第三章

Bitcoin是一个开源项目，源代码可以根据开放（MIT）许可证提供，可免费下载并用于任何目的。 开源意味着不仅仅是自由使用。 这也意味着比特币是由一个开放的志愿者社区开发的。 起初这个社区只有中本聪。 到2016年，比特币的源代码有超过400个贡献者，大约十几位开发人员几乎全职工作，几十名开发人员兼职。 任何人都可以为代码贡献 - 包括你！

当由中本聪创建比特币时，软件实际上是在后来大名鼎鼎的[satoshi_whitepaper]白皮书之前完成的。 中本聪想在写作之前确保它有效工作。 那么这个第一个实践，就叫做“比特币（Bitcoin）”或者“Satoshi客户”，实际上已经被大大的修改和改进了。它已经演变成所谓的比特币核心，以区别于其他兼容的实现方式。 比特币核心是比特币系统的参考实现，这意味着它是如何实施的权威参考。 Bitcoin Core实现了比特币的所有方面，包括钱包，交易和区块验证引擎，以及P2P网络中的完整网络节点。

警示 即使Bitcoin Core包含钱包的参考实现，但这并不意味着可以用作用户或应用程序的生产钱包。 建议应用程序开发人员使用现代标准（如BIP-39和BIP-32）构建钱包（请参阅助记词]和[hd钱包]章节）。 BIP就是比特币改进提案（Bitcoin Improvement Proposal）。

下图为比特币核心的架构。

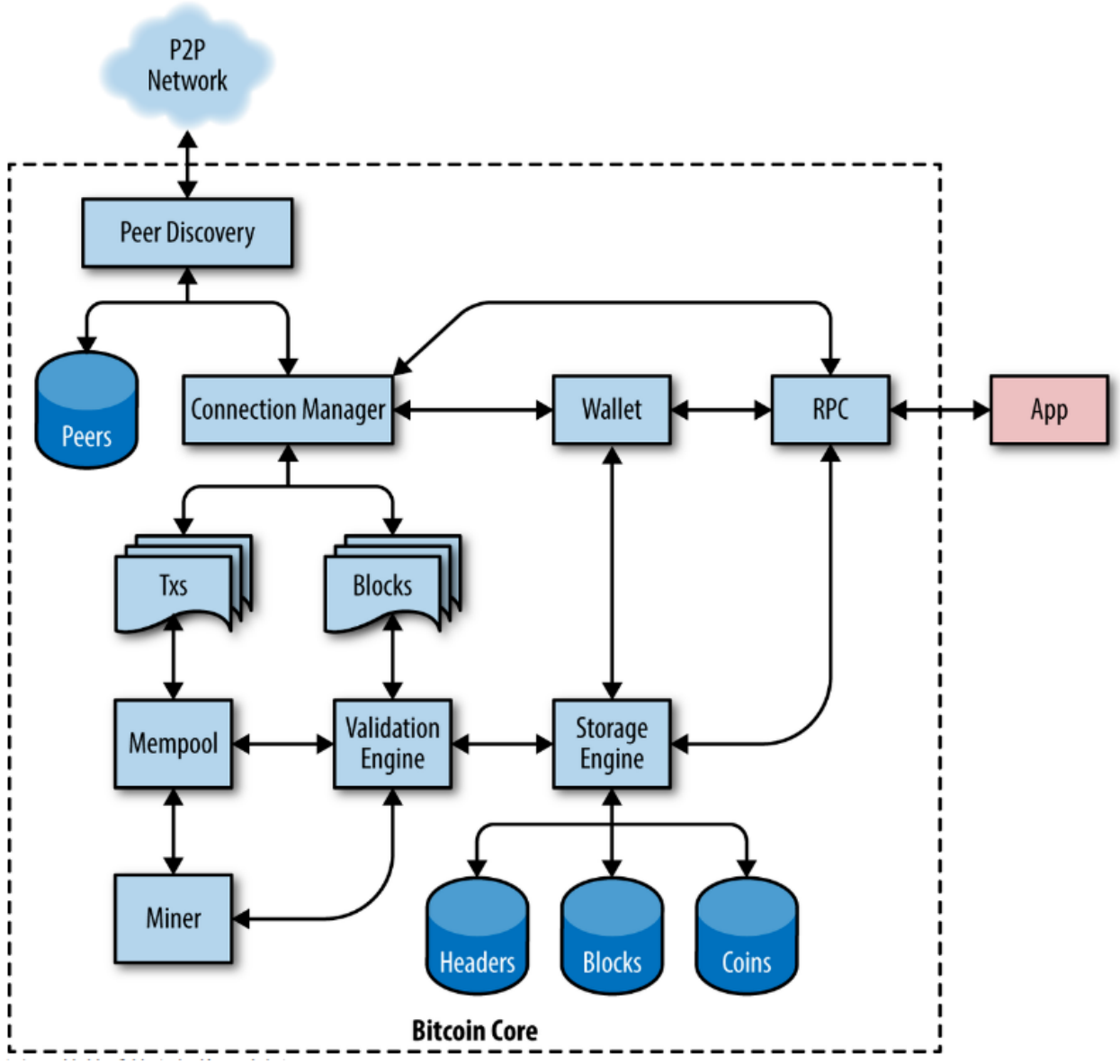

图3-1比特币核心架构（来源Eric Lombrozo）

3.1比特币开发环境

如果您是开发人员，您将需要使用所有工具，库和支持软件来设置开发环境，以编写比特币应用程序。 这一章涉及的技术细节较深，我们将逐步介绍该过程。 如果你觉得过于繁琐（并且您实际上并没有设置开发环境），建议你跳到下一章，技术性比本章浅显一些。

3.2从源码编译比特币核心

Bitcoin Core的源代码可以作为ZIP存档下载，也可以从GitHub克隆权威的源代码库。 在GitHub比特币页面GitHub bitcoin page上，选择“下载ZIP”。 或者，使用git命令行在系统上创建源代码的本地副本。

提示在本章的许多例子中，我们将使用操作系统的命令行界面（也称为“shell”），通过“terminal”应用程序访问。 shell将显示提示你键入命令，并且shell反馈一些文本和一个新的提示您的下一个命令。 提示符可能在您的系统上看起来不同，但在以下示例中，它由符号表示（非用户）。在示例中，当您在符号后面看到文本时，不要键入符号，而是在其后面键入命令，然后按键执行该命令。在示例中，每个命令下面的行是操作系统对该命令的响应。当

你看到下一个前缀时，应该继续输入下一个新的命令，可以一直重复这个过程。

在本例中，我们使用git命令来创建源代码的本地副本（“clone”）：

```
$ git clone https://github.com/bitcoin/bitcoin.git
Cloning into 'bitcoin'...
remote: Counting objects: 66193, done.
remote: Total 66193 (delta 0), reused 0 (delta 0), pack-reused 66193
Receiving objects: 100% (66193/66193), 63.39 MiB | 574.00 KiB/s, done.
Resolving deltas: 100% (48395/48395), done.
Checking connectivity... done.
$
```

提示Git是最广泛使用的分布式版本控制系统，是任何软件开发人员工具包的重要组成部分。 您可能需要在操作系统上安装git命令或git的图形用户界面。

当git克隆操作完成后，您将在目录比特币中拥有源代码存储库的完整本地副本。 在提示符下键入“cd bitcoin”，进入为此目录：

$ cd bitcoin

3.2.1选择比特币核心版本

默认情况下，本地副本将与最新的代码同步，这可能是不稳定的或Beta版的比特币。 在编译代码之前，先查看一个发布标签tag，选择一个特定的版本。 这将使本地副本与关键字标签所标识的代码库的特定快照同步。 开发人员使用标签来标记版本号的特定版本的代码。 首先，要找到可用的标签，我们使用git tag命令：

```
$ git tag
v0.1.5
v0.1.6test1
v0.10.0
...
v0.11.2
v0.11.2rc1
v0.12.0rc1
v0.12.0rc2
...
```

tag列表显示所有发布的比特币版本。 根据惯例，用于测试的发布候选版本具有后缀“rc”。 可以在生产系统上运行的稳定版本没有后缀。 从上面的列表中，选择最高版本的版本，在编写时是v0.11.2。 要使本地代码与此版本同步，请使用git checkout命令：

```
$ git checkout v0.11.2
HEAD is now at 7e27892... Merge pull request #6975
```

您可以通过输入命令git status来确认您有所需的版本“checkout”：

```
$ git status
HEAD detached at v0.11.2
nothing to commit, working directory clean
```

3.2.2配置构建比特币核心

源代码中包括文档，可以在多个文件中找到。通过在提示符下键入“more README.md”并使用空格键进入下一页，查看bitcoin目录中README.md中的主要文档。在本章中，我们将在Linux上构建命令行比特币客户端，也称为比特币（bitcoind）。在系统中查看编译bitcoind命令行客户端的说明，方法是输入“more doc / build-unix.md”。可以在doc目录中找到macOS和Windows的替代说明，分别为build-osx.md或build-windows.md。

仔细查看build前提条件，这些前提是build文档的第一部分。这些是在您开始编译比特币之前必须存在于系统上的库。如果缺少这些先决条件，build过程将失败并显示错误。如果发生这种情况是因为您缺失先决条件，则可以安装它，然后从您所在的地方恢复build过程。假设安装了先决条件，您可以通过使用autogen.sh脚本生成一组build脚本来启动build过程。

注意Bitcoin Core build过程已经从0.9开始更改为使用autogen / configure / make系统。 旧版本使用简单的Makefile，与以下示例的方法略有不同。 因此要按照要编译的版本的说明进行操作。 在0.9中引入的autogen / configure / make可能是用于所有未来版本代码的build系统，并且是以下示例中演示的系统。

```
$ ./autogen.sh
...
glibtoolize: copying file 'build-aux/m4/libtool.m4'
glibtoolize: copying file 'build-aux/m4/ltoptions.m4'
glibtoolize: copying file 'build-aux/m4/ltsugar.m4'
glibtoolize: copying file 'build-aux/m4/ltversion.m4'
...
configure.ac:10: installing 'build-aux/compile'
configure.ac:5: installing 'build-aux/config.guess'
configure.ac:5: installing 'build-aux/config.sub'
configure.ac:9: installing 'build-aux/install-sh'
configure.ac:9: installing 'build-aux/missing'
Makefile.am: installing 'build-aux/depcomp'
...
```

autogen.sh脚本创建一组自动配置脚本，它会询问系统以发现正确的设置，并确保您拥有编译代码所需的所有库。其中最重要的是配置脚本，它提供了许多不同的选项来自定义构建过程。 键入“./configure --help”查看各种选项：

```
$ ./configure --help
`configure' configures Bitcoin Core 0.11.2 to adapt to many kinds of systems.

Usage: ./configure [OPTION]... [VAR=VALUE]...

...
Optional Features:
  --disable-option-checking  ignore unrecognized --enable/--with options
  --disable-FEATURE       do not include FEATURE (same as --enable-FEATURE=no)
  --enable-FEATURE[=ARG]  include FEATURE [ARG=yes]
```

```
  --enable-wallet         enable wallet (default is yes)

  --with-gui[=no|qt4|qt5|auto]
...
```

配置脚本允许您通过使用--enable-FEATURE和--disable-FEATURE标志来启用或禁用bitcoind的某些功能，其中FEATURE由功能名称替换，如帮助输出中所列。 在本章中，我们将构建具有所有默认功能的bitcoind客户端。 我们不会使用配置标志，但您应该查看它们以了解可选功能是客户端的一部分。 如果您处于学术环境中，计算机实验室的限制可能需要您在主目录中安装应用程序（例如，使用--prefix = $ HOME）。

以下是一些有用的选项，可以覆盖configure脚本的默认行为：

--prefix=$HOME

这将覆盖生成的可执行文件的默认安装位置（它是/ usr / local /）。 使用$ HOME将所有内容放在您的主目录或不同的路径中。

--disable-wallet

这用于禁用参考钱包的实现。

--with-incompatible-bdb

如果您正在构建钱包，请允许使用不兼容版本的Berkeley DB库。

--with-gui=no

不要构建图形用户界面，图形界面需要Qt库。 这只构建服务器和命令行。

接下来，运行configure脚本来自动发现所有必需的库，并为您的系统创建一个自定义的构建脚本：

```
$ ./configure
checking build system type... x86_64-unknown-linux-gnu
checking host system type... x86_64-unknown-linux-gnu
checking for a BSD-compatible install... /usr/bin/install -c
checking whether build environment is sane... yes
checking for a thread-safe mkdir -p... /bin/mkdir -p
checking for gawk... gawk
checking whether make sets $(MAKE)... yes
...
[many pages of configuration tests follow]
...
$
```

如果一切顺利，configure命令将会以创建可定制的构建脚本结束。这些构建脚本允许我们编译bitcoind。如果有缺失的库或是错误，configure命令将会以错误信息终止。如果出现了错误，可能是因为缺少库或是有不兼容的库。重新检查构建文档，确认你已经安装缺失的必备条件。然后运行configure，看看错误是否消失了。

3.2.3构建Bitcoin核心可执行文件

下一步，你将编译源代码，这个过程根据CPU和内存资源不同，但一般可能需要1个小时完成。在编译的过程中，你应该过几秒或是几分钟看一下输出结果。如果出现了问题，你会看到错误。如果中断了，编译的过程可以在任何时候恢复。输入make命令就可以开始编译了：

```
$ make
Making all in src
  CXX      crypto/libbitcoinconsensus_la-hmac_sha512.lo
  CXX      crypto/libbitcoinconsensus_la-ripemd160.lo
  CXX      crypto/libbitcoinconsensus_la-sha1.lo
  CXX      crypto/libbitcoinconsensus_la-sha256.lo
  CXX      crypto/libbitcoinconsensus_la-sha512.lo
  CXX      libbitcoinconsensus_la-hash.lo
  CXX      primitives/libbitcoinconsensus_la-transaction.lo
  CXX      libbitcoinconsensus_la-pubkey.lo
  CXX      script/libbitcoinconsensus_la-bitcoinconsensus.lo
  CXX      script/libbitcoinconsensus_la-interpreter.lo

[... many more compilation messages follow ...]

$
```

如果一切顺利，bitcoind现在已经编译完成。最后一步就是通过sudo make install 命令，安装 bitcoind 可执行文件到你的系统路径下，可能会提示您输入用户密码，因为此步骤需要管理员权限：

```
$ sudo make install
Password:
Making install in src
 ../build-aux/install-sh -c -d '/usr/local/lib'
libtool: install: /usr/bin/install -c bitcoind /usr/local/bin/bitcoind
libtool: install: /usr/bin/install -c bitcoin-cli /usr/local/bin/bitcoin-cli
libtool: install: /usr/bin/install -c bitcoin-tx /usr/local/bin/bitcoin-tx
...
$
```

bitcoind 默认的安装位置是/usr/local/bin。你可以通过询问系统下面2个可执行文件的路径，来确认bitcoin是否安装成功。

```
$ which bitcoind
/usr/local/bin/bitcoind

$ which bitcoin-cli
/usr/local/bin/bitcoin-cli
```

3.2.4运行比特币核心节点

比特币的对等网络由网络“节点”组成，主要由志愿者和一些构建比特币应用程序的商业机构运行。 那些运行的比特币节点具有直接和权威的比特币区块链视图，并且具有所有交易的本地副本，由其自己的系统独立验证。 通过运行节点，您不必依赖任何第三方来验证交易。 此外，通过运行比特币节点，您可以通过使其更健壮的方式为比特币网络做出贡献。

但是，运行节点需要一个具有足够资源来处理所有比特币交易的永久连接的系统。 根据您是否选择索引所有交易并保留块的完整副本，您可能还需要大量的磁盘空间和RAM。 到2016年底，全索引节点需要2 GB的RAM和125 GB的磁盘空间，以便它有增长的空间。 比特币节点还传输和接收比特币交易和块，消耗互联网带宽。 如果您的互联网连接受限，有带宽上限或按流量计费，建议您不要在其上运行比特币全节点，或以限制其带宽的方式运行它（请参阅资源有限的系统）。

提示 Bitcoin Core默认情况下保留区块链的完整副本，与2009年成立以来在比特币网络上发生的每一笔交易相关。此数据集的大小为120GB，下载可能需要几天或几周，具体取决于 CPU和互联网连接的速度。直到完整的块链数据集被下载完成之前，Bitcoin Core将无法处理交易或更新帐户余额。 确保您有足够的磁盘空间，带宽和时间来完成初始同步。 您可以配置Bitcoin Core通过丢弃旧块来减少块链的大小（请参阅资源有限的系统），但是在丢弃数据之前仍将下载整个数据集。

尽管有这些资源需求，但仍有成千上万的志愿者运行比特币节点。 一些在简单的系统上运行，就像树莓派Raspberry Pi（一块35美元的计算机，一张卡的大小）。 许多志愿者还在租用的服务器上运行比特币节点，通常是Linux的一些变体。 虚拟专用服务器（VPS）或云计算服务器实例可用于运行比特币节点。 这些服务器可以从各种供应商每月租用25至50美元。

为什么要运行一个节点？ 以下是一些最常见的原因：

如果您正在开发比特币软件，并且需要依靠比特币节点进行可编程（API）访问网络和区块链。

如果您正在构建必须根据比特币共识规则验证交易的应用程序。 比特币软件公司通常运行几个节点。

如果你想支持比特币。 运行节点使网络更加健壮，能够提供更多的钱包，更多的用户和更多的交易。

如果您不想依赖任何第三方来处理或验证您的交易。

如果您正在阅读本书并对开发比特币软件感兴趣，那么您应该运行自己的节点。

3.2.5首次运行比特币核心

当你第一次运行bitcoind时，它会提醒你用一个安全密码给JSON-RPC接口创建一个配置文件。该密码控制对Bitcoin Core提供的应用程序编程接口（API）的访问。

通过在终端输入bitcoind就可以运行bitcoind了：

```
$ bitcoind
Error: To use the "-server" option, you must set a rpcpassword in the configuration
file:
/home/ubuntu/.bitcoin/bitcoin.conf
It is recommended you use the following random password:
rpcuser=bitcoinrpc
rpcpassword=2XA4DuKNCbtZXsBQRRNDEwEY2nM6M4H9Tx5dFjoAVVbK
(you do not need to remember this password)
The username and password MUST NOT be the same.
If the file does not exist, create it with owner-readable-only file permissions.
It is also recommended to set alertnotify so you are notified of problems;
for example: alertnotify=echo %s | mail -s "Bitcoin Alert" admin@foo.com
```

你可以看到，第一次运行bitcoind它会告诉你，你需要建立一个配置文件，至少有一个rpcuser和rpcpassword条目。 另外，建议您设置警报机制。 在下一节中，我们将介绍各种配置选项，并设置一个配置文件。

3.2.6配置比特币核心节点

在首选编辑器中编辑配置文件，并设置参数，用bitcoind推荐的强密码替换密码。 请勿使用本书中显示的密码。 在.bitcoin目录（在用户的主目录下）中创建一个文件，以便它被命名为.bitcoin / bitcoin.conf并提供用户名和密码：

```
rpcuser=bitcoinrpc
rpcpassword=CHANGE_THIS
```

除了rpcuser和rpcpassword选项，Bitcoin Core还提供了100多个配置选项，可以修改网络节点的行为，区块链的存储以及其操作的许多其他方面。

要查看这些选项的列表，请运行bitcoind --help：

```
 bitcoind --help
Bitcoin Core Daemon version v0.11.2

Usage:
  bitcoind [options]                     Start Bitcoin Core Daemon

Options:

  -?
       This help message

  -alerts
       Receive and display P2P network alerts (default: 1)

  -alertnotify=<cmd>
       Execute command when a relevant alert is received or we see a really
       long fork (%s in cmd is replaced by message)
...
[many more options]
...

  -rpcsslciphers=<ciphers>
       Acceptable ciphers (default:
       TLSv1.2+HIGH:TLSv1+HIGH:!SSLv2:!aNULL:!eNULL:!3DES:@STRENGTH)
```

以下是您可以在配置文件中设置的一些最重要的选项，或作为bitcoind的命令行参数：

alertnotify

运行指定的命令或脚本，通常通过电子邮件将紧急警报发送给该节点的所有者。

conf

配置文件的另一个位置。 这只是作为bitcoind的命令行参数有意义，因为它不能在它引用的配置文件内。

datadir

选择要放入所有块链数据的目录和文件系统。 默认情况下，这是您的主目录的.bitcoin子目录。 确保这个文件系统具有几GB的可用空间。

prune

通过删除旧的块，将磁盘空间要求降低到这个兆字节。 在资源受限的节点上不能满足完整块的节点使用这个。

txindex

维护所有交易的索引。 这意味着可以通过ID以编程方式检索任何交易的块链的完整副本。

maxconnections

设置接受连接的最大节点数。 从默认值减少该值将减少您的带宽消耗。 如果您的网络是按照流量计费，请使用。

maxmempool

将交易内存池限制在几兆字节。 使用它来减少节点的内存使用。

maxreceivebuffer/maxsendbuffer

将每连接内存缓冲区限制为1000字节的多个倍数。 在内存受限节点上使用。

minrelaytxfee

设置您将继续的最低费用交易。 低于此值，交易被视为零费用。 在内存受限的节点上使用它来减少内存中交易池的大小。

交易数据库索引和txindex选项

默认情况下，Bitcoin Core构建一个仅包含与用户钱包有关的交易的数据库。 如果您想要使用诸如getrawtransaction（参见探索和解码交易)之类的命令访问任何交易，则需要配置Bitcoin Core以构建完整的交易索引，这可以通过txindex选项来实现。 在Bitcoin Core配置文件中设置txindex = 1。 如果不想一开始设置此选项，后期再想设置为完全索引，则需要使用-reindex选项重新启动bitcoind，并等待它重建索引。

下面的完整索引节点的例子配置显示了如何将上述选项与完全索引节点组合起来，作为比特币应用程序的API后端运行。

例3-1完整索引节点的例子

```
alertnotify=myemailscript.sh "Alert: %s"
datadir=/lotsofspace/bitcoin
txindex=1
rpcuser=bitcoinrpc
rpcpassword=CHANGE_THIS
```

下面是小型服务器资源不足配置示例。

例3-2小型服务器资源不足配置示例

```
alertnotify=myemailscript.sh "Alert: %s"
maxconnections=15
prune=5000
minrelaytxfee=0.0001
maxmempool=200
maxreceivebuffer=2500
maxsendbuffer=500
rpcuser=bitcoinrpc
rpcpassword=CHANGE_THIS
```

编辑配置文件并设置最符合您需求的选项后，可以使用此配置测试 bitcoind。 运行Bitcoin Core，使用选项printtoconsole在前台运行输出到控制台：

```
$ bitcoind -printtoconsole

Bitcoin version v0.11.20.0
Using OpenSSL version OpenSSL 1.0.2e 3 Dec 2015
Startup time: 2015-01-02 19:56:17
Using data directory /tmp/bitcoin
Using config file /tmp/bitcoin/bitcoin.conf
Using at most 125 connections (275 file descriptors available)
Using 2 threads for script verification
scheduler thread start
HTTP: creating work queue of depth 16
No rpcpassword set - using random cookie authentication
Generated RPC authentication cookie /tmp/bitcoin/.cookie
HTTP: starting 4 worker threads
Bound to [::]:8333
Bound to 0.0.0.0:8333
Cache configuration:
* Using 2.0MiB for block index database
* Using 32.5MiB for chain state database
* Using 65.5MiB for in-memory UTXO set
init message: Loading block index...
Opening LevelDB in /tmp/bitcoin/blocks/index
Opened LevelDB successfully

[... more startup messages ...]
```

一旦您确信正在加载正确的设置并按预期运行，您可以按Ctrl-C中断进程。要在后台运行Bitcoin Core作为进程，请使用守护程序选项启动它，如bitcoind -daemon。要监视比特币节点的进度和运行状态，请使用命令bitcoin-cli getinfo：

```
$ bitcoin-cli getinfo
{
    "version" : 110200,
    "protocolversion" : 70002,
    "blocks" : 396328,
    "timeoffset" : 0,
    "connections" : 15,
    "proxy" : "",
    "difficulty" : 120033340651.23696899,
    "testnet" : false,
    "relayfee" : 0.00010000,
    "errors" : ""
}
```

这显示运行Bitcoin Core版本0.11.2的节点，块链接高度为396328个块和15个活动网络连接。

一旦您对所选择的配置选项感到满意，您应该将bitcoin添加到操作系统中的启动脚本中，以使其连续运行，并在操作系统重新启动时自动启动。 您可以在contrib / init下的bitcoin的源目录中的各种操作系统和README.md文件中找到一些示例启动脚本，显示哪个系统使用哪个脚本。

3.3 通过命令行使用比特币核心的JSON-RPC API接口

比特币核心客户端实现了JSON-RPC接口，这个接口也可以通过命令行帮助程序bitcoin-cli访问。命令行可以使用API进行编程，让我们有能力进行交互实验。开始前，调用help命令查看可用的比特币RPC命令列表：

```
$ bitcoin-cli help
addmultisigaddress nrequired ["key",...] ( "account" )
addnode "node" "add|remove|onetry"
backupwallet "destination"
createmultisig nrequired ["key",...]
createrawtransaction [{"txid":"id","vout":n},...] {"address":amount,...}
decoderawtransaction "hexstring"
...
...
verifymessage "bitcoinaddress" "signature" "message"
walletlock
walletpassphrase "passphrase" timeout
walletpassphrasechange "oldpassphrase" "newpassphrase"
```

这些命令中的每一个可能需要多个参数。 要获得更多帮助，详细说明和参数信息，请在帮助后添加命令名称。 例如，要查看getblockhash RPC命令的帮助：

```
$ bitcoin-cli help getblockhash
getblockhash index

Returns hash of block in best-block-chain at index provided.
```

```
Arguments:
1. index         (numeric, required) The block index

Result:
"hash"         (string) The block hash

Examples:
> bitcoin-cli getblockhash 1000
> curl --user myusername --data-binary '{"jsonrpc": "1.0", "id":"curltest", "method":
"getblockhash", "params": [1000] }' -H 'content-type: text/plain;'
http://127.0.0.1:8332/
```

在帮助信息的最后，您将看到RPC命令的两个示例，使用bitcoin-cli helper或HTTP客户端的curl。 这些例子演示如何调用命令。 复制第一个示例并查看结果：

```
$ bitcoin-cli getblockhash 1000
00000000c937983704a73af28acdec37b049d214adbda81d7e2a3dd146f6ed09
```

结果是一个区块哈希，这在下面的章节中有更详细的描述。 但是现在，该命令应该在您的系统上返回相同的结果，表明您的Bitcoin Core节点正在运行，正在接受命令，并且有关于块1000的信息返回给您。

在下一节中，我们将演示一些非常有用的RPC命令及其预期输出。

3.3.1 获得比特币核心客户端状态的信息

命令： getinfo

比特币 getinfo RPC命令显示关于比特币网络节点、钱包、区块链数据库状态的基础信息。使用 bitcoin-cli 运行它：

```
$ bitcoin-cli getinfo
{
    "version" : 110200,
    "protocolversion" : 70002,
    "blocks" : 396367,
    "timeoffset" : 0,
    "connections" : 15,
    "proxy" : "",
    "difficulty" : 120033340651.23696899,
    "testnet" : false,
    "relayfee" : 0.00010000,
    "errors" : ""
}
```

数据以JavaScript对象表示法（JSON）返回，这是一种格式，可以轻松地被所有编程语言“消费”，但也是非常人性化的。 在这些数据中，我们看到比特币软件客户端（110200）和比特币协议（70002）的版本号。 我们看到当前的块高度，显示了这个客户端知道了多少块（396367）。 我们还会看到有关比特币网络和与此客户端相关的设置的各种统计信息。

提示 比特币特客户端"赶上"当前的blockchain高度需要一些时间，因为它从其他bitcoin客户端下载块。 您可以使用getinfo检查其进度，以查看已知块的数量。

3.3.1 .1探索和解码交易

命令：getrawtransaction，decodeawtransaction

在买咖啡的故事中，Alice从Bob咖啡厅买了一杯咖啡。 她的交易记录在交易ID（txid）0627052b6f28912f2703066a912ea577f2ce4da4caa5a5fbd8a57286c345c2f2的封锁上。 我们使用API通过传递交易ID作为参数来检索和检查该交易：

```
$ bitcoin-cli getrawtransaction 0627052b6f28912f2703066a912ea577f2ce4da4caa5a↵
5fbd8a57286c345c2f2

0100000001186f9f998a5aa6f048e51dd8419a14d8a0f1a8a2836dd734d2804fe65fa35779000↵
000008b483045022100884d142d86652a3f47ba4746ec719bbfbd040a570b1deccbb6498c75c4↵
ae24cb02204b9f039ff08df09cbe9f6addac960298cad530a863ea8f53982c09db8f6e3813014↵
10484ecc0d46f1918b30928fa0e4ed99f16a0fb4fde0735e7ade8416ab9fe423cc54123363767↵
89d172787ec3457eee41c04f4938de5cc17b4a10fa336a8d752adfffffffff0260e3160000000↵
0001976a914ab68025513c3dbd2f7b92a94e0581f5d50f654e788acd0ef8000000000001976a9↵
147f9b1a7fb68d60c536c2fd8aeaa53a8f3cc025a888ac00000000
```

提示交易ID在交易被确认之前不具有权威性。 在区块链中缺少交易哈希并不意味着交易未被处理。 这被称为"交易可扩展性"，因为在块中确认之前可以修改交易哈希。 确认后，txid是不可改变的和权威的。

命令getrawtransaction以十六进制返回顺序交易。 为了解码，我们使用decodeawtransaction命令，将十六进制数据作为参数传递。 您可以复制getrawtransaction返回的十六进制，并将其作为参数粘贴到decodeawtransaction中：

```
$ bitcoin-cli decoderawtransaction 0100000001186f9f998a5aa6f048e51dd8419a14d8↵
a0f1a8a2836dd734d2804fe65fa35779000000008b483045022100884d142d86652a3f47ba474↵
6ec719bbfbd040a570b1deccbb6498c75c4ae24cb02204b9f039ff08df09cbe9f6addac960298↵
cad530a863ea8f53982c09db8f6e381301410484ecc0d46f1918b30928fa0e4ed99f16a0fb4fd↵
e0735e7ade8416ab9fe423cc5412336376789d172787ec3457eee41c04f4938de5cc17b4a10fa↵
336a8d752adfffffffff0260e31600000000001976a914ab68025513c3dbd2f7b92a94e0581f5↵
d50f654e788acd0ef8000000000001976a9147f9b1a7fb68d60c536c2fd8aeaa53a8f3cc025a8↵
88ac00000000
```

```
{
  "txid": "0627052b6f28912f2703066a912ea577f2ce4da4caa5a5fbd8a57286c345c2f2",
  "size": 258,
  "version": 1,
  "locktime": 0,
  "vin": [
    {
      "txid": "7957a35fe64f80d234d76d83a2...8149a41d81de548f0a65a8a999f6f18",
      "vout": 0,
      "scriptSig": {
        "asm":"3045022100884d142d86652a3f47ba4746ec719bbfbd040a570b1decc...",
        "hex":"483045022100884d142d86652a3f47ba4746ec719bbfbd040a570b1de..."
```

```
      },
      "sequence": 4294967295
    }
  ],
  "vout": [
    {
      "value": 0.01500000,
      "n": 0,
      "scriptPubKey": {
        "asm": "OP_DUP OP_HASH160 ab68...5f654e7 OP_EQUALVERIFY OP_CHECKSIG",
        "hex": "76a914ab68025513c3dbd2f7b92a94e0581f5d50f654e788ac",
        "reqSigs": 1,
        "type": "pubkeyhash",
        "addresses": [
          "1GdK9UzpHBzqzX2A9JFP3Di4weBwqgmoQA"
        ]
      }
    },
    {
      "value": 0.08450000,
      "n": 1,
      "scriptPubKey": {
        "asm": "OP_DUP OP_HASH160 7f9b1a...025a8 OP_EQUALVERIFY OP_CHECKSIG",
        "hex": "76a9147f9b1a7fb68d60c536c2fd8aeaa53a8f3cc025a888ac",
        "reqSigs": 1,
        "type": "pubkeyhash",
        "addresses": [
          "1Cdid9KFAaatwczBwBttQcwXYCpvK8h7FK"
        ]
      }
    }
  ]
}
```


交易解码展示这笔交易的所有成分，包括交易的输入及输出。在这个例子中，我们可以看到这笔给我们新地址存入50mBTC的交易使用了一个输入并且产生两个输出。这笔交易的输入是前一笔确认交易的输出（展示位以d3c7开头的 vin txid）。两个输出则是50mBTC存入额度及返回给发送者的找零。

我们可以使用相同命令（例如 gettransaction ）通过检查由本次交易的txid索引的前一笔交易进一步探索区块链。通过从 一笔交易跳到另外一笔交易，我们可以追溯一连串的交易，因为币值一定是从一个拥有者的地址传送到另一个拥有者的地址。

3.3.2 探索区块

命令： getblock 、 getblockhash

探索区块类似于探索交易。 但是，块可以由块高度或块哈希引用。 首先，让我们找到一个块的高度。 在买咖啡故事中，我们看到Alice的交易已被包含在框277316中。我们使用getblockhash命令，它将块高度作为参数，并返回该块的块哈希值：

```
$ bitcoin-cli getblockhash 277316
000000000000001b6b9a13b095e96db41c4a928b97ef2d944a9b31b2cc7bdc4
```

既然我们知道我们的交易在哪个区块中，我们可以使用getblock命令，并把区块哈希值作为参数来查询对应的区块：

```
$ bitcoin-cli getblock 000000000000001b6b9a13b095e96db41c4a928b97ef2d944a9b3↵
1b2cc7bdc4
```


```
{
  "hash": "000000000000001b6b9a13b095e96db41c4a928b97ef2d944a9b31b2cc7bdc4",
  "confirmations": 37371,
  "size": 218629,
  "height": 277316,
  "version": 2,
  "merkleroot": "c91c008c26e50763e9f548bb8b2fc323735f73577effbc55502c51eb4cc7cf2e",
  "tx": [
    "d5ada064c6417ca25c4308bd158c34b77e1c0eca2a73cda16c737e7424afba2f",
    "b268b45c59b39d759614757718b9918caf0ba9d97c56f3b91956ff877c503fbe",
    "04905ff987ddd4cfe603b03cfb7ca50ee81d89d1f8f5f265c38f763eea4a21fd",
    "32467aab5d04f51940075055c2f20bbd1195727c961431bf0aff8443f9710f81",
    "561c5216944e21fa29dd12aaa1a45e3397f9c0d888359cb05e1f79fe73da37bd",
[... hundreds of transactions ...]
    "78b300b2a1d2d9449b58db7bc71c3884d6e0579617e0da4991b9734cef7ab23a",
    "6c87130ec283ab4c2c493b190c20de4b28ff3caf72d16ffa1ce3e96f2069aca9",
    "6f423dbc3636ef193fd8898dfdf7621dcade1bbe509e963ffbff91f696d81a62",
    "802ba8b2adabc5796a9471f25b02ae6aeee2439c679a5c33c4bbcee97e081196",
    "eaaf6a048588d9ad4d1c092539bd571dd8af30635c152a3b0e8b611e67d1a1af",
    "e67abc6bd5e2cac169821afc51b207127f42b92a841e976f9b752157879ba8bd",
    "d38985a6a1bfd35037cb7776b2dc86797abbb7a06630f5d03df2785d50d5a2ac",
    "45ea0a3f6016d2bb90ab92c34a7aac9767671a8a84b9bcce6c019e60197c134b",
    "c098445d748ced5f178ef2ff96f2758cbec9eb32cb0fc65db313bcac1d3bc98f"
  ],
  "time": 1388185914,
  "mediantime": 1388183675,
  "nonce": 924591752,
  "bits": "1903a30c",
  "difficulty": 1180923195.258026,
  "chainwork": "0000000000000000000000000000000000000000000000934695e92aaf53afa1a",
  "previousblockhash":
"0000000000000002a7bbd25a417c0374cc55261021e8a9ca74442b01284f0569",
  "nextblockhash": "000000000000000010236c269dd6ed714dd5db39d36b33959079d78dfd431ba7"
}
```


该块包含419笔交易，列出的第64笔交易（0627052b ...）是Alice的咖啡付款。 高度条目告诉我们这是区块链中的第277316块。

3.3.3使用比特币核心的编程接口

bitcoin-cli helper对于探索Bitcoin Core API和测试功能非常有用。 但是应用编程接口的全部要点是以编程方式访问功能。 在本节中，我们将演示从另一个程序访问Bitcoin Core。

Bitcoin Core的API是一个JSON-RPC接口。 JSON代表JavaScript对象符号，它是一种非常方便的方式来呈现人类和程序可以轻松阅读的数据。 RPC代表远程过程调用，这意味着我们通过网络协议调用远程（位于比特币核心节点）的过程（函数）。 在这种情况下，网络协议是HTTP或HTTPS（用于加密连接）。

当我们使用bitcoin-cli命令获取命令的帮助时，它给了我们一个例子，它使用curl，通用的命令行HTTP客户端来构造这些JSON-RPC调用之一：

```
$ curl --user myusername --data-binary '{"jsonrpc": "1.0", "id":"curltest", "method":
"getinfo", "params": [] }' -H 'content-type: text/plain;' http://127.0.0.1:8332/
```

此命令显示curl向本地主机（127.0.0.1）提交HTTP请求，连接到默认比特币端口（8332），并使用text / plain编码向getinfo方法提交jsonrpc请求。

如果您在自己的程序中实现JSON-RPC调用，则可以使用通用的HTTP库构建调用，类似于前面的curl示例所示。

然而，大多数编程语言中都使用库，以“包装”比特币核心API的方式使其简单得多。 我们将使用python-bitcoinlib库来简化API访问。 请记住，这需要您有一个运行的Bitcoin Core实例，将用于进行JSON-RPC调用。

下面的例子通运行getinfo中的Python脚本进行简单的getinfo调用，并从Bitcoin Core返回的数据中打印区块参数。

例3-3通过Bitcoin Core的JSON-RPC API运行

```
link:code/rpc_example.py[]
```

运行结果如下：

```
$ python rpc_example.py
394075
```

它告诉我们，我们的本地Bitcoin Core节点在其块链中有394075个块。 这不是一个惊人的结果，但它演示了使用库作为Bitcoin Core的JSON-RPC API的简化接口的基本使用。

接下来，我们使用getrawtransaction和decodetransaction调用来检索Alice咖啡付款的详细信息。 在下面的例子中，我们检索Alice的交易并列出交易的输出。 对于每个输出，我们显示收件人地址和值。 作为提醒，Alice的交易有一个输出支付Bob的咖啡馆和一个输出找回Alice。

例3-4检索交易并迭代其输出

```
link:code/rpc_transaction.py[]
```

运行结果如下：

```
$ python rpc_transaction.py
([u'1GdK9UzpHBzqzX2A9JFP3Di4weBwqgmoQA'], Decimal('0.01500000'))
([u'1Cdid9KFAaatwczBwBttQcwXYCpvK8h7FK'], Decimal('0.08450000'))
```

上述两个例子都比较简单。 你真的不需要一个程序来运行它们; 你可以很容易地使用bitcoin-cli helper。 然而，下一个示例需要数百个RPC调用，并更清楚地说明了使用编程接口的方便。

在时，我们首先检索块277316，然后通过引用每个交易ID来检索每个419个交易。 接下来，我们迭代每个交易的输出并将其加起来。例3-5检索块并添加所有交易输出

```
link:code/rpc_block.py[]
```

运行结果如下：

```
$ python rpc_block.py

('Total value in block: ', Decimal('10322.07722534'))
```

我们的示例代码计算出，此块中交易的总价值为10,322.07722534 个BTC（包括25 BTC奖励和0.0909 BTC费用）。 通过搜索块哈希或高度来比较区块浏览器站点报告的数量。 一些区块浏览器报告不包括奖励的总价值，不包括交易费用。 看看是否可以发现差异。

3.4 其他替代客户端、资料库、工具包

除了参考客户端（bitcoind），还可以使用其他的客户端和资料库去连接比特币网络和数据结构。这些工具都由一系列 的编程语言执行，用他们各自的语言为比特币程序提供原生的交互。以下列出了一部分由编程语言组织的一些最好的库，客户端和工具包：

C/C++

Bitcoin Core The reference implementation of bitcoin

libbitcoin Cross-platform C++ development toolkit, node, and consensus library

bitcoin explorer Libbitcoin's command-line tool

picocoin A C language lightweight client library for bitcoin by Jeff Garzik

JavaScript

bcoin A modular and scalable full-node implementation with API

Bitcore Full node, API, and library by Bitpay

BitcoinJS A pure JavaScript Bitcoin library for node.js and browsers

Java

bitcoinj A Java full-node client library

Bits of Proof (BOP) A Java enterprise-class implementation of bitcoin

Python

python-bitcoinlib A Python bitcoin library, consensus library, and node by Peter Todd

pycoin A Python bitcoin library by Richard Kiss

pybitcointools A Python bitcoin library by Vitalik Buterin

Ruby

bitcoin-client A Ruby library wrapper for the JSON-RPC API

Go

btcd A Go language full-node bitcoin client

Rust

rust-bitcoin Rust bitcoin library for serialization, parsing, and API calls

C#

NBitcoin Comprehensive bitcoin library for the .NET framework

Objective-C

CoreBitcoin Bitcoin toolkit for ObjC and Swift

更多的库存在各种其他编程语言，也会一直更新的。