3 Bitcoin Core：参考实现

Bitcoin is an *open source* project and the source code is available under an open (MIT) license, free to download and use for any purpose. Open source means more than simply free to use. It also means that bitcoin is developed by an open community of volunteers. At first, that community consisted of only Satoshi Nakamoto. By 2016, bitcoin's source code had more than 400 contributors with about a dozen developers working on the code almost full-time and several dozen more on a part-time basis. Anyone can contribute to the code—including you!
比特币是一个开源项目，源代码根据一个开放（MIT）许可证提供，可免费下载和用于任何目的。
开源不仅仅是自由使用，也意味着比特币是由一个开放的志愿者社区开发的。
最初，这个社区只有中本聪。到了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全职工作，几十名开发人员兼职。任何人都可以为代码做贡献！

When bitcoin was created by Satoshi Nakamoto, the software was actually completed before the whitepaper reproduced in [satoshi_whitepaper] was written. Satoshi wanted to make sure it worked before writing about it. That first implementation, then simply known as "Bitcoin" or "Satoshi client," has been heavily modified and improved. It has evolved into what is known as *Bitcoin Core*, to differentiate it from other compatible implementations. Bitcoin Core is the *reference implementation* of the bitcoin system, meaning that it is the authoritative reference on how each part of the technology should be implemented. Bitcoin Core implements all aspects of bitcoin, including wallets, a transaction and block validation engine, and a full network node in the peer-to-peer bitcoin network.
当由中本聪创建比特币时，在白皮书发表之前，这个软件实际上已经完成。
中本聪想在白皮书之前确保它能有效工作。
这个第一个实现，称为"比特币Bitcoin"或"Satoshi客户端"，实际上已经被大大修改和改进了。
它已经演变成了Bitcoin Core，以区别于其它兼容的实现。
Bitcoin Core是比特币系统的参考实现，这意味着它权威参考：应该如何实现这个技术的每个部分。
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，以及在P2P比特币网络中的全网络节点。

Warning: Even though Bitcoin Core includes a reference implementation of a wallet, this is not intended to be used as a production wallet for users or for applications. Application developers are advised to build wallets using modern standards such as BIP-39 and BIP-32 (see [mnemonic_code_words] and [hd_wallets]). BIP stands for Bitcoin Improvement Proposal.
警告：虽然Bitcoin Core包含钱包的参考实现，但并不意味着要用作用户或应用的实际钱包。
建议应用开发人员使用现代标准（如BIP-39和BIP-32）构建钱包。
BIP表示Bitcoin Improvement Proposal（比特币改进提案）。

Bitcoin Core architecture (Source: Eric Lombrozo) shows the architecture of Bitcoin Core.
下图为Bitcoin Core的架构。

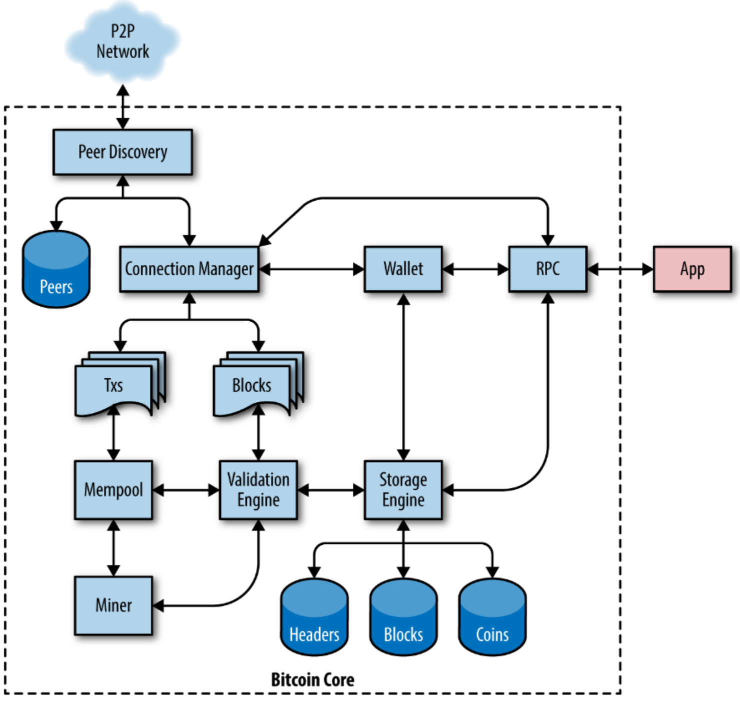

Figure 1. Bitcoin Core architecture (Source: Eric Lombrozo)

3.1比特币开发环境

If you’re a developer, you will want to set up a development environment with all the tools, libraries, and support software for writing bitcoin applications. In this highly technical chapter, we’ll walk through that process step-by-step. If the material becomes too dense (and you’re not actually setting up a development environment) feel free to skip to the next chapter, which is less technical.
如果你是开发人员，你要设置一个开发环境，有所有工具、库和支持软件，以便编写比特币应用程序。
本章涉及的技术细节较多，我们将逐步介绍这个过程。
如果你觉得过于繁琐（并且你实际上并不打算设置开发环境），可以跳到下一章，下一章的技术会浅显一些。

3.2从源码编译Bitcoin Core

Bitcoin Core’s source code can be downloaded as a archive or by cloning the authoritative source repository from GitHub. On the Bitcoin Core download page, select the most recent version and download the compressed archive of the source code, e.g., bitcoin-0.15.0.2.tar.gz. Alternatively, use the git command line to create a local copy of the source code from the GitHub bitcoin page.
Bitcoin Core的源代码可以下载为压缩文件，也可以从GitHub克隆权威的源码库。

- 在Bitcoin Core下载页，选择最近的版本，下载源码的压缩文件，例如bitcoin-0.15.0.2.tar.gz。
  https://bitcoincore.org/bin/
- 使用git命令从GitHub bitcoin页面下载源码。
  https://github.com/bitcoin/bitcoin

Tip: In many of the examples in this chapter we will be using the operating system’s command-line interface (also known as a "shell"), accessed via a "terminal" application.

The shell will display a prompt; you type a command; and the shell responds with some text and a new prompt for your next command. The prompt may look different on your system, but in the following examples it is denoted by a $ symbol. In the examples, when you see text after a $ symbol, don't type the $ symbol but type the command immediately following it, then press Enter to execute the command. In the examples, the lines below each command are the operating system's responses to that command. When you see the next $ prefix, you'll know it's a new command and you should repeat the process.
提示：在本章的许多例子中，我们使用操作系统的命令行界面（shell），通过terminal应用程序访问。 shell提示你输入命令，并且输出一些文本和一个新的提示，你输入下一个命令。
提示符可能与你的系统上看起来不同，在以下示例中，用$符号表示。
在示例中，不要输入$，而是输入它之后的命令。然后按回车键执行。
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$时，可以输入新的命令，可以一直重复这个过程。

In this example, we are using the git command to create a local copy ("clone") of the source code:
在本例中，我们用git命令把源代码clone到本地。

```
$ git clone https://github.com/bitcoin/bitcoin.git
Cloning into 'bitcoin'...
remote: Counting objects: 102071, done.
remote: Compressing objects: 100% (10/10), done.
Receiving objects: 100% (102071/102071), 86.38 MiB | 730.00 KiB/s, done.
remote: Total 102071 (delta 4), reused 5 (delta 1), pack-reused 102060
Resolving deltas: 100% (76168/76168), done.
Checking connectivity... done.
$
```

Tip：Git is the most widely used distributed version control system, an essential part of any software developer's toolkit. You may need to install the git command, or a graphical user interface for git, on your operating system if you do not have it already.
提示：Git是最广泛使用的分布式版本控制系统，是软件开发人员工具包的重要组成部分。
你可能需要在操作系统上安装git命令或图形用户界面。

When the git cloning operation has completed, you will have a complete local copy of the source code repository in the directory *bitcoin*. Change to this directory by typing **cd bitcoin** at the prompt:
当git克隆操作完成后，将在bitcoin目录中有完整的源代码。
输入下面命令进入为此目录。

```
$ cd bitcoin
```

3.2.1选择一个Bitcoin Core版本

By default, the local copy will be synchronized with the most recent code, which might be an unstable or beta version of bitcoin. Before compiling the code, select a specific version by checking out a release *tag*. This will synchronize the local copy with a specific snapshot of the code repository identified by a keyword tag. Tags are used by the developers to mark specific releases of the code by version number. First, to find the available tags, we use the git tag command:
默认情况下，下载的源代码是最新的代码，这可能是一个不稳定版本或Beta版本。
在编译代码之前，先获取一个release tag，来选择一个特定的版本。
这将使本地副本与keyword tag所标识的代码库的特定快照同步。
开发人员使用tag来用版本号标记代码特定release。
首先，要找到可用的tags，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
...
```

The list of tags shows all the released versions of bitcoin. By convention, *release candidates*, which are intended for testing, have the suffix "rc." Stable releases that can be run on production systems have no suffix. From the preceding list, select the highest version release, which at the time of writing was v0.15.0. To synchronize the local code with this version, use the git checkout command:
tag列表显示所有的发布版本。
根据惯例，用于测试的发布版本有后缀“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，在本书写作时是v0.15.0。
为了使本地代码与此版本同步，使用git checkout命令：

```
$ git checkout v0.15.0
HEAD is now at 3751912... Merge #11295: doc: Old fee_estimates.dat are discarded by
0.15.0
```

You can confirm you have the desired version "checked out" by issuing the command git status:
使用git status命令，可以确认你有了所需的版本。

```
$ git status
HEAD detached at v0.15.0
nothing to commit, working directory clean
```

3.2.2配置Bitcoin Core Build

The source code includes documentation, which can be found in a number of files. Review the main documentation located in *README.md* in the *bitcoin* directory by typing **more README.md** at the prompt and using the spacebar to progress to the next page. In this chapter, we will build the command-line bitcoin client, also known as bitcoind on Linux. Review the instructions for compiling the bitcoind command-line client on your platform by typing **more doc/build-unix.md**. Alternative instructions for macOS and Windows can be found in the *doc* directory, as *build-osx.md* or *build-windows.md*, respectively.
源代码中包括文档，可以在多个文件中找到。
主文档是bitcoin目录中的README.md。
在本章中，我们将构建命令行比特币客户端，在Linux也称为bitcoind。
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的构建说明，分别为build-osx.md或build-windows.md。

Carefully review the build prerequisites, which are in the first part of the build documentation. These are libraries that must be present on your system before you can begin to compile bitcoin. If these prerequisites are missing, the build process will fail with an error. If this happens because you missed a prerequisite, you can install it and then resume the build process from where you left off. Assuming the prerequisites are installed, you start the build process by generating a set of build scripts using the *autogen.sh* script.
仔细查看构建的前提条件，这些前提在构建文档的第一部分。
这些是在你开始编译比特币之前必须存在于系统上的库。
如果缺少这些条件，构建过程会失败，并提示错误。

如果失败是因为您缺失条件，可以安装它，然后恢复构建过程。
假设要求的库都已经安装了，可以通过使用autogen.sh脚本生成一组构建脚本来启动构建过程。

```
$ ./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'
...
```

The *autogen.sh* script creates a set of automatic configuration scripts that will interrogate your system to discover the correct settings and ensure you have all the necessary libraries to compile the code. The most important of these is the configure script that offers a number of different options to customize the build process. Type **./configure --help** to see the various options:
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，并确保你有编译代码所需的所有库。
其中最重要的是configure脚本，它提供了许多不同的选项来定制构建过程。
输入“./configure --help”查看各种选项。

```
$ ./configure --help
`configure' configures Bitcoin Core 0.15.0 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]
...
```

The configure script allows you to enable or disable certain features of bitcoind through the use of the --enable-FEATURE and --disable-FEATURE flags, where FEATURE is replaced by the feature name, as listed in the help output.
configure脚本允许你使用--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，就像help输出列出的那样。

In this chapter, we will build the bitcoind client with all the default features. We won't be using the configuration flags, but you should review them to understand what optional features are part of the client. If you are in an academic setting, computer lab restrictions may require you to install applications in your home directory (e.g., using --prefix=$HOME).
在本章中，我们构建的bitcoind客户端有所有默认功能。
我们不使用配置标志，但你应该查看它们，以了解客户端提供了哪些可选功能。
如果你在学校环境，计算机实验室限制可能需要你在主目录中安装应用程序（例如，使用--prefix = $ HOME）。

Here are some useful options that override the default behavior of the configure script:
以下是一些有用的选项，修改了configure脚本的默认行为：

--prefix=$HOME
This overrides the default installation location (which is */usr/local/*) for the resulting executable. Use `$HOME` to put everything in your home directory, or a different path.
这将修改生成的可执行文件的默认安装位置，默认是 /usr/local/。
使用$HOME将所有内容放在主目录中。

--disable-wallet
This is used to disable the reference wallet implementation.
用于禁用参考钱包的实现。

--with-incompatible-bdb
If you are building a wallet, allow the use of an incompatible version of the Berkeley DB library.
如果你在构建一个钱包，这允许使用不兼容版本的Berkeley DB库。

--with-gui=no
Don't build the graphical user interface, which requires the Qt library. This builds server and command-line bitcoin only.
不构建GUI（GUI要要Qt库）。
这只构建服务器和命令行。

Next, run the configure script to automatically discover all the necessary libraries and create a customized build script for your system:
接下来，运行configure脚本来自动发现所有必需的库，并为你的系统创建一个定制的构建脚本。

```
$ ./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]
...
$
```

If all went well, the configure command will end by creating the customized build scripts that will allow us to compile bitcoind. If there are any missing libraries or errors, the configure command will terminate with an error instead of creating the build scripts. If an error occurs, it is most likely because of a missing or incompatible library. Review the build documentation again and make sure you install the missing prerequisites. Then run configure again and see if that fixes the error.
如果一切顺利，configure命令将会创建定制的构建脚本，可用于编译bitcoind。
如果缺失库或有错误，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，确认你已经安装缺少的条件。然后运行configure，看看错误是否解决了。

3.2.3构建Bitcoin Core可执行程序

Next, you will compile the source code, a process that can take up to an hour to complete, depending on the speed of your CPU and available memory. During the compilation process you should see output every few seconds or every few minutes, or an error if

something goes wrong. If an error occurs, or the compilation process is interrupted, it can be resumed any time by typing make again. Type **make** to start compiling the executable application:
下一步，你将编译源代码，这个过程一般可能需要1个小时完成。
在编译的过程中，你应该经常看一下输出结果。如果出现了问题，会显示错误。
如果发生了错误，编译过程会中断，可以输入make来恢复执行。
输入make，开始编译这个可执行应用程序。

```
$ 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 ...]

$
```

On a fast system with more than one CPU, you might want to set the number of parallel compile jobs. For instance, make -j 2 will use two cores if they are available. If all goes well, Bitcoin Core is now compiled. You should run the unit test suite with make check to ensure the linked libraries are not broken in obvious ways. The final step is to install the various executables on your system using the make install command. You may be prompted for your user password, because this step requires administrative privileges:
如果系统由多个CPU，可以并行编译。例如使用 make —j2。
如果一切顺利，Bitcoin Core现在已经编译完成。
你应该使用make check来运行单元测试包，以保证链接库没有被破坏。
最后一步是使用make install命令安装各种可执行程序。
可能会提示你输入用户密码，因为这一步需要管理员权限。

```
$ make check && 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
...
$
```

The default installation of bitcoind puts it in */usr/local/bin*. You can confirm that Bitcoin Core is correctly installed by asking the system for the path of the executables, as follows:
bitcoind 默认的安装位置是/usr/local/bin。
你可以通过询问系统中可执行文件的路径，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运行一个Bitcoin Core节点

Bitcoin’s peer-to-peer network is composed of network "nodes," run mostly by volunteers and some of the businesses that build bitcoin applications. Those running bitcoin nodes have a direct and authoritative view of the bitcoin blockchain, with a local copy of all the transactions, independently validated by their own system. By running a node, you don’t have to rely on any third party to validate a transaction. Moreover, by running a bitcoin node you contribute to the bitcoin network by making it more robust.
比特币的P2P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有一个直接和权威的比特币区块链视图，并且有所有交易，由各自的系统独立验证。

通过运行一个节点，你不必依赖任何第三方来验证一个交易。
此外，通过运行一个比特币节点，你为比特币网络做了贡献：使之更加健壮。

Running a node, however, requires a permanently connected system with enough resources to process all bitcoin transactions. Depending on whether you choose to index all transactions and keep a full copy of the blockchain, you may also need a lot of disk space and RAM. As of early 2018, a full-index node needs 2 GB of RAM and a minimum of 160 GB of disk space (see https://blockchain.info/charts/blocks-size). Bitcoin nodes also transmit and receive bitcoin transactions and blocks, consuming internet bandwidth. If your internet connection is limited, has a low data cap, or is metered (charged by the gigabit), you should probably not run a bitcoin node on it, or run it in a way that constrains its bandwidth (see Sample configuration of a resource-constrained system).
但是，运行一个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。
根据你是否选择索引所有交易并保留完整区块链，你可能还需要大量的磁盘空间和RAM。
到2018年初，一个全索引节点需要2GB RAM和160 GB磁盘空间。
参见：https://blockchain.info/charts/blocks-size
比特币节点还传输和接收比特币交易和区块，消耗互联网带宽。
如果你的互联网连接受限，有带宽上限或按流量计费，建议你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）。

Tip：Bitcoin Core keeps a full copy of the blockchain by default, with every transaction that has ever occurred on the bitcoin network since its inception in 2009. This dataset is dozens of gigabytes in size and is downloaded incrementally over several days or weeks, depending on the speed of your CPU and internet connection. Bitcoin Core will not be able to process transactions or update account balances until the full blockchain dataset is downloaded. Make sure you have enough disk space, bandwidth, and time to complete the initial synchronization. You can configure Bitcoin Core to reduce the size of the blockchain by discarding old blocks (see Sample configuration of a resource-constrained system), but it will still download the entire dataset before discarding data.
提示：Bitcoin Core默认情况下保留完整区块链，从2009年以来在比特币网络上发生的每一笔交易。
此数据集的大小为160GB，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。
在完整的区块链数据集被下载完成之前，Bitcoin Core无法处理交易或更新帐户余额。
确保你有足够的磁盘空间、带宽和时间来完成初始同步。
你可以配置Bitcoin Core来减少区块链大小，方法是：丢弃旧的区块（参阅资源有限的系统）。但是在丢弃数据之前，仍要下载整个数据集。

Despite these resource requirements, thousands of volunteers run bitcoin nodes. Some are running on systems as simple as a Raspberry Pi (a $35 USD computer the size of a pack of cards). Many volunteers also run bitcoin nodes on rented servers, usually some variant of Linux. A *Virtual Private Server* (VPS) or *Cloud Computing Server* instance can be used to run a bitcoin node. Such servers can be rented for $25 to $50 USD per month from a variety of providers.
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。
一些运行在简单的系统上。 许多志愿者也在租用的服务器上运行比特币节点，这些服务器通常是Linux系统。

“虚拟私有服务器（VPS）”或“云计算服务器”可用于运行比特币节点。这些服务器可以从各种供应商那里租，每月25至50美元。

Why would you want to run a node? Here are some of the most common reasons:

- If you are developing bitcoin software and need to rely on a bitcoin node for programmable (API) access to the network and blockchain.
- If you are building applications that must validate transactions according to bitcoin’s consensus rules. Typically, bitcoin software companies run several nodes.
- If you want to support bitcoin. Running a node makes the network more robust and able to serve more wallets, more users, and more transactions.
- If you do not want to rely on any third party to process or validate your transactions.

为什么要运行一个节点?
以下是一些最常见的原因:

- 你在开发比特币软件，并且需要依赖比特币节点来编程（API）访问网络和区块链。
- 你正在构建应用程序，它必须根据比特币共识规则验证交易。一般来说，比特币软件公司通常运行几个节点。
- 你想支持比特币。运行节点使网络更加健壮，能够服务更多的钱包、更多的用户和更多的交易。
- 你不想依赖任何第三方来处理或验证你的交易。

If you’re reading this book and interested in developing bitcoin software, you should be running your own node.
如果你正在阅读本书，并对开发比特币软件感兴趣，那么你应该运行自己的节点。

3.2.5配置Bitcoin Core节点

Bitcoin Core will look for a configuration file in its data directory on every start. In this section we will examine the various configuration options and set up a configuration file. To locate the configuration file, run bitcoind -printtoconsole in your terminal and look for the first couple of lines.
当第一次运行时，Bitcoin Core会寻找一个配置文件。
在本节中，我们看看各种配置选项，并建立一个配置文件。
为了定位这个配置文件，运行下面命令，在前面几行中找。

```
$ bitcoind -printtoconsole
Bitcoin version v0.15.0
Using the 'standard' SHA256 implementation
Using data directory /home/ubuntu/.bitcoin/
Using config file /home/ubuntu/.bitcoin/bitcoin.conf
...
[a lot more debug output]
...
```

You can hit Ctrl-C to shut down the node once you determine the location of the config file. Usually the configuration file is inside the *.bitcoin* data directory under your user’s home directory. Open the configuration file in your preferred editor.
一旦你找到了配置文件的位置，可以按Ctrl-C来关闭这个节点。
通常，配置文件是在你的home目录的.bitcoin数据目录中。
用编辑器打开这个配置文件。

Bitcoin Core offers more than 100 configuration options that modify the behavior of the network node, the storage of the blockchain, and many other aspects of its operation. To see a listing of these options, run bitcoind --help:
BitCoin Core提供了100个配置选项，它们可以修改网络节点的行为、区块链的存储、以及操作的许多其它方面。为了查看这些选项列表，运行下面命令。

```
$ bitcoind --help
Bitcoin Core Daemon version v0.15.0
```

```
Usage:
  bitcoind [options]                     Start Bitcoin Core Daemon

Options:

  -?
       Print this help message and exit

  -version
       Print version and exit

  -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]
...

  -rpcthreads=<n>
       Set the number of threads to service RPC calls (default: 4)
```

Here are some of the most important options that you can set in the configuration file, or as command-line parameters to bitcoind:
下面是你可以在配置文件中设置的一些最重要的选项，或者作为bitcoind命令行参数：

alertnotify

Run a specified command or script to send emergency alerts to the owner of this node, usually by email.

运行一个指定的命令或脚本，给节点所有者发送一个紧急警报，通常用email。

conf

An alternative location for the configuration file. This only makes sense as a command-line parameter to bitcoind, as it can’t be inside the configuration file it refers to.

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，不能在配置文件中。

datadir

Select the directory and filesystem in which to put all the blockchain data. By default this is the *.bitcoin* subdirectory of your home directory. Make sure this filesystem has several gigabytes of free space.

选择要放置所有区块链数据的目录和文件系统。默认情况下，是.bitcoin子目录。确保这个文件系统具足够的可用空间。

prune

Reduce the disk space requirements to this many megabytes, by deleting old blocks. Use this on a resource-constrained node that can’t fit the full blockchain.

通过删除旧的区块，将磁盘空间要求降低到指定大小。在资源受限的节点上使用这个，它不能放置完整的区块链。

txindex

Maintain an index of all transactions. This means a complete copy of the blockchain that allows you to programmatically retrieve any transaction by ID.

维护所有交易的一个索引。这意味着一个完整的区块链，你可以通过编程获取任何交易。

dbcache
The size of the UTXO cache. The default is 300 MiB. Increase this on high-end hardware and reduce the size on low-end hardware to save memory at the expense of slow disk IO.

UTXO缓存的大小。缺省是300MiB。

maxconnections
Set the maximum number of nodes from which to accept connections. Reducing this from the default will reduce your bandwidth consumption. Use if you have a data cap or pay by the gigabyte.

设置接受连接的最大数量。 从默认值减少该值将减少你的带宽消耗。如果你的网络是按照流量计费，可以使用它。

maxmempool
Limit the transaction memory pool to this many megabytes. Use it to reduce memory use on memory-constrained nodes.

将交易内存池限制在这个大小。

maxreceivebuffer/maxsendbuffer
Limit per-connection memory buffer to this many multiples of 1000 bytes. Use on memory-constrained nodes.

将每个连接的内存缓冲区限制为指定大小。 在内存受限节点上使用。

minrelaytxfee
Set the minimum fee rate for transaction you will relay. Below this value, the transaction is treated nonstandard, rejected from the transaction pool and not relayed.

设置你会传播的交易的最低交易费。低于此值，交易被视为非标准，会被交易池拒绝，并且不传播。

Transaction Database Index and txindex Option
交易数据库索引和txindex选项

By default, Bitcoin Core builds a database containing *only* the transactions related to the user's wallet. If you want to be able to access *any* transaction with commands like getrawtransaction (see Exploring and Decoding Transactions), you need to configure Bitcoin Core to build a complete transaction index, which can be achieved with the txindex option. Set txindex=1 in the Bitcoin Core configuration file. If you don't set this option at first and later set it to full indexing, you need to restart bitcoind with the -reindex option and wait for it to rebuild the index.
默认情况下，Bitcoin Core构建一个数据库，仅包含与用户钱包有关的交易。
如果你想要使用诸如getrawtransaction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，以后再设置为完全索引，则需要使用-reindex选项重新启动bitcoind，并等待它重建索引。

Sample configuration of a full-index node shows how you might combine the preceding options, with a fully indexed node, running as an API backend for a bitcoin application.
下面的例子说明了如何将上述选项与一个全索引节点组合起来，为一个比特币应用程序运行一个API后端。

Example 1. Sample configuration of a full-index node

例3-1全索引节点的例子

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

Sample configuration of a resource-constrained system shows a resource-constrained node running on a smaller server.
下面例子是一个资源限制型节点，运行在一个小服务器上。

Example 2. Sample configuration of a resource-constrained system

```
alertnotify=myemailscript.sh "Alert: %s"
maxconnections=15
prune=5000
dbcache=150
maxmempool=150
maxreceivebuffer=2500
maxsendbuffer=500
```

Once you've edited the configuration file and set the options that best represent your needs, you can test bitcoind with this configuration. Run Bitcoin Core with the option printtoconsole to run in the foreground with output to the console:
编辑配置文件并设置最符合需求的选项后，可以使用此配置来测试 bitcoind。
使用选项printtoconsole运行Bitcoin Core，它在前台运行，有输出到控制台。

```
$ bitcoind -printtoconsole

Bitcoin version v0.15.0
InitParameterInteraction: parameter interaction: -whitelistforcerelay=1 ->
setting -whitelistrelay=1
Assuming ancestors of block
0000000000000000003b9ce759c2a087d52abc4266f8f4ebd6d768b89defa50a have valid
signatures.
Using the 'standard' SHA256 implementation
Default data directory /home/ubuntu/.bitcoin
Using data directory /lotsofspace/.bitcoin
Using config file /home/ubuntu/.bitcoin/bitcoin.conf
Using at most 125 automatic connections (1048576 file descriptors available)
Using 16 MiB out of 32/2 requested for signature cache, able to store 524288
elements
Using 16 MiB out of 32/2 requested for script execution cache, able to store
524288 elements
Using 2 threads for script verification
HTTP: creating work queue of depth 16
No rpcpassword set - using random cookie authentication
Generated RPC authentication cookie /lotsofspace/.bitcoin/.cookie
HTTP: starting 4 worker threads
init message: Verifying wallet(s)...
Using BerkeleyDB version Berkeley DB 4.8.30: (April  9, 2010)
Using wallet wallet.dat
CDBEnv::Open: LogDir=/lotsofspace/.bitcoin/database ErrorFile=/
lotsofspace/.bitcoin/db.log
scheduler thread start
Cache configuration:
* Using 250.0MiB for block index database
* Using 8.0MiB for chain state database
* Using 1742.0MiB for in-memory UTXO set (plus up to 286.1MiB of unused
mempool space)
init message: Loading block index...
Opening LevelDB in /lotsofspace/.bitcoin/blocks/index
Opened LevelDB successfully
```

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

You can hit Ctrl-C to interrupt the process once you are satisfied that it is loading the correct settings and running as you expect.
一旦你确认它加载了正确的配置，并按预期运行，就可以按Ctrl-C中断进程。

To run Bitcoin Core in the background as a process, start it with the daemon option, as bitcoind -daemon.
要在后台作为一个进程来运行Bitcoin Core，用daemon选项启动它：bitcoind -daemon。

To monitor the progress and runtime status of your bitcoin node, use the command bitcoin-cli getblockchaininfo:
要查看你的比特币节点的进度和运行状态，使用命令：bitcoin-cli getblockchaininfo

```
$ bitcoin-cli getblockchaininfo

{
  "chain": "main",
  "blocks": 0,
  "headers": 83999,
  "bestblockhash":
"000000000019d6689c085ae165831e934ff763ae46a2a6c172b3f1b60a8ce26f",
  "difficulty": 1,
  "mediantime": 1231006505,
  "verificationprogress": 3.783041623201835e-09,
  "chainwork":
"0000000000000000000000000000000000000000000000000000000100010001",
  "pruned": false,
  [...]
}
```

This shows a node with a blockchain height of 0 blocks and 83999 headers. The node currently fetches the block headers of the best chain and afterward continues to download the full blocks.
这显示了这个节点，有0个区块，83999个头。
这个节点当前取了最佳链的头，然后会继续下载全部区块。

Once you are happy with the configuration options you have selected, you should add bitcoin to the startup scripts in your operating system, so that it runs continuously and restarts when the operating system restarts. You will find a number of example startup scripts for various operating systems in bitcoin’s source directory under *contrib/init* and a *README.md* file showing which system uses which script.
一旦你对所选择的配置选项感到满意，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。
contrib/init下的bitcoin的源目录中有各种操作系统的启动脚本例子，README.md文件说明了哪个系统使用哪个脚本。

3.3 BitCoin Core API

The Bitcoin Core client implements a JSON-RPC interface that can also be accessed using the command-line helper bitcoin-cli. The command line allows us to experiment interactively with the capabilities that are also available programmatically via the API. To start, invoke the help command to see a list of the available bitcoin RPC commands:
Bitcoin Core客户端实现了一个JSON-RPC接口，还可以使用命令行程序bitcoin-cli访问它。
这个命令行可以用交互式能力进行实验，也能通过API进行编程。
开始前，使用help看看可用的bitcoin 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"
```

Each of these commands may take a number of parameters. To get additional help, a detailed description, and information on the parameters, add the command name after help. For example, to see help on the getblockhash RPC command:
每个命令可能都需要一些参数。
要获得更多帮助、详细说明和参数信息，请在help后加上命令名称。
例如，要查看getblockhash RPC命令的帮助。

```
$ bitcoin-cli help getblockhash
getblockhash height

Returns hash of block in best-block-chain at height provided.

Arguments:
1. height         (numeric, required) The height 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/
```

At the end of the help information you will see two examples of the RPC command, using the bitcoin-cli helper or the HTTP client curl. These examples demonstrate how you might call the command. Copy the first example and see the result:
在帮助信息的最后，有这个RPC命令的两个例子，它们分别使用bitcoin-cli helper和HTTP客户端curl。 这些例子说明了如何调用这个命令。下面是使用第一个例子的结果。

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

The result is a block hash, which is described in more detail in the following chapters. But for now, this command should return the same result on your system, demonstrating that

your Bitcoin Core node is running, is accepting commands, and has information about block 1000 to return to you.
结果是一个区块哈希，下面的章节中有更详细的描述。
但是现在，该命令应该在你的系统上返回相同的结果，表示你的Bitcoin Core节点正在运行，能够接受命令，并且有关于区块1000的信息返回给你。

In the next sections we will demonstrate some very useful RPC commands and their expected output.
在下一节，我们要说明一些非常有用的RPC命令及其预期输出。

3.3.1 获得Bitcoin Core客户端状态的信息

Bitcoin Core provides status reports on diffent modules through the JSON-RPC interface. The most important commands include getblockchaininfo, getmempoolinfo, getnetworkinfo and getwalletinfo.
Bitcoin Core提通过JSON-RPC接口提供了不同模块的状态报告。
最重要的命令包括：

- getblockchaininfo
- getmempoolinfo
- getnetworkinfo
- getwalletinfo

Bitcoin’s getblockchaininfo RPC command was introduced earlier. The getnetworkinfo command displays basic information about the status of the bitcoin network node. Use bitcoin-cli to run it:
getblockchaininfo前面介绍了，它显示这个比特币网络节点的状态的基本信息。

```
$ bitcoin-cli getnetworkinfo
  "version": 150000,
  "subversion": "/Satoshi:0.15.0/",
  "protocolversion": 70015,
  "localservices": "000000000000000d",
  "localrelay": true,
  "timeoffset": 0,
  "networkactive": true,
  "connections": 8,
  "networks": [
    ...
    detailed information about all networks (ipv4, ipv6 or onion)
    ...
  ],
  "relayfee": 0.00001000,
  "incrementalfee": 0.00001000,
  "localaddresses": [
  ],
  "warnings": ""
}
```

The data is returned in JavaScript Object Notation (JSON), a format that can easily be "consumed" by all programming languages but is also quite human-readable. Among this data we see the version numbers for the bitcoin software client (150000) and bitcoin protocol (70015). We see the current number of connections (8) and various information about the bitcoin network and the settings related to this client.
这个数据是用JSON返回的，JSON是一种格式，可以很容易被所有编程语言使用，但也是可读的。
在这个数据中，有比特币软件客户端版本号（150000）和比特币协议版本号（70015）。
当前连接数（8），以及比特币网络和与客户端设置相关的各种信息。

Tip：It will take some time, perhaps more than a day, for the bitcoind client to "catch up" to the current blockchain height as it downloads blocks from other bitcoin clients. You can check its progress using getblockchaininfo to see the number of known blocks.
提示：比特币特客户端可能要花几天时间才能获取到最新的区块链高度，因为它要从其它比特币客户端下载区块。你可以使用getblockchaininfo查看已知的区块的数量。

3.3.2查看和解码交易

Commands: getrawtransaction, decoderawtransaction
命令：

- getrawtransaction
- decodeawtransaction

In [cup_of_coffee], Alice bought a cup of coffee from Bob's Cafe. Her transaction was recorded on the blockchain with transaction ID (txid) 0627052b6f28912f2703066a912ea577f2ce4da4caa5a5fbd8a57286c345c2f2. Let's use the API to retrieve and examine that transaction by passing the transaction ID as a parameter:
Alice在Bob的咖啡店买了一杯咖啡，她的交易记录在区块链中，交易ID（txid）是：
0627052b6f28912f2703066a912ea577f2ce4da4caa5a5fbd8a57286c345c2f2
我们使用这个API来查看这个交易，提供的参数是这个交易ID。

```
$ bitcoin-cli getrawtransaction
0627052b6f28912f2703066a912ea577f2ce4da4caa5a5fbd8a57286c345c2f2

0100000001186f9f998a5aa6f048e51dd8419a14d8a0f1a8a2836dd734d2804fe65fa3577900
0↲
000008b483045022100884d142d86652a3f47ba4746ec719bbfbd040a570b1deccbb6498c75c
4↲
ae24cb02204b9f039ff08df09cbe9f6addac960298cad530a863ea8f53982c09db8f6e381301
4↲
10484ecc0d46f1918b30928fa0e4ed99f16a0fb4fde0735e7ade8416ab9fe423cc5412336376
7↲
89d172787ec3457eee41c04f4938de5cc17b4a10fa336a8d752adfffffffff0260e316000000
0↲
0001976a914ab68025513c3dbd2f7b92a94e0581f5d50f654e788acd0ef8000000000001976a
9↲
147f9b1a7fb68d60c536c2fd8aeaa53a8f3cc025a888ac00000000
```

Tip：A transaction ID is not authoritative until a transaction has been confirmed. Absence of a transaction hash in the blockchain does not mean the transaction was not processed. This is known as "transaction malleability," because transaction hashes can be modified prior to confirmation in a block. After confirmation, the txid is immutable and authoritative.
提示：在交易被确认之前，交易ID不具有权威性。
一个交易哈希不在区块链中并不意味着这个交易未被处理。
这被称为"交易可塑性（malleability）"，因为在区块确认之前可能修改交易哈希。
确认后，txid就是不可改变的，具有权威性。

The command getrawtransaction returns a serialized transaction in hexadecimal notation. To decode that, we use the decoderawtransaction command, passing the hex data as a parameter. You can copy the hex returned by getrawtransaction and paste it as a parameter to decoderawtransaction:
命令getrawtransaction以十六进制数据返回一个序列化的交易。
为了解码，我们使用decodeawtransaction命令，提供的参数是十六进制数据。
你可以复制getrawtransaction返回的十六进制数据 并将其作为参数粘贴到decodeawtransaction中。

```
$ bitcoin-cli decoderawtransaction
0100000001186f9f998a5aa6f048e51dd8419a14d8↲
a0f1a8a2836dd734d2804fe65fa35779000000008b483045022100884d142d86652a3f47ba47
4↲
6ec719bbfbd040a570b1deccbb6498c75c4ae24cb02204b9f039ff08df09cbe9f6addac96029
8↲
cad530a863ea8f53982c09db8f6e381301410484ecc0d46f1918b30928fa0e4ed99f16a0fb4f
d↲
e0735e7ade8416ab9fe423cc5412336376789d172787ec3457eee41c04f4938de5cc17b4a10f
a↲
336a8d752adfffffffff0260e31600000000001976a914ab68025513c3dbd2f7b92a94e0581f
5↲
d50f654e788acd0ef8000000000001976a9147f9b1a7fb68d60c536c2fd8aeaa53a8f3cc025a
8↲
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"
        ]
      }
    }
  ]
}
```

The transaction decode shows all the components of this transaction, including the transaction inputs and outputs. In this case we see that the transaction that credited our new address with 15 millibits used one input and generated two outputs. The input to this transaction was the output from a previously confirmed transaction (shown as the vin txid starting with 7957a35fe). The two outputs correspond to the 15 millibit credit and an output with change back to the sender.
这个交易解码显示了这个交易的所有成分，包括交易的输入和输出。
在这个例子中，可以看到这个交易有一个输入、两个输出。
这个交易的输入是前一个已确认交易的输出，例子中是`vin txid` 7957a35fe…..

We can further explore the blockchain by examining the previous transaction referenced by its txid in this transaction using the same commands (e.g., getrawtransaction). Jumping from transaction to transaction we can follow a chain of transactions back as the coins are transmitted from owner address to owner address.
我们可以进一步探索这个区块链，使用相同的命令（例如 `gettransaction` ）进一步查看前一个交易（使用它的`txid`）。
这样，我们可以可以追溯一连串交易，比特币在所有者地址之间转移。

3.3.3 查看区块

Commands: getblock, getblockhash
命令：

- `getblock`
- `getblockhash`

Exploring blocks is similar to exploring transactions. However, blocks can be referenced either by the block *height* or by the block *hash*. First, let’s find a block by its height. In [cup_of_coffee], we saw that Alice’s transaction was included in block 277316.
查看区块类似于查看交易。但是，可以用区块高度或区块哈希引用区块。
我们先用区块区块查看区块。
在买咖啡故事中，`Alice`的交易已包含在区块`277316`中。

We use the getblockhash command, which takes the block height as the parameter and returns the block hash for that block:
我们使用`getblockhash`命令，提供的参数是区块高度，返回区块的哈希。

```
$ bitcoin-cli getblockhash 277316
0000000000000001b6b9a13b095e96db41c4a928b97ef2d944a9b31b2cc7bdc4
```

Now that we know which block Alice’s transaction was included in, we can query that block. We use the getblock command with the block hash as the parameter:
因为我们知道`Alice`的交易包含在哪个区块中，就可以查看那个区块。
我们使用`getblock`命令，参数是这个区块的哈希。

```
$ bitcoin-cli getblock
0000000000000001b6b9a13b095e96db41c4a928b97ef2d944a9b31b2cc7bdc4
{
  "hash":
"0000000000000001b6b9a13b095e96db41c4a928b97ef2d944a9b31b2cc7bdc4",
  "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":
"000000000000000000000000000000000000000000000934695e92aaf53afa1a",
  "previousblockhash":
"0000000000000002a7bbd25a417c0374cc55261021e8a9ca74442b01284f0569",
  "nextblockhash":
"000000000000000010236c269dd6ed714dd5db39d36b33959079d78dfd431ba7"
}
```

The block contains 419 transactions and the 64th transaction listed (0627052b...) is Alice's coffee payment. The height entry tells us this is the 277316th block in the blockchain.
这个区块包含419个交易，第64个交易（0627052b…）是Alice的交易。
height表示它是区块链中第277316个区块。

3.3.4使用Bitcoin Core的编程接口

The bitcoin-cli helper is very useful for exploring the Bitcoin Core API and testing functions. But the whole point of an application programming interface is to access functions programmatically. In this section we will demonstrate accessing Bitcoin Core from another program.
bitcoin-cli helper对于查看Bitcoin Core API和测试功能非常有用。
但是，API的重点是以编程方式访问它的功能。
在本节中，我们将演示用一个程序来访问Bitcoin Core。

Bitcoin Core's API is a JSON-RPC interface. JSON stands for JavaScript Object Notation and it is a very convenient way to present data that both humans and programs can easily read. RPC stands for Remote Procedure Call, which means that we are calling procedures (functions) that are remote (on the Bitcoin Core node) via a network protocol. In this case, the network protocol is HTTP, or HTTPS (for encrypted connections).
Bitcoin Core的API是一个JSON-RPC接口。
JSON是一种非常方便的方式来呈现人和程序都可以轻松读取的数据。
RPC意味着，我们通过网络协议调用远端（在Bitcoin Core节点上）的过程（函数）。
在本例中，网络协议是HTTP或HTTPS（用于加密连接）。

When we used the bitcoin-cli command to get help on a command, it showed us an example of using curl, the versatile command-line HTTP client to construct one of these JSON-RPC calls:
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curl，它是一个通用的命令行HTTP客户端，可用来构造JSON-RPC调用。

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

This command shows that curl submits an HTTP request to the local host (127.0.0.1), connecting to the default bitcoin port (8332), and submitting a jsonrpc request for the getblockchaininfo method using text/plain encoding.
这个命令的意思是：curl向本地主机（127.0.0.1）提交了一个HTTP请求，使用的端口是8332（默认的比特币端口），并使用text/plain编码提交了一个jsonrpc请求，请求的是getblockchaininfo 方法。

You might notice that curl will ask for credentials to be sent along with the request. Bitcoin Core will create a random password on each start and place it in the data directory under the name .cookie. The bitcoin-cli helper can read this password file given the data directory. Similarly, you can copy the password and pass it to curl (or any higher level Bitcoin Core RPC wrappers). Alternatively, you can create a static password with the helper script provided in *./share/rpcuser/rpcuser.py* in Bitcoin Core's source directory.
你可能注意到，curl要求发送请求时要有凭证。
Bitcoin Core在每个启动时会创建一个随机密码，放在数据目录的.cookie文件中。
bitcoin-cli helper可以从数据目录读取这个密码文件。
类似的，你可以拷贝这个密码，传给cure（或任何高级的Bitcoin Core RPC warppers）。
你还可以用helper脚本创建一个静态密码，这个脚本在Bitcoin Core源目录的*./share/rpcuser/rpcuser.py*

If you're implementing a JSON-RPC call in your own program, you can use a generic HTTP library to construct the call, similar to what is shown in the preceding curl example.
如果你要在自己的程序中实现JSON-RPC调用，可以使用一个通用的HTTP库构建这个调用，类似于前面的curl示例中所示的。

However, there are libraries in most every programming language that "wrap" the Bitcoin Core API in a way that makes this a lot simpler. We will use the python-bitcoinlib library to simplify API access. Remember, this requires you to have a running Bitcoin Core instance, which will be used to make JSON-RPC calls.
但是，大多数编程语言中都有库封装了Bitcoin Core API，使调用更加简单。
我们将使用python-bitcoinlib库来简化API访问。
记住，这需要你运行一个Bitcoin Core实例，用于进行JSON-RPC调用。

The Python script in Running getblockchaininfo via Bitcoin Core's JSON-RPC API makes a simple getblockchaininfo call and prints the block parameter from the data returned by Bitcoin Core.
下面例子的Python脚本做了一个简单的getblockchaininfo调用，输出的区块参数来自Bitcoin Core返回的数据。

Example 3. Running getblockchaininfo via Bitcoin Core's JSON-RPC API
例3：通过Bitcoin Core的JSON-RPC API运行getblockchaininfo

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

Running it gives us the following result:
运行结果如下。

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

It tells us that our local Bitcoin Core node has 394075 blocks in its blockchain. Not a spectacular result, but it demonstrates the basic use of the library as a simplified interface to Bitcoin Core's JSON-RPC API.
它告诉我们，Bitcoin Core节点在其区块链中有394075个区块。
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。

Next, let's use the getrawtransaction and decodetransaction calls to retrieve the details of Alice's coffee payment. In Retrieving a transaction and iterating its outputs, we retrieve Alice's transaction and list the transaction's outputs. For each output, we show the recipient address and value. As a reminder, Alice's transaction had one output paying Bob's Cafe and one output for change back to Alice.
接下来，我们使用getrawtransaction和decodetransaction调用来查看Alice的交易信息。
在下面的例子中，我们查看Alice的交易，并列出交易的输出。
对于每个输出，显示了收款人的地址和金额。

Example 4. Retrieving a transaction and iterating its outputs
例4：检索一个交易，并列出它的输出

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

Running this code, we get:
运行结果如下。

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

Both of the preceding examples are rather simple. You don't really need a program to run them; you could just as easily use the bitcoin-cli helper. The next example, however, requires several hundred RPC calls and more clearly demonstrates the use of a programmatic interface.
上述两个例子都比较简单，你可以很容易地使用bitcoin-cli helper来实现。
但是，下一个例子需要数百个RPC调用，并更清楚地说明了API的使用。

In Retrieving a block and adding all the transaction outputs, we first retrieve block 277316, then retrieve each of the 419 transactions within by reference to each transaction ID. Next, we iterate through each of the transaction's outputs and add up the value.
在下例中，我们首先检索区块277316，然后通过引用每个交易ID来检索区块中的419个交易。
然后，我们将每个交易的输出都加起来。

Example 5. Retrieving a block and adding all the transaction outputs
例5：检索一个区块，把所有交易的输出相加

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

Running this code, we get:
运行结果如下。

```
$ python rpc_block.py
('Total value in block: ', Decimal('10322.07722534'))
```

Our example code calculates that the total value transacted in this block is 10,322.07722534 BTC (including 25 BTC reward and 0.0909 BTC in fees). Compare that to the amount reported by a block explorer site by searching for the block hash or height. Some block explorers report the total value excluding the reward and excluding the fees. See if you can spot the difference.
我们的例子代码计算出，这个区块中交易的总价值为10,322.07722534 BTC（包括25 BTC奖励和0.0909 BTC交易费）。
可以与区块浏览器站点报告的金额进行比较，方法是搜索这个区块哈希和高度。
有些区块浏览器报告的总额不包括奖励和交易费。

看看你是否能发现差异。

3.4 其它客户端、库、工具包

There are many alternative clients, libraries, toolkits, and even full-node implementations in the bitcoin ecosystem. These are implemented in a variety of programming languages, offering programmers native interfaces in their preferred language.
在比特币生态系统中，有许多其它的客户端、库、工具包，甚至全节点实现。
这些是用各种编程语言实现的，提供了编程语言原生接口。

The following sections list some of the best libraries, clients, and toolkits, organized by programming languages.
以下列出了一些最好的库、客户端和工具包。

Many more libraries exist in a variety of other programming languages and more are created all the time.
还有很多库在用各种其它编程语言中。

编程语言	工具	说明
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