

==以太坊基础知识

在本章中，我们将开始介绍以太坊，学习如何使用钱包，如何创建交易，以及如何运行基本的智能合约。

===以太币的货币单位

以太坊的货币单位称为_ether_，也标识为“ETH”或符号Ξ（从希腊字母开始“Xi”看起来像一个程式化的大写字母E），或者更不常见的是◆：例如1个ether，1个ETH，Ξ1，或◆1。

对#926使用Unicode字符+U+039E+，对#9830使用+U+2666+。

以太币被细分为更小的单位，可能的最小单位为_wei_，翻译为维。一个以太币是1 quintillion wei（1 * 10 ^ 18 ^或1,000,000,000,000,000,000）。您可能会听到人们也提到货币“以太坊”，但这是一个常见的初学者的错误。以太坊是系统，以太币是货币。

ether的值始终在以太坊内部始终表示为以wei表示的无符号整数值。当您交易1个以太币时，交易会将值编码为1000000000000000000 wei。

以太币的各种计量单位都有使用国际单位制（SI）的科学名称和通俗的名称，以向计算机学和密码学的诸位伟大先驱致敬。

[\[ether_denominations\]](#)显示各种单位，其俗语（通用）名称及其SI名称。为了与价值的内部表示保持一致，该表在wei（第一行）中显示了所有面额，在第7行中以太显示为10 ^ 18 ^ wei。

以太币计量单位名称

==	价值（wei）	指数	通用名称	SI名称	1	1	wei	Wei	1,000	10 ^ 3 ^	Babbage Kilowei或Femtoether	1,000,000	10 ^ 6 ^	Lovelace Megawei或picoether	1,000,000,000	10 ^ 9 ^	Shannon Gigawei或nanoether	1,000,000,000,000	10 ^ 12 ^	Szabo Microether 或 micro	1,000,000,000,000,000	10 ^ 15 ^	Finney Milliether 或 milli	1,000,000,000,000,000,000	10 ^ 18 ^	_ _Ether Ether	1,000,000,000,000,000,000,000	10 ^ 21	Grand Kiloether	1,000,000,000,000,000,000,000,000	10 ^ 24	Megaether	==
----	---------	----	------	------	---	---	-----	-----	-------	----------	-----------------------------	-----------	----------	------------------------------	---------------	----------	----------------------------	-------------------	-----------	----------------------------	-----------------------	-----------	-----------------------------	---------------------------	-----------	--------------------	-------------------------------	---------	-------------------	-----------------------------------	---------	-----------	----

===选择以太坊钱包

“钱包”这个名词可以代表着很多事务，虽然它们都是相关的，并且每天在做几乎相同的事情。我们将使用“钱包”一词来表示可帮助你管理以太坊账户的软件应用程序。简而言之，以太坊钱包是你通往以太坊系统的门户。它拥有你的密钥，可以代表你创建和广播交易。选择以太坊钱包可能很困难，因为存在许多具有不同功能和设计的不同选择。有些更适合初学者，有些更适合专家。以太坊平台本身仍在改进中，“最佳”钱包通常是适应平台升级而随之升级的钱包。

但是不用担心！如果你选择了一个钱包并且不喜欢它的工作方式，或者你一开始喜欢它，但后来又想尝试其他钱包，则可以很容易地更换它们。你所要做的就是进行一笔交易，将你的资金从旧钱包发送到新钱包，或者从旧钱包导出你的私钥并将其导入新的钱包。

作为整本书的示例，我们将选择三种不同类型的钱包：移动钱包，桌面钱包和基于网络的钱包。我们选择了这三款钱包，因为它们代表了广泛的复杂性和功能。然而，选择这些钱包并不是对其质量或安全性的认可。这些钱包只是便于示范和测试。

请记住，要使钱包应用程序正常工作，它必须有权访问你的私钥，因此至关重要的是，你只能从可信任的来源下载和使用钱包应用程序。幸运的是，总的来说，钱包应用程序越受欢迎，它就越值得信赖。尽管如此，还是要避免

将“所有鸡蛋都放在一个篮子里”，并且将以太坊账户分配到几个钱包中，这是一个好习惯。

以下是一些不错的入门钱包：

MetaMask ::MetaMask是在你的浏览器（Chrome，Firefox，Opera或Brave Browser）中运行的浏览器扩展钱包。它易于使用且便于测试，因为它能够连接到各种以太坊节点并测试区块链。MetaMask是一个基于网页的钱包。

Jaxx

Jaxx是一种多平台，多币种的钱包，可在多种操作系统上运行，包括Android，iOS，Windows，macOS，和Linux。对于新用户来说，这通常是一个不错的选择，因为它的操作简单并便于使用。Jaxx是移动钱包还是台式机钱包，具体取决于你如何安装它。

MyEtherWallet (MEW)

MyEtherWallet是基于Web的钱包，可在任何浏览器中运行。它具有多种复杂的功能，我们将在许多示例中进行探讨。MyEtherWallet是一个基于Web的钱包。

Emerald Wallet

Emerald Wallet是设计为与以太坊经典区块链一起使用的钱包，也可以兼容基于以太坊的区块链。它是一个开源的桌面应用程序，可在Windows，macOS和Linux下运行。Emerald Wallet可以在“轻量”模式下运行完整的节点或连接到公共远程节点。它还具有一个辅助工具，可从命令行执行所有操作。

我们将从在桌面上安装MetaMask开始-但首先，我们将简要地讨论一下如何控制和管理密钥。

===控制与责任

"Ethereum (generally)","control and responsibility", id="ix_02intro-asciidoc1", range="startofrange")像以太坊这样的开放式区块链很重要，因为它们以_decentralized_系统的身份运行。这意味着很多事情，但是一个关键方面是，以太坊的每个用户都可以并且应该控制自己的私钥，这些私钥控制着对资金和智能合约的访问。有时我们将获得资金和智能合约的组合称为“帐户”或“钱包”。这些术语的功能可能会变得非常复杂，因此我们将在后面详细介绍。但是，作为一项基本原则，就像一个私钥等于一个“帐户”一样容易。一些用户选择通过使用第三方保管人（例如在线交易所）来放弃对其私钥的控制。在本书中，我们将教你如何控制和管理自己的私钥。

控制带来了巨大的责任。如果丢失私钥，则将无法访问资金和合约。没有人可以帮助你重新获得访问权限-你的资金将永远被锁定。这里有一些方法可以帮助你管理此职责：

*不要临时才考虑安全问题。使用经过使用和考验的标准方法。

*帐户越重要（例如，受控资金的价值越高，或者可调用的智能合约越重要），则应采取更高的安全措施。

*最高的安全性是需要有无缝可钻的设备来支持的，但并非每个帐户都需要此级别。

*切勿以非加密形式（尤其是数字形式）存储私钥。幸运的是，当今大多数用户界面甚至都不会让你看到原始私钥。

- 私钥可以加密形式存储为数字“keystore”文件。被加密后，他们需要密码才能解锁。当提示你选择一个密码时，请使其强度高（即长而随机），备份并不要共享。如果你没有密码管理器，请将其写下并存储在安全隐秘的地方。要访问你的帐户，你需要密钥库文件和密码。

*请勿在数字文档，数字照片，屏幕截图，在线驱动器，加密的PDF等文件中存储任何密码。同样，请勿即刻提高安全性。使用密码管理器或笔和纸。

*当提示你将钱包的助记词序列进行备份时，请使用笔和纸进行物理备份。不要将该任务“留作以后”；你会忘记的。这些备份可用于重建私钥，以防丢失所有保存在系统中的数据，或者忘记或丢失密码。但是，攻击者也可以使用它

们来获取你的私钥，因此切勿以数字方式存储它们，并将物理副本安全地存储在上锁的抽屉中或保险箱中。

*在转账任何大额资金（尤其是到新地址）之前，请先进行一次小额测试交易（例如，价值小于\$ 1的交易），然后等待收据确认。

*创建新帐户时，首先仅将一小笔测试交易发送到新地址。收到测试交易后，请尝试再次从该帐户发送回。帐户创建有很多原因可能会出错，如果出错，最好找出一个很小的损失。如果测试正常，一切都很好。

*公共区块浏览器是一种独立查看交易是否已被网络接受的简便方法。但是，这种便利会对你的隐私产生负面影响，因为这些浏览器可能会根据你公开账户地址来追踪你的交易行为。

*请勿汇款至本书中显示的任何地址。书中列出了私钥，有人会立即偷走这笔钱。

现在，我们已经介绍了密钥管理和安全性的一些基本操作的最佳用法，让我们开始使用MetaMask！

MetaMask入门

打开Google Chrome浏览器，然后导航至<https://chrome.google.com/webstore/category/extensions> []。

搜索“MetaMask”，然后单击狐狸的标识。你应该看到类似[\[metamask_download\]](#)的结果。

MetaMask Chrome扩展件的详细页面

image

[images / metamask_download.png](#) [“MetaMask详细页面”]

确认你正在下载真正的MetaMask扩展件很重要，因为有时恶意扩展件可以通过Google的过滤器。真正的软件的标识是：

*在地址栏中显示ID + `nkbihfboegaoehlfinkodbefgpgknn`

*由 <https://metamask.io> 提供 *超过1,500条评论 *拥有超过1,000,000用户

当你确认使用了正确的扩展程序后，单击“添加到Chrome”进行安装。

====创建钱包

一旦安装了MetaMask，你会在浏览器的工具栏中看到一个新图标（狐狸头）。点击它开始。系统将要求您接受条款和条件，然后通过输入密码来创建新的以太坊钱包（请参见[\[metamask_password\]](#)）。

MetaMask Chrome扩展件的密码页面

image

[images / metamask_password.png](#) [“MetaMask密码页面”]

该密码控制对MetaMask的访问，因此任何有权访问你的浏览器的人都无法使用它。

一旦你设置了密码，MetaMask将为你生成一个钱包并向您显示_mnemonic backup_由12个英语单词组成（请参见[\[metamask_mnemonic\]](#)）。如果MetaMask或你的计算机发生故障，则可以在任何兼容的钱包中使用这些词来恢复对账户资金的访问。你无需密码即可进行此恢复；12个单词就足够了。

将你的助记符（12个单词）备份在纸上两次。将两个纸张备份存储在两个单独的安全位置，例如耐火保险箱，上锁的抽屉或保险箱。将纸质备份视为与以太坊钱包中存储的现金等价的现金。能够使用这些字词的任何人都可以访问并窃取你账户中的资金。

由MetaMask创建的钱包助记符备份 image :: images / metamask_mnemonic.png [“ MetaMask助记符页面”]

确认安全存储了助记符后，您将能够查看以太坊帐户的详细信息，如[\[metamask_account\]](#).

您在MetaMask中的以太坊账户

image::images/metamask_account.png [“ MetaMask帐户页面”]

你的帐户页面显示你的帐户名称（默认情况下为“帐户1”），以太坊地址（示例中为 +0x9E713 ... +）和彩色图标，以帮助你直观地将该帐户与其他帐户区分开。在帐户页面的顶部，你可以看到当前正在使用的以太坊网络（示例中为“主网络”）。

恭喜你！你已经设置了第一个以太坊钱包。

====切换网络

正如你在MetaMask帐户页面上看到的那样，用户可以在多个以太坊网络之间进行选择。默认情况下，MetaMask将尝试连接到主网络。其他选择是公共测试网，你选择的任何以太坊节点或在你自己的计算机（本地主机）上运行的私有区块链节点：

以太坊主网络::公共以太坊区块链主网络。真实的ETH，真实的价值和真实的结果。

Ropsten测试网络：：以太坊公共区块链测试网络。该网络上的ETH没有价值。

Kovan测试网络::以Aura共识协议为基础的以太坊公共测试区块链和网络，并具有Proof of Authority授权证明（联合签名）。该网络上的ETH没有价值。仅"Parity"客户端支持Kovan测试网络。其他以太坊客户使用后来提出的Clique共识协议，用于基于权限的验证。

Rinkeby测试网络

使用带有授权证明（联合签名）的Clique共识协议，以太坊公共测试区块链和网络。该网络上的ETH没有价值。

Localhost 8545

连接到与浏览器在同一台计算机上运行的节点。该节点可以是任何公共区块链（主或测试网）的一部分，也可以是私有测试网的一部分。

自定义RPC：：允许您使用与Geth兼容的远程过程调用（RPC）接口将MetaMask连接到任何节点。该节点可以是任何公共或私有区块链的一部分。

NOTE

你的MetaMask钱包在其连接到的所有网络上使用相同的私钥和以太坊地址。但是，每个以太坊网络上的以太坊地址余额将有所不同。例如，你的密钥可以控制Ropsten上的以太币和合约，但不能控制主网络的以太币和合约。

====获得一些测试用以太ether

"ether (generally)", "testnet")你的首先需要为你的钱包注入资金才能使用。你不用在主网络上执行此操作，因为真正的以太坊需要花费金钱，而处理它需要更多的经验。下面将介绍你如何使用一些测试网的 ether 来加载你的钱包。

将MetaMask切换到_Ropsten测试网络_。单击“存款”，然后单击“ Ropsten测试水龙头”。MetaMask将打开一个新的网页，如 [\[metamask_ropsten_faucet\]](#)。

MetaMask Ropsten测试水龙头

image

images / metamask_ropsten_faucet.png [“MetaMask Ropsten测试水龙头”]

你可能会注意到网页已经包含你的MetaMask钱包的以太坊地址。MetaMask自动将启用了以太坊的网页与你的MetaMask钱包集成在一起，并且可以在网页上“查看”以太坊地址，例如，允许你向显示以太坊地址的在线商店付款。如果网页请求，MetaMask也可以使用你自己的钱包地址作为收件人地址来填充网页。在此页面中，水龙头应用程序正在向MetaMask询问要发送测试以太币的钱包地址。

单击绿色的“从水龙头请求1以太”按钮。您会看到交易ID出现在页面的下部。水龙头应用程序创建了一笔交易

2014-向您付款。交易ID如下所示：

```
0x7c7ad5aaea6474adccf6f5c5d6abed11b70a350fbc6f9590109e099568090c57
```

几秒钟后，新交易将由Ropsten网络的矿工进行确认，你的MetaMask钱包将显示1 ETH余额。单击交易ID，浏览器将带跳转到_block explorer_，这是一个网站，你可以通过它浏览和查看区块，地址和交易。MetaMask使用<https://etherscan.io/>[Etherscan区块浏览器]，它是较流行的以太坊区块浏览器之一。包含来自Ropsten测试水龙头的付款的交易显示在[\[ropsten_block_explorer\]](#)。

Etherscan Ropsten测试网区块浏览器

image

images / ropsten_block_explorer.png [“Etherscan Ropsten Block Explorer”]

该交易已记录在Ropsten区块链上，任何人都可以随时查看，只需搜索交易ID或<http://bit.ly/2Q860Wk> [访问链接]即可。

你可以尝试访问该链接，或在_ropsten.etherscan.io_网站中输入交易哈希并亲自查看。

====从MetaMask发送以太币

("test ether","sending", id="ix_02intro-asciidoc7", range="startofrange" 一旦你从Ropsten测试水龙头收到了第一个测试用ether，就可以尝试发送ether的操作，比如将一些ether送回给水龙头。我们将试着发送一些ether回到faucet。正如你在Ropsten Test Faucet页面上看到的那样，你可以选择“donate”1个ETH。这个选项是可用的，所以一旦你完成了测试，你可以返回剩余的测试ether，以便其他人可以使用它。尽管测试ether没有价值，但有些人囤积测试ether，使其他人难以使用测试网络。囤积测试ether有害无益！

还好，我们不是以太坊测试币的囤积者。单击橙色的“1 ether”按钮，告诉MetaMask创建一个支付水龙头1 ether的交易。MetaMask将准备交易并弹出带有确认的窗口，如[\[send_to_faucet\]](#)。

向水龙头发送1个以太币

image::images/send_to_faucet.png [“向水龙头发送1个以太币”]

糟糕！你可能会注意到你无法完成交易。MetaMask说你的余额不足。乍一看，这似乎令人困惑：你有1 ETH，你想发送1 ETH，那么为什么MetaMask表示你的账户资金不足？

答案是因为_gas_的成本没有计算在内。每笔以太坊交易都需要支付费用，矿工会收取一定费用以验证交易。以太坊的费用以一种称为“gas”的虚拟货币收费。在交易中，你需要用以太币ether支付燃料gas费用。

NOTE

测试网络也需要收费。没有费用，测试网络的行为将不同于主网络，从而使其无法成为合格的测试平台。收费还可以保护测试网络免受DoS攻击和合约中恶意架构（例如，无限循环）的攻击，就像它们保护主网络一样。

当你发送交易时，MetaMask计算出最近成功交易的平均燃料gas价格为3 gwei，代表gigawei。正如我们在[\[ether_units\]](#)中所讨论的那样，Wei是以太坊的最小单位量：[细分]。燃料gas限额的确定是以发送基本交易为代价的，即21,000个燃料gas单位。因此，你这个交易将花费的最大ETH数量为3 * 21,000 gwei = 63,000 gwei = 0.000063 ETH。（请注意，平均燃料gas价格可能会波动，这主要由矿工决定。我们将在下一章中看到如何增加/减少燃料gas限额，以确保在需要时优先交易。）

这就是说：进行1 ETH交易需要花费1.000063 ETH。当显示总数时，MetaMask令人困惑地将_down_四舍五入为1 ETH，但是实际所需的金额为1.000063 ETH，而帐户里面只有1 ETH，所以交易无法完成。单击“Reject”按钮以取消该交易。

让我们再获取一些测试ether！再次单击绿色的“从水龙头请求1以太坊”按钮，然后等待几秒钟。不用担心，水龙头中应该有大量的ether，如果需要的话，还会给你更多。

当你的账户余额达到2 ETH后，你可以重试。这次，当你单击橙色的“1 ether”捐赠按钮时，账户中有足够的余额来完成交易。当MetaMask弹出付款窗口时，单击提交。完成所有这些操作后，你应该会看到0.999937 ETH的余额，因为你向水龙头中发送了1 ETH，所消耗的燃料gas中含0.000063 ETH。

====浏览一个账户公开地址的交易记录

现在，你已经了解如何使用MetaMask发送和接收测试以太坊。你的钱包已收到至少两笔付款，并至少发送了一笔。你可以使用_ropsten.etherscan.io_的区块浏览器查看所有这些交易。你可以复制钱包地址并将其粘贴到区块浏览器的搜索框中，也可以让MetaMask为你打开页面。在MetaMask中你的帐户图标旁边，您将看到一个显示三个点的按钮。点击它以显示与帐户相关的选项菜单（请参见[\[metamask_account_context_menu\]](#)）。

MetaMask帐户交易记录菜单

image

images / metamask_account_context_menu.png [“MetaMask帐户记录菜单”]

选择“在Etherscan上查看帐户”以在区块浏览器中打开一个网页，显示帐户的交易历史记录，如[\[block_explorer_account_history\]](#)。



Figure 1. Etherscan上的账户公开地址交易历史记录

在这里，你可以查看以太坊账户的整个交易历史。它显示了Ropsten区块链上记录的所有交易，你的地址是交易发起者或接收者。单击其中一些交易以查看更多详细信息。

你可以浏览任何地址的交易记录。查看Ropsten测试水龙头地址的交易历史记录（提示：这是你的地址中最早的付款项中列出的“发件人”地址）。你可以看到从水龙头发送到你和其它地址的所有测试以太坊。你看到的每笔交易都可以引导你查到更多的地址和更多的交易。不久之后，你将迷失在这些相连数据的迷宫中。公共区块链包含大量信息，所有这些信息都可以通过编程方式进行查找，我们将在以后的示例中为你展示这些方法。

====介绍世界计算机

现在，你已经创建了一个钱包，并发送和接收了以太坊。到目前为止，我们已经将以太坊视为一种加密货币。但是以太坊的功能还有很多。实际上，加密货币功能仅仅是以太坊作为分布式世界计算机的众多功能之一。以太坊用于支付运行_smart contract_的费用，这些是在称为_Ethereum Virtual Machine_（EVM）的仿真计算机上运行的计算机程序。

EVM是全局单例，这意味着它可以像运行在各处的全局单实例计算机一样运行。以太坊网络上的每个节点都运行EVM的本地副本以验证智能合约的执行，而以太坊区块链则记录了该世界计算机在处理交易和智能合约时的变化状态。我们将在[\[evm_chapter\]](#)。

===外部拥有的帐户（EOA）和智能合约

在MetaMask钱包中创建的帐户类型称为外部拥有的帐户_externally owned account_（EOA）。外部拥有的帐户是具有私钥的帐户；拥有私钥意味着控制对资金或合约的访问。现在，你可能正在猜测还有另一种帐户。其他类型的帐户是合约帐户_contract account_。合约帐户具有智能合约代码，而简单的EOA则没有。此外，合约帐户没有私钥。相反，它由其智能合约代码的逻辑所拥有（并控制）：在合约帐户创建时记录在以太坊区块链上并由EVM执行的软件程序。

部署在以太坊的智能合约有地址，就像EOA一样。合约也可以像EOA一样发送和接收以太币。但是，当交易目的地是合约地址时，它将使用交易以及交易数据作为输入，使该合约可以在EVM中执行_run_。除以太币外，交易还可以包含数据_data_，该_data_指示合约中要运行的特定功能函数以及要传递给该函数的参数。通过这种方式，交易可以在合约中调用功能函数并获得结果。

请注意，由于合约帐户没有私钥，因此它无法直接启动一个交易。只有EOA可以启动交易，但是合约可以通过调用其他合约来构建交易，从而构建复杂的执行路径。EOA的一种典型用法是EOA将请求交易发送到多签名智能合约钱包，以将ETH发送到另一个地址。典型的DApp编程模式是让合约A调用合约B，以便在Contract A的用户之间维持共享状态。

在接下来的几节中，我们将编写我们的第一份智能合约。然后，你将学习如何与MetaMask钱包一起创建，充值和使用该合约，并在Ropsten测试网络上用测试以太币进行测试。

===一个简单的合约：测试用以太币水龙头

以太坊支持许多不同的高级语言，所有这些语言均可用于编写合约并产生EVM二进制代码。你可以在[\[high_level_languages\]](#)中浏览众多有名和有趣的文章。目前智能合约编程中使用最多的高级编程语言是：Solidity。Solidity是由本书的合著者Gavin Wood博士创建，并已成为以太坊（及其它基于以太坊的区块链平台）中使用最广泛的语言。我们将使用Solidity编写我们的第一份智能合约。

对于我们的第一个示例（[\[solidity_faucet_example\]](#)），我们将写一个控制_faucet_的合约。你已经使用过水龙头在Ropsten测试网络上获取测试以太币ether。水龙头是一个相对简单的合约：它向任何请求提现的账户地址发出以太币，并且可以不时重新充值。你可以将水龙头改造为由人工或网页服务器控制的钱包。

Example 1. Faucet.sol：实施水龙头功能的Solidity合约

```
// SPDX-License-Identifier: CC-BY-SA-4.0

// Version of Solidity compiler this program was written for
pragma solidity 0.6.4;

// Our first contract is a faucet!
contract Faucet {
    // Accept any incoming amount
    receive() external payable {}

    // Give out ether to anyone who asks
    function withdraw(uint withdraw_amount) public {
        // Limit withdrawal amount
        require(withdraw_amount <= 1000000000000000000);

        // Send the amount to the address that requested it
        msg.sender.transfer(withdraw_amount);
    }
}
```

NOTE

你可以在 <https://github.com/ethereumbook/ethereumbook/> [该书的GitHub存储库]的_code_子目录中找到该书的所有代码示例。具体来说，我们的_Faucet.sol_合约位于：

```
code/Solidity/Faucet.sol
```

目前这是一个非常简单的合约，实现了基本功能。它也是一个有缺陷的合约，展示了许多不良做法和安全漏洞。我们将在后面的部分中研究其所有缺陷，以达到学习的目的。但是现在，让我们逐行看一下该合约的用法及其工作方式。你会很快注意到，Solidity的许多元素与现有的编程语言类似，例如JavaScript，Java或C++：[++]。

第一行是一条注释：

```
// Our first contract is a faucet!
```

注释是供人类阅读，并且不包含在可执行的EVM字节码中。我们通常将它们放在试图解释的代码之前，或者有时放在同一行。注释以两个正斜杠开头：+//+。从第一个斜杠到该行末尾的所有内容都被视为空白行并被忽略。

下一行是我们实际合约的开始位置：

```
contract Faucet {
```

该行声明了一个合约 contract 对象，类似于其他面向对象的语言中的 class 声明。合约定义包括花括号之间的所有行（pass: [<code>{}</code>]），它们定义了一个_scope_，就像在许多其他编程语言中如何使用花括号一样。

接下来，我们构建 Faucet 合约的第一个函数：

```
function withdraw(uint withdraw_amount) public {
```

该函数名为提现 withdraw，它采用一个无符号整数（uint）参数，命名为 withdraw_amount。它被声明为公共功能函数，这意味着它可以被其他合约调用。函数定义在花括号之间。withdraw 函数的第一部分设置了提现的金额限制：

```
require(withdraw_amount <= 1000000000000000000);
```

它使用内置的Solidity函数 require 来测试前提条件，即 withdraw_amount 小于或等于100,000,000,000,000,000 wei，它是以太坊的基本单位（请参阅 [\[ether_denominations\]](#)）并等于0.1以太币ether。如果调用 withdraw 函数且其 withdraw_amount 大于该数量，则此处的 require 函数将导致合约执行停止并以_exception_失败。请注意，在Solidity中，语句必须以分号终止。

智能合约的这一部分是我们水龙头的主要逻辑。它通过限制取款限额来控制合约中资金的流出。这是一个非常简单的控件，但可以让您了解可编程区块链的功能：去中心化的软件来控制资金。

接下来是实际提现过程：

```
msg.sender.transfer(withdraw_amount);
```

这里发生了一些有趣的事情。msg 对象是所有合约均可访问的输入之一。它代表触发该合约执行的交易。属性 sender 是交易的发送者地址。函数 transfer 是一个内置函数，可将以太币ether从当前合约转移到发送者的地址。从后往前看的话，这意味着 transfer 这个+msg+ 到 sender 会触发该合同的执行。transfer 函数将一个数量作为唯一参数。我们将 withdraw_amount 值作为参数传递给前面几行声明的 withdraw 函数。

下一行是右花括号，表示+ withdraw +函数定义的结尾。

接下来，我们来定义另一个函数：

```
function () external payable {}
```

此函数是所谓的_fallback_或_default_函数，如果触发合约的交易没有指定合约中任何已声明的函数，或根本没有指定任何函数，或者不包含数据，则调用此函数。合约可以设定一个这样的默认功能函数（没有名称），通常是具有接收以太币的功能。这就是为什么它被定义为外部可付款功能，这意味着它可以将以太币存储到合约中。除接受以太币的功能以外，它什么都不做，如花括号pass: [(<code>{}</code>)]中的空定义所指示。如果我们进行的交易将以太币发送到合约地址，就好像它是一个钱包一样，那么此功能将处理这个交易并将以太币存入合约地址。

我们的默认功能正下方是最后一个大括号，它关闭了合约+Faucet+ 的定义。就是这个！

===编译水龙头合约

现在我们有第一个示例合约，我们需要使用Solidity编译器将Solidity代码转换为EVM二进制码，以便可以在区块链本身的EVM上执行。

Solidity编译器是独立的可执行文件，是各种框架的一部分，并捆绑在集成开发环境（IDE）中。为简单起见，我们将使用一种非常普遍的IDE，称为_Remix_。

使用你的Chrome浏览器（带有你先前安装的MetaMask钱包）浏览至Remix IDE，网址为 <https://remix.ethereum.org> 。

首次加载Remix时，它将以一个名为_ballot.sol_的示例合约开始。我们不需要这个示例合约，因此可以通过单击选项卡角上的+x+来关闭它，如[\[remix_close_tab\]](#)。



Figure 2. 关闭默认示例选项

现在，通过单击左上方工具栏中的圆形加号来添加新标签，如[\[remix_toolbar\]](#)。将新文件命名为_Faucet.sol_。



Figure 3. 单击加号打开一个新选项

打开新标签后，请复制并粘贴示例_Faucet.sol_中的代码，如 [\[remix_faucet_load\]](#)。



Figure 4. 将水龙头示例代码复制到新的选项卡中

将_Faucet.sol_合同加载到Remix IDE中之后，IDE将自动编译代码。如果一切顺利，你将在右侧的“编译”选项卡下看到一个带有“水龙头”的绿色框，确认编译成功（请参见[\[remix_compile\]](#)）。

□

Figure 5. Remix 成功编译了Faucet.sol 合约

如果出现问题，最可能的问题是Remix IDE使用的Solidity编译器版本不同于0.5.12。在这种情况下，我们的pragma指令将阻止_Faucet.sol_编译。要更改编译器版本，请转到“设置”选项卡，将版本设置为0.5.12，然后重试。

现在，Solidity编译器已将_Faucet.sol_编译为EVM二进制码。如果你感到好奇，二进制码如下所示：

```
PUSH1 0x60 PUSH1 0x40 MSTORE CALLVALUE ISZERO PUSH2 0xF JUMPI PUSH1 0x0 DUP1
REVERT JUMPDEST PUSH1 0xE5 DUP1 PUSH2 0x1D PUSH1 0x0 CODECOPY PUSH1 0x0 RETURN
STOP PUSH1 0x60 PUSH1 0x40 MSTORE PUSH1 0x4 CALLDATASIZE LT PUSH1 0x3F JUMPI
PUSH1 0x0 CALLDATALOAD PUSH29
0x1000000000000000000000000000000000000000000000000000000000000000
SWAP1 DIV PUSH4 0xFFFFFFFF AND DUP1 PUSH4 0x2E1A7D4D EQ PUSH1 0x41 JUMPI
JUMPDEST STOP JUMPDEST CALLVALUE ISZERO PUSH1 0x4B JUMPI PUSH1 0x0 DUP1 REVERT
JUMPDEST PUSH1 0x5F PUSH1 0x4 DUP1 DUP1 CALLDATALOAD SWAP1 PUSH1 0x20 ADD SWAP1
SWAP2 SWAP1 POP POP PUSH1 0x61 JUMP JUMPDEST STOP JUMPDEST PUSH8
0x16345785D8A0000 DUP2 GT ISZERO ISZERO ISZERO PUSH1 0x77 JUMPI PUSH1 0x0 DUP1
REVERT JUMPDEST CALLER PUSH20 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF AND
PUSH2 0x8FC DUP3 SWAP1 DUP2 ISZERO MUL SWAP1 PUSH1 0x40 MLOAD PUSH1 0x0 PUSH1
0x40 MLOAD DUP1 DUP4 SUB DUP2 DUP6 DUP9 DUP9 CALL SWAP4 POP POP POP POP ISZERO
ISZERO PUSH1 0xB6 JUMPI PUSH1 0x0 DUP1 REVERT JUMPDEST POP JUMP STOP LOG1 PUSH6
0x627A7A723058 KECCAK256 PUSH9 0x13D1EA839A4438EF75 GASLIMIT CALLVALUE LOG4 0x5f
PUSH24 0x7541F409787592C988A079407FB28B4AD000290000000000
```

看完这些编码后，你是不是很高兴使用诸如Solidity之类的高级语言，而不是直接用EVM的二进制码进行编程？我也是！

===在区块链上创建合约

我们有一份智能合约。我们已经将其编译为字节码。现在，我们需要在以太坊区块链上“注册”合约。我们将使用Ropsten测试网来测试我们的合约，这就是我们要提交给它的区块链。

在区块链上注册合约涉及创建特殊交易，其目的地为地址 0x00，也称为“零地址”。零地址是一个特殊的地址，它告诉以太坊区块链你要注册一个合约。幸运的是，Remix IDE将为你处理所有的这些交易，并将交易发送到MetaMask。

首先，切换到“运行”选项卡，然后在“环境”下拉选择框中选择“内置的Web3”。这将Remix IDE连接到MetaMask钱包，并通过MetaMask连接到Ropsten测试网络。完成此操作后，您可以在“环境”下看到“Ropsten”。另外，在“帐户选择”框中，它会显示你的钱包地址（请参见[\[remix_run\]](#)）。



Figure 6. 在Remix IDE的“Run”选项中，选择“Injected Web3”环境

你刚刚确认的“运行”设置正下方是准备创建的 Faucet 合约。单击[\[remix_run\]](#)中显示的“部署”按钮。

Remix将构建特殊的“创建”交易，如[\[remix_metamask_create\]](#)所示，MetaMask将要求你批准这个交易。你会注意到创建合约的交易中没有以太币，但是它有262字节的数据（已编译合约），将消耗10 gwei的燃料gas。单击提交以批准它。

MetaMask显示合约创建过程的交易

image::images/remix_metamask_create.png [“MetaMask显示合约创建过程的交易”]

现在你必须等待合约被区块链网络所确认。在Ropsten上确认合约大约需要15到30秒。Remix虽然看起来没在工作，但是要耐心等待。

当合约被创建完成后，它会出现在“Run”选项卡的底部（请参见[\[remix_contract_interact\]](#)）。



Figure 7. 水龙头合约上线了！

请注意，Faucet 合约现在具有其自己的地址：Remix将其显示为“0x72e ... c7829上的水龙头”（尽管你的地址，随机字母和数字将有所不同）。右侧的小剪贴板符号可让你将合约地址复制到剪贴板。我们将在下一节中使用它。

===操作合约

回顾到目前为止我们已经学到的东西：以太坊合约是可以控制资金的程序，这些程序在称为EVM的虚拟机中运行。

它们由特殊交易所创建，该交易提交其二进制码以记录在区块链上。一旦在区块链上创建合约，它们就有一个以太坊地址，就像钱包一样。每当有人将交易发送到合约地址时，都会使合约的代码在EVM中运行，并将交易的内容作为输入。发送至pass: [合同]地址的交易可能包含以太坊或数据，或两者都有。如果它们包含ether，它将“存入”合约余额。如果它们包含数据指令，则数据指令可以在合约中指定一个函数名并调用它，并将参数传递给该函数。

====在区块浏览器中查看合约地址

我们现在在区块链上记录了一个合约，我们可以看到它具有以太坊地址。让我们在_ropsten.etherscan.io_区块浏览器中检查一下，看看合约是什么样的。在Remix IDE中，通过单击合约名称旁边的剪贴板图标来复制合约的地址（请参见[\[remix_contract_address\]](#)）。

从Remix复制合约地址 image::images/remix_contract_address.png["从Remix复制合约地址"]

保持Remix打开；我们稍后会使用。现在，将浏览器导航到_ropsten.etherscan.io_并将地址粘贴到搜索框中。你应该看到合约在Ropsten以太坊测试网上的地址历史记录，如[\[etherscan_contract_address\]](#)所示。

在Etherscan区块浏览器中查看水龙头合约的地址 image :: images / etherscan_contract_address.png [“在etherscan块浏览器中查看水龙头合约的地址”]

====为合约充值

"Faucet.sol contract (test example)","sending ether to", id="ix_02intro-asciidoc18", range="startofrange")目前，该合约在其历史记录中只有一笔交易：创建合约的交易。如你所见，该合约也没有以太坊（零余额）。这是因为我们并没有在创建交易时，向合约发送任何以太坊。

我们的水龙头需要资金！我们的第一个项目将是使用MetaMask将以太坊发送给合约。你仍然应该在剪贴板中保留合约的地址（如果没有，请从Remix中再次复制它）。打开MetaMask，向其中发送1个以太坊，就像发送到其他以太坊地址一样（请参见[\[metamask_send_to_contract\]](#)）。

将1个以太坊发送到合约地址

image::images/metamask_send_to_contract.png [“”]

一分钟后，如果你重新加载Etherscan区块浏览器，它将显示另一笔交易到合约地址，并将余额更新为1个以太坊。

还记得我们 *Faucet.sol* 代码中未命名的默认外部应付账款功能吗？它看起来像这样：

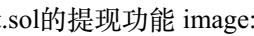
```
function () external payable {}
```

当你将交易发送到合约地址时，没有数据指定要调用的函数，它将调用此默认函数。因为我们将其声明为 `payed`，所以它接受了1个以太坊并将其存入合约的帐户余额中。你的交易将会在EVM的合约中运行，从而更新了余额。你已经完成了水龙头合约的充值！

====从我们的合约中提现

接下来，让我们从水龙头中取出一些资金。要提取资金，我们必须构造一个调用 `withdraw` 函数并向其传递 `withdraw_amount` 参数的。利用Remix使得实现这些操作变得简单，它将为我们的构建该交易，而MetaMask将发起该交易并让我们批准。

返回到Remix选项卡，然后在“运行”选项卡上查看合约。你应该看到一个标有 `withdraw` 的橙色框，其中有一个标有 `uint256 withdraw_amount` 的字段条目（请参见[\[remix_contract_withdraw\]](#)）。

在Remix中Faucet.sol的提现功能  [“在Remix中Faucet.sol的提现功能”]

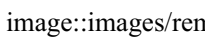
这是合约的Remix接口。它允许我们构造交易，用来调用合约中定义的功能。我们将输入+withdraw_amount+，然后单击“提现”按钮以创建交易。

首先，让我们找出 withdraw_amount。我们想尝试提取0.1 ether，这是我们合约允许的最大金额。请记住，以太坊中的所有货币值都在内部以wei计价，我们的 withdraw 函数也期望 withdraw_amount 也以wei计价。我们想要的数量是0.1 ether，即100,000,000,000,000,000 wei（1后面是17个零）。

由于JavaScript的限制，Remix无法直接处理大于 10^{17} 的数值。所以，我们将其括在双引号中，以使Remix可以将其接收为字符串并将其作为+ BigNumber +进行操作。如果我们不将其包括在引号中，则Remix IDE将无法对其进行处理，并显示如下错误"Error encoding arguments: Error: Assertion failed."。

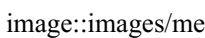
在+ withdraw_amount +框中输入“ 100000000000000000”（带引号），然后单击“withdraw”提现按钮（请参见<<remix_withdraw>>）。

在Remix中单击“提现”以创建提现交易

 [“”]

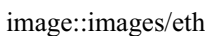
MetaMask将弹出一个交易窗口供您批准。点击确认，将你的提现通知发送到合约（请参见[\[metamask_withdraw\]](#)）。

MetaMask调用提现功能的交易

 [“MetaMask调用提现功能的交易”]

请稍等片刻，然后重新加载Etherscan区块浏览器，以查看 Faucet 合约地址历史记录中反映的交易（请参见[\[etherscan_withdrawal_tx\]](#)）。

Etherscan显示在调用提现功能的交易

 [“ Etherscan显示调用提现功能的交易”]

现在，我们看到一个新的交易，目的地为合约地址，交易金额为0 ether。合约的余额已更改，现在为0.9以太，因为它已按要求向我们发送了0.1以太。但是，我们在_合同地址历史记录_中看不到“ OUT”交易。

那么这个提现的操作到哪里去了？合同的地址历史记录页面上出现了一个新标签，名为内部交易。由于源自合约代码的0.1 ether转移是内部交易（也称为_message_）。点击该标签以查看它（请参见[\[etherscan_withdrawal_internal\]](#)）。

合约通过以下代码发送“内部交易”（来自_Faucet.sol_中的pass: [`withdraw`]函数）：

```
msg.sender.transfer(withdraw_amount);
```

现在我们来回顾一下：你从MetaMask钱包发送了一笔交易，其中包含合约调用指令，以 withdraw_amount 参数为0.1 ether调用 withdraw 函数。该交易导致合约在EVM中运行。当EVM运行 Faucet+合约的 +withdraw 函数时，它首先调用 require 函数，并验证所请求的金额小于或等于允许的最大0.1个ether取款。然后它调用 transfer 函数向你传入的地址发送ether。运行 transfer 功能生成了一笔内部交易，该交易从合约的余额中将0.1 ether存入了你的钱包地址。这就是Etherscan上内部交易一栏中显示出来的交易。

Etherscan显示将以太币从合约中转出的内部交易

image

 [“ Etherscan显示将以太币从合约中转出的内部交易”]

===本章小结

在本章中，你学习了使用MetaMask来设置钱包，并利用Ropsten测试网络上的水龙头为其提供资金。你将以太币发送到钱包的以太坊地址中，然后将以太币发送到水龙头的以太坊地址中。

之后，你在Solidity中编写了一个水龙头合约，并使用Remix IDE工具将合约编译为EVM二进制代码，然后使用Remix工具发送交易并在Ropsten区块链上创建了一个+ 水龙头 合约。当合约创建完成，水龙头 合约就有一个以太坊地址，然后你可以发送了一些测试以太币。最后，你构造了一个事务来调用 withdraw +函数，并成功地请求0.1个测试以太币。水龙头合约自动检查了请求，并通过内部交易向你的账户发送了0.1个测试以太币。

可能看起来并不多，但是你已经成功地通过在去中心化世界计算机上运行的软件进行了资金的交换。

我们将在[\[smart_contracts_chapter>>中学习更多的智能合约编程，并在<<smart_contract_security\]](#)中了解合约编程的最佳做法和安全注意事项。

Last updated 2022-01-12 22:01:07 +0800