的，有助于创建数字签名，给发送的交易提供安全保障，这个密钥永远不应该被共享，它是本地使用的，对源节点来说是唯一的。

为了使用前面的参数，我们将依靠两个存储库，其中存储在keystore.db数据库中的"私钥"地址，将是每个节点的本地使用，而不会在系统中共享；另一个存储库中存储的所有"公钥"地址，即publickeys.db数据库，将通过"点对点"协议在系统的所有节点中共享。

数据库"keystore.db"和"publickeys.db"已经在附件"KeyStore & PublicKeys数据库创建"中创建。

在"RESTful Mini Sentinel网络的安装和配置"一节中已经创建了事务队列的数据库和系统。我们已经为交易创建了一个名为op.sqlite3的数据库，在这个数据库中，我们有两个表，一个是"trans"，负责处理交易（**交易队列**），另一个名为"sign"，每个操作的数字签名都存储在这里。

SQLite RESTful系统是这次的备份网络，为了方便和测试备份系统，我们将使用它，但是，如果要在"Peer to Peer"模型中改变和使用同一个数据库op.sqlite3，我们只有在同步系统中再次使用共享模型中的数据库，这个我们在后面会看到。

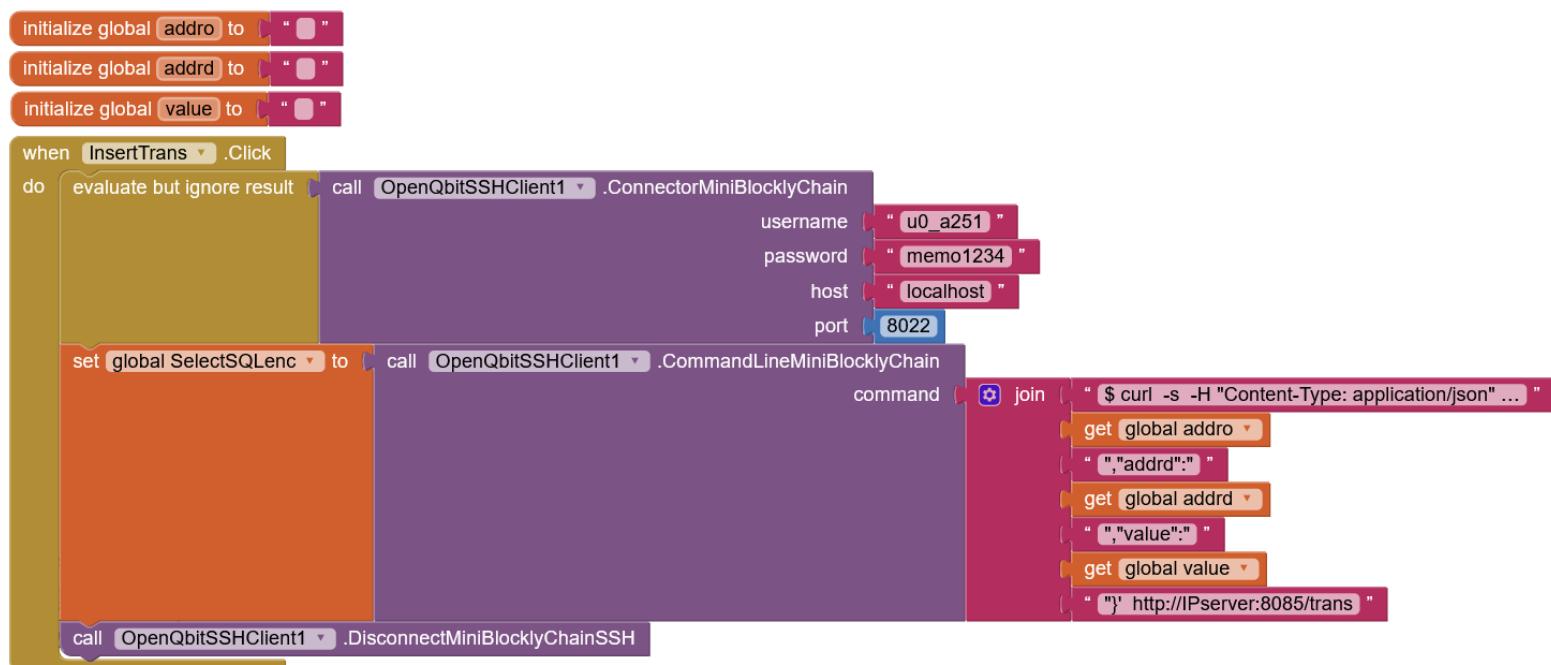
我们将开始使用应用于op.sqlite数据库的RESTful向事务队列发送操作。

我们在"trans"表中创建了一个新的事务。

```
$ curl -s -H "Content-Type: application/json" -d
'{"addr": "MBCNqkUDgQ6t48WhyoLTf5XxypgfzLUQ7Kj9",
"addrd": "MBCTDY1F2FvRjKCEabwTUa3EaE8BiM1Qqsh", "value": "999"}'
http://IPserver:8085/trans
```

```
#产出
{
  "id": 1,
  "addr": "MBCNqkUDgQ6t48WhyoLTf5XxypgfzLUQ7Kj9",
  "addrd": "MBCTDY1F2FvRjKCEabwTUa3EaE8BiM1Qqsh",
  "值": "999"
}
```

在App Inventor中实现。



输入参数：addro、addrd、value是可以输入算法的变量，是从keystore.db数据库中提取出来的

参见附录"KeyStore数据库的创建"。

我们现在插入上一笔交易的数字签名。在"标志"表中

```
$ curl -s -H "Content-Type: application/json" -d '{ }。
"trans_id":1
"标志" :"59ahGVn8pZ4oduxtTWCCvRiqjWEEy1c62"。
"hash": " f6a6b509376deebc8a5d70da9d0436943b76c3933f35c419ed48e78abab59a68"}'
http://IPserver:8085/sign

#产出
{
"id": 1,
"trans_id":1,
"标志" :"59ahGVn8pZ4oduxtTWCCvRiqjWEEy1c62"。
"has" :"f6a6b509376deebc8a5d70da9d0436943b76c3933f35c419ed48e78ab59a68"
}
```

注意：数字签名（sign）必须在发送curl命令语句之前获得，它是由块（GenerateSignature）生成的。

输入参数：Trans_id、sign、hash是可以输入算法的变量，交易的数字签名将随着区块（GenerateSignature）生成。

现在我们将看到生成数字签名的过程，该签名将应用于每一笔交易。

为交易生成数字签名；当我们使用块(GenerateSignature)时，我们会有一个结果文件类型为.sig这个文件我们会用它来保存在表"sign"的sign字段中，这个signature将在交易队列处理时使用，它是另一个参数，用来给始发地和目的地之间的安全，以及他们之间的交易金额或类型。

为了处理像file.sig这样的二进制数据，我们必须将其转换为base64，然后存储在"sign"表的sign变量中。为此，我们将使用块（EncoderFileBase64）。

每次交易的"标志"表的哈希字段值如何计算。

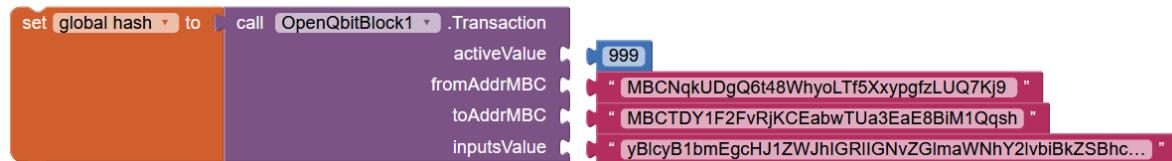
交易哈希=SHA256(源地址+目的地址+资产+fileBase64.sig)

哈希类型SHA256的例子。

SHA256(MBCNqkUDgQ6t48WhyoLTf5XxypgfzLUQ7Kj9+MBCTDY1F2FvRjKCEabwTUa3EaE8BiM1Qqsh+999+)。

产出：9d45198faaef624f2e7d1897dd9b3cede6ecca7fbac516ed1756b350fe1d56b4。

对于哈希值的计算，我们将不得不依靠Block(**Transaction**)。



输出参数是系统中任何节点发送的每一个事务的哈希值，它存储在基础op.sqlite中"sign"表的sign字段中。

通过前面的操作，我们确保只有一个唯一且不可重复的哈希值才能代表每一个发送到队列中的事务，而这个代表哈希值就是传输信息的全部。

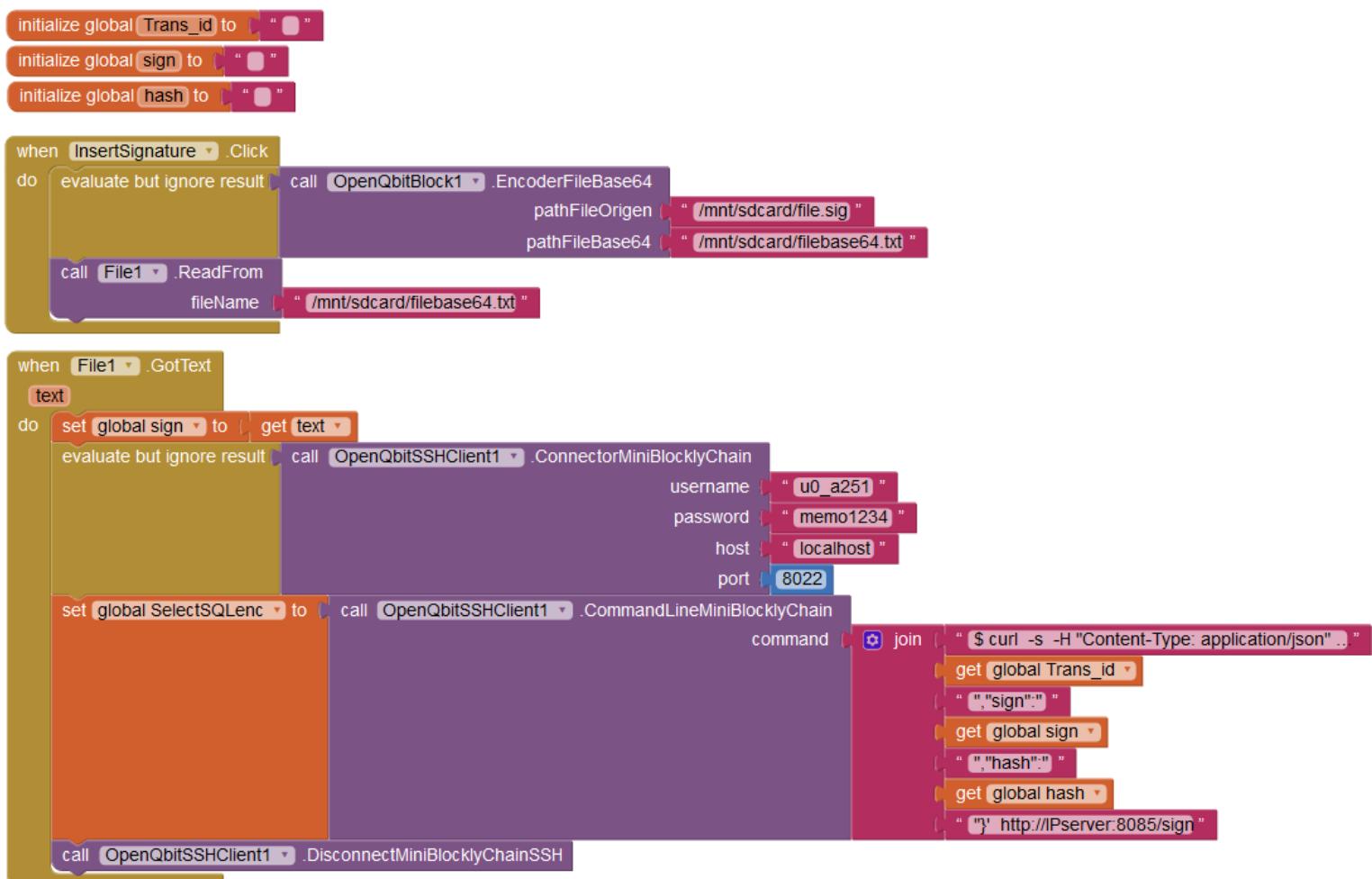
唯一缺少的就是包含Trans_id变量，它可以作为一个标识符来区分每笔交易发送的是哪个节点，在我们的例子中，我们会把Trans_id变量的固定值为"1"。因为它是我们网络中配置的第一个节点。这个值可以根据每个特定的设计而改变语法，所以为了简单起见，我们选择了一个简单的标识符。

由于我们已经定义了所有的变量，我们将在App Inventor中创建块的结构和执行顺序，以便在表"sign"中执行INSERT。

注意：必须考虑到每一次发送事务都是由op.sqlite3数据库中的两个INSERTS组成，其中一个必须在"trans"表中完成，而它们各自的值则在"sign"表中完成。

在op.sqlite3数据库的设计中，由于将来有可能只对"sign"表进行加密，因为该表中包含了不应在网络中平铺直叙的信息，所以建立了两张独立的表，这个方案留待将来每次设计时考虑。

我们将在表"sign"的INSERT的App Inventor里面创建块和方法的执行结构。



到此为止，我们已经完成了对"sign"表和"trans"表的INSERT，这两张表都在op.sqlite3数据库中，插入到这两张表中的数据将被用来创建事务队列，并发送给所有节点进行处理。

此时我们将使用Java SQLite-Redis连接器，该连接器将执行op.sqlite3数据库中的数据向Redis数据库的转换。参见附录"Java SQLite-Redis代码连接器"。

在实现了SQLite-Redis连接器后，事务队列将通过Redis系统传递给系统的所有节点。

我们将开始如何以适当的方式来接收事务队列，以便能够处理它。

事务队列将通过Redis服务交付。这是一个实时数据库，在App Inventor环境中拥有各自的控制块。

在App Inventor的Blockly系统中配置Redis环境。

需要注意的重要一点是，截至编写本手册时，用于控制Redis服务器的块只在App Inventor系统中可用。如果使用与App Inventor不同的Blockly系统，你将不得不使用Redis CLI（命令行）执行，通过扩展（OpenQbitSSHClient）向Termux终端发送命令。

Redis CLI（命令行）中的使用实例。

要在Redis获得所有的标签或密钥。

\$ redis-cli KEYS '*'。

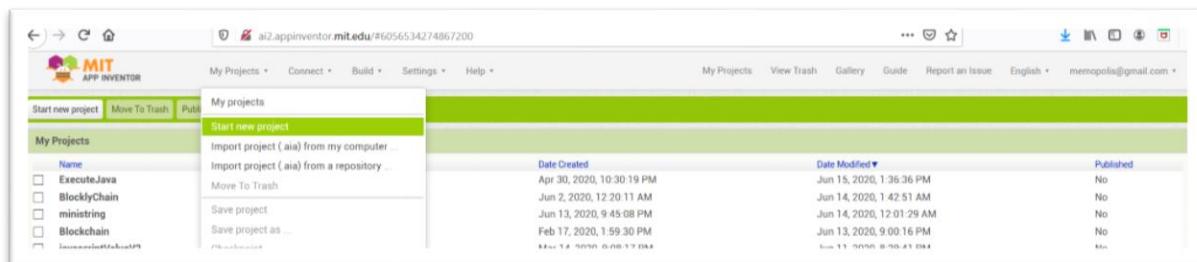
要从一个键中获取值。

\$ redis-cli GET <key>

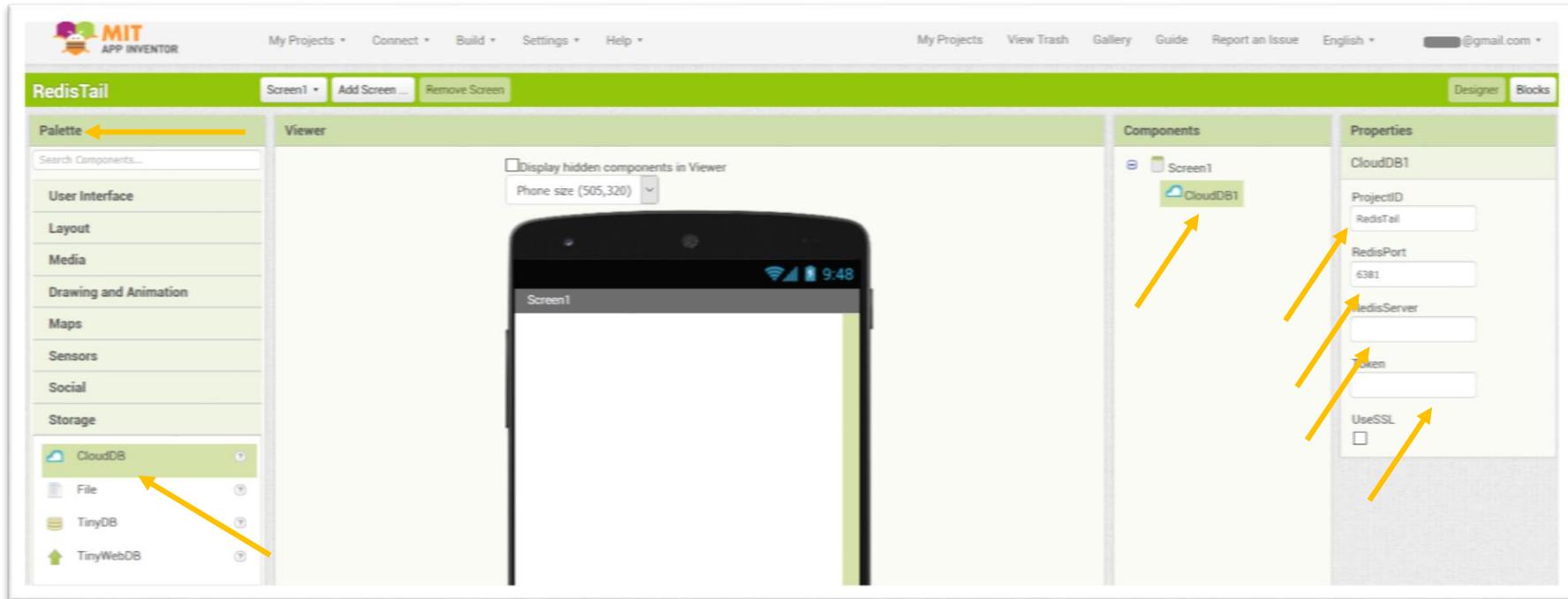
在我们的案例中，因为我们使用的是App Inventor，我们将回顾如何使用区块与Redis一起工作。

在App Inventor里面，我们有几个区块可以使用Redis，我们开始创建一个新的项目，我们会。我的项目 > 开始新项目

迷你积木链 - DIY - "自己动手"。



创建项目后，我们进入左上角，在“调色板”部分点击“存储”，并拖动对象“CloudDB”，这将集成所有功能，配置连接到Redis服务器。



配置非常简单，我们只需要给出以下参数就可以连接到我们的Redis服务器（本地），该服务器运行在我们的节点（手机）中。

项目ID： - 这是Redis中标识事务队列的标签的起始ID。

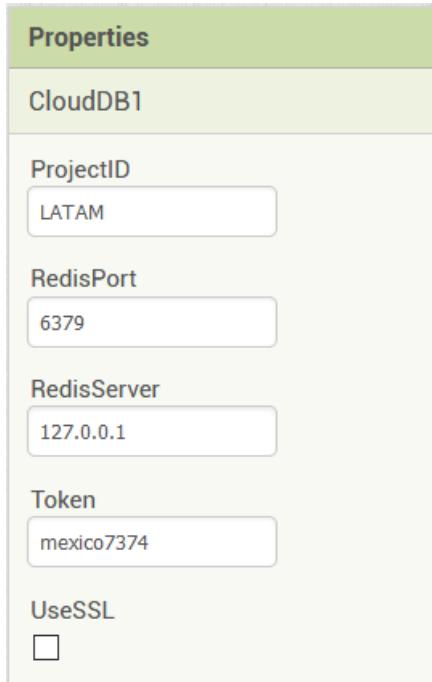
RedisPort。 - 这是Redis服务器的端口，我们默认连接的端口是6379。

RedisServer。 - 在我们的例子中，这是节点的域名或本地IP，而所有节点的IP为127.0.0.1。

代币。 - 这是启用连接的Redis服务器的密码。

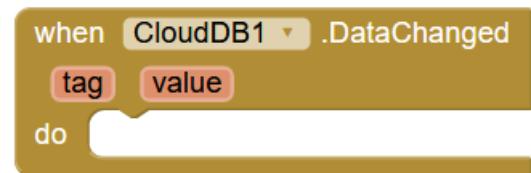
使用SSL。 - 这个选项给了我们使用SSL证书的机会，在我们的情况下，我们不启用它。

在我们的模型和系统设计中，我们将在所有的网络节点中设置以下参数。



现在是时候回顾一下为我们提供对Redis数据库环境的控制和数据处理的块了。

我们从方法块（DataChanged）开始说起



这个方法会在我们配置的Redis服务器每次发生变化时给我们两个值。

标签。- 该值是标识交易队列的标签。

价值。- 该值包含所有发送处理的交易列表。该值是一个类型为<String>的字符串列表。

值"的情况下，我们将开始进行如下的信息处理。

由于你向我们提供了一条所有哈希值交易的链子，我们将首先检查这条链子是否有效，并验证它是否自始至终没有被修改过。我们通过使用一种非常有用的算法来验证大量的信息。

我们将使用Merkle的树算法。应用这种算法，使我们收到的信息（交易队列）的完整性有了保障。

我们来看下面这个Redis中事务队列的例子，我们从Termux终端执行Redis CLI(Command-Line)来查询"LATAM:HASH"键，我们必须已经执行了SQLite-Redis Connector才能够查询这个键。

```
C:memor>redis-cli
127.0.0.1:6379> 获取LATAM:HASH。
"9d45198faaef624f2e7d1897dd9b3cde6ecc7fbac516ed1756b350fe1d56b4,
"f71c801a5fd25fc303ebc8c616204b4877ffb93006ec6a88bc30acf43ec250f5。
"60f8a3bcac1ea7d38e86efbc3e3e00480807d23f980391000766e804ed14ecb2 ,
8a6dfe1d38c22e0f9212052efa6136da3edf1fb1b2a3e25224ac3d689124b754]
127.0.0.1:6379>
```

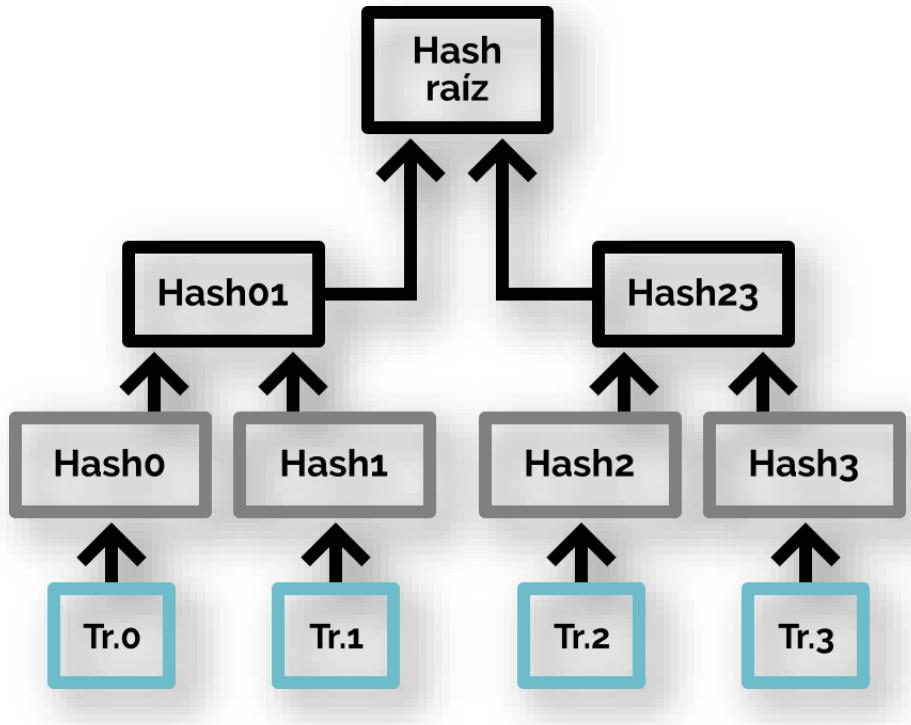
前面的事务队列只有三个元素，因为我们看到四个哈希代表一个单独的事务组成，它们是最初注入op.sqlite3数据库的"trans"和"sign"表内的每个事务的哈希。

现在是时候了解梅克尔树的工作原理了。

哈希梅克尔树是一种二进制或非二进制的数据树结构，其中每个不是表的节点都被标记为其子节点的标签或值的连接的哈希值。它们是哈希列表和哈希字符串的泛化。

在我们的例子中，我们将进行以下计算，以计算四个元素的merkle树，这是通过计算取一对数据的连接结果，并得到它们各自的哈希值，第一级的结果将应用于第二级的结果，直到只有一个最终元素，这将被称为哈希merkleroot（根哈希）。

我们看下面的图来描述这个过程。



基于哈希的交易队列将通过Block（GetMerkleRoot）拉出。



哈希根：51431822de7c94b90dc06d47b8f6275f315a4976c8479d30c32747fa90325432。

然后将结果与本地Redis系统中的LATAM:merkleroot密钥进行比较，两个密钥必须匹配以验证数据的完整性。

merkle树提供的另一个安全点是确认某项具体的交易从源头开始就包含在交易队列中，并且没有通过任何外部或内部通信手段以欺诈的方式引入。

如果链是奇数元素的求和，最后一个元素重复，以有一个安排，并开始执行算法。

另一个不太重要的点是，验证每笔交易对应其来源地-目的地的时间，以及资产是来源地地址发送的资产。

这就是Block (VerifySignature) 的位置。

call OpenQbitBlock1 .VerifySignature

在执行前面的块之前，必须先从"标志"表中下载二进制格式的文件 (file.sig)。

```
sqlite3 op.sqlite3 "select sign from sign where=id_addr;"
```

id_addr。这是"跨"表的源地址的id。

前面的搜索请求会给我们一个当前交易的数字签名的Base64格式数据，这个要把它转换为原始格式（二进制），我们需要Block(DecoderFileBase64)。

既然我们有了二进制文件 (file.sig)，我们就必须将源头的公钥和接收方的公钥也以二进制格式加载到系统中，有了这四种格式的数据，就可以在系统中运行数字签名的验证了。

- ✓ 公钥家庭地址
- ✓ 接收者的公开密钥地址
- ✓ 主动发送。
- ✓ 数字签名(file.sig)

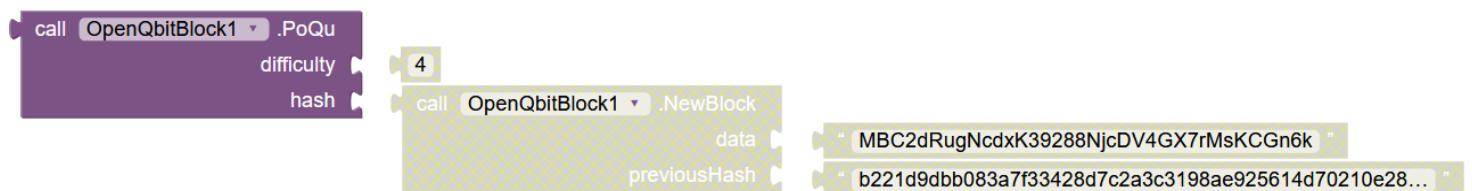
二进制格式的公钥可以从共享数据库publickeys.db中以适当的格式（二进制）下载。

这个过程必须为事务队列中的每一个操作执行。

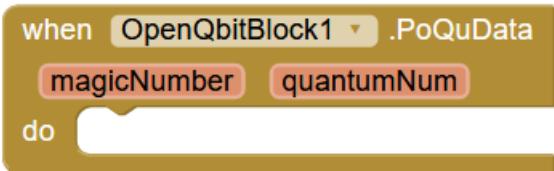
最后我们将回顾一下系统如何以一种合意的方式选择一个节点，能够被选择为区块链添加下一个区块，成为处理交易队列的节点。

这种选择中标节点处理交易队列的方式是基于我们为Mini BlocklyChain系统开发的算法。

我们如何实现PoQu"量子证明"的共识。这个共识过程是基于量子随机数的生成，并使用(PoQu)块进行应用。



首先，我们得到两个参数，块 (PoQu) 给我们交付的方法 (PoQuData) 的参数是 magicNumber 和 quantumNum。



magicNumber 是数字"nonce"，是一个整数，是在难度不大于5的内部PoW中得到的，这个数字负责给节点提出第一个要求，有了magicNumber，节点就能提出要求，得到

一个在0到1范围内的量子随机数的生成系统的数字，这个数字将给执行中的节点随机建立一个概率，这个概率将存储在名为"投票"的表中。

"投票"表将存储每个节点计算出的quantumNum，这个存储将随着节点数量的增加而完成，直到每个系统设计中建立的某个时间，建议有多个条目 $((N/2)+1)$ ，其中"N"是系统中可用的节点数量，它可以通过每个节点的"cron"任务管理工具中的一个操作来建立或控制。

很重要的一点是，在使用"点对点"网络传输交易队列的情况下，此时你应该已经通过手机的自动系统建立了各节点的本地时间同步。

节点中的cron代理的配置。详见"系统节点中的时间同步（手机）分、秒"部分。

在本例中由于我们使用的是备份通信网络，所以我们不需要节点的分秒同步，因为我们的例子占用的是"客户端-服务器"方案，这种通信方式只是在事务队列的传输过程中。节点之间的其他进程都是通过"点对点"通信。

现在我们来看看"投票"表的结构、设计和创建，该表将位于本例中我们也要创建的名为"quorum.db"的数据库中。

\$ sqlite3

SQLite版本3.32.2 2020-06-20 15:25:24

输入".help"以获得使用提示。

连接到一个短暂的内存数据库。

使用".open FILENAME"在持久性数据库上重新打开。

sqlite> .open quorum.db

sqlite> CREATE TABLE vote (id integer primary key AUTOINCREMENT NOT NULL, node_imei
VARCHAR NOT NULL, quantumNum INTEGER NOT NULL, nonce INTEGER NOT NULL);

sqlite> .quit

同样的票数表的创建如下图所示，但它是通过引入SQL语句，以分段的形式。

\$ sqlite3

SQLite版本3.32.2 2020-06-20 15:25:24

输入 ".help" 以获得使用提示。

连接到一个短暂的内存数据库。

使用 ".open FILENAME" 在持久性数据库上重新打开。

```
sqlite> .open quorum.db
```

```
sqlite> CREATE TABLE vote (
```

```
...>   id 整数 AUTOINCREMENT  
...>   node_name 文本 NOT NULL,  
...>   QuantumNumber 整数 NOT NULL,  
...>   nonce 整数 NOT NULL  
...> );
```

```
sqlite> .tables
```

投票

```
sqlite> .quit
```

我们将在创建基础"quorum.db"和表格投票中看到类似的东西。

The screenshot shows a terminal window on a smartphone. The terminal output is as follows:

```
$ sqlite3
SQLite version 3.32.2 2020-06-04 12:58:43
Enter ".help" for usage hints.
Connected to a transient in-memory database.
Use ".open FILENAME" to reopen on a persistent database.
sqlite> .open quorum.db
sqlite> CREATE TABLE vote (
...>     id INTEGER primary key AUTOINCREMENT,
...>     node_imie VARCHAR NOT NULL,
...>     quantumNum INTEGER NOT NULL,
...>     time INTEGER NOT NULL
...> );
sqlite> .tables
vote
sqlite> .quit
$
```

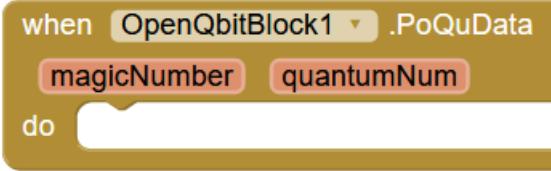
Below the terminal is a standard QWERTY virtual keyboard.

现在让我们看看如何从"投票"表中获取数值。

node_imie。 - 我们使用块(GetDeviceID)来获取这个值。

call OpenQbitBlock1 .GetDeviceID

quantumNum - 这个值是使用(PoQu)块获得的(PoQuData)方法的结果之一。



nce。 - 这个值是已经以积分的方式执行了块(PoQu)而得到的，与方法(PoQuData)的magicNumber相同。

在完成"投票"表中的INSERTS后，需要对数据库进行复制。

在一定时间后，根据各系统的设计，"cron"服务会在每个网络节点上执行，并在"投票"表中有下一个要处理的SELECT。

SELECT node_imei FROM vote WHERE magicNumber= (SELECT max(magicNumber) FROM vote)。

之前的SELECT以较高的概率返回IMEI结果，现在每个运行SELECT的节点都会将自己的IMEI与结果IMEI进行比较，只有匹配的节点才会建立一个概率数字最高的文件，该文件将在网络"Peer to Peer"中被一个格式为IMEI.mbc的文件复制，该文件将包含获胜节点的IMEI。

中标节点就可以开始处理交易队列。根据以上所有的块。

有两点很重要，那就是根据每个系统创建的情况，每个设计者都要审查和定制三个流程。

1.-当中标节点开始处理交易队列时，必须执行一种方法或程序，在这种方法或程序中必须检查中标节点是否在线，与网络的通信是否丢失。

2.-交易队列处理开始时，中标节点必须启动两个控制标志，表示处理开始，另一个标志确认交易队列处理已经结束，这两个标志必须在网络中共享给所有节点，这样有助于定位连接失败或处理失败或处理时间过长。

3.-在通信失败或其他事件中，中标节点一直无法处理交易队列的情况下，必须选择眼前概率范围内的下一个节点。

前面一点可以用验证中奖节点是否在线的服务来控制，可以使用"cron"服务，必须为每个设计的案例开发脚本，不过，下面展示了一个通用的shell脚本的例子，这样可以根据每个Mini Blocklychain系统的需求来修改。

```
# ! /bin/bash
```

```
dir="/data/data/com.termux/files/home/Sync/imei"。
```

```
如果 [!$(ls $directory)]
```

然后

```
sqlite3 quorum.db "UPDATE vote SET magicNumber=0 WHERE magicNumber= (SELECT  
max(magicNumber) FROM vote);"
```

别的

```
MAX_NUM=$(sqlite3 quorum.db "SELECT max(magicNumber) FROM vote;")
```

```
IMEI_quorum=$(sqlite3 quorum.db "SELECT node_id FROM vote WHERE=MAX_NUM")
```

```
IMEI_local=$(cat device_imei) //使用块(GetDevice)
```

```
如果 [ IMEI_quorum -eq IMEI_local ] 。
```

然后

```
触摸 $MAX_NUM > IMEI.mbc
```

```
fi
```

```
fi
```

退出

31. 附件"与Ethereum和Bitcoin环境的整合"。

现在，我们将看到我们如何整合世界上最著名的两个区块链系统，专门用于加密货币，如Ethereum和比特币。

让我们从安装软件开始，帮助我们在Ethereum环境中执行所有可能的交易。

什么是Ethereum？

Ethereum是一个开源平台，去中心化，不像其他区块链，Ethereum可以做得更多。它是可编程的，这意味着开发人员可以使用它来创建新类型的应用程序。

这些去中心化的应用（或“dapp”）得到了加密技术和区块链技术的好处。它们是可靠和可预测的，这意味着一旦它们被“加载”到Ethereum中，它们将始终按计划运行。他们可以控制数字资产，创造新型的金融应用。它们可以是分散的，这意味着没有一个实体或个人控制它们。

眼下，全世界成千上万的开发者正在Ethereum上创建应用，并发明了新的应用类型，其中许多应用你今天就可以使用。

- 允许你用ETH或其他资产进行廉价、即时支付的加密货币组合
- 允许您借贷或投资您的数字资产的金融应用。
- 去中心化的市场，允许你交换数字资产，甚至交换对现实世界事件的“预测”。
- 游戏中，你在游戏中拥有资产，甚至可以赢得真金白银。
- 它有智能合约，这些合约是有协议的程序，当它被阐述或创建的前提满足时，就会被执行。

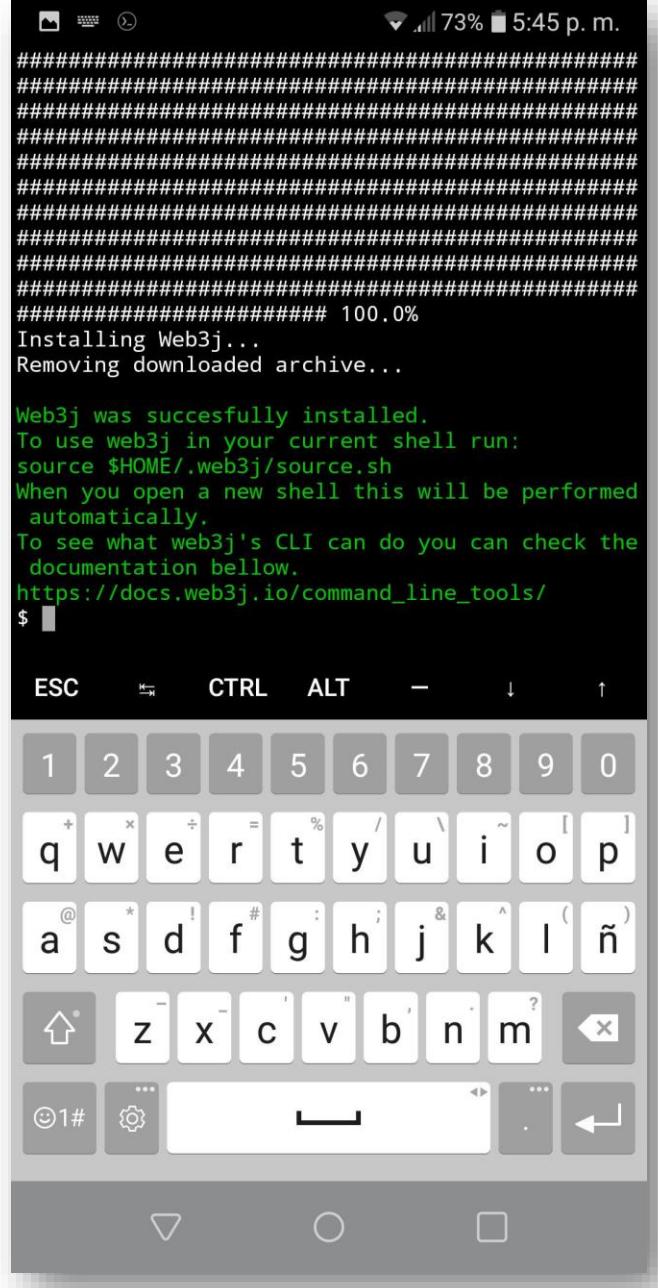
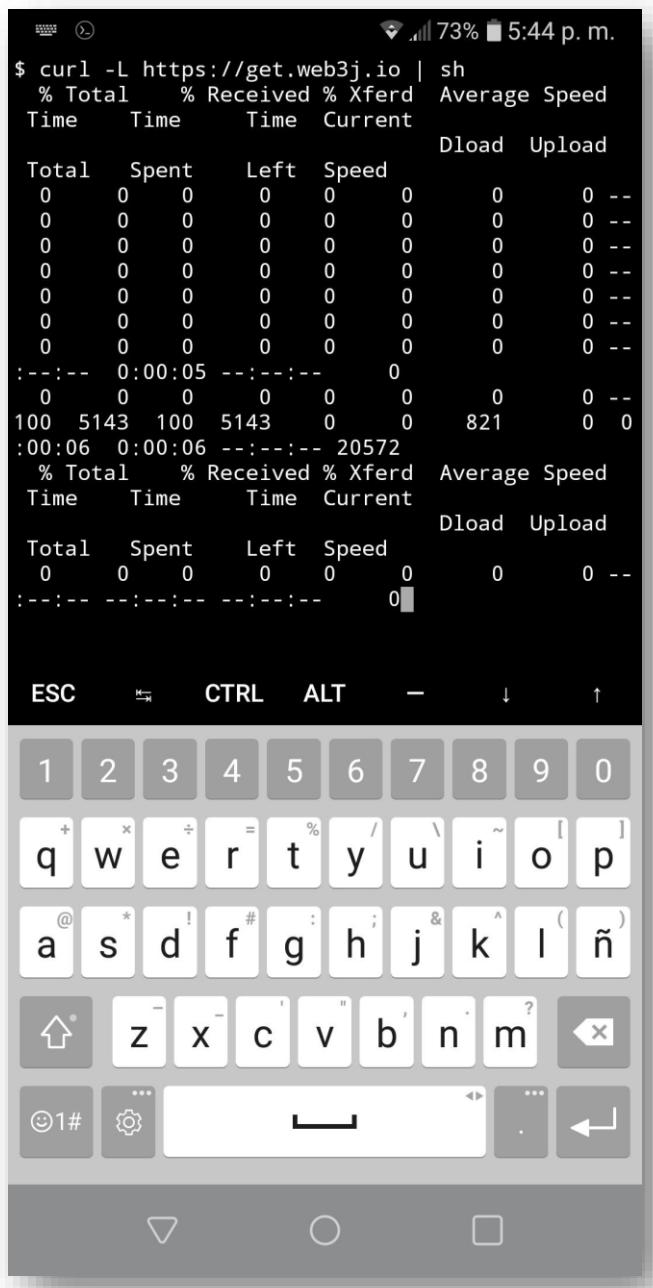
智能合约与DApps(去中心化应用)有相似之处，但也有一些重要的区别。

和智能合约一样，DApp也是一个接口，它通过去中心化的对等网络将用户与提供商的服务连接起来。但是，智能合约的创建需要固定的参与者数量，而DApps则没有用户数量的限制。此外，它们不仅仅局限于金融应用，如智能合约：一个DApp可以服务于任何你能想到的目的。

在我们的案例中，我们将使用名为"Web3j"的库，该库是用Java开发的，它允许以简单和直观的方式与Ethereum区块链进行交互。

执行以下命令，安装"Web3j"。

```
$ curl -L get.web3j.io | sh
```



然后我们必须用可执行文件"java"所在的路径创建环境变量\$JAVA_HOME, 因为库会搜索这个变量, 以便能够成功执行。

```
$ JAVA_HOME= /data/data/com.termux/files/usr/bin
```

变量创建完成后, 我们要到安装"Web3j"库的目录下执行下面的命令, 重要的一点是这个目录是隐藏的, 在dr命令的"cd"后我们放一个点"."然后目录下没有空格, 如下图

◦

```
$ cd .web3j
```

进入后我们用下面的命令测试库是否正常运行。

```
./web3j版本
```

它的结果是非常类似于。

```
$ ls
source.sh  web3j  web3j-4.5.16
$ ./web3j version

Version: 4.5.16
Build timestamp: 2020-03-06 14:13:49.943 UTC
$
```

后面我们将通过以下方式创建一个钱包, 用于Ethereum的区块链环境中。

\$./web3j钱包创建

前面的命令给了我们ethereum的地址。

4598fe2fd6afe2508f58343c7d42f2ab492edf34

```
$ ./web3j wallet create

Please enter a wallet file password:
Please re-enter the password:
Please enter a destination directory location [/data/data/com.termux/files/home/.ethereum/testnet/keystore]:
Wallet file UTC--2020-06-27T06-12-23.819752000Z-4598fe2fd6afe2508f58343c7d42f2ab492edf34.json successfully created in: /data/data/com.termux/files/home/.ethereum/testnet/keystore
$
```

它的结果是一个JSON格式的文件，其中包含地址和加密数据，以便稍后用于生成刚刚创建的账户的私钥。

这个JSON类型的文件将被创建在一个名为keystore的默认目录中。

从这里开始，我们已经可以使用和/或执行Web3j命令进行操作。

我们可以通过使用扩展（ConnectorSSHClient）来实现。

要知道如何使用"Web3j"库的不同参数和操作，我们可以通过查阅其官方网站上的文档来帮助我们。

<https://docs.web3j.io/>

对于比特币环境，我们有两种选择：使用生成比特币账户及其各自的公钥和私钥的块（）。我们可以通过安装"Bitcoinj"java库，使用这些密钥集成到比特币环境中。

\$ npm install bitcoinj



```
$ npm install bitcoinj
+ bitcoinj@0.0.0
added 1 package from 1 contributor and audited 2
09 packages in 20.528s
$
```

为了使用这个图书馆，我们将依靠其官方网站。

<https://bitcoinj.github.io/>

集成到比特币区块链环境中的第二种选择是为Termux安装Bitcoind包，如下图所示。

\$ apt install bitcoin



```
$ pkg install bitcoin
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following NEW packages will be installed:
  bitcoin
0 upgraded, 1 newly installed, 0 to remove and 1
9 not upgraded.
Need to get 3601 kB of archives.
After this operation, 15.0 MB of additional disk
space will be used.
Get:1 https://dl.bintray.com/termux/termux-pac
ges-24/stable/main arm bitcoin arm 0.20.0 [3601
kB]
Fetched 3601 kB in 2s (1253 kB/s)
Selecting previously unselected package bitcoin.
(Reading database ... 19110 files and directo
ries currently installed.)
Preparing to unpack .../bitcoin_0.20.0_arm.deb .
..
Unpacking bitcoin (0.20.0) ...
Setting up bitcoin (0.20.0) ...
$
```

现在，当我们在Termux终端上运行bitcoind代理时，它会自动在该目录下创建一个地址。

/data/data/com.termux/files/hme/.bitcoin。

在比特币的情况下，我们将依靠"Bitcoind"代理的以下文档。

在这种情况下，我们也可以根据不同的需要，使用扩展（ConnectorSSHClient）来执行"bitcoind"代理。

bitcoind使用的参数说明。

名称

bitcoind - 启动比特币核心守护进程。

SYNOPSIS

bitcoind [选项] 启动比特币核心守护进程

描述

启动比特币核心守护进程

选项

-?

打印此帮助信息并退出

-alertnotify=<cmd>。

当收到相关警报或我们看到一个很长的分叉时，执行该命令（cmd中的%s被信息所取代）。

-assumevalid=<hex>

如果这个区块在字符串中，假设他和他的祖先是有效的，有可能跳过你的写验证（0为验证所有，默认：

000000000000f1c54590ee18d15ec70e68c8cd4cfbadb1b4f11697eee, testnet :
000000000037a8cd3e06cd5edbfe9dd1dbcc5dacab279376ef7fcf2b4c75)

-blocknotify=<cmd>

当最佳区块发生变化时，执行该命令（cmd中的%s被区块哈希值所取代

-blockreconstructionextratxn=<n>。

额外的事务要保留在内存中，以便进行紧凑的块重构（默认：100

-blocksdir=<dir>。

指定区块目录（默认：`<datadir>/blocks`

-仅在区块内

如果您拒绝网络合作伙伴的交易。投资组合或RPC交易不受影响。（默认：0）

-conf=<档案>

指定配置文件。相对路径将以数据目录的位置为前缀。（默认：`bitcoin.conf`）

-daemon

它就像一个恶魔一样在后台运行，并接受以下命令

-datadir=<dir>。

指定数据目录

-dbcache=<n>

最大数据库缓存大小<n> MiB（4到16384，默认：450）。此外，未使用的mempool内存也会被共享给这个缓存（参见**-maxmempool**）。

-debuglogfile=<file>。

指定调试日志文件的位置。相对路径将以网络特定的datadir位置为前缀。（**-nodebuglogfile**禁用；默认：`debug.log`）

-includeconf=<file>。

指定一个额外的配置文件，相对于**-datadir**路径（只能从配置文件中使用，不能从命令行使用）。

-loadblock=<file>

在启动时从外部文件blk000?.dat中导入块。

-maxmempool=<n>

将事务内存池保持在<n>兆字节以下（默认：300）。

-maxorphantx=<n>

在内存中保留最大<n>个可断开连接的事务（默认：100）。

-mempoolexpiry=<n>。

不要在mempool中保存交易超过<n>小时（默认：336）。

-par=<n>

设置破折号中的验证线程数（-6到16，0=自动，<0=让所有核心空闲， 默认：0）。

-persistmempool

如果您在关闭时保存mempool，并在重新启动时加载它（默认：1）。

-pid=<file>

指定PID文件。相对路径将以网络特定的datadir位置为前缀。（默认：bitcoind.pid）

-punza=<n>

允许修剪(去除)旧的区块，从而减少储存需求。这允许调用RPC修剪链来删除特定的块，如果提供了MIB中的目标大小，则允许自动修剪旧的块。该模式与**-txindex**和**-rescan**不兼容。警告：如果要反转这个设置，需要重新下载整个区块链（默认：0 = 禁用区块修剪，1 = 允许通过RPC手动修剪，>=550 = 自动修剪区块文件，使其保持在MIB指定的目标大小以下）。

-reindex

重建磁盘上blk*.dat文件的字符串状态和块索引。

-reindex-chainstate

从当前的索引块重建链的状态。在修剪模式下，或者如果磁盘上的块可能已经损坏，请使用完整的**重新索引**。

-叹息

用默认的系统权限创建新文件，而不是Umask 077(只有在禁用组合功能时才有效)

-txindex

维护一个完整的交易索引，由getrawtransaction rpc调用使用(默认：0)

-版本

打印版

连接选项：

-addnode=<ip>

添加一个连接到的节点，并尝试保持连接的开放性（更多信息请参见“addendum”的RPC命令帮助）。这个选项可以指定多次，以增加多个节点。

-banscore=<n>

断开行为不端的同事的阈值（默认：100）。

-同时..

防止不守规矩的同事重新连接的秒数（默认：86400）。

-bind=<addr>

把自己绑在给出的地址上，时刻听从。对IPv6使用[host]:port的符号。

-connection=<ip>

只连接到指定的节点；**-noconnect**禁用自动连接（这对的规则与**-addnode**相同）。这个选项可以多次指定，连接到多个节点。

-发现

发现你自己的IP地址（默认：监听时为1，而不是**-externalip**或**-proxy**）。

-dns

允许DNS搜索**-addnode**、**-seednode**和**-connect**（默认：1）。

-dnsseed

如果地址较少，则通过DNS查询配对地址（默认：1，除非使用**-connection**）。

-enablebip61

通过BIP61发送拒绝信息（默认：1）。

-externalip=<ip>

指定您自己的公共地址

-逼迫式种子

总是通过DNS查询同事的地址（默认：0）。

-听

接受来自外部的连接（如果没有**-proxy**或**-connection**，则默认为1）。

-聆听ionion

自动创建Tor隐藏服务（默认：1）。

-maxconnections=<n>。

最多保持<n>个同事的联系（默认：125）。

-maxreceivebuffer=<n>。

每个连接的最大接收缓冲区, <n>*1000字节(默认 : 5000)

-maxsendbuffer=<n>。

每个连接的最大发送缓冲区, <n>*1000字节(默认 : 1000)

-最大时间设定

对平均时间补偿的调整所允许的最大值。本地时间视角可以通过前进或后退对这个量来影响。(默认 : 4200秒)

-maxuploadtarget=<n>。

尽量将出站流量控制在给定的目标范围内 (24小时内以MiB为单位) , 0=无限制 (默认 : 0) 。

-onion=<<ip:port>>。

使用一个单独的SOCKS5代理来通过Tor的隐藏服务与对等体联系, 设置-**noonion**为禁用 (默认 : **-proxy**) 。

-onlynet=<net>

只通过<net>网络 (ipv4、 ipv6或onion) 进行外发连接。传入的连接不受此选项的影响。这个选项可以指定多次, 允许多个网络。

-成对过滤器

支持开花过滤器的阻断过滤和交易 (默认 : 1) 。

-许可证multisig

中继器不是P2SH多协议 (默认 : 1) 。

-port=<port>

监听'端口'中的连接 (默认 : 8333, testnet : 18333, regtest : 18444) 。

-proxy=<ip:port>

通过SOCKS5代理连接，设置-**noproxy**为关闭（默认：关闭）。

-proxyrandomize

随机设置每个代理连接的凭证这将启用 Tor 流量隔离（默认值：1）。

-seednode=<ip>

连接到一个节点上检索对地址，然后断开连接。这个选项可以多次指定，连接到多个节点。

-timeout=<n>

指定连接超时时间，单位为毫秒（最小值：1， 默认值：5000）。

-torcontrol=<ip>:<port>。

如果启用了onion监听，要使用的Tor控制端口（默认：127.0.0.1:9051）。

-password=<pass>

Tor 控制端口密码（默认：空白

-UPNP

使用UPnP指定监听端口（默认：0）。

-whitebind=<addr>。

链接到某个地址和连接到该地址的白名单伙伴。对IPv6使用[host]:port的符号。

-whitelist=<IP地址或网络>。

白名单上的对从给定的IP地址（如1.2.3.4）或CIDR的注释网络（如1.2.3.0/24）连接。
可以多次指定。白名单对不能被DoS禁止。

投资组合选项：

-地址类型

使用什么类型的地址 ("legacy"、"p2sh-segwit"或"bech32"，默认："p2sh-segwit")。

-避免部分费用

按方向对输出进行分组，选择全部或不选择，而不是按每个输出进行选择。由于地址只使用一次（除非有人在消费后才发送），隐私性得到了加强，但由于增加了限制，货币的选择可能不理想，因此可能会导致费率略高（默认：0）。

-类型变化

使用哪种汇率 ("legacy"， "p2sh-segwit"， 或"bech32")。默认值与**-addresstype**相同，除了**-addresstype=p2sh-segwit**在发送至本地segwit地址时使用本地segwit输出。)

-从钱包里拿出来

不要加载钱包，并停用钱包的RPC调用。

-discardfee=<amt>。

收费率（单位：BTC/kB），表示其通过将变化添加到收费率中而放弃变化的容忍度（默认：0.0001）。注意：如果一个输出是尘埃，在这个速率下就会被丢弃，但我们会一直丢弃到尘埃重传速率，而超过这个速率的丢弃速率则受限于较长目标的速率估计值

-fallbackfee=<amt>。

当费用估算数据不足时使用的费率率（单位：BTC/kB）（默认：0.0002）。

-keypool=<n>

设置密钥池大小为<n>（默认：1000）。

-mintxfee=<amt>。

低于此值的费率(单位：BTC/kB)被认为是创建交易的零费率(默认：0.00001)

-paytxfee=<amt>。

要添加到您发送的交易中的速率（单位：BTC/kB）（默认：0.00）。

-重新扫描

重新扫描区块链，查找在开始时丢失的投资组合交易。

-拯救钱包

试图检索启动中损坏的钱包的私钥。

-浪费信息交流；

发送交易时，花费未确认的零钱（默认：1）。

-txconfirmtarget=<n>。

如果没有设置支付费用，请为平均在n个区块内开始确认的交易加入足够的费用（默认：6）。

-投资组合维护

在开始时将投资组合更新为最新的格式

-wallet=<路径>。

指定组合数据库的路径。您可以多次指定它来加载多个投资组合。如果路径不是绝对的，则解释为与<walletdir>的关系，如果路径不存在，则将创建该路径（如包含钱包.dat文件和日志文件的目录）。为了向后兼容，它也将接受<walletdir>中现有数据文件的名称）。

-walletbroadcast

进行投资组合扩散交易（默认：1

-walletdir=<dir>。

指定保存投资组合的目录（默认：`<datadir>/portfolios`，如果存在，否则`<datadir>`）。

-walletnotify=<cmd>。

当投资组合交易发生变化时，执行命令（cmd中的%s由TxID代替）。

-walletrbf

发送交易时，可选择包括整个RBF的激活（仅RPC， 默认为：0

-zapwallettxes=<mode>。

从组合中删除所有交易，只在开始时通过**-rescan**检索阻塞链的部分（1=保留tx-metadata，如付款请求信息，2=放弃tx-metadata）。

ZeroMQ通知选项。

-zmqpubhashblock=<address>。

启用"地址"中的发布块

-zmqpubhashblockhwm=<n>。

设置高水印的外发信息发布块（默认：1000）。

-zmqpubhashtx=<地址>。

启用在'地址'中发布哈希什交易。

-zmqpubhashtxhwm=<n>。

设置发布高水印哈希事务的输出消息（默认：1000）。

-zmqpubrawblock=<address>。

在"地址"中启用原始发布块

-zmqpubrawblockhwm=<n>。

设置原始块发信高水印（默认：1000）。

-zmqpubrawtx=<地址>。

启用在'地址'中发布原始交易

-zmqpubrawtxhwm=<n>。

设置发布总交易输出消息高水印（默认：1000）。

调试测试选项。

-debug=<category>

输出调试信息（默认：**-nodebug**，提供“类别”是可选的）。如果没有提供<category>，或者<category>=1，则输出所有调试信息。<类别>可以是：net, tor, mempool, http, bench, zmq, db, rpc, estimatefee, addrman, selectcoins, reindex, cmpctblock, rand, prune, proxy, mempoolrej, libevent, coindb, qt, leveldb。

-debugexclude=<category>。

将调试信息从类别中排除。可以和**-debug=1**一起使用，生成除一个或多个指定类别以外的所有类别的调试记录。

-help-debug

打印带有调试选项的帮助信息并退出。

-logips

在调试输出中包含IP地址（默认：0）。

-logtimestamps

准备带有时间戳的调试输出（默认：1）

-maxtxfee=<amt>。

单笔投资组合交易或总交易使用的最大总费用（以BTC为单位）；如果设置得太低，可能会中止大额交易（默认：0.10）。

-为控制台打印

向控制台发送跟踪/调试信息（当没有**-daemon**时默认为1。要禁用日志记录到文件，设置**-nodebuglogfile**）

-shrinkdebugfile

在客户端启动时减少debug.log文件（默认：当没有`-debug`时为1）。

-uacomment=<cmt>。

为用户代理字符串添加注释

链条选择选项。

-testnet

使用测试链...

节点重传选项。

-bytespersigop

广播和采矿交易中每个sigop的字节当量(默认：20)

-数据载体

中继和地雷数据载体交易(默认：1)

-数据载波化

我们重传和提取的数据载体的交易中的最大数据量（默认：83）。

-mempool-replacement

启用替换内存池中的事务（默认：1）。

-minrelaytxfee=<amt>。

低于此值的费用（单位：BTC/kB）被认为是转发、提取和创建交易的零费用（默认：0.00001）。

-白名单为cerelay

强制重传白名单伙伴的交易，即使交易已经在mempool中，或者违反了本地重传策略（默认：0）。

-whitelistrelay

接受从白名单对收到的传输交易，即使没有传输交易（默认：1）。

阻止创建选项。

-blockmaxweight=<n>

设置BIP141块的最大权重（默认：3996000）。

-blockmintxfee=<amt>。

设置包含在区块创建中的交易的最低佣金率（单位：BTC/kB）。(默认：0.00001)

RPC服务器选项。

-饭店

接受公共REST请求（默认：0）。

-rpccallowip=<ip>

允许从指定的源头进行JSON-RPC连接。它们对<ip>单个IP（如1.2.3.4）、网络/网络掩码（如1.2.3.4/255.255.255.0）或网络/CIDR（如1.2.3.4/24）有效。这个选项可以指定多次

-rpcauth=<userpw>。

JSON-RPC连接的用户名和密码HMAC-SHA-256。'userpw'字段的格式为：'username'：'Salt'：\$ 'HASH'。share/rpcauth中包含了一个规范的python脚本。然后，客户端使用rpcuser=<USERNAME>/rpcpassword=<PASSWORD>参数对进行正常连接。这个选项可以指定多次

-rpccbind=<addr>[:端口]

链接到某个地址来监听JSON-RPC连接，不要将RPC服务器暴露在不可靠的网络中，比如公共互联网！除非同时通过**-rpccallowip**，否则该选项将被忽略。该端口是可选的，并覆盖**-rpccport**。使用[host]:port符号来表示IPv6。这个选项可以指定多次（默认：127.0.0.1和::1即localhost）。

-rpccookiefile=<loc>。

授权cookie的位置。相对路径将以网络特定的datadir位置为前缀。(默认 : dir)

-rpcpassword=<pw>

JSON-RPC连接的密码

-rpcport=<port>

监听'端口'中的JSON-RPC连接 (默认 : 8332, testnet : 18332, regtest : 18443)。

-rpcserialversion

设置原始事务序列化或非语言模式下返回的块十六进制，而不是segwit(0)或segwit(1)(
默认 : 1)

-rpcthreads=<n>

设置处理RPC调用的线程数 (默认 : 4)。

-rpcuser=<用户>

JSON-RPC连接的用户名

-服务器

接受命令行命令和JSON-RPC

32. 软件的许可和使用：

安卓系统

<https://source.android.com/setup/start/licenses>

Termux

<https://github.com/termux/termux-app/blob/master/LICENSE.md>

节点

<https://raw.githubusercontent.com/nodejs/node/master/LICENSE>

SQLite

<https://www.sqlite.org/copyright.html>

捷信

<https://git-scm.com/about/free-and-open-source>

sqlit-to-rest

<https://github.com/olsonpm/sqlite-to-rest/blob/dev/license.txt>

Redis DB

<https://redis.io/topics/license>

世界时间API NTP

<http://worldtimeapi.org/pages/faqs#commercial-apps>

Tor网络

<https://github.com/torproject/tor/blob/master/LICENSE>

同步网络

<https://forum.syncthing.net/t/syncthing-is-now-mplv2-licensed/2133>

OpenSSH

<https://www.openssh.com/features.html>

Putty SSH

<https://www.chiark.greenend.org.uk/~sgtatham/putty/licence.html>

麻省理工学院App Inventor 2同伴和App Inventor Blockly。

<https://appinventor.mit.edu/about/termsofservice>

SQLite Expert Personal -免费软件

<http://www.sqliteexpert.com/download.html>

Apache蚂蚁

<https://ant.apache.org/license.html>

WGET

<https://www.gnu.org/software/wget/>

OpenJDK

<https://openjdk.java.net/legal/>

外部扩展：

JSONTOOLs

<https://thunkableblocks.blogspot.com/2017/07/jsontools-extension.html>

Mini BlocklyChain系统的开源和商业版本的授权请咨询官方网站

<http://www.openqbit.com.>

迷你BlocklyChain, MiniBlockly, BlocklyCode, MiniBlockMiniChain, QBlockly是OpenQbit注册的商标。

迷你BlocklyChain是在公共领域。

Mini BlocklyChain中的所有代码和文档都被作者献给了公共领域。所有的代码作者和他们所工作的公司代表都签署了宣誓书，将他们的贡献献给了公共领域，这些宣誓书的原件都存放在OpenQbit墨西哥总部的保险箱里。任何人都可以自由地发布、使用或分发原始的Mini BlocklyChain (OpenQbit)扩展，无论是源代码还是编译后的二进制文件，用于任何目的，商业或非商业，以及以任何方式。

前一段适用于Mini BlocklyChain中可交付的代码和文档，即Mini BlocklyChain库中那些实际上与一个更大的应用程序一起分组和出货的部分。编译过程中使用的一些脚本（例如，由autoconf生成的“配置”脚本）可能包含在其他开放源码许可证中。然而，这些编译脚本都没有进入最终的Mini BlocklyChain可交付库中，因此在评估您复制和使用Mini BlocklyChain库的权利时，与这些脚本相关的许可证不应该是一个因素。

Mini BlocklyChain中所有可交付的代码都是从头开始编写的。没有从其他项目或公开的互联网上获取任何代码。每一行代码都可以追溯到它的原作者，所有这些作者都有公共领域的奉献档案。因此，Mini BlocklyChain的代码库是干净的，没有被其他开源项目授权的代码污染，而不是开放贡献的。

迷你BlocklyChain是开源的，这意味着你可以制作任意数量的副本，并且可以不受限制地使用这些副本做你想做的事情。但Mini BlocklyChain并不开源。为了使Mini BlocklyChain处于公共领域，并确保代码不受专有或授权内容的污染，该项目不接受不明人士的补丁。Mini BlocklyChain中的所有代码都是原创的，因为它是专门为Mini BlocklyChain使用而编写的。没有从互联网上抄袭不明来源的代码。

迷你BlocklyChain是在公共领域，不需要许可证。即便如此，一些组织还是希望得到法律证明，证明他们有权使用Mini BlocklyChain。发生这种情况的情况包括：

- 你的公司希望对侵犯版权的索赔进行赔偿。
- 您在一个不承认公共领域的司法管辖区使用Mini BlocklyChain。
- 您使用Mini BlocklyChain的司法管辖区不承认作者将其作品置于公共领域的权利。
- 您希望有一份有形的法律文件作为证据，证明您拥有使用和分发Mini BlocklyChain的合法权利。
- 你的法律部门告诉你，你必须买一个许可证。

如果上述情况适用于您，OpenQbit公司（雇佣所有Mini BlocklyChain开发者的公司）将向您出售Mini BlocklyChain产权担保。产权保证书是一份法律文件，声明Mini BlocklyChain的宣称作者是真正的作者，作者拥有将Mini BlocklyChain献给公共领域的合法权利，OpenQbit将针对授权要求进行有力的辩护。Mini BlocklyChain的产权担保销售收入全部用于资助Mini BlocklyChain的持续改进和支持。

贡献代码

为了保持Mini BlocklyChain完全免费和免版税，该项目不接受补丁。如果你想做一个建议性的修改，并包含一个补丁作为概念验证，那将是非常好的。不过，如果我们从头开始重写你的补丁，也不要生气。非商业或开源许可证的类型谁使用它在这种模式和一些类似的不购买的支持，无论是个人或企业使用，无论公司的规模将受以下法律前提。

保修免责声明。除非适用法律要求或书面同意，许可方以"原样"提供本作品(每个贡献者提供其贡献)，**没有任何形式的明示或暗示的保证或条件**，包括但不限于所有权、非侵权、适销性或特定用途的适用性的任何保证或条件。您应自行负责确定本作品的正确使用或再分配，并承担与您行使本许可下的权限有关的任何风险。

因使用本软件而造成的任何经济或其他损失，均由受影响方承担。所有的法律纠纷将提交给墨西哥国家墨西哥城的法院审理。

对于商业支持、使用和许可，必须在OpenQbit或其公司与相关方之间建立协议或合同。

◦

分销营销条款可能会在不通知的情况下发生变化, 请到官方网站www.openqbit.com,
查看任何非商业和商业的支持和授权条款的变化。

任何个人, 用户, 任何法律性质的私人或公共实体, 或来自世界任何地方的人, 只要使用本软件, 就无条件地接受本文件中的条款, 以及那些可以在任何时候在www.openqbit.co门户网站上修改的条款, 而无需事先通知, 并可由OpenQbit决定应用于非商业或商业用途。

关于Mini BlocklyChain的任何问题和信息都应直接向App Inventor社区或各个Blockly系统社区反映, 因为它们是。AppBuilder、Trunkable等和/或向邮件opensource@openqbit.com, 对于需求的问题可以在3到5个工作日内得到答复。

支持与商业用途。

support@openqbit.com

商业用途的销售。

sales@openqbit.com

法律信息和许可问题或关切

legal@openqbit.com

