关于 ethereum.org {#about-ethereumorg}

ethereum.org 是一个为以太坊社区打造的公共且开源的网站，任何人都可以贡献其中。 我们有一个由以太坊基金会资助的小组致力于维护和开发该网站。

我们的愿景 {#our-vision}

ethereum.org 的使命是成为不断增长的以太坊社区的最佳门户。 {#mission}

我们提供相应的教育资源，旨在帮助新用户熟悉以太坊及其关键概念。 我们希望：

- 向新用户解释以太坊相关技术
- 帮助新用户使用以太币和以太坊
- 帮助新开发者开始工作
- 推广以太坊领域的进展
- 展示社区创建的资源
- 使以太坊相关教育覆盖更多的语言

我们的核心理念能帮助我们做到这一点。

核心理念 {#core-principles}

1. ethereum.org 是通往以太坊的门户网站 {#core-principles-1}

我们希望用户的兴趣得到激发，他们的问题得到答复。 因此，我们的门户网站需要将信息、“奇迹的瞬间”和与通往优秀社区资源的链接综合起来。 我们收录内容的目标是作为“入门通道”，而不是取代现存的大量资源。 我们致力于支持和整合社区开发资源，提高它们的可见度，并使它们更易被追索和使用。

以太坊社区的核心是：不仅仅服务于社区，更要与之合作并吸取反馈意见。 这个网站不仅仅是为了我们现在拥有的社区，更是为了我们希望成长为的社区。 我们必须记住，我们的社会是全球性的，它包含来自许多语言、区域和文化的人。

2. ethereum.org 一直在进化 🛠️ {#core-principles-2}

以太坊和社区一直在不断演变，而 ethereum.org 也将如此。 这就是为什么网站使用了简单的设计系统和模块结构的原因。 当我们了解到更多关于人们如何使用该网站和社区对该网站的需求时，我们会进行反复的改进。

我们是开源的，一个拥有众多的贡献者的社群，所以您可以提出建议或帮助我们解决问题。

了解如何贡献

3. ethereum.org 不是一个卖普通商品的网站 🦄 {#core-principles-3}

以太坊是一个庞然大物：它包括社区、技术、思想和意识形态等等。 这意味着网站需要处理许多不同的用户的需求， 比如“需要特定工具的开发者”和“刚刚购买了一些 ETH 却不知道钱包是什么的新人”

"哪个网站是了解和学习区块链平台的最佳网站？"这仍然是一个未解之谜，但我们就是先驱。 构建它仍需要不断地试验。

设计原则 {#design-principles}

我们使用一组设计原则来帮助我们做关于内容和设计网站上的决定。

风格指南 {#style-guide}

我们有一个风格指南来规范内容，从而以使贡献过程更加容易。

我们欢迎大家对设计原则和风格指南提出反馈意见。 请记住，ethereum.org 来自社区，服务社区。

如果您想要对官网做贡献，请先阅读我们的宗旨 ，以及我们的风格指南。

愿景与计划 {#roadmap}

正在招聘 {#open-jobs}

虽然这个网站是开源的，任何人都可以在上面协作，但我们确实有一个团队专门负责 ethereum.org 和其他以太坊基金会的项目。

我们会在这里发布空缺的职位。 如果你在这里看不到适合你的角色，请加入Discord，并让我们知道你想如何与我们合作！

您想找除了 ethereum.org 以外的工作？ 查看其他与以太坊相关的工作。

title: 以太坊活动
description: 如何加入以太坊社区。
sidebar: true

lang: zh

活动预告 {#events}

每个月，世界各地都会举办大型以太坊活动。您可以考虑参加附近的一个活动，认识更多的社区成员，了解就业机会，并培养新的技能。

这是一个由我们社区维护的不完整列表。 想要将了解到的即将举行的以太坊活动添加到此列表？ 请添加！

以太坊聚会 {#meetups}

找不到适合您的活动？ 尝试加入一个聚会。 聚会是指由各个以太坊爱好者团体举办的小型活动——这是对以太坊感兴趣的人们聚集在一起的机会，他们可以一起谈论并了解以太坊的最新发展。

有兴趣举办自己的聚会？ 查看 BUIDL 网络，这是由 ConsenSys 发起的一项举措，旨在帮助支持以太坊的各个聚会社区。

这是一个由我们社区制定的不完整列表。 您可以在此处查找更多以太坊聚会。 想要将了解到的活跃聚会小组添加到此列表？ 请添加！

title: 以太坊帐户
description: 对 以太坊帐户的解释--它们的数据结构以及它们与密钥对密码学的关系。
lang: zh

sidebar: true

一个以太坊帐户是一个具有以太币 (ETH) 余额的实体，可以在以太坊上发送交易。 帐户可以由用户控制，也可以作为智能合约部署。

前置要求 {#prerequisites}

帐户是一个很适合初学者的主题。 但为了帮助您更好地理解这个页面，我们建议您首先阅读我们的以太坊简介。

帐户类型 {#types-of-account}

以太坊有两种帐户类型：

- 外部持有 – 私钥的所有者控制
- 合约 – 一种由代码控制，部署在网络上的智能合约。 了解智能合约。

这两种帐户类型都能：

- 接收、持有和发送 ETH 和 token
- 与已部署的智能合约进行交互

主要区别 {#key-differences}

外部持有

- 创建帐户是免费的
- 可以发起交易
- 外部所有的帐户之间只能进行 ETH 和代币交易

合约

- 创建合约存在成本，因为需要使用网络存储空间
- 只能在收到交易时发送交易
- 从外部帐户向合约帐户发起的交易能触发可执行多种操作的代码，例如转移代币甚至创建新合约

理解帐户 {#an-account-examined}

以太坊帐户有四个字段：

- `nonce` – 显示从帐户发送的交易数量的计数器。 这将确保交易只处理一次。 在合约帐户中，这个数字代表该帐户创建的合约数量
- `balance` – 这个地址拥有的 Wei 数量。 Wei 是以太币的计数单位，每个 ETH 有 1e+18 Wei。

- `codeHash` - 该哈希表示以太坊虚拟机 (EVM) 上的帐户*代码*。 合约帐户具有编程的代码片段，可以执行不同的操作。 如果帐户收到消息调用，则执行此 EVM 代码。 与其他帐户字段不同，不能更改。 所有代码片段都被保存在状态数据库的相应哈希下，供后续检索。 此哈希值称为 codeHash。 对于外部所有的帐户，codeHash 字段是空字符串的哈希。
- `storageRoot` – 有时被称为存储哈希。 Merkle Patricia trie 根节点的 256 位哈希已编码了帐户的存储内容（256 位整数值映射），并编码为 Trie，作为来自 256 的 Keccak 256 位哈希的映射位整数键，用于 RLP 编码的 256 位整数值。 此 Trie 对此帐户存储内容的哈希进行编码，默认情况下为空。

显示帐户组成部分的图表 *图表来自 [Ethereum EVM illustrated]*

外部持有的帐户和密钥对 {#externally-owned-accounts-and-key-pairs}

帐户由公钥和私钥加密对组成。 它们有助于证明交易实际上是由发送者签名的，并防止伪造。 您的私钥是您用来签名交易的密钥，所以它保障您对与您帐户相关的资金进行管理。 您从未真正持有加密货币，您持有私钥 – 资金总是在以太坊的账本上。

这将防止恶意参与者广播虚假交易，因为您总是可以验证交易的发送者。

如果 Alice 想要从她自己的帐户发送 ETH 到 Bob 的帐户，Alice 需要创建交易请求并将其发送到网络进行验证。 以太坊对公钥加密的使用确保了 Alice 可以证明她最初发起了交易请求。 没有加密机制，恶意对手 Eve 可以简单地公开广播一个看起来像“从 Alice 的帐户发送 5 ETH 到 Eve 帐户”的请求。而且没有人能够证实它不是来自 Alice 的。

帐户创建 {#account-creation}

当你想要创建一个帐户时，大多数库将生成一个随机的私钥。

私钥由 64 个十六进制字符组成，可以用密码加密保存。

例如：

`ffffffffffffffffffffffffffffffffebaaedce6af48a03bbfd25e8cd036415f`

使用[椭圆曲线数字签名算法]从私钥生成公钥。 通过获取公钥 Keccak-256 哈希的最后 20 个字节并校验码前面添加 `0x`，可以为帐户获取公共地址。

下面是使用 GETH 的 `personal_newAccount` 在控制台中创建一个帐户的例子

```
> personal.newAccount()
Passphrase:
Repeat passphrase:
"0x5e97870f263700f46aa00d967821199b9bc5a120"

> personal.newAccount("h4ck3r")
"0x3d80b31a78c30fc628f20b2c89d7ddbf6e53cedc"
```

[GETH 文档]

可以通过您的私钥获取公钥，但您不能通过公钥获取私钥。 这意味着保持私人密钥的安全至关重要，如同名称所建议的 **PRIVATE**。

您需要一个私钥来签署消息和交易并输出签名。 然后其他人可以使用签名获取您的公钥，证明信息的作者。 在您的应用程序中，您可以使用 javascript 库向网络发送交易。

合约帐户 {#contract-accounts}

合约帐户也有一个 42 个字符组成的十六进制地址：

例如：

`0x06012c8cf97bead5deae237070f9587f8e7a266d`

合约地址通常在将合约部署到以太坊区块链时给出。 地址产生自创建人的地址和从创建人地址发送的交易数量（“nonce”）。

关于钱包的说明 {#a-note-on-wallets}

帐户和钱包不同。 账户是用户拥有的以太坊账户的密钥和地址对。 钱包是一个界面或者说应用程序，可以让您与以太坊账户交互。

直观演示 {#a-visual-demo}

跟随 Austin 了解哈希函数和密钥对。

延伸阅读 {#further-reading}

还有哪些社区资源对您有所帮助？ 编辑并添加本页面！

相关主题 {#related-topics}

- 智能合约
- 交易

---

title: "Solidity 和 Truffle 持续集成设置"
description: 如何为 Truffle 测试设置 Travis 或 Circle CI 以及有用的插件
author: Markus Waas
lang: zh
sidebar: true
tags:

- "solidity"
- "智能合约"
- "测试"
- "truffle"
- "持续集成"
- "ganache"
  skill: 中等难度
  published: 2020-06-05
  source: soliditydeveloper.com
  sourceUrl: https://soliditydeveloper.com/continuous-integration

---

与 Truffle 的持续集成 (CI) 非常适合在实施一组基本测试后进行开发。 它允许您运行非常长的测试。在合并拉取请求之前确保所有测试通过，并使用其他工具跟踪各种统计数据。

我们将使用 Truffle Metacoin Box 来设置我们的持续集成。 您可以选择 Travis CI 或 Circle CI。

设置 Travis CI {#setting-up-travis-ci}

添加 Travis CI 很简单。 您只需要将 `.travis.yml` 配置文件添加到项目的根目录：

```
language: node_js
node_js:
  - 10

cache: npm

before_script:
  - echo fs.inotify.max_user_watches=524288 | sudo tee -a /etc/sysctl.conf &&
sudo sysctl -p

script:
  - npm test
```

我们现在保持简单，只运行执行 Truffle 单元测试的测试脚本。 但我们有一个问题，在 Travis CI 机器上没有区块链。 一个简单的解决方法是使用命令 `npm install ganache-cli` 安装 ganache-cli，并在测试前运行它。 您可以通过在 `npx truffle test` 调用前添加一个带有 npx `ganache-cli > /dev/null` 行的 bash 脚本来实现这一点。 完整示例 bash 脚本。

设置 Circle CI {#setting-up-circle-ci}

CircleCi 需要更长的配置文件。 额外的 `npm ci` 命令在 Travis 中自动完成。 它安装依赖项比 `npm install` 更快和更安全。 在测试之前，我们再次使用 Travis 版本中的同一个脚本来运行 ganache-cli。

```
version: 2

aliases:
  - &defaults
    docker:
      - image: circleci/node:10

  - &cache_key_node_modules
    key: v1-node_modules-{{ checksum "package-lock.json" }}

jobs:
  dependencies:
    <<: *defaults
    steps:
      - checkout
      - restore_cache:
          <<: *cache_key_node_modules
      - run:
          name: Install npm dependencies
          command: |
            if [ ! -d node_modules ]; then
              npm ci
            fi
      - persist_to_workspace:
          root: .
          paths:
```

```
            - node_modules
            - build
      - save_cache:
          paths:
            - node_modules
          <<: *cache_key_node_modules

  test:
    <<: *defaults
    steps:
      - checkout
      - attach_workspace:
          at: .
      - run:
          name: Unit tests
          command: npm test

workflows:
  version: 2
  everything:
    jobs:
      - dependencies
      - test:
          requires:
            - dependencies
```

添加 eth-gas-reporter 插件 {#adding-the-eth-gas-reporter-plugin}

Eth-gas-reporter 插件对于记录您的智能合约函数的 gas 成本相当有用。 在您的 CI 中使用它，将进一步有助于在添加拉取请求时显示差异。

第 1 步：安装 eth-gas-reporter 插件和 Codechecks {#step-1-install-the-eth-gas-reporter-plugin-and-codechecks}

```
$ npm install --save-dev eth-gas-reporter
$ npm install --save-dev @codechecks/client
```

第 2 步：在您的 truffle-config.js 内的 mocha 设置中加入该插件 {#step-2-add-the-plugin-to-the-mocha-settings-inside-your-truffle-configjs}

查看选项

```
module.exports = {
  networks: { ... },
  mocha: {
    reporter: 'eth-gas-reporter',
    reporterOptions: {
      excludeContracts: ['Migrations']
    }
  }
};
```

第 3 步：在您的项目的根目录中添加一个 codechecks.yml {#step-3-add-a-codechecksyml-to-your-projects-root-directory}

```
checks:
  - name: eth-gas-reporter/codechecks
```

第 4 步：在 test 命令后运行 codechecks {#step-4-run-codechecks-after-the-test-command}

```
- npm test
- npx codechecks
```

第 5 步：创建一个 Codechecks 帐户 {#step-5-create-a-codechecks-account}

- 使用 Codechecks 创建一个帐户。
- 将 Github repo 添加到其中。
- 复制密钥并将 `CC_SECRET=COPIED SECRET` 添加到您的 CI（Travis 参见这里，CircleCi 参见这里）。
- 现在继续创建拉取请求。

这就完成了。 现在，您将看到一份关于您的拉取请求的 gas 成本变化的报告。

Gas 成本报告示例

添加 solidity-coverage 插件 {#adding-the-solidity-coverage-plugin}

通过 solidity-coverage 插件，您可以检查您的代码路径有多少被您的测试所覆盖。 将此插件添加到您的 CI，设置好后，使用非常方便。

第 1 步：创建一个 metacoin 项目并安装覆盖工具 {#step-1-create-a-metacoin-project-and-install-coverage-tools}

```
$ npm install --save-dev truffle
$ npm install --save-dev coveralls
$ npm install --save-dev solidity-coverage
```

第 2 步：将 solidity-coverage 添加到 truffle-config.js 的插件数组中。 {#step-2-add-solidity-coverage-to-the-plugins-array-in-truffle-configjs}

```
module.exports = {
  networks: {...},
  plugins: ["solidity-coverage"]
}
```

第 3 步：将 coverage 命令添加到 .travis.yml 或 Circle CI config.yml {#step-3-add-the-coverage-commands-to-the-travisyml-or-circle-ci-configyml}

```
- npx truffle run coverage
- cat coverage/lcov.info | npx coveralls
```

Solidity coverage 启动了它自己的 ganache-cli，所以我们不必担心这个问题。 但不要替换常规测试命令，coverage 的 ganache-cli 工作方式不同，因此不能代替运行正常的单位测试。

第 4 步：将 repository 添加到 Coveralls {#step-4-add-repository-to-coveralls}

- 使用 Coveralls 创建一个帐户
- 添加 Github repo 到 Coverall
- 创建拉取请求

Coverall 示例

进一步的想法 {#further-ideas}

- MythX：使用 MythX，您可以自动分析智能合约的安全性。 因此， 将其添加到您的 CI 是非常有意义的。
- 检查：好的代码在某种程度上可以通过检查工具来实现。 Eslint 对于 JavaScript 来说非常有用，很容易设置，Solhint 可以用于 Solidity。
- 长测试：有时您可能想要添加极端测试，例如使用数百名用户测试一个合约。 这需要很长时间。 不要在每次测试运行中都运行它们，而是将它们添加到 CI 中。

这是全部内容了。 持续集成是您开发中非常有用的战略。 您可以在 Truffle-CI-example 查看完整的示例。 请务必移除 Circle-CI 或 Travis，只使用一个就够了！

title: 使用 Waffle 测试智能合约
description: 学习如何测试 Solidity 智能合约，以及如何将智能合约匹配器与 Waffle 配合使用
author: Vladislav Starostenko
tags:

- "waffle"
- "智能合约"
- "solidity"

- "测试"
  skill: 中等难度
  lang: zh
  sidebar: true
  published: 2020-10-16

在本教程中，您将学习如何：

- 使用 Waffle 编写智能合约测试
- 使用一些流行的匹配器，配合 Waffle 来测试智能合约

本文假定：

- 您熟悉终端操作，
- 您能够创建一个新的 `JavaScript` 项目，
- 您写过一些 `Solidity` 代码，
- 您使用 `JavaScript` 写过一些测试，
- 您使用过 JavaScripts 的包安装程序，`yarn` 或 `npm`。

再次强调，如果您不符合其中任何一条，或者您不打算敲本文中的代码，您照着学仍然可以学得很好。

关于 Waffle 的几句话 {#a-few-words-about-waffle}

Waffle 是用于编写和测试智能合约的最先进的库。

使用 JavaScript API ethers-js。

您可以在 Waffle 文档 中阅读更多详细信息！

快速教程 {#the-quick-tutorial}

首先，创建新的 `JavaScript` 或 `TypeScript` 项目（我将使用 `TS`，但如果您使用 `JS`，问题也不大）：

有点像这样：

▶ package.json

```
{
  "name": "tutorial",
  "version": "1.0.0",
  "main": "index.js",
  "license": "MIT",
  "scripts": {
    "test": "export NODE_ENV=test && mocha",
    "lint": "eslint '{src,test}/**/*.ts'",
    "lint:fix": "eslint --fix '{src,test}/**/*.ts'",
    "build": "waffle"
  },
  "devDependencies": {
    "@types/mocha": "^5.2.7",
    "@typescript-eslint/eslint-plugin": "^2.30.0",
    "@typescript-eslint/parser": "^2.30.0",
    "eslint": "^6.8.0",
    "eslint-plugin-import": "^2.20.2",
```

```
    "ethers": "^5.0.17",
    "mocha": "^7.1.2",
    "ts-node": "^8.9.1",
    "typescript": "^3.8.3"
  }
}
```

▶ tsconfig.json

```
{
  "compilerOptions": {
    "declaration": true,
    "esModuleInterop": true,
    "lib": [
      "ES2018"
    ],
    "module": "CommonJS",
    "moduleResolution": "node",
    "outDir": "dist",
    "resolveJsonModule": true,
    "skipLibCheck": true,
    "strict": true,
    "target": "ES2018"
  }
}
```

▶ .gitignore

```
node_modules
build
```

▶ .eslintrc.js

```
module.exports = {
  "env": {
  "es6": true
},
  "extends": [
  "plugin:@typescript-eslint/recommended",
  "plugin:import/errors",
  "plugin:import/warnings",
  "plugin:import/typescript"
],
  "parser": "@typescript-eslint/parser",
  "parserOptions": {
  "project": "./tsconfig.json",
    "sourceType": "module"
},
  "rules": {
  "@typescript-eslint/camelcase": "off",
    "@typescript-eslint/explicit-function-return-type": "off",
    "@typescript-eslint/explicit-member-accessibility": [
    "error",
    {
```

```
      "accessibility": "no-public",
      "overrides": {
        "parameterProperties": "off"
      }
    }
  ],
    "@typescript-eslint/indent": [
    "error",
    2,
    {
      "ArrayExpression": 1,
      "CallExpression": {
        "arguments": 1
      },
      "FunctionDeclaration": {
        "body": 1,
        "parameters": 1
      },
      "FunctionExpression": {
        "body": 1,
        "parameters": 1
      },
      "ImportDeclaration": 1,
      "MemberExpression": 1,
      "ObjectExpression": 1,
      "SwitchCase": 1,
      "VariableDeclarator": 1,
      "flatTernaryExpressions": false,
      "ignoreComments": false,
      "outerIIFEBody": 1
    }
  ],
    "@typescript-eslint/interface-name-prefix": "off",
    "@typescript-eslint/member-delimiter-style": [
    "error",
    {
      "multiline": {
        "delimiter": "semi",
        "requireLast": true
      },
      "singleline": {
        "delimiter": "semi",
        "requireLast": false
      }
    }
  ],
    "@typescript-eslint/no-explicit-any": "off",
    "@typescript-eslint/no-parameter-properties": "off",
    "@typescript-eslint/no-unused-vars": [
    "error",
    {
      "args": "none",
      "ignoreRestSiblings": true,
      "vars": "all"
    }
```

```
  ],
    "@typescript-eslint/no-use-before-define": "off",
    "@typescript-eslint/no-useless-constructor": "error",
    "@typescript-eslint/no-var-requires": "warn",
    "accessor-pairs": "error",
    "array-bracket-spacing": [
    "error",
    "never"
  ],
    "arrow-spacing": [
    "error",
    {
      "after": true,
      "before": true
    }
  ],
    "block-spacing": [
    "error",
    "always"
  ],
    "brace-style": [
    "error",
    "1tbs",
    {
      "allowSingleLine": true
    }
  ],
    "camelcase": "off",
    "comma-dangle": [
    "error",
    {
      "arrays": "never",
      "exports": "never",
      "functions": "never",
      "imports": "never",
      "objects": "never"
    }
  ],
    "comma-spacing": [
    "error",
    {
      "after": true,
      "before": false
    }
  ],
    "comma-style": [
    "error",
    "last"
  ],
    "computed-property-spacing": [
    "error",
    "never"
  ],
    "constructor-super": "error",
    "curly": [
```

```
    "error",
    "multi-line"
  ],
    "dot-location": [
    "error",
    "property"
  ],
    "eol-last": "error",
    "eqeqeq": [
    "error",
    "always",
    {
      "null": "ignore"
    }
  ],
    "func-call-spacing": [
    "error",
    "never"
  ],
    "generator-star-spacing": [
    "error",
    {
      "after": true,
      "before": true
    }
  ],
    "handle-callback-err": [
    "error",
    "^(err|error)$"
  ],
    "import/default": "off",
    "import/named": "off",
    "import/no-extraneous-dependencies": [
    "error",
    {
      "devDependencies": false
    }
  ],
    "import/no-unresolved": "off",
    "indent": "off",
    "key-spacing": [
    "error",
    {
      "afterColon": true,
      "beforeColon": false
    }
  ],
    "keyword-spacing": [
    "error",
    {
      "after": true,
      "before": true
    }
  ],
    "linebreak-style": [
```

```
    "error",
    "unix"
  ],
    "lines-between-class-members": [
    "error",
    "always",
    {
      "exceptAfterSingleLine": true
    }
  ],
    "max-len": [
    "error",
    {
      "code": 120
    }
  ],
    "new-cap": [
    "error",
    {
      "capIsNew": false,
      "newIsCap": true
    }
  ],
    "new-parens": "error",
    "no-array-constructor": "error",
    "no-async-promise-executor": "error",
    "no-caller": "error",
    "no-class-assign": "error",
    "no-compare-neg-zero": "error",
    "no-cond-assign": "error",
    "no-const-assign": "error",
    "no-constant-condition": [
    "error",
    {
      "checkLoops": false
    }
  ],
    "no-control-regex": "error",
    "no-debugger": "error",
    "no-delete-var": "error",
    "no-dupe-args": "error",
    "no-dupe-keys": "error",
    "no-duplicate-case": "error",
    "no-empty-character-class": "error",
    "no-empty-pattern": "error",
    "no-eval": "error",
    "no-ex-assign": "error",
    "no-extend-native": "error",
    "no-extra-bind": "error",
    "no-extra-boolean-cast": "error",
    "no-extra-parens": [
    "error",
    "functions"
  ],
    "no-fallthrough": "error",
```

```
    "no-floating-decimal": "error",
    "no-func-assign": "error",
    "no-global-assign": "error",
    "no-implied-eval": "error",
    "no-inner-declarations": [
    "error",
    "functions"
  ],
    "no-invalid-regexp": "error",
    "no-irregular-whitespace": "error",
    "no-iterator": "error",
    "no-label-var": "error",
    "no-labels": [
    "error",
    {
      "allowLoop": false,
      "allowSwitch": false
    }
  ],
    "no-lone-blocks": "error",
    "no-misleading-character-class": "error",
    "no-mixed-operators": [
    "error",
    {
      "allowSamePrecedence": true,
      "groups": [
        [
          "==",
          "!=",
          "===",
          "!==",
          ">",
          ">=",
          "<",
          "<="
        ],
        [
          "&&",
          "||"
        ],
        [
          "in",
          "instanceof"
        ]
      ]
    }
  ],
    "no-mixed-spaces-and-tabs": "error",
    "no-multi-spaces": "error",
    "no-multi-str": "error",
    "no-multiple-empty-lines": [
    "error",
    {
      "max": 1,
      "maxEOF": 0
```

```
    }
  ],
    "no-negated-in-lhs": "error",
    "no-new": "error",
    "no-new-func": "error",
    "no-new-object": "error",
    "no-new-require": "error",
    "no-new-symbol": "error",
    "no-new-wrappers": "error",
    "no-obj-calls": "error",
    "no-octal": "error",
    "no-octal-escape": "error",
    "no-path-concat": "error",
    "no-proto": "error",
    "no-prototype-builtins": "error",
    "no-redeclare": [
    "error",
    {
      "builtinGlobals": false
    }
  ],
    "no-regex-spaces": "error",
    "no-return-assign": [
    "error",
    "except-parens"
  ],
    "no-return-await": "error",
    "no-self-assign": "error",
    "no-self-compare": "error",
    "no-sequences": "error",
    "no-shadow-restricted-names": "error",
    "no-sparse-arrays": "error",
    "no-tabs": "error",
    "no-template-curly-in-string": "error",
    "no-this-before-super": "error",
    "no-throw-literal": "error",
    "no-trailing-spaces": "error",
    "no-unexpected-multiline": "error",
    "no-unmodified-loop-condition": "error",
    "no-unneeded-ternary": [
    "error",
    {
      "defaultAssignment": false
    }
  ],
    "no-unreachable": "error",
    "no-unsafe-finally": "error",
    "no-unsafe-negation": "error",
    "no-use-before-define": [
    "error",
    {
      "classes": false,
      "functions": false,
      "variables": false
    }
```

```
  ],
    "no-useless-call": "error",
    "no-useless-catch": "error",
    "no-useless-computed-key": "error",
    "no-useless-escape": "error",
    "no-useless-rename": "error",
    "no-useless-return": "error",
    "no-whitespace-before-property": "error",
    "no-with": "error",
    "object-curly-spacing": [
    "error",
    "never"
  ],
    "object-property-newline": [
    "error",
    {
      "allowMultiplePropertiesPerLine": true
    }
  ],
    "one-var": [
    "error",
    {
      "initialized": "never"
    }
  ],
    "operator-linebreak": [
    "error",
    "after",
    {
      "overrides": {
        ":": "before",
        "?": "before"
      }
    }
  ],
    "padded-blocks": [
    "error",
    {
      "blocks": "never",
      "classes": "never",
      "switches": "never"
    }
  ],
    "prefer-const": [
    "error",
    {
      "destructuring": "all"
    }
  ],
    "prefer-promise-reject-errors": "error",
    "quote-props": [
    "error",
    "as-needed"
  ],
    "quotes": [
```

```
    "error",
    "single"
  ],
    "rest-spread-spacing": [
    "error",
    "never"
  ],
    "semi": [
    "error",
    "always"
  ],
    "semi-spacing": [
    "error",
    {
      "after": true,
      "before": false
    }
  ],
    "space-before-blocks": [
    "error",
    "always"
  ],
    "space-before-function-paren": [
    "error",
    {
      "anonymous": "always",
      "named": "never",
      "asyncArrow": "always"
    }
  ],
    "space-in-parens": [
    "error",
    "never"
  ],
    "space-infix-ops": "error",
    "space-unary-ops": [
    "error",
    {
      "nonwords": false,
      "words": true
    }
  ],
    "spaced-comment": [
    "error",
    "always",
    {
      "block": {
        "balanced": true,
        "exceptions": [
          "*"
        ],
        "markers": [
          "*package",
          "!",
          ",",
```

```
          ":",
          "::",
          "flow-include"
        ]
      },
      "line": {
        "markers": [
          "*package",
          "!",
          "/",
          ",",
          "="
        ]
      }
    }
  ],
    "symbol-description": "error",
    "template-curly-spacing": [
    "error",
    "never"
  ],
    "template-tag-spacing": [
    "error",
    "never"
  ],
    "unicode-bom": [
    "error",
    "never"
  ],
    "use-isnan": "error",
    "valid-typeof": [
    "error",
    {
      "requireStringLiterals": true
    }
  ],
    "wrap-iife": [
    "error",
    "any",
    {
      "functionPrototypeMethods": true
    }
  ],
    "yield-star-spacing": [
    "error",
    "both"
  ],
    "yoda": [
    "error",
    "never"
  ]
},
  "overrides": [
  {
    "files": [
```

```
      "test/**/*.ts"
    ],
    "rules": {
      "@typescript-eslint/no-explicit-any": "off",
      "@typescript-eslint/no-non-null-assertion": "off",
      "@typescript-eslint/no-var-requires": "off",
      "no-unused-expressions": "off",
      "prefer-promise-reject-errors": "off",
      "import/no-extraneous-dependencies": [
        "error",
        {
          "devDependencies": true
        }
      ]
    }
  }
]
}
```

第 1 步：在项目中安装 waffle 文档链接 {#step-1-install-waffle-in-your-project}

准备开始，请安装 `ethereum-waffle`。 在本教程中，我会使用 `yarn`，因此，要安装 `ethereum-waffle`，请运行：

```
yarn add --dev ethereum-waffle
```

第 2 步：编写智能合约 文档链接 {#step-2-write-a-smart-contract}

在本教程中，我会使用 ERC20 代币，它们来自 OpenZepelin。

因此，通过使用 `yarn` 安装 `OpenZepelin` 来添加它：

```
yarn add @openzeppelin/contracts -D
```

然后在 `src` 目录中创建 `BasicToken.sol` 合约：

```
pragma solidity ^0.6.0;

import "@openzeppelin/contracts/token/ERC20/ERC20.sol";

// Example class - a mock class using delivering from ERC20
contract BasicToken is ERC20 {
    constructor(uint256 initialBalance) ERC20("Basic", "BSC") public {
        _mint(msg.sender, initialBalance);
    }
}
```

第 3 步：编译智能合约 文档链接 {#step-3-compile-your-smart-contract}

要编译智能合约，在您项目的 `package.json` 中添加以下条目：

```
{
  "scripts": {
    "build": "waffle"
  }
}
```

在项目的根目录添加一个 `waffle.json` 文件：

一个 `waffle.json` 配置实例：

```
{
  "compilerType": "solcjs",
  "compilerVersion": "0.6.2",
  "sourceDirectory": "./src",
  "outputDirectory": "./build"
}
```

您可以在这里阅读更多关于 Waffle 配置的信息。

然后运行 `yarn build` 来编译智能合约。

您应该看到 Waffle 编译了您的合约，并将由此产生的 JSON 输出放置在 `build` 目录内。

▶ BasicToken.json

```
{
  "abi": [
    {
      "inputs": [
        {
          "internalType": "uint256",
          "name": "initialBalance",
          "type": "uint256"
        }
      ],
      "stateMutability": "nonpayable",
      "type": "constructor"
    },
    {
      "anonymous": false,
      "inputs": [
        {
          "indexed": true,
          "internalType": "address",
          "name": "owner",
          "type": "address"
        },
        {
          "indexed": true,
```

```
          "internalType": "address",
          "name": "spender",
          "type": "address"
        },
        {
          "indexed": false,
          "internalType": "uint256",
          "name": "value",
          "type": "uint256"
        }
      ],
      "name": "Approval",
      "type": "event"
    },
    {
      "anonymous": false,
      "inputs": [
        {
          "indexed": true,
          "internalType": "address",
          "name": "from",
          "type": "address"
        },
        {
          "indexed": true,
          "internalType": "address",
          "name": "to",
          "type": "address"
        },
        {
          "indexed": false,
          "internalType": "uint256",
          "name": "value",
          "type": "uint256"
        }
      ],
      "name": "Transfer",
      "type": "event"
    },
    {
      "inputs": [
        {
          "internalType": "address",
          "name": "owner",
          "type": "address"
        },
        {
          "internalType": "address",
          "name": "spender",
          "type": "address"
        }
      ],
      "name": "allowance",
      "outputs": [
        {
```

```
          "internalType": "uint256",
          "name": "",
          "type": "uint256"
        }
      ],
      "stateMutability": "view",
      "type": "function"
    },
    {
      "inputs": [
        {
          "internalType": "address",
          "name": "spender",
          "type": "address"
        },
        {
          "internalType": "uint256",
          "name": "amount",
          "type": "uint256"
        }
      ],
      "name": "approve",
      "outputs": [
        {
          "internalType": "bool",
          "name": "",
          "type": "bool"
        }
      ],
      "stateMutability": "nonpayable",
      "type": "function"
    },
    {
      "inputs": [
        {
          "internalType": "address",
          "name": "account",
          "type": "address"
        }
      ],
      "name": "balanceOf",
      "outputs": [
        {
          "internalType": "uint256",
          "name": "",
          "type": "uint256"
        }
      ],
      "stateMutability": "view",
      "type": "function"
    },
    {
      "inputs": [],
      "name": "decimals",
      "outputs": [
```

```
        {
          "internalType": "uint8",
          "name": "",
          "type": "uint8"
        }
      ],
      "stateMutability": "view",
      "type": "function"
    },
    {
      "inputs": [
        {
          "internalType": "address",
          "name": "spender",
          "type": "address"
        },
        {
          "internalType": "uint256",
          "name": "subtractedValue",
          "type": "uint256"
        }
      ],
      "name": "decreaseAllowance",
      "outputs": [
        {
          "internalType": "bool",
          "name": "",
          "type": "bool"
        }
      ],
      "stateMutability": "nonpayable",
      "type": "function"
    },
    {
      "inputs": [
        {
          "internalType": "address",
          "name": "spender",
          "type": "address"
        },
        {
          "internalType": "uint256",
          "name": "addedValue",
          "type": "uint256"
        }
      ],
      "name": "increaseAllowance",
      "outputs": [
        {
          "internalType": "bool",
          "name": "",
          "type": "bool"
        }
      ],
      "stateMutability": "nonpayable",
```

```
      "type": "function"
    },
    {
      "inputs": [],
      "name": "name",
      "outputs": [
        {
          "internalType": "string",
          "name": "",
          "type": "string"
        }
      ],
      "stateMutability": "view",
      "type": "function"
    },
    {
      "inputs": [],
      "name": "symbol",
      "outputs": [
        {
          "internalType": "string",
          "name": "",
          "type": "string"
        }
      ],
      "stateMutability": "view",
      "type": "function"
    },
    {
      "inputs": [],
      "name": "totalSupply",
      "outputs": [
        {
          "internalType": "uint256",
          "name": "",
          "type": "uint256"
        }
      ],
      "stateMutability": "view",
      "type": "function"
    },
    {
      "inputs": [
        {
          "internalType": "address",
          "name": "recipient",
          "type": "address"
        },
        {
          "internalType": "uint256",
          "name": "amount",
          "type": "uint256"
        }
      ],
      "name": "transfer",
```

```
      "outputs": [
        {
          "internalType": "bool",
          "name": "",
          "type": "bool"
        }
      ],
      "stateMutability": "nonpayable",
      "type": "function"
    },
    {
      "inputs": [
        {
          "internalType": "address",
          "name": "sender",
          "type": "address"
        },
        {
          "internalType": "address",
          "name": "recipient",
          "type": "address"
        },
        {
          "internalType": "uint256",
          "name": "amount",
          "type": "uint256"
        }
      ],
      "name": "transferFrom",
      "outputs": [
        {
          "internalType": "bool",
          "name": "",
          "type": "bool"
        }
      ],
      "stateMutability": "nonpayable",
      "type": "function"
    }
  ],
  "evm": {
    "bytecode": {
      "linkReferences": {},
```

"object":
"6080604052348015620000115760008 0fd5b506040516200153938038062001539833981810160 4
0526020811015620000375760008 0fd5b810190808051906020019092919050505060405180604 00
1604052806005815260200 17f426173696300000000000000000000000000000000000000000000000
0000000008152506040518060400160405280600381526020017f425343000000000000000000000
0000000000000000000000000000000000000815250816003908051906020019062000 0cc9291906
2000389565b508060049080519060200190620000e59291906200038956 5b506012600560006101 0
00a81548160ff021916908360ff160217905550505062000116338262000 11d60201b60201c565b5
062000438565b600073ffffffffffffffffffffffffffffffffffffffff168273ffffffffffffffff
ffffffffffffffffffffffffff161415620001c1576040517f08c379a000000000000000000000000
0000000000000000000000000000000000815260040180806020018281038252601f8152602001807
f45524332303a206d696e7420746f20746865207a65726f206164647265737300815250602001915
0506040518091039 0fd5b620001d560008383620002fb60201b60201c565b620001f181600254620
0030060201b62000f2d1790919060201c565b600281905550620002 4f816000808573ffffffffffff
ffffffffffffffffffffffffff1673ffffffffffffffffffffffffffffffffffffffff1681526
0200190815260200160002054620003006020 1b62000f2d1790919060201c565b6000808473fffff
ffffffffffffffffffffffffffffffffff1673ffffffffffffffffffffffffffffffffffffffff1
6815260200190815260200160002081905550817 3ffffffffffffffffffffffffffffffffffffff
f16600073ffffffffffffffffffffffffffffffffffffffff167fddf252ad1be2c89b69c2b068fc3
78daa952ba7f163c4a11628f55a4df523b3ef836040518082815260200191505060405180910390a
35050565b505050565b600080828401905083811015620003 7f576040517f08c379a000000000000
0000000000000000000000000000000000000000000000000000000815260040180806020018281038252601b8
152602001807f536166654d6174683a206164646974696f6e206f766572666c6f770000000000815
250602001915050604051809103 90fd5b809150509291505056 5b828054600181600116156101000
2031660029004906000526020600020906 01f016020900481019282601f10620003cc57805160ff1
916838001178555620003fd565b828001600101855582156200 03fd579182015b82811115620003f
c578251825591602001919060010190620003df565b5b5090506200040c919062000410565b50905
65b6200043591905b808211156200043157600081600090555506001016200041 7565b5090565b905
65b6110f1806200044860003960 00f3fe6080604052348015610010576000 80fd5b5060043610610
0a95760003560e01c806339509351116100715780633950935114610 25f57806370a08231146102c
557806395d89b411461031d578063a457c2d7146103a0578063a9059cbb14610406578063dd62ed3
e1461046c576100a9565b806306fdde03146100ae578063095ea7b3146101315780631 8160ddd146
1019757806323b872dd146101b5578063313ce5671461023b575b600080fd5b6100b66104e4565b6
0405180806020018281038252838181518152602001915080519060200190808383 60005b8381101
56100f657808201518184015260208101905061 00db565b505050509050908101906 01f168015610
1235780820380516001836020036101000a031916815260200191505b509250505060405180910390
0f35b61017d60048036036040811015610147576000 80fd5b81019080803573ffffffffffffffff
ffffffffffffffffffffffff169060200190929190803590602001909291905050506105 86565b604
0518082151515158152602001915050604051809103 90f35b61019f6105a4565b604051808281526
0200191505060405180910390f35b610221600480360360608110156101cb57600080fd5b8101908
0803573ffffffffffffffffffffffffffffffffffffffff169060200190929190803573ffffffff
ffffffffffffffffffffffffffff169060200190929190803590602001909291905050506105a
e565b604051808215151515815260200191505060405180910390f35b610243610687565b6040518
08260ff1660ff168152602001915050604051809103 90f35b6102ab600480360360408110156102 7
557600080fd5b81019080803573ffffffffffffffffffffffffffffffffffffffff1690602001909
291908035906020019092919050505061069e565b604051808215151515815260200191505060405
180910390f35b610307600480360360208110156102db57600080fd5b81019080803573ffffffff
ffffffffffffffffffffffffffff169060200190929190505050610751565b604051808281526
0200191505060405180910390f35b610325610799565b604051808060200182810382528381518
1526020019150805190602001908083836000 5b838110156103657808201518184015260208101 9
05061034a565b505050509050908101906 01f168015610392578082038051600183602003610100 0
a031916815260200191505b509250505060405180910390f35b6103ec600480360360408110156 10
3b657600080fd5b81019080803573ffffffffffffffffffffffffffffffffffffffff16906020019
092919080359060200190929190505050610 83b565b604051808215151515815260200191505060 4
0518091039 0f35b610452600480360360408110156104 1c57600080fd5b81019080803573fffffff

```
ffffffffffffffffffffffffffffffffff16906020019092919080359060200190929190505050610
908565b604051808215151515815260200191505060405180910390f35b6104ce600480360360408
110156104825760008 0fd5b81019080803573ffffffffffffffffffffffffffffffffffffffff169
06020019092919080357 3ffffffffffffffffffffffffffffffffffffffff169060200190929190 5
05050610926565b604051808281526020019150506040518091039 0f35b606060038054600181600
116156101000203166002900480601f016020809104026020016040519081016040528092919081 8
152602001828054600181600116156101000203166002900480156 1057c5780601f1061055157610
100808354040283529160200191610 57c565b820191906000526020600020905b815481529060010
190602001808311610 55f57829003601f168201915b505050505090509056 5b600061059a6105936
109ad565b84846109b5565b60019050929150505 65b60006002549050905 65b60006105bb8484846
10bac565b61067c846105c76109ad565b61067785604051806060016040528060288152602001611
026602891396001600 08b73ffffffffffffffffffffffffffffffffffffffff1673ffffffffffffff
ffffffffffffffffffffffffffff16815260200190815260200160002060006106 2d6109ad565b73f
ffffffffffffffffffffffffffffffffffffffff1673ffffffffffffffffffffffffffffffffffff
fff16815260200190815260200160002054610e6d9092919063ffffffff16565b6109b5565b60019
0509392505050565b60006005600090549061010 00a900460ff16905090565b60006107476106ab6
109ad565b846107428560016000610 6bc6109ad565b73ffffffffffffffffffffffffffffffffff
fffff1673ffffffffffffffffffffffffffffffffffffffff16815260200190815260200160002060
0008973ffffffffffffffffffffffffffffffffffffffff1673ffffffffffffffffffffffffffff
ffffffffff16815260200190815260200160002054610f2d90919063ffffffff16565b6109b5565
b60019050929150505 65b60008060008373ffffffffffffffffffffffffffffffffffffffff1673f
ffffffffffffffffffffffffffffffffffffffff16815260200190815260200160002054905091905
0565b6060600480546001816001161561010002031660029004806 01f016020809104026020016 04
051908101604052809291908181526020018280546001816001161561010002031660029004801 56
108315780601f1061080657610100808354040283529160200191610831565b82019190600052602
0600020905b815481529060010190602001808311610814578290036 01f168201915b50505050509
05090565b60006108fe6108486109ad565b846108f985604051806060016040528060258152602 00
161109760259139600160006108726109ad565b73ffffffffffffffffffffffffffffffffffffff
f1673ffffffffffffffffffffffffffffffffffffffff16815260200190815260200160002060008
a73ffffffffffffffffffffffffffffffffffffffff1673ffffffffffffffffffffffffffffffff
fffffff16815260200190815260200160002054610e6d9092919063ffffffff16565b6109b5565b6
00190509291505056 5b600061091c6109156109ad565b8484610bac565b60019050929150505 65b6
000600160008473ffffffffffffffffffffffffffffffffffffffff1673ffffffffffffffffffff
ffffffffffffffffff16815260200190815260200160002060008373ffffffffffffffffffffff
ffffffffffffffff1673ffffffffffffffffffffffffffffffffffffffff16815260200190815260
02001600020549050929150505 65b600033905090565b600073ffffffffffffffffffffffffffff
ffffffffff168373ffffffffffffffffffffffffffffffffffffffff161415610a3b576040517f0
8c379a0000000000000000000000000000000000000000000000000000000000000815260040180806 02
00182810382526024815260200180611073602491396040019150506040518091039 0fd5b600073f
ffffffffffffffffffffffffffffffffffffffff168273ffffffffffffffffffffffffffffffffff
fffff161415610ac1576040517f08c379a0000000000000000000000000000000000000000000000
000000000008152600401808060200182810382526022815260200180610fde60229139604001915
05060405180910390fd5b80600160008573ffffffffffffffffffffffffffffffffffffffff1673f
ffffffffffffffffffffffffffffffffffffffff16815260200190815260200160002060008473fff
ffffffffffffffffffffffffffffffffffffff1673ffffffffffffffffffffffffffffffffffff
f16815260200190815260200160002081905550817 3ffffffffffffffffffffffffffffffffffff
fff168373ffffffffffffffffffffffffffffffffffffffff167f8c5be1e5ebec7d5bd14f71427d1
e84f3dd0314c0f7b2291e5b200ac8c7c3b925836040518082815260200191505060405180910390a
3505050565b600073ffffffffffffffffffffffffffffffffffffffff168373ffffffffffffffff
ffffffffffffffffffffffff161415610c32576040517f08c379a0000000000000000000000000000
000000000000000000000000000000815260040180806020018281038252602581526020018061104
e602591396040019150506040518091039 0fd5b600073ffffffffffffffffffffffffffffffffff
fffff168273ffffffffffffffffffffffffffffffffffffffff161415610cb8576040517f08c379a
0000000000000000000000000000000000000000000000000000000000000815260040180806020018 28
103825260238152602001806 10fbb602391396040019150506040518091039 0fd5b610cc38383836
```

10fb5565b610d2e8160405180606001604052806026815260200161100060269139600080877 3fff
ffffffffffffffffffffffffffffffffffffff1673ffffffffffffffffffffffffffffffffffffffff
f16815260200190815260200160002054610e6d9092919063ffffffff16565b6000808573ffffff
ffffffffffffffffffffffffffffffffff1673ffffffffffffffffffffffffffffffffffffffff168
15260200190815260200160002081905550610dc1816000808573ffffffffffffffffffffffffff
ffffffffffff1673ffffffffffffffffffffffffffffffffffffffff16815260200190815260200
160002054610f2d90919063ffffffff16565b6000808473ffffffffffffffffffffffffffffffff
fffffff1673ffffffffffffffffffffffffffffffffffffffff16815260200190815260200160002
0819055508173ffffffffffffffffffffffffffffffffffffffff168373ffffffffffffffffffff
ffffffffffffffffff167fddf252ad1be2c89b69c2b068fc378daa952ba7f163c4a11628f55a4df
523b3ef836040518082815260200191505060405180910390a3505050565b6000838311158290610
f1a576040517f08c379a0000000000000000000000000000000000000000000000000000000815
2600401808060200182810382528381815181526020019150805190602001908083836000 5b83811
015610edf578082015181840152602081019050610ec4565b50505050905090810190601f1680156
10f0c578082038051600183602003610100 0a031916815260200191505b50925050506040518091 0
390fd5b506000838503905080915050939250505056 5b6000808284019050838110156 10fab57604
0517f08c379a0000000000000000000000000000000000000000000000000000000081526004018
0806020018281038252601b8152602001807f536166654d6174683a206164646974696f6e206f766
572666c6f770000000000815250602001915050604051809103 90fd5b809150509291505056 5b505
05056fe45524332303a207472616e7366657220746f20746865207a65726f206164647265737345 5
24332303a20617070726f766520746f20746865207a65726f206164647265737345524332303a207
472616e7366657220616d6f756e74206578636565647320616c616e636545524332303a2074726
16e7366657220616d6f756e74206578636565647320616c6c6f77616e636545524332303a2074726
16e736665722066726f6d20746865207a65726f206164647265737345524332303a20617070726f7
6652066726f6d20746865207a65726f206164647265737345524332303a206465637265617365642
0616c6c6f77616e63652062656c6f77207a65726fa2646970667358221220 81c840f087cef92fecc
b03fadc678b2708c331896ec5432b5d4c675f27b6d3e664736f6c63430006020033",

"opcodes": "PUSH1 0x80 PUSH1 0x40 MSTORE CALLVALUE DUP1 ISZERO PUSH3 0x11 JUMPI PUSH1 0x0 DUP1 REVERT JUMPDEST POP PUSH1 0x40 MLOAD PUSH3 0x1539 CODESIZE SUB DUP1 PUSH3 0x1539 DUP4 CODECOPY DUP2 DUP2 ADD PUSH1 0x40 MSTORE PUSH1 0x20 DUP2 LT ISZERO PUSH3 0x37 JUMPI PUSH1 0x0 DUP1 REVERT JUMPDEST DUP2 ADD SWAP1 DUP1 DUP1 MLOAD SWAP1 PUSH1 0x20 ADD SWAP1 SWAP3 SWAP2 SWAP1 POP POP POP PUSH1 0x40 MLOAD DUP1 PUSH1 0x40 ADD PUSH1 0x40 MSTORE DUP1 PUSH1 0x5 DUP2 MSTORE PUSH1 0x20 ADD PUSH32 0x4261736963000000000000000000000000000000000000000000000000000000 DUP2 MSTORE POP PUSH1 0x40 MLOAD DUP1 PUSH1 0x40 ADD PUSH1 0x40 MSTORE DUP1 PUSH1 0x3 DUP2 MSTORE PUSH1 0x20 ADD PUSH32 0x4253430000000000000000000000000000000000000000000000000000000000 DUP2 MSTORE POP DUP2 PUSH1 0x3 SWAP1 DUP1 MLOAD SWAP1 PUSH1 0x20 ADD SWAP1 PUSH3 0xCC SWAP3 SWAP2 SWAP1 PUSH3 0x389 JUMP JUMPDEST POP DUP1 PUSH1 0x4 SWAP1 DUP1 MLOAD SWAP1 PUSH1 0x20 ADD SWAP1 PUSH3 0xE5 SWAP3 SWAP2 SWAP1 PUSH3 0x389 JUMP JUMPDEST POP PUSH1 0x12 PUSH1 0x5 PUSH1 0x0 PUSH2 0x100 EXP DUP2 SLOAD DUP2 PUSH1 0xFF MUL NOT AND SWAP1 DUP4 PUSH1 0xFF AND MUL OR SWAP1 SSTORE POP POP POP PUSH3 0x116 CALLER DUP3 PUSH3 0x11D PUSH1 0x20 SHL PUSH1 0x20 SHR JUMP JUMPDEST POP PUSH3 0x438 JUMP JUMPDEST PUSH1 0x0 PUSH20 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF AND DUP3 PUSH20 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF AND EQ ISZERO PUSH3 0x1C1 JUMPI PUSH1 0x40 MLOAD PUSH32 0x8C379A0000000000000000000000000000000000000000000000000000000000 DUP2 MSTORE PUSH1 0x4 ADD DUP1 DUP1 PUSH1 0x20 ADD DUP3 DUP2 SUB DUP3 MSTORE PUSH1 0x1F DUP2 MSTORE PUSH1 0x20 ADD DUP1 PUSH32 0x45524332303A206D696E7420746F20746865207A65726F206164647265737300 DUP2 MSTORE POP PUSH1 0x20 ADD SWAP2 POP POP PUSH1 0x40 MLOAD DUP1 SWAP2 SUB SWAP1 REVERT JUMPDEST PUSH3 0x1D5 PUSH1 0x0 DUP4 DUP4 PUSH3 0x2FB PUSH1 0x20 SHL PUSH1 0x20 SHR JUMP JUMPDEST PUSH3 0x1F1 DUP2 PUSH1 0x2 SLOAD PUSH3 0x300 PUSH1 0x20 SHL PUSH3 0xF2D OR SWAP1 SWAP2 SWAP1 PUSH1 0x20 SHR JUMP JUMPDEST PUSH1 0x2 DUP2 SWAP1 SSTORE POP PUSH3 0x24F DUP2 PUSH1 0x0 DUP1 DUP6 PUSH20 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF AND PUSH20 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF AND DUP2 MSTORE PUSH1 0x20 ADD SWAP1 DUP2 MSTORE PUSH1 0x20 ADD PUSH1 0x0 KECCAK256 SLOAD PUSH3 0x300 PUSH1 0x20 SHL PUSH3 0xF2D OR SWAP1 SWAP2 SWAP1 PUSH1 0x20 SHR JUMP JUMPDEST PUSH1 0x0 DUP1 DUP5 PUSH20 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF AND PUSH20 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF AND DUP2 MSTORE PUSH1 0x20 ADD SWAP1 DUP2 MSTORE PUSH1 0x20 ADD PUSH1 0x0 KECCAK256 DUP2 SWAP1 SSTORE POP DUP2 PUSH20 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF AND PUSH1 0x0 PUSH20 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF AND PUSH32 0xDDF252AD1BE2C89B69C2B068FC378DAA952BA7F163C4A11628F55A4DF523B3EF DUP4 PUSH1 0x40 MLOAD DUP1 DUP3 DUP2 MSTORE PUSH1 0x20 ADD SWAP2 POP POP PUSH1 0x40 MLOAD DUP1 SWAP2 SUB SWAP1 LOG3 POP POP JUMP JUMPDEST POP POP POP JUMP JUMPDEST PUSH1 0x0 DUP1 DUP3 DUP5 ADD SWAP1 POP DUP4 DUP2 LT ISZERO PUSH3 0x37F JUMPI PUSH1 0x40 MLOAD PUSH32 0x8C379A0000000000000000000000000000000000000000000000000000000000 DUP2 MSTORE PUSH1 0x4 ADD DUP1 DUP1 PUSH1 0x20 ADD DUP3 DUP2 SUB DUP3 MSTORE PUSH1 0x1B DUP2 MSTORE PUSH1 0x20 ADD DUP1 PUSH32 0x536166654D6174683A206164646974696F6E206F766572666C6F770000000000 DUP2 MSTORE POP PUSH1 0x20 ADD SWAP2 POP POP PUSH1 0x40 MLOAD DUP1 SWAP2 SUB SWAP1 REVERT JUMPDEST DUP1 SWAP2 POP POP SWAP3 SWAP2 POP POP JUMP JUMPDEST DUP3 DUP1 SLOAD PUSH1 0x1 DUP2 PUSH1 0x1 AND ISZERO PUSH2 0x100 MUL SUB AND PUSH1 0x2 SWAP1 DIV SWAP1 PUSH1 0x0 MSTORE PUSH1 0x20 PUSH1 0x0 KECCAK256 SWAP1 PUSH1 0x1F ADD PUSH1 0x20 SWAP1 DIV DUP2 ADD SWAP3 DUP3 PUSH1 0x1F LT PUSH3 0x3CC JUMPI DUP1 MLOAD PUSH1 0xFF NOT AND DUP4 DUP1 ADD OR DUP6 SSTORE PUSH3 0x3FD JUMP JUMPDEST DUP3 DUP1 ADD PUSH1 0x1 ADD DUP6 SSTORE DUP3 ISZERO PUSH3 0x3FD JUMPI SWAP2 DUP3 ADD JUMPDEST DUP3 DUP2 GT ISZERO PUSH3 0x3FC JUMPI DUP3 MLOAD DUP3 SSTORE SWAP2 PUSH1 0x20 ADD SWAP2 SWAP1 PUSH1 0x1 ADD SWAP1 PUSH3 0x3DF JUMP JUMPDEST JUMPDEST POP

```
SWAP1 POP PUSH3 0x40C SWAP2 SWAP1 PUSH3 0x410 JUMP JUMPDEST POP SWAP1 JUMP
JUMPDEST PUSH3 0x435 SWAP2 SWAP1 JUMPDEST DUP1 DUP3 GT ISZERO PUSH3 0x431 JUMPI
PUSH1 0x0 DUP2 PUSH1 0x0 SWAP1 SSTORE POP PUSH1 0x1 ADD PUSH3 0x417 JUMP JUMPDEST
POP SWAP1 JUMP JUMPDEST SWAP1 JUMP JUMPDEST PUSH2 0x10F1 DUP1 PUSH3 0x448 PUSH1
0x0 CODECOPY PUSH1 0x0 RETURN INVALID PUSH1 0x80 PUSH1 0x40 MSTORE CALLVALUE DUP1
ISZERO PUSH2 0x10 JUMPI PUSH1 0x0 DUP1 REVERT JUMPDEST POP PUSH1 0x4 CALLDATASIZE
LT PUSH2 0xA9 JUMPI PUSH1 0x0 CALLDATALOAD PUSH1 0xE0 SHR DUP1 PUSH4 0x39509351
GT PUSH2 0x71 JUMPI DUP1 PUSH4 0x39509351 EQ PUSH2 0x25F JUMPI DUP1 PUSH4
0x70A08231 EQ PUSH2 0x2C5 JUMPI DUP1 PUSH4 0x95D89B41 EQ PUSH2 0x31D JUMPI DUP1
PUSH4 0xA457C2D7 EQ PUSH2 0x3A0 JUMPI DUP1 PUSH4 0xA9059CBB EQ PUSH2 0x406 JUMPI
DUP1 PUSH4 0xDD62ED3E EQ PUSH2 0x46C JUMPI PUSH2 0xA9 JUMP JUMPDEST DUP1 PUSH4
0x6FDDE03 EQ PUSH2 0xAE JUMPI DUP1 PUSH4 0x95EA7B3 EQ PUSH2 0x131 JUMPI DUP1
PUSH4 0x18160DDD EQ PUSH2 0x197 JUMPI DUP1 PUSH4 0x23B872DD EQ PUSH2 0x1B5 JUMPI
DUP1 PUSH4 0x313CE567 EQ PUSH2 0x23B JUMPI JUMPDEST PUSH1 0x0 DUP1 REVERT
JUMPDEST PUSH2 0xB6 PUSH2 0x4E4 JUMP JUMPDEST PUSH1 0x40 MLOAD DUP1 DUP1 PUSH1
0x20 ADD DUP3 DUP2 SUB DUP3 MSTORE DUP4 DUP2 DUP2 MLOAD DUP2 MSTORE PUSH1 0x20
ADD SWAP2 POP DUP1 MLOAD SWAP1 PUSH1 0x20 ADD SWAP1 DUP1 DUP4 DUP4 PUSH1 0x0
JUMPDEST DUP4 DUP2 LT ISZERO PUSH2 0xF6 JUMPI DUP1 DUP3 ADD MLOAD DUP2 DUP5 ADD
MSTORE PUSH1 0x20 DUP2 ADD SWAP1 POP PUSH2 0xDB JUMP JUMPDEST POP POP POP POP
SWAP1 POP SWAP1 DUP2 ADD SWAP1 PUSH1 0x1F AND DUP1 ISZERO PUSH2 0x123 JUMPI DUP1
DUP3 SUB DUP1 MLOAD PUSH1 0x1 DUP4 PUSH1 0x20 SUB PUSH2 0x100 EXP SUB NOT AND
DUP2 MSTORE PUSH1 0x20 ADD SWAP2 POP JUMPDEST POP SWAP3 POP POP POP PUSH1 0x40
MLOAD DUP1 SWAP2 SUB SWAP1 RETURN JUMPDEST PUSH2 0x17D PUSH1 0x4 DUP1
CALLDATASIZE SUB PUSH1 0x40 DUP2 LT ISZERO PUSH2 0x147 JUMPI PUSH1 0x0 DUP1
REVERT JUMPDEST DUP2 ADD SWAP1 DUP1 DUP1 CALLDATALOAD PUSH20
0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF AND SWAP1 PUSH1 0x20 ADD SWAP1 SWAP3
SWAP2 SWAP1 DUP1 CALLDATALOAD SWAP1 PUSH1 0x20 ADD SWAP1 SWAP3 SWAP2 SWAP1 POP
POP POP PUSH2 0x586 JUMP JUMPDEST PUSH1 0x40 MLOAD DUP1 DUP3 ISZERO ISZERO ISZERO
ISZERO DUP2 MSTORE PUSH1 0x20 ADD SWAP2 POP POP PUSH1 0x40 MLOAD DUP1 SWAP2 SUB
SWAP1 RETURN JUMPDEST PUSH2 0x19F PUSH2 0x5A4 JUMP JUMPDEST PUSH1 0x40 MLOAD DUP1
DUP3 DUP2 MSTORE PUSH1 0x20 ADD SWAP2 POP POP PUSH1 0x40 MLOAD DUP1 SWAP2 SUB
SWAP1 RETURN JUMPDEST PUSH2 0x221 PUSH1 0x4 DUP1 CALLDATASIZE SUB PUSH1 0x60 DUP2
LT ISZERO PUSH2 0x1CB JUMPI PUSH1 0x0 DUP1 REVERT JUMPDEST DUP2 ADD SWAP1 DUP1
DUP1 CALLDATALOAD PUSH20 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF AND SWAP1
PUSH1 0x20 ADD SWAP1 SWAP3 SWAP2 SWAP1 DUP1 CALLDATALOAD PUSH20
0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF AND SWAP1 PUSH1 0x20 ADD SWAP1 SWAP3
SWAP2 SWAP1 DUP1 CALLDATALOAD SWAP1 PUSH1 0x20 ADD SWAP1 SWAP3 SWAP2 SWAP1 POP
POP POP PUSH2 0x5AE JUMP JUMPDEST PUSH1 0x40 MLOAD DUP1 DUP3 ISZERO ISZERO ISZERO
ISZERO DUP2 MSTORE PUSH1 0x20 ADD SWAP2 POP POP PUSH1 0x40 MLOAD DUP1 SWAP2 SUB
SWAP1 RETURN JUMPDEST PUSH2 0x243 PUSH2 0x687 JUMP JUMPDEST PUSH1 0x40 MLOAD DUP1
DUP3 PUSH1 0xFF AND PUSH1 0xFF AND DUP2 MSTORE PUSH1 0x20 ADD SWAP2 POP POP PUSH1
0x40 MLOAD DUP1 SWAP2 SUB SWAP1 RETURN JUMPDEST PUSH2 0x2AB PUSH1 0x4 DUP1
CALLDATASIZE SUB PUSH1 0x40 DUP2 LT ISZERO PUSH2 0x275 JUMPI PUSH1 0x0 DUP1
REVERT JUMPDEST DUP2 ADD SWAP1 DUP1 DUP1 CALLDATALOAD PUSH20
0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF AND SWAP1 PUSH1 0x20 ADD SWAP1 SWAP3
SWAP2 SWAP1 DUP1 CALLDATALOAD SWAP1 PUSH1 0x20 ADD SWAP1 SWAP3 SWAP2 SWAP1 POP
POP POP PUSH2 0x69E JUMP JUMPDEST PUSH1 0x40 MLOAD DUP1 DUP3 ISZERO ISZERO ISZERO
ISZERO DUP2 MSTORE PUSH1 0x20 ADD SWAP2 POP POP PUSH1 0x40 MLOAD DUP1 SWAP2 SUB
SWAP1 RETURN JUMPDEST PUSH2 0x307 PUSH1 0x4 DUP1 CALLDATASIZE SUB PUSH1 0x20 DUP2
LT ISZERO PUSH2 0x2DB JUMPI PUSH1 0x0 DUP1 REVERT JUMPDEST DUP2 ADD SWAP1 DUP1
DUP1 CALLDATALOAD PUSH20 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF AND SWAP1
PUSH1 0x20 ADD SWAP1 SWAP3 SWAP2 SWAP1 POP POP POP PUSH2 0x751 JUMP JUMPDEST
PUSH1 0x40 MLOAD DUP1 DUP3 DUP2 MSTORE PUSH1 0x20 ADD SWAP2 POP POP PUSH1 0x40
MLOAD DUP1 SWAP2 SUB SWAP1 RETURN JUMPDEST PUSH2 0x325 PUSH2 0x799 JUMP JUMPDEST
PUSH1 0x40 MLOAD DUP1 DUP1 PUSH1 0x20 ADD DUP3 DUP2 SUB DUP3 MSTORE DUP4 DUP2
```

```
DUP2 MLOAD DUP2 MSTORE PUSH1 0x20 ADD SWAP2 POP DUP1 MLOAD SWAP1 PUSH1 0x20 ADD
SWAP1 DUP1 DUP4 DUP4 PUSH1 0x0 JUMPDEST DUP4 DUP2 LT ISZERO PUSH2 0x365 JUMPI
DUP1 DUP3 ADD MLOAD DUP2 DUP5 ADD MSTORE PUSH1 0x20 DUP2 ADD SWAP1 POP PUSH2
0x34A JUMP JUMPDEST POP POP POP POP SWAP1 POP SWAP1 DUP2 ADD SWAP1 PUSH1 0x1F AND
DUP1 ISZERO PUSH2 0x392 JUMPI DUP1 DUP3 SUB DUP1 MLOAD PUSH1 0x1 DUP4 PUSH1 0x20
SUB PUSH2 0x100 EXP SUB NOT AND DUP2 MSTORE PUSH1 0x20 ADD SWAP2 POP JUMPDEST POP
SWAP3 POP POP POP PUSH1 0x40 MLOAD DUP1 SWAP2 SUB SWAP1 RETURN JUMPDEST PUSH2
0x3EC PUSH1 0x4 DUP1 CALLDATASIZE SUB PUSH1 0x40 DUP2 LT ISZERO PUSH2 0x3B6 JUMPI
PUSH1 0x0 DUP1 REVERT JUMPDEST DUP2 ADD SWAP1 DUP1 DUP1 CALLDATALOAD PUSH20
0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF AND SWAP1 PUSH1 0x20 ADD SWAP1 SWAP3
SWAP2 SWAP1 DUP1 CALLDATALOAD SWAP1 PUSH1 0x20 ADD SWAP1 SWAP3 SWAP2 SWAP1 POP
POP POP PUSH2 0x83B JUMP JUMPDEST PUSH1 0x40 MLOAD DUP1 DUP3 ISZERO ISZERO ISZERO
ISZERO DUP2 MSTORE PUSH1 0x20 ADD SWAP2 POP POP PUSH1 0x40 MLOAD DUP1 SWAP2 SUB
SWAP1 RETURN JUMPDEST PUSH2 0x452 PUSH1 0x4 DUP1 CALLDATASIZE SUB PUSH1 0x40 DUP2
LT ISZERO PUSH2 0x41C JUMPI PUSH1 0x0 DUP1 REVERT JUMPDEST DUP2 ADD SWAP1 DUP1
DUP1 CALLDATALOAD PUSH20 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF AND SWAP1
PUSH1 0x20 ADD SWAP1 SWAP3 SWAP2 SWAP1 DUP1 CALLDATALOAD SWAP1 PUSH1 0x20 ADD
SWAP1 SWAP3 SWAP2 SWAP1 POP POP POP PUSH2 0x908 JUMP JUMPDEST PUSH1 0x40 MLOAD
DUP1 DUP3 ISZERO ISZERO ISZERO ISZERO DUP2 MSTORE PUSH1 0x20 ADD SWAP2 POP POP
PUSH1 0x40 MLOAD DUP1 SWAP2 SUB SWAP1 RETURN JUMPDEST PUSH2 0x4CE PUSH1 0x4 DUP1
CALLDATASIZE SUB PUSH1 0x40 DUP2 LT ISZERO PUSH2 0x482 JUMPI PUSH1 0x0 DUP1
REVERT JUMPDEST DUP2 ADD SWAP1 DUP1 DUP1 CALLDATALOAD PUSH20
0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF AND SWAP1 PUSH1 0x20 ADD SWAP1 SWAP3
SWAP2 SWAP1 DUP1 CALLDATALOAD PUSH20 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
AND SWAP1 PUSH1 0x20 ADD SWAP1 SWAP3 SWAP2 SWAP1 POP POP POP PUSH2 0x926 JUMP
JUMPDEST PUSH1 0x40 MLOAD DUP1 DUP3 DUP2 MSTORE PUSH1 0x20 ADD SWAP2 POP POP
PUSH1 0x40 MLOAD DUP1 SWAP2 SUB SWAP1 RETURN JUMPDEST PUSH1 0x60 PUSH1 0x3 DUP1
SLOAD PUSH1 0x1 DUP2 PUSH1 0x1 AND ISZERO PUSH2 0x100 MUL SUB AND PUSH1 0x2 SWAP1
DIV DUP1 PUSH1 0x1F ADD PUSH1 0x20 DUP1 SWAP2 DIV MUL PUSH1 0x20 ADD PUSH1 0x40
MLOAD SWAP1 DUP2 ADD PUSH1 0x40 MSTORE DUP1 SWAP3 SWAP2 SWAP1 DUP2 DUP2 MSTORE
PUSH1 0x20 ADD DUP3 DUP1 SLOAD PUSH1 0x1 DUP2 PUSH1 0x1 AND ISZERO PUSH2 0x100
MUL SUB AND PUSH1 0x2 SWAP1 DIV DUP1 ISZERO PUSH2 0x57C JUMPI DUP1 PUSH1 0x1F LT
PUSH2 0x551 JUMPI PUSH2 0x100 DUP1 DUP4 SLOAD DIV MUL DUP4 MSTORE SWAP2 PUSH1
0x20 ADD SWAP2 PUSH2 0x57C JUMP JUMPDEST DUP3 ADD SWAP2 SWAP1 PUSH1 0x0 MSTORE
PUSH1 0x20 PUSH1 0x0 KECCAK256 SWAP1 JUMPDEST DUP2 SLOAD DUP2 MSTORE SWAP1 PUSH1
0x1 ADD SWAP1 PUSH1 0x20 ADD DUP1 DUP4 GT PUSH2 0x55F JUMPI DUP3 SWAP1 SUB PUSH1
0x1F AND DUP3 ADD SWAP2 JUMPDEST POP POP POP POP POP SWAP1 POP SWAP1 JUMP
JUMPDEST PUSH1 0x0 PUSH2 0x59A PUSH2 0x593 PUSH2 0x9AD JUMP JUMPDEST DUP5 DUP5
PUSH2 0x9B5 JUMP JUMPDEST PUSH1 0x1 SWAP1 POP SWAP3 SWAP2 POP POP JUMP JUMPDEST
PUSH1 0x0 PUSH1 0x2 SLOAD SWAP1 POP SWAP1 JUMP JUMPDEST PUSH1 0x0 PUSH2 0x5BB
DUP5 DUP5 DUP5 PUSH2 0xBAC JUMP JUMPDEST PUSH2 0x67C DUP5 PUSH2 0x5C7 PUSH2 0x9AD
JUMP JUMPDEST PUSH2 0x677 DUP6 PUSH1 0x40 MLOAD DUP1 PUSH1 0x60 ADD PUSH1 0x40
MSTORE DUP1 PUSH1 0x28 DUP2 MSTORE PUSH1 0x20 ADD PUSH2 0x1026 PUSH1 0x28 SWAP2
CODECOPY PUSH1 0x1 PUSH1 0x0 DUP12 PUSH20
0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF AND PUSH20
0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF AND DUP2 MSTORE PUSH1 0x20 ADD SWAP1
DUP2 MSTORE PUSH1 0x20 ADD PUSH1 0x0 KECCAK256 PUSH1 0x0 PUSH2 0x62D PUSH2 0x9AD
JUMP JUMPDEST PUSH20 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF AND PUSH20
0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF AND DUP2 MSTORE PUSH1 0x20 ADD SWAP1
DUP2 MSTORE PUSH1 0x20 ADD PUSH1 0x0 KECCAK256 SLOAD PUSH2 0xE6D SWAP1 SWAP3
SWAP2 SWAP1 PUSH4 0xFFFFFFFF AND JUMP JUMPDEST PUSH2 0x9B5 JUMP JUMPDEST PUSH1
0x1 SWAP1 POP SWAP4 SWAP3 POP POP POP JUMP JUMPDEST PUSH1 0x0 PUSH1 0x5 PUSH1 0x0
SWAP1 SLOAD SWAP1 PUSH2 0x100 EXP SWAP1 DIV PUSH1 0xFF AND SWAP1 POP SWAP1 JUMP
JUMPDEST PUSH1 0x0 PUSH2 0x747 PUSH2 0x6AB PUSH2 0x9AD JUMP JUMPDEST DUP5 PUSH2
0x742 DUP6 PUSH1 0x1 PUSH1 0x0 PUSH2 0x6BC PUSH2 0x9AD JUMP JUMPDEST PUSH20
```

```
0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF AND PUSH20
0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF AND DUP2 MSTORE PUSH1 0x20 ADD SWAP1
DUP2 MSTORE PUSH1 0x20 ADD PUSH1 0x0 KECCAK256 PUSH1 0x0 DUP10 PUSH20
0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF AND PUSH20
0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF AND DUP2 MSTORE PUSH1 0x20 ADD SWAP1
DUP2 MSTORE PUSH1 0x20 ADD PUSH1 0x0 KECCAK256 SLOAD PUSH2 0xF2D SWAP1 SWAP2
SWAP1 PUSH4 0xFFFFFFFF AND JUMP JUMPDEST PUSH2 0x9B5 JUMP JUMPDEST PUSH1 0x1
SWAP1 POP SWAP3 SWAP2 POP POP JUMP JUMPDEST PUSH1 0x0 DUP1 PUSH1 0x0 DUP4 PUSH20
0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF AND PUSH20
0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF AND DUP2 MSTORE PUSH1 0x20 ADD SWAP1
DUP2 MSTORE PUSH1 0x20 ADD PUSH1 0x0 KECCAK256 SLOAD SWAP1 POP SWAP2 SWAP1 POP
JUMP JUMPDEST PUSH1 0x60 PUSH1 0x4 DUP1 SLOAD PUSH1 0x1 DUP2 PUSH1 0x1 AND ISZERO
PUSH2 0x100 MUL SUB AND PUSH1 0x2 SWAP1 DIV DUP1 PUSH1 0x1F ADD PUSH1 0x20 DUP1
SWAP2 DIV MUL PUSH1 0x20 ADD PUSH1 0x40 MLOAD SWAP1 DUP2 ADD PUSH1 0x40 MSTORE
DUP1 SWAP3 SWAP2 SWAP1 DUP2 DUP2 MSTORE PUSH1 0x20 ADD DUP3 DUP1 SLOAD PUSH1 0x1
DUP2 PUSH1 0x1 AND ISZERO PUSH2 0x100 MUL SUB AND PUSH1 0x2 SWAP1 DIV DUP1 ISZERO
PUSH2 0x831 JUMPI DUP1 PUSH1 0x1F LT PUSH2 0x806 JUMPI PUSH2 0x100 DUP1 DUP4
SLOAD DIV MUL DUP4 MSTORE SWAP2 PUSH1 0x20 ADD SWAP2 PUSH2 0x831 JUMP JUMPDEST
DUP3 ADD SWAP2 SWAP1 PUSH1 0x0 MSTORE PUSH1 0x20 PUSH1 0x0 KECCAK256 SWAP1
JUMPDEST DUP2 SLOAD DUP2 MSTORE SWAP1 PUSH1 0x1 ADD SWAP1 PUSH1 0x20 ADD DUP1
DUP4 GT PUSH2 0x814 JUMPI DUP3 SWAP1 SUB PUSH1 0x1F AND DUP3 ADD SWAP2 JUMPDEST
POP POP POP POP POP SWAP1 POP SWAP1 JUMP JUMPDEST PUSH1 0x0 PUSH2 0x8FE PUSH2
0x848 PUSH2 0x9AD JUMP JUMPDEST DUP5 PUSH2 0x8F9 DUP6 PUSH1 0x40 MLOAD DUP1 PUSH1
0x60 ADD PUSH1 0x40 MSTORE DUP1 PUSH1 0x25 DUP2 MSTORE PUSH1 0x20 ADD PUSH2
0x1097 PUSH1 0x25 SWAP2 CODECOPY PUSH1 0x1 PUSH1 0x0 PUSH2 0x872 PUSH2 0x9AD JUMP
JUMPDEST PUSH20 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF AND PUSH20
0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF AND DUP2 MSTORE PUSH1 0x20 ADD SWAP1
DUP2 MSTORE PUSH1 0x20 ADD PUSH1 0x0 KECCAK256 PUSH1 0x0 DUP11 PUSH20
0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF AND PUSH20
0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF AND DUP2 MSTORE PUSH1 0x20 ADD SWAP1
DUP2 MSTORE PUSH1 0x20 ADD PUSH1 0x0 KECCAK256 SLOAD PUSH2 0xE6D SWAP1 SWAP3
SWAP2 SWAP1 PUSH4 0xFFFFFFFF AND JUMP JUMPDEST PUSH2 0x9B5 JUMP JUMPDEST PUSH1
0x1 SWAP1 POP SWAP3 SWAP2 POP POP JUMP JUMPDEST PUSH1 0x0 PUSH2 0x91C PUSH2 0x915
PUSH2 0x9AD JUMP JUMPDEST DUP5 DUP5 PUSH2 0xBAC JUMP JUMPDEST PUSH1 0x1 SWAP1 POP
SWAP3 SWAP2 POP POP JUMP JUMPDEST PUSH1 0x0 PUSH1 0x1 PUSH1 0x0 DUP5 PUSH20
0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF AND PUSH20
0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF AND DUP2 MSTORE PUSH1 0x20 ADD SWAP1
DUP2 MSTORE PUSH1 0x20 ADD PUSH1 0x0 KECCAK256 PUSH1 0x0 DUP4 PUSH20
0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF AND PUSH20
0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF AND DUP2 MSTORE PUSH1 0x20 ADD SWAP1
DUP2 MSTORE PUSH1 0x20 ADD PUSH1 0x0 KECCAK256 SLOAD SWAP1 POP SWAP3 SWAP2 POP
POP JUMP JUMPDEST PUSH1 0x0 CALLER SWAP1 POP SWAP1 JUMP JUMPDEST PUSH1 0x0 PUSH20
0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF AND DUP4 PUSH20
0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF AND EQ ISZERO PUSH2 0xA3B JUMPI PUSH1
0x40 MLOAD PUSH32
0x8C379A0000000000000000000000000000000000000000000000000000000000 DUP2 MSTORE
PUSH1 0x4 ADD DUP1 DUP1 PUSH1 0x20 ADD DUP3 DUP2 SUB DUP3 MSTORE PUSH1 0x24 DUP2
MSTORE PUSH1 0x20 ADD DUP1 PUSH2 0x1073 PUSH1 0x24 SWAP2 CODECOPY PUSH1 0x40 ADD
SWAP2 POP POP PUSH1 0x40 MLOAD DUP1 SWAP2 SUB SWAP1 REVERT JUMPDEST PUSH1 0x0
PUSH20 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF AND DUP3 PUSH20
0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF AND EQ ISZERO PUSH2 0xAC1 JUMPI PUSH1
0x40 MLOAD PUSH32
0x8C379A0000000000000000000000000000000000000000000000000000000000 DUP2 MSTORE
PUSH1 0x4 ADD DUP1 DUP1 PUSH1 0x20 ADD DUP3 DUP2 SUB DUP3 MSTORE PUSH1 0x22 DUP2
MSTORE PUSH1 0x20 ADD DUP1 PUSH2 0xFDE PUSH1 0x22 SWAP2 CODECOPY PUSH1 0x40 ADD
```

```
SWAP2 POP POP PUSH1 0x40 MLOAD DUP1 SWAP2 SUB SWAP1 REVERT JUMPDEST DUP1 PUSH1
0x1 PUSH1 0x0 DUP6 PUSH20 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF AND PUSH20
0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF AND DUP2 MSTORE PUSH1 0x20 ADD SWAP1
DUP2 MSTORE PUSH1 0x20 ADD PUSH1 0x0 KECCAK256 PUSH1 0x0 DUP5 PUSH20
0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF AND PUSH20
0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF AND DUP2 MSTORE PUSH1 0x20 ADD SWAP1
DUP2 MSTORE PUSH1 0x20 ADD PUSH1 0x0 KECCAK256 DUP2 SWAP1 SSTORE POP DUP2 PUSH20
0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF AND DUP4 PUSH20
0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF AND PUSH32
0x8C5BE1E5EBEC7D5BD14F71427D1E84F3DD0314C0F7B2291E5B200AC8C7C3B925 DUP4 PUSH1
0x40 MLOAD DUP1 DUP3 DUP2 MSTORE PUSH1 0x20 ADD SWAP2 POP POP PUSH1 0x40 MLOAD
DUP1 SWAP2 SUB SWAP1 LOG3 POP POP POP JUMP JUMPDEST PUSH1 0x0 PUSH20
0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF AND DUP4 PUSH20
0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF AND EQ ISZERO PUSH2 0xC32 JUMPI PUSH1
0x40 MLOAD PUSH32
0x8C379A0000000000000000000000000000000000000000000000000000000000 DUP2 MSTORE
PUSH1 0x4 ADD DUP1 DUP1 PUSH1 0x20 ADD DUP3 DUP2 SUB DUP3 MSTORE PUSH1 0x25 DUP2
MSTORE PUSH1 0x20 ADD DUP1 PUSH2 0x104E PUSH1 0x25 SWAP2 CODECOPY PUSH1 0x40 ADD
SWAP2 POP POP PUSH1 0x40 MLOAD DUP1 SWAP2 SUB SWAP1 REVERT JUMPDEST PUSH1 0x0
PUSH20 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF AND DUP3 PUSH20
0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF AND EQ ISZERO PUSH2 0xCB8 JUMPI PUSH1
0x40 MLOAD PUSH32
0x8C379A0000000000000000000000000000000000000000000000000000000000 DUP2 MSTORE
PUSH1 0x4 ADD DUP1 DUP1 PUSH1 0x20 ADD DUP3 DUP2 SUB DUP3 MSTORE PUSH1 0x23 DUP2
MSTORE PUSH1 0x20 ADD DUP1 PUSH2 0xFBB PUSH1 0x23 SWAP2 CODECOPY PUSH1 0x40 ADD
SWAP2 POP POP PUSH1 0x40 MLOAD DUP1 SWAP2 SUB SWAP1 REVERT JUMPDEST PUSH2 0xCC3
DUP4 DUP4 DUP4 PUSH2 0xFB5 JUMP JUMPDEST PUSH2 0xD2E DUP2 PUSH1 0x40 MLOAD DUP1
PUSH1 0x60 ADD PUSH1 0x40 MSTORE DUP1 PUSH1 0x26 DUP2 MSTORE PUSH1 0x20 ADD PUSH2
0x1000 PUSH1 0x26 SWAP2 CODECOPY PUSH1 0x0 DUP1 DUP8 PUSH20
0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF AND PUSH20
0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF AND DUP2 MSTORE PUSH1 0x20 ADD SWAP1
DUP2 MSTORE PUSH1 0x20 ADD PUSH1 0x0 KECCAK256 SLOAD PUSH2 0xE6D SWAP1 SWAP3
SWAP2 SWAP1 PUSH4 0xFFFFFFFF AND JUMP JUMPDEST PUSH1 0x0 DUP1 DUP6 PUSH20
0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF AND PUSH20
0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF AND DUP2 MSTORE PUSH1 0x20 ADD SWAP1
DUP2 MSTORE PUSH1 0x20 ADD PUSH1 0x0 KECCAK256 DUP2 SWAP1 SSTORE POP PUSH2 0xDC1
DUP2 PUSH1 0x0 DUP1 DUP6 PUSH20 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF AND
PUSH20 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF AND DUP2 MSTORE PUSH1 0x20 ADD
SWAP1 DUP2 MSTORE PUSH1 0x20 ADD PUSH1 0x0 KECCAK256 SLOAD PUSH2 0xF2D SWAP1
SWAP2 SWAP1 PUSH4 0xFFFFFFFF AND JUMP JUMPDEST PUSH1 0x0 DUP1 DUP5 PUSH20
0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF AND PUSH20
0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF AND DUP2 MSTORE PUSH1 0x20 ADD SWAP1
DUP2 MSTORE PUSH1 0x20 ADD PUSH1 0x0 KECCAK256 DUP2 SWAP1 SSTORE POP DUP2 PUSH20
0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF AND DUP4 PUSH20
0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF AND PUSH32
0xDDF252AD1BE2C89B69C2B068FC378DAA952BA7F163C4A11628F55A4DF523B3EF DUP4 PUSH1
0x40 MLOAD DUP1 DUP3 DUP2 MSTORE PUSH1 0x20 ADD SWAP2 POP POP PUSH1 0x40 MLOAD
DUP1 SWAP2 SUB SWAP1 LOG3 POP POP POP JUMP JUMPDEST PUSH1 0x0 DUP4 DUP4 GT ISZERO
DUP3 SWAP1 PUSH2 0xF1A JUMPI PUSH1 0x40 MLOAD PUSH32
0x8C379A0000000000000000000000000000000000000000000000000000000000 DUP2 MSTORE
PUSH1 0x4 ADD DUP1 DUP1 PUSH1 0x20 ADD DUP3 DUP2 SUB DUP3 MSTORE DUP4 DUP2 DUP2
MLOAD DUP2 MSTORE PUSH1 0x20 ADD SWAP2 POP DUP1 MLOAD SWAP1 PUSH1 0x20 ADD SWAP1
DUP1 DUP4 DUP4 PUSH1 0x0 JUMPDEST DUP4 DUP2 LT ISZERO PUSH2 0xEDF JUMPI DUP1 DUP3
ADD MLOAD DUP2 DUP5 ADD MSTORE PUSH1 0x20 DUP2 ADD SWAP1 POP PUSH2 0xEC4 JUMP
JUMPDEST POP POP POP POP SWAP1 POP SWAP1 DUP2 ADD SWAP1 PUSH1 0x1F AND DUP1
```

ISZERO PUSH2 0xF0C JUMPI DUP1 DUP3 SUB DUP1 MLOAD PUSH1 0x1 DUP4 PUSH1 0x20 SUB PUSH2 0x100 EXP SUB NOT AND DUP2 MSTORE PUSH1 0x20 ADD SWAP2 POP JUMPDEST POP SWAP3 POP POP POP PUSH1 0x40 MLOAD DUP1 SWAP2 SUB SWAP1 REVERT JUMPDEST POP PUSH1 0x0 DUP4 DUP6 SUB SWAP1 POP DUP1 SWAP2 POP POP SWAP4 SWAP3 POP POP POP JUMP JUMPDEST PUSH1 0x0 DUP1 DUP3 DUP5 ADD SWAP1 POP DUP4 DUP2 LT ISZERO PUSH2 0xFAB JUMPI PUSH1 0x40 MLOAD PUSH32 0x8C379A000000000000000000000000000000000000000000000000000000000 DUP2 MSTORE PUSH1 0x4 ADD DUP1 DUP1 PUSH1 0x20 ADD DUP3 DUP2 SUB DUP3 MSTORE PUSH1 0x1B DUP2 MSTORE PUSH1 0x20 ADD DUP1 PUSH32 0x536166654D6174683A206164646974696F6E206F766572666C6F770000000000 DUP2 MSTORE POP PUSH1 0x20 ADD SWAP2 POP POP PUSH1 0x40 MLOAD DUP1 SWAP2 SUB SWAP1 REVERT JUMPDEST DUP1 SWAP2 POP POP SWAP3 SWAP2 POP POP JUMP JUMPDEST POP POP POP JUMP INVALID GASLIMIT MSTORE NUMBER ORIGIN ADDRESS GASPRICE KECCAK256 PUSH21 0x72616E7366657220746F20746865207A65726F2061 PUSH5 0x6472657373 GASLIMIT MSTORE NUMBER ORIGIN ADDRESS GASPRICE KECCAK256 PUSH2 0x7070 PUSH19 0x6F766520746F20746865207A65726F20616464 PUSH19 0x65737345524332303A207472616E7366657220 PUSH2 0x6D6F PUSH22 0x6E7420657863656564732062616C616E636545524332 ADDRESS GASPRICE KECCAK256 PUSH21 0x72616E7366657220616D6F756E7420657863656564 PUSH20 0x20616C6C6F77616E636545524332303A20747261 PUSH15 0x736665722066726F6D20746865207A PUSH6 0x726F20616464 PUSH19 0x65737345524332303A20617070726F76652066 PUSH19 0x6F6D20746865207A65726F2061646472657373 GASLIMIT MSTORE NUMBER ORIGIN ADDRESS GASPRICE KECCAK256 PUSH5 0x6563726561 PUSH20 0x656420616C6C6F77616E63652062656C6F77207A PUSH6 0x726FA2646970 PUSH7 0x735822122081C8 BLOCKHASH CREATE DUP8 0xCE 0xF9 0x2F 0xEC 0xCB SUB STATICCALL 0xDC PUSH8 0x8B2708C331896EC5 NUMBER 0x2B 0x5D 0x4C PUSH8 0x5F27B6D3E664736F PUSH13 0x6343000602003300000000000 ",
      "sourceMap":
"142:152:5:-:0;;;177:115;8:9:-1;5:2;;;30:1;27;20:12;5:2;177:115:5;;;;;;;;;;;;;;
13:2:-1;8:3;5:11;2:2;;;29:1;26;19:12;2:2;177:115:5;;;;;;;;;;;;;;;2013:141:2;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;2093:4;2085:5;:12;;;;;;;;;;;;;;i;;;:-;;2117:6;210
7:7;:16;;;;;;;;;;;;;;i;;;:-;;2145:2;2133:9;;:14;;;;;;;;;;;;;;;;;;2013:141;;252:
33:5::1;258:10;270:14;252:5;;;:33;;;;:i;;;:-;177:115:::0;142:152;;7835:370:2;793
7:1;7918:21;;:7;:21;;;;7910:65;;7986:49;8
015:1;8019:7;8028:6;7986:20;;;:49;;;;:i;;;:-;8061:24;8078:6;8061:12;;:16;;;;;;:2
4;;;;;;:i;;;:-;8046:12;:39;;;;8116:30;8139:6;8116:9;:18;8126:7;8116:18;;;;;;;;;
;;;;;;:22;;;;;;:30;;;;;;;:i;;;:-;8095:9;:18;8105:7;8095:18;;;;;;;;;;;;;;;:51;;;;8
182:7;8161:37;;8178:1;8161:37;;;8191:6;8161:37;;;;;;;;;;;;;;;;;;7835:370;;;;:o;1
0695:92::-;;;;;;:o;874:176:1:-;932:7;951:9;967:1;963;:5;951:17;;991:1;986;:6;;97
8:46;;1042:1;1035:8;;;874:176;;;;;;:o;142
:152:5:-;;;;;;;;;;;;;;;;;;;

"object":
"6080604052348015610010576000 80fd5b50600436106100a95760003560e01c806339509351116
1007157806339509351146102 5f57806370a08231146102c557806395d89b411461031d578063a45
7c2d7146103a0578063a9059cbb14610406578063dd62ed3e1461046c576100a9565b806306fdde0
3146100ae578063095ea7b31461013157806318160ddd146101979757806323b872dd146101b557806
3313ce5671461023b575b600080fd5b6100b66104e4565b604051808060200182810382528381815
18152602001915080519060200190808383600005b838110156100f65780820151818401526020810
190506100db565b505050509050908101906601f1680156101235780820380516001836020036101010
00a031916815260200191505b509250505060405180910390f35b61017d600480360360408110156
1014757600080fd5b81019080803573ffffffffffffffffffffffffffffffffffffffff169060200
190929190803590602001909291905050506105865b604051808215151515815260200191505060
405180910390f35b61019f6105a4565b604051808281526020019150506040518091039 0f35b610
221600480360360608110156101cb57600080fd5b81019080803573ffffffffffffffffffffffff
ffffffffffffffff169060200190929190803573ffffffffffffffffffffffffffffffffffffffff1
69060200190929190803590602001909291905050506105ae565b604051808215151515815260200
191505060405180910390f35b610243610687565b604051808260ff1660ff168152602001915050506
0405180910390f35b6102ab600480360360408110156102755760008 0fd5b81019080803573fffff
ffffffffffffffffffffffffffffffffff169060200190929190803590602001909291905050506
1069e565b604051808215151515815260200191505060405180910390f35b610307600480360360 2
08110156102db57600080fd5b81019080803573ffffffffffffffffffffffffffffffffffffffff1
69060200190929190505050610751565b604051808281526020019150506040518091039 0f35b610
325610799565b604051808060200182810382528381815181526020019150805190602001908083 8
360005b838110156103655780820151818401526020810190506103 4a565b505050509050908101 9
0601f168015610392578082038051600183602003610100 0a031916815260200191505b509250505
060405180910390f35b6103ec600480360360408110156103b657600080fd5b81019080803573fff
ffffffffffffffffffffffffffffffffffff169060200190929190803590602001909291905050505
061083b565b604051808215151515815260200191505060405180910390f35b61045260048036036
04081101561041c57600080fd5b81019080803573ffffffffffffffffffffffffffffffffffffffff
f169060200190929190803590602001909291905050506109 08565b604051808215151515815260 2
00191505060405180910390f35b6104ce600480360360408110156104825760008 0fd5b810190808
03573ffffffffffffffffffffffffffffffffffffffff169060200190929190803573ffffffffffff
ffffffffffffffffffffffffffff169060200190929190505050610926565b604051808281526 02
00191505060405180910390f35b606060038054600181600116156101000203166002900480601f0
1602080910402602001604051908101604052809291908181526020018280546001816001161561 0
1000203166002900480156105 7c5780601f106105515761010080835404028352916020019161057
c565b820191906000526020600020905b815481529060010190602001808311610 55f57829003601
f168201915b505050505090509 0565b600061059a6105936109ad565b84846109b5565b600190509
2915050565b600060025490509 0565b60006105bb848484610bac565b61067c846105c76109ad565
b610677856040518060600160 40528060288152602001611026602891396001600 08b73ffffffff
ffffffffffffffffffffffffffff1673ffffffffffffffffffffffffffffffffffffffff16815
2602001908152602001600020600061062d6109ad565b73ffffffffffffffffffffffffffffffff
ffffff1673ffffffffffffffffffffffffffffffffffffffff168152602001908152602001600 02
054610e6d9092919063ffffffff16565b6109b5565b600190509392505050565b600060056000905
4906101000a900460ff16905090565b60006107476106ab6109ad565b846107428560016000 6106b
c6109ad565b73ffffffffffffffffffffffffffffffffffffffff1673ffffffffffffffffffffff
ffffffffffffffff168152602001908152602001600020600 8973ffffffffffffffffffffffff
ffffffffffffff1673ffffffffffffffffffffffffffffffffffffffff168152602001908152602
00160002054610f2d90919063ffffffff16565b6109b5565b600190509291505056 5b60008060008
373ffffffffffffffffffffffffffffffffffffffff1673ffffffffffffffffffffffffffffffff
ffffff168152602001908152602001600020549050919050565b6060600480546001816001161 56
1010002031660029004806 01f01602080910402602001604051908101604052809291908181526 02
00182805460018160011615610100020316600290048015610831578060 1f106108065761010080 8
3540402835291602001916108315 65b820191906000526020600020905b815481529060010190602
00180831161081457829003601f168201915b505050505090509 0565b60006108fe6108486109ad5
65b846108f985604051806060016040528060258152602001611097602591396001600061087261 0

9ad565b73ffffffffffffffffffffffffffffffffffffffff1673ffffffffffffffffffffffffffff
ffffffffffffff16815260200190815260200160002060008a73ffffffffffffffffffffffffffff
ffffffffffff1673ffffffffffffffffffffffffffffffffffffffff16815260200190815260200160
0002054610e6d9092919063ffffffff16565b6109b5565b600190509291505056 5b600061091c610
9156109ad565b8484610bac565b600190509291505056 5b600600160008473ffffffffffffffff
ffffffffffffffffffffff1673ffffffffffffffffffffffffffffffffffffffff1681526020019
0815260200160002060008373ffffffffffffffffffffffffffffffffffffffff1673ffffffffff
ffffffffffffffffffffffffff16815260200190815260200160002054905092915050565b600
033905090565b600073ffffffffffffffffffffffffffffffffffffffff168373ffffffffffffff
ffffffffffffffffffffffff161415610a3b576040517f08c379a000000000000000000000000
000000000000000000000000000000815260040180806020018281038252602481526020018061 1
0736024913960400191505060405180910390fd5b600073ffffffffffffffffffffffffffffffff
fffffff168273ffffffffffffffffffffffffffffffffffffffff161415610ac1576040517f08c37
9a000000000000000000000000000000000000000000000000000000000815260040180806020018
2810382526022815260200180610fde602291396040019150506040518091039 0fd5b80600160008
573ffffffffffffffffffffffffffffffffffffffff1673ffffffffffffffffffffffffffffffff
fffffff16815260200190815260200160002060008473ffffffffffffffffffffffffffffffff
fffff1673ffffffffffffffffffffffffffffffffffffffff16815260200190815260200160000208
19055508173ffffffffffffffffffffffffffffffffffffffff168373ffffffffffffffffffffff
ffffffffffffff167f8c5be1e5ebec7d5bd14f71427d1e84f3dd0314c0f7b2291e5b200ac8c7c
3b925836040518082815260200191505060405180910390a3505050565b600073ffffffffffffff
ffffffffffffffffffffffff168373ffffffffffffffffffffffffffffffffffffffff161415610
c32576040517f08c379a000000000000000000000000000000000000000000000000000000000815
2600401808060200182810382526025815260200180611 04e6025913960400191505060405180910
390fd5b600073ffffffffffffffffffffffffffffffffffffffff168273ffffffffffffffffffff
ffffffffffffffffff161415610cb8576040517f08c379a000000000000000000000000000000
000000000000000000000000000815260040180806020018281038252602381526020018061 0fbb602
39139604001915050604051809103 90fd5b610cc3838383610fb5565b610d2e81604051806060016
0405280602681526020016110006026913960008087 73ffffffffffffffffffffffffffffffff
fffff1673ffffffffffffffffffffffffffffffffffffffff16815260200190815260200160002054
610e6d9092919063ffffffff16565b6000808573ffffffffffffffffffffffffffffffffffffff
f1673ffffffffffffffffffffffffffffffffffffffff16815260200190815260200160002081905
550610dc1816000808573ffffffffffffffffffffffffffffffffffffffff1673ffffffffffffff
ffffffffffffffffffffffff16815260200190815260200160002054610f2d90919063ffffffff1
6565b6000808473ffffffffffffffffffffffffffffffffffffffff1673ffffffffffffffffffff
ffffffffffffffffff16815260200190815260200160002081905550817 3ffffffffffffffffff
ffffffffffffffffffff168373ffffffffffffffffffffffffffffffffffffffff167fddf252ad1
be2c89b69c2b068fc378daa952ba7f163c4a11628f55a4df523b3ef836040518082815260200191 5
0506040518091039 0a3505050565b600083831115829061 0f1a576040517f08c379a000000000000
000000000000000000000000000000000000000000000815260040180806020018281038252838 18
15181526020019150805190602001908083836000 5b83811015610edf578082015181840152602 08
1019050610ec4565b505050509050908101906 01f168015610f0c578082038051600183602003610
1000a031916815260200191505b509250505060405180910390fd5b506000838503905080915050 9
392505050565b600080828401905083811015610fab576040517f08c379a00000000000000000000
000000000000000000000000000000000000000000815260040180806020018281038252601b815260200
1807f536166654d6174683a206164646974696f6e206f766572666c6f770000000000081525060200
191505060405180910390fd5b809150509291505056 5b50505056fe45524332303a207472616e736
6657220746f20746865207a65726f206164647265737345524332303a20617070726f766520746f2
0746865207a65726f206164647265737345524332303a207472616e7366657220616d6f756e74206
578636565647320626c616e636545524332303a207472616e7366657220616d6f756e742065786
36565647320616c6c6f77616e636545524332303a207472616e736665722066726f6d20746865207
a65726f206164647265737345524332303a20617070726f76652066726f6d20746865207a65726f2
0616464726573734552433230 3a206465372656173656420616c6c6f77616e63652062656c6f772
07a65726fa2646970667358221220 81c840f087cef92feccb03fadc678b2708c331896ec5432b5d4
c675f27b6d3e664736f6c63430006020033",

"opcodes": "PUSH1 0x80 PUSH1 0x40 MSTORE CALLVALUE DUP1 ISZERO PUSH2 0x10 JUMPI PUSH1 0x0 DUP1 REVERT JUMPDEST POP PUSH1 0x4 CALLDATASIZE LT PUSH2 0xA9 JUMPI PUSH1 0x0 CALLDATALOAD PUSH1 0xE0 SHR DUP1 PUSH4 0x39509351 GT PUSH2 0x71 JUMPI DUP1 PUSH4 0x39509351 EQ PUSH2 0x25F JUMPI DUP1 PUSH4 0x70A08231 EQ PUSH2 0x2C5 JUMPI DUP1 PUSH4 0x95D89B41 EQ PUSH2 0x31D JUMPI DUP1 PUSH4 0xA457C2D7 EQ PUSH2 0x3A0 JUMPI DUP1 PUSH4 0xA9059CBB EQ PUSH2 0x406 JUMPI DUP1 PUSH4 0xDD62ED3E EQ PUSH2 0x46C JUMPI PUSH2 0xA9 JUMP JUMPDEST DUP1 PUSH4 0x6FDDE03 EQ PUSH2 0xAE JUMPI DUP1 PUSH4 0x95EA7B3 EQ PUSH2 0x131 JUMPI DUP1 PUSH4 0x18160DDD EQ PUSH2 0x197 JUMPI DUP1 PUSH4 0x23B872DD EQ PUSH2 0x1B5 JUMPI DUP1 PUSH4 0x313CE567 EQ PUSH2 0x23B JUMPI JUMPDEST PUSH1 0x0 DUP1 REVERT JUMPDEST PUSH2 0xB6 PUSH2 0x4E4 JUMP JUMPDEST PUSH1 0x40 MLOAD DUP1 DUP1 PUSH1 0x20 ADD DUP3 DUP2 SUB DUP3 MSTORE DUP4 DUP2 DUP2 MLOAD DUP2 MSTORE PUSH1 0x20 ADD SWAP2 POP DUP1 MLOAD SWAP1 PUSH1 0x20 ADD SWAP1 DUP1 DUP4 DUP4 PUSH1 0x0 JUMPDEST DUP4 DUP2 LT ISZERO PUSH2 0xF6 JUMPI DUP1 DUP3 ADD MLOAD DUP2 DUP5 ADD MSTORE PUSH1 0x20 DUP2 ADD SWAP1 POP PUSH2 0xDB JUMP JUMPDEST POP POP POP POP SWAP1 POP SWAP1 DUP2 ADD SWAP1 PUSH1 0x1F AND DUP1 ISZERO PUSH2 0x123 JUMPI DUP1 DUP3 SUB DUP1 MLOAD PUSH1 0x1 DUP4 PUSH1 0x20 SUB PUSH2 0x100 EXP SUB NOT AND DUP2 MSTORE PUSH1 0x20 ADD SWAP2 POP JUMPDEST POP SWAP3 POP POP POP PUSH1 0x40 MLOAD DUP1 SWAP2 SUB SWAP1 RETURN JUMPDEST PUSH2 0x17D PUSH1 0x4 DUP1 CALLDATASIZE SUB PUSH1 0x40 DUP2 LT ISZERO PUSH2 0x147 JUMPI PUSH1 0x0 DUP1 REVERT JUMPDEST DUP2 ADD SWAP1 DUP1 DUP1 CALLDATALOAD PUSH20 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF AND SWAP1 PUSH1 0x20 ADD SWAP1 SWAP3 SWAP2 SWAP1 DUP1 CALLDATALOAD SWAP1 PUSH1 0x20 ADD SWAP1 SWAP3 SWAP2 SWAP1 POP POP POP PUSH2 0x586 JUMP JUMPDEST PUSH1 0x40 MLOAD DUP1 DUP3 ISZERO ISZERO ISZERO ISZERO DUP2 MSTORE PUSH1 0x20 ADD SWAP2 POP POP PUSH1 0x40 MLOAD DUP1 SWAP2 SUB SWAP1 RETURN JUMPDEST PUSH2 0x19F PUSH2 0x5A4 JUMP JUMPDEST PUSH1 0x40 MLOAD DUP1 DUP3 DUP2 MSTORE PUSH1 0x20 ADD SWAP2 POP POP PUSH1 0x40 MLOAD DUP1 SWAP2 SUB SWAP1 RETURN JUMPDEST PUSH2 0x221 PUSH1 0x4 DUP1 CALLDATASIZE SUB PUSH1 0x60 DUP2 LT ISZERO PUSH2 0x1CB JUMPI PUSH1 0x0 DUP1 REVERT JUMPDEST DUP2 ADD SWAP1 DUP1 DUP1 CALLDATALOAD PUSH20 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF AND SWAP1 PUSH1 0x20 ADD SWAP1 SWAP3 SWAP2 SWAP1 DUP1 CALLDATALOAD PUSH20 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF AND SWAP1 PUSH1 0x20 ADD SWAP1 SWAP3 SWAP2 SWAP1 DUP1 CALLDATALOAD SWAP1 PUSH1 0x20 ADD SWAP1 SWAP3 SWAP2 SWAP1 POP POP POP PUSH2 0x5AE JUMP JUMPDEST PUSH1 0x40 MLOAD DUP1 DUP3 ISZERO ISZERO ISZERO ISZERO DUP2 MSTORE PUSH1 0x20 ADD SWAP2 POP POP PUSH1 0x40 MLOAD DUP1 SWAP2 SUB SWAP1 RETURN JUMPDEST PUSH2 0x243 PUSH2 0x687 JUMP JUMPDEST PUSH1 0x40 MLOAD DUP1 DUP3 PUSH1 0xFF AND PUSH1 0xFF AND DUP2 MSTORE PUSH1 0x20 ADD SWAP2 POP POP PUSH1 0x40 MLOAD DUP1 SWAP2 SUB SWAP1 RETURN JUMPDEST PUSH2 0x2AB PUSH1 0x4 DUP1 CALLDATASIZE SUB PUSH1 0x40 DUP2 LT ISZERO PUSH2 0x275 JUMPI PUSH1 0x0 DUP1 REVERT JUMPDEST DUP2 ADD SWAP1 DUP1 DUP1 CALLDATALOAD PUSH20 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF AND SWAP1 PUSH1 0x20 ADD SWAP1 SWAP3 SWAP2 SWAP1 DUP1 CALLDATALOAD SWAP1 PUSH1 0x20 ADD SWAP1 SWAP3 SWAP2 SWAP1 POP POP POP PUSH2 0x69E JUMP JUMPDEST PUSH1 0x40 MLOAD DUP1 DUP3 ISZERO ISZERO ISZERO ISZERO DUP2 MSTORE PUSH1 0x20 ADD SWAP2 POP POP PUSH1 0x40 MLOAD DUP1 SWAP2 SUB SWAP1 RETURN JUMPDEST PUSH2 0x307 PUSH1 0x4 DUP1 CALLDATASIZE SUB PUSH1 0x20 DUP2 LT ISZERO PUSH2 0x2DB JUMPI PUSH1 0x0 DUP1 REVERT JUMPDEST DUP2 ADD SWAP1 DUP1 DUP1 CALLDATALOAD PUSH20 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF AND SWAP1 PUSH1 0x20 ADD SWAP1 SWAP3 SWAP2 SWAP1 POP POP POP PUSH2 0x751 JUMP JUMPDEST PUSH1 0x40 MLOAD DUP1 DUP3 DUP2 MSTORE PUSH1 0x20 ADD SWAP2 POP POP PUSH1 0x40 MLOAD DUP1 SWAP2 SUB SWAP1 RETURN JUMPDEST PUSH2 0x325 PUSH2 0x799 JUMP JUMPDEST PUSH1 0x40 MLOAD DUP1 DUP1 PUSH1 0x20 ADD DUP3 DUP2 SUB DUP3 MSTORE DUP4 DUP2 DUP2 MLOAD DUP2 MSTORE PUSH1 0x20 ADD SWAP2 POP DUP1 MLOAD SWAP1 PUSH1 0x20 ADD SWAP1 DUP1 DUP4 DUP4 PUSH1 0x0 JUMPDEST DUP4 DUP2 LT ISZERO PUSH2 0x365 JUMPI DUP1 DUP3 ADD MLOAD DUP2 DUP5 ADD MSTORE PUSH1 0x20 DUP2 ADD SWAP1 POP PUSH2 0x34A JUMP JUMPDEST POP POP POP POP SWAP1 POP SWAP1 DUP2 ADD SWAP1 PUSH1 0x1F AND DUP1 ISZERO PUSH2 0x392 JUMPI DUP1

```
DUP3 SUB DUP1 MLOAD PUSH1 0x1 DUP4 PUSH1 0x20 SUB PUSH2 0x100 EXP SUB NOT AND
DUP2 MSTORE PUSH1 0x20 ADD SWAP2 POP JUMPDEST POP SWAP3 POP POP POP PUSH1 0x40
MLOAD DUP1 SWAP2 SUB SWAP1 RETURN JUMPDEST PUSH2 0x3EC PUSH1 0x4 DUP1
CALLDATASIZE SUB PUSH1 0x40 DUP2 LT ISZERO PUSH2 0x3B6 JUMPI PUSH1 0x0 DUP1
REVERT JUMPDEST DUP2 ADD SWAP1 DUP1 DUP1 CALLDATALOAD PUSH20
0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF AND SWAP1 PUSH1 0x20 ADD SWAP1 SWAP3
SWAP2 SWAP1 DUP1 CALLDATALOAD SWAP1 PUSH1 0x20 ADD SWAP1 SWAP3 SWAP2 SWAP1 POP
POP POP PUSH2 0x83B JUMP JUMPDEST PUSH1 0x40 MLOAD DUP1 DUP3 ISZERO ISZERO ISZERO
ISZERO DUP2 MSTORE PUSH1 0x20 ADD SWAP2 POP POP PUSH1 0x40 MLOAD DUP1 SWAP2 SUB
SWAP1 RETURN JUMPDEST PUSH2 0x452 PUSH1 0x4 DUP1 CALLDATASIZE SUB PUSH1 0x40 DUP2
LT ISZERO PUSH2 0x41C JUMPI PUSH1 0x0 DUP1 REVERT JUMPDEST DUP2 ADD SWAP1 DUP1
DUP1 CALLDATALOAD PUSH20 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF AND SWAP1
PUSH1 0x20 ADD SWAP1 SWAP3 SWAP2 SWAP1 DUP1 CALLDATALOAD SWAP1 PUSH1 0x20 ADD
SWAP1 SWAP3 SWAP2 SWAP1 POP POP POP PUSH2 0x908 JUMP JUMPDEST PUSH1 0x40 MLOAD
DUP1 DUP3 ISZERO ISZERO ISZERO ISZERO DUP2 MSTORE PUSH1 0x20 ADD SWAP2 POP POP
PUSH1 0x40 MLOAD DUP1 SWAP2 SUB SWAP1 RETURN JUMPDEST PUSH2 0x4CE PUSH1 0x4 DUP1
CALLDATASIZE SUB PUSH1 0x40 DUP2 LT ISZERO PUSH2 0x482 JUMPI PUSH1 0x0 DUP1
REVERT JUMPDEST DUP2 ADD SWAP1 DUP1 DUP1 CALLDATALOAD PUSH20
0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF AND SWAP1 PUSH1 0x20 ADD SWAP1 SWAP3
SWAP2 SWAP1 DUP1 CALLDATALOAD PUSH20 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
AND SWAP1 PUSH1 0x20 ADD SWAP1 SWAP3 SWAP2 SWAP1 POP POP POP PUSH2 0x926 JUMP
JUMPDEST PUSH1 0x40 MLOAD DUP1 DUP3 DUP2 MSTORE PUSH1 0x20 ADD SWAP2 POP POP
PUSH1 0x40 MLOAD DUP1 SWAP2 SUB SWAP1 RETURN JUMPDEST PUSH1 0x60 PUSH1 0x3 DUP1
SLOAD PUSH1 0x1 DUP2 PUSH1 0x1 AND ISZERO PUSH2 0x100 MUL SUB AND PUSH1 0x2 SWAP1
DIV DUP1 PUSH1 0x1F ADD PUSH1 0x20 DUP1 SWAP2 DIV MUL PUSH1 0x20 ADD PUSH1 0x40
MLOAD SWAP1 DUP2 ADD PUSH1 0x40 MSTORE DUP1 SWAP3 SWAP2 SWAP1 DUP2 DUP2 MSTORE
PUSH1 0x20 ADD DUP3 DUP1 SLOAD PUSH1 0x1 DUP2 PUSH1 0x1 AND ISZERO PUSH2 0x100
MUL SUB AND PUSH1 0x2 SWAP1 DIV DUP1 ISZERO PUSH2 0x57C JUMPI DUP1 PUSH1 0x1F LT
PUSH2 0x551 JUMPI PUSH2 0x100 DUP1 DUP4 SLOAD DIV MUL DUP4 MSTORE SWAP2 PUSH1
0x20 ADD SWAP2 PUSH2 0x57C JUMP JUMPDEST DUP3 ADD SWAP2 SWAP1 PUSH1 0x0 MSTORE
PUSH1 0x20 PUSH1 0x0 KECCAK256 SWAP1 JUMPDEST DUP2 SLOAD DUP2 MSTORE SWAP1 PUSH1
0x1 ADD SWAP1 PUSH1 0x20 ADD DUP1 DUP4 GT PUSH2 0x55F JUMPI DUP3 SWAP1 SUB PUSH1
0x1F AND DUP3 ADD SWAP2 JUMPDEST POP POP POP POP POP SWAP1 POP SWAP1 JUMP
JUMPDEST PUSH1 0x0 PUSH2 0x59A PUSH2 0x593 PUSH2 0x9AD JUMP JUMPDEST DUP5 DUP5
PUSH2 0x9B5 JUMP JUMPDEST PUSH1 0x1 SWAP1 POP SWAP3 SWAP2 POP POP JUMP JUMPDEST
PUSH1 0x0 PUSH1 0x2 SLOAD SWAP1 POP SWAP1 JUMP JUMPDEST PUSH1 0x0 PUSH2 0x5BB
DUP5 DUP5 DUP5 PUSH2 0xBAC JUMP JUMPDEST PUSH2 0x67C DUP5 PUSH2 0x5C7 PUSH2 0x9AD
JUMP JUMPDEST PUSH2 0x677 DUP6 PUSH1 0x40 MLOAD DUP1 PUSH1 0x60 ADD PUSH1 0x40
MSTORE DUP1 PUSH1 0x28 DUP2 MSTORE PUSH1 0x20 ADD PUSH2 0x1026 PUSH1 0x28 SWAP2
CODECOPY PUSH1 0x1 PUSH1 0x0 DUP12 PUSH20
0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF AND PUSH20
0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF AND DUP2 MSTORE PUSH1 0x20 ADD SWAP1
DUP2 MSTORE PUSH1 0x20 ADD PUSH1 0x0 KECCAK256 PUSH1 0x0 PUSH2 0x62D PUSH2 0x9AD
JUMP JUMPDEST PUSH20 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF AND PUSH20
0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF AND DUP2 MSTORE PUSH1 0x20 ADD SWAP1
DUP2 MSTORE PUSH1 0x20 ADD PUSH1 0x0 KECCAK256 SLOAD PUSH2 0xE6D SWAP1 SWAP3
SWAP2 SWAP1 PUSH4 0xFFFFFFFF AND JUMP JUMPDEST PUSH2 0x9B5 JUMP JUMPDEST PUSH1
0x1 SWAP1 POP SWAP4 SWAP3 POP POP POP JUMP JUMPDEST PUSH1 0x0 PUSH1 0x5 PUSH1 0x0
SWAP1 SLOAD SWAP1 PUSH2 0x100 EXP SWAP1 DIV PUSH1 0xFF AND SWAP1 POP SWAP1 JUMP
JUMPDEST PUSH1 0x0 PUSH2 0x747 PUSH2 0x6AB PUSH2 0x9AD JUMP JUMPDEST DUP5 PUSH2
0x742 DUP6 PUSH1 0x1 PUSH1 0x0 PUSH2 0x6BC PUSH2 0x9AD JUMP JUMPDEST PUSH20
0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF AND PUSH20
0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF AND DUP2 MSTORE PUSH1 0x20 ADD SWAP1
DUP2 MSTORE PUSH1 0x20 ADD PUSH1 0x0 KECCAK256 PUSH1 0x0 DUP10 PUSH20
0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF AND PUSH20
```

0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF AND DUP2 MSTORE PUSH1 0x20 ADD SWAP1 DUP2 MSTORE PUSH1 0x20 ADD PUSH1 0x0 KECCAK256 SLOAD PUSH2 0xF2D SWAP1 SWAP2 SWAP1 PUSH4 0xFFFFFFFF AND JUMP JUMPDEST PUSH2 0x9B5 JUMP JUMPDEST PUSH1 0x1 SWAP1 POP SWAP3 SWAP2 POP POP JUMP JUMPDEST PUSH1 0x0 DUP1 PUSH1 0x0 DUP4 PUSH20 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF AND PUSH20 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF AND DUP2 MSTORE PUSH1 0x20 ADD SWAP1 DUP2 MSTORE PUSH1 0x20 ADD PUSH1 0x0 KECCAK256 SLOAD SWAP1 POP SWAP2 SWAP1 POP JUMP JUMPDEST PUSH1 0x60 PUSH1 0x4 DUP1 SLOAD PUSH1 0x1 DUP2 PUSH1 0x1 AND ISZERO PUSH2 0x100 MUL SUB AND PUSH1 0x2 SWAP1 DIV DUP1 PUSH1 0x1F ADD PUSH1 0x20 DUP1 SWAP2 DIV MUL PUSH1 0x20 ADD PUSH1 0x40 MLOAD SWAP1 DUP2 ADD PUSH1 0x40 MSTORE DUP1 SWAP3 SWAP2 SWAP1 DUP2 DUP2 MSTORE PUSH1 0x20 ADD DUP3 DUP1 SLOAD PUSH1 0x1 DUP2 PUSH1 0x1 AND ISZERO PUSH2 0x100 MUL SUB AND PUSH1 0x2 SWAP1 DIV DUP1 ISZERO PUSH2 0x831 JUMPI DUP1 PUSH1 0x1F LT PUSH2 0x806 JUMPI PUSH2 0x100 DUP1 DUP4 SLOAD DIV MUL DUP4 MSTORE SWAP2 PUSH1 0x20 ADD SWAP2 PUSH2 0x831 JUMP JUMPDEST DUP3 ADD SWAP2 SWAP1 PUSH1 0x0 MSTORE PUSH1 0x20 PUSH1 0x0 KECCAK256 SWAP1 JUMPDEST DUP2 SLOAD DUP2 MSTORE SWAP1 PUSH1 0x1 ADD SWAP1 PUSH1 0x20 ADD DUP1 DUP4 GT PUSH2 0x814 JUMPI DUP3 SWAP1 SUB PUSH1 0x1F AND DUP3 ADD SWAP2 JUMPDEST POP POP POP POP POP SWAP1 POP SWAP1 JUMP JUMPDEST PUSH1 0x0 PUSH2 0x8FE PUSH2 0x848 PUSH2 0x9AD JUMP JUMPDEST DUP5 PUSH2 0x8F9 DUP6 PUSH1 0x40 MLOAD DUP1 PUSH1 0x60 ADD PUSH1 0x40 MSTORE DUP1 PUSH1 0x25 DUP2 MSTORE PUSH1 0x20 ADD PUSH2 0x1097 PUSH1 0x25 SWAP2 CODECOPY PUSH1 0x1 PUSH1 0x0 PUSH2 0x872 PUSH2 0x9AD JUMP JUMPDEST PUSH20 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF AND PUSH20 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF AND DUP2 MSTORE PUSH1 0x20 ADD SWAP1 DUP2 MSTORE PUSH1 0x20 ADD PUSH1 0x0 KECCAK256 PUSH1 0x0 DUP11 PUSH20 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF AND PUSH20 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF AND DUP2 MSTORE PUSH1 0x20 ADD SWAP1 DUP2 MSTORE PUSH1 0x20 ADD PUSH1 0x0 KECCAK256 SLOAD PUSH2 0xE6D SWAP1 SWAP3 SWAP2 SWAP1 PUSH4 0xFFFFFFFF AND JUMP JUMPDEST PUSH2 0x9B5 JUMP JUMPDEST PUSH1 0x1 SWAP1 POP SWAP3 SWAP2 POP POP JUMP JUMPDEST PUSH1 0x0 PUSH2 0x91C PUSH2 0x915 PUSH2 0x9AD JUMP JUMPDEST DUP5 DUP5 PUSH2 0xBAC JUMP JUMPDEST PUSH1 0x1 SWAP1 POP SWAP3 SWAP2 POP POP JUMP JUMPDEST PUSH1 0x0 PUSH1 0x1 PUSH1 0x0 DUP5 PUSH20 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF AND PUSH20 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF AND DUP2 MSTORE PUSH1 0x20 ADD SWAP1 DUP2 MSTORE PUSH1 0x20 ADD PUSH1 0x0 KECCAK256 PUSH1 0x0 DUP4 PUSH20 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF AND PUSH20 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF AND DUP2 MSTORE PUSH1 0x20 ADD SWAP1 DUP2 MSTORE PUSH1 0x20 ADD PUSH1 0x0 KECCAK256 SLOAD SWAP1 POP SWAP3 SWAP2 POP POP JUMP JUMPDEST PUSH1 0x0 CALLER SWAP1 POP SWAP1 JUMP JUMPDEST PUSH1 0x0 PUSH20 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF AND DUP4 PUSH20 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF AND EQ ISZERO PUSH2 0xA3B JUMPI PUSH1 0x40 MLOAD PUSH32 0x8C379A0000000000000000000000000000000000000000000000000000000000 DUP2 MSTORE PUSH1 0x4 ADD DUP1 DUP1 PUSH1 0x20 ADD DUP3 DUP2 SUB DUP3 MSTORE PUSH1 0x24 DUP2 MSTORE PUSH1 0x20 ADD DUP1 PUSH2 0x1073 PUSH1 0x24 SWAP2 CODECOPY PUSH1 0x40 ADD SWAP2 POP POP PUSH1 0x40 MLOAD DUP1 SWAP2 SUB SWAP1 REVERT JUMPDEST PUSH1 0x0 PUSH20 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF AND DUP3 PUSH20 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF AND EQ ISZERO PUSH2 0xAC1 JUMPI PUSH1 0x40 MLOAD PUSH32 0x8C379A0000000000000000000000000000000000000000000000000000000000 DUP2 MSTORE PUSH1 0x4 ADD DUP1 DUP1 PUSH1 0x20 ADD DUP3 DUP2 SUB DUP3 MSTORE PUSH1 0x22 DUP2 MSTORE PUSH1 0x20 ADD DUP1 PUSH2 0xFDE PUSH1 0x22 SWAP2 CODECOPY PUSH1 0x40 ADD SWAP2 POP POP PUSH1 0x40 MLOAD DUP1 SWAP2 SUB SWAP1 REVERT JUMPDEST DUP1 PUSH1 0x1 PUSH1 0x0 DUP6 PUSH20 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF AND PUSH20 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF AND DUP2 MSTORE PUSH1 0x20 ADD SWAP1 DUP2 MSTORE PUSH1 0x20 ADD PUSH1 0x0 KECCAK256 PUSH1 0x0 DUP5 PUSH20

```
0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF AND PUSH20
0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF AND DUP2 MSTORE PUSH1 0x20 ADD SWAP1
DUP2 MSTORE PUSH1 0x20 ADD PUSH1 0x0 KECCAK256 DUP2 SWAP1 SSTORE POP DUP2 PUSH20
0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF AND DUP4 PUSH20
0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF AND PUSH32
0x8C5BE1E5EBEC7D5BD14F71427D1E84F3DD0314C0F7B2291E5B200AC8C7C3B925 DUP4 PUSH1
0x40 MLOAD DUP1 DUP3 DUP2 MSTORE PUSH1 0x20 ADD SWAP2 POP POP PUSH1 0x40 MLOAD
DUP1 SWAP2 SUB SWAP1 LOG3 POP POP POP JUMP JUMPDEST PUSH1 0x0 PUSH20
0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF AND DUP4 PUSH20
0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF AND EQ ISZERO PUSH2 0xC32 JUMPI PUSH1
0x40 MLOAD PUSH32
0x8C379A0000000000000000000000000000000000000000000000000000000000 DUP2 MSTORE
PUSH1 0x4 ADD DUP1 DUP1 PUSH1 0x20 ADD DUP3 DUP2 SUB DUP3 MSTORE PUSH1 0x25 DUP2
MSTORE PUSH1 0x20 ADD DUP1 PUSH2 0x104E PUSH1 0x25 SWAP2 CODECOPY PUSH1 0x40 ADD
SWAP2 POP POP PUSH1 0x40 MLOAD DUP1 SWAP2 SUB SWAP1 REVERT JUMPDEST PUSH1 0x0
PUSH20 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF AND DUP3 PUSH20
0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF AND EQ ISZERO PUSH2 0xCB8 JUMPI PUSH1
0x40 MLOAD PUSH32
0x8C379A0000000000000000000000000000000000000000000000000000000000 DUP2 MSTORE
PUSH1 0x4 ADD DUP1 DUP1 PUSH1 0x20 ADD DUP3 DUP2 SUB DUP3 MSTORE PUSH1 0x23 DUP2
MSTORE PUSH1 0x20 ADD DUP1 PUSH2 0xFBB PUSH1 0x23 SWAP2 CODECOPY PUSH1 0x40 ADD
SWAP2 POP POP PUSH1 0x40 MLOAD DUP1 SWAP2 SUB SWAP1 REVERT JUMPDEST PUSH2 0xCC3
DUP4 DUP4 DUP4 PUSH2 0xFB5 JUMP JUMPDEST PUSH2 0xD2E DUP2 PUSH1 0x40 MLOAD DUP1
PUSH1 0x60 ADD PUSH1 0x40 MSTORE DUP1 PUSH1 0x26 DUP2 MSTORE PUSH1 0x20 ADD PUSH2
0x1000 PUSH1 0x26 SWAP2 CODECOPY PUSH1 0x0 DUP1 DUP8 PUSH20
0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF AND PUSH20
0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF AND DUP2 MSTORE PUSH1 0x20 ADD SWAP1
DUP2 MSTORE PUSH1 0x20 ADD PUSH1 0x0 KECCAK256 SLOAD PUSH2 0xE6D SWAP1 SWAP3
SWAP2 SWAP1 PUSH4 0xFFFFFFFF AND JUMP JUMPDEST PUSH1 0x0 DUP1 DUP6 PUSH20
0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF AND PUSH20
0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF AND DUP2 MSTORE PUSH1 0x20 ADD SWAP1
DUP2 MSTORE PUSH1 0x20 ADD PUSH1 0x0 KECCAK256 DUP2 SWAP1 SSTORE POP PUSH2 0xDC1
DUP2 PUSH1 0x0 DUP1 DUP6 PUSH20 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF AND
PUSH20 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF AND DUP2 MSTORE PUSH1 0x20 ADD
SWAP1 DUP2 MSTORE PUSH1 0x20 ADD PUSH1 0x0 KECCAK256 SLOAD PUSH2 0xF2D SWAP1
SWAP2 SWAP1 PUSH4 0xFFFFFFFF AND JUMP JUMPDEST PUSH1 0x0 DUP1 DUP5 PUSH20
0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF AND PUSH20
0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF AND DUP2 MSTORE PUSH1 0x20 ADD SWAP1
DUP2 MSTORE PUSH1 0x20 ADD PUSH1 0x0 KECCAK256 DUP2 SWAP1 SSTORE POP DUP2 PUSH20
0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF AND DUP4 PUSH20
0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF AND PUSH32
0xDDF252AD1BE2C89B69C2B068FC378DAA952BA7F163C4A11628F55A4DF523B3EF DUP4 PUSH1
0x40 MLOAD DUP1 DUP3 DUP2 MSTORE PUSH1 0x20 ADD SWAP2 POP POP PUSH1 0x40 MLOAD
DUP1 SWAP2 SUB SWAP1 LOG3 POP POP POP JUMP JUMPDEST PUSH1 0x0 DUP4 DUP4 GT ISZERO
DUP3 SWAP1 PUSH2 0xF1A JUMPI PUSH1 0x40 MLOAD PUSH32
0x8C379A0000000000000000000000000000000000000000000000000000000000 DUP2 MSTORE
PUSH1 0x4 ADD DUP1 DUP1 PUSH1 0x20 ADD DUP3 DUP2 SUB DUP3 MSTORE DUP4 DUP2 DUP2
MLOAD DUP2 MSTORE PUSH1 0x20 ADD SWAP2 POP DUP1 MLOAD SWAP1 PUSH1 0x20 ADD SWAP1
DUP1 DUP4 DUP4 PUSH1 0x0 JUMPDEST DUP4 DUP2 LT ISZERO PUSH2 0xEDF JUMPI DUP1 DUP3
ADD MLOAD DUP2 DUP5 ADD MSTORE PUSH1 0x20 DUP2 ADD SWAP1 POP PUSH2 0xEC4 JUMP
JUMPDEST POP POP POP POP SWAP1 POP SWAP1 DUP2 ADD SWAP1 PUSH1 0x1F AND DUP1
ISZERO PUSH2 0xF0C JUMPI DUP1 DUP3 SUB DUP1 MLOAD PUSH1 0x1 DUP4 PUSH1 0x20 SUB
PUSH2 0x100 EXP SUB NOT AND DUP2 MSTORE PUSH1 0x20 ADD SWAP2 POP JUMPDEST POP
SWAP3 POP POP POP PUSH1 0x40 MLOAD DUP1 SWAP2 SUB SWAP1 REVERT JUMPDEST POP PUSH1
0x0 DUP4 DUP6 SUB SWAP1 P
```

第 4 步：编译智能合约 文档链接 {#step-4-test-your-smart-contract}

第 4.1 步：安装必要依赖项 文档链接 {#step-4-1}

我们成功编写智能合同后，就可以对其进行测试。 我们将使用 `Waffle` 来进行测试。

`Waffle` 的测试是使用 `Mocha` 和 `Chai` 一起编写的。 我们可以使用不同的测试环境，但 `Waffle` 匹配器只适用于 `Chai`。

因此，我们需要将 `Chai` 添加到我们的依赖关系中：

```
yarn add --dev mocha chai
```

第 4.2 步：创建测试文件 文档链接 {#step-4-2}

要编写测试，我们需要在测试目录中创建 `BasicToken.test.ts` 文件。

```
import { expect, use } from "chai"
import { Contract } from "ethers"
import { deployContract, MockProvider, solidity } from "ethereum-waffle"
import BasicToken from "../build/BasicToken.json"

use(solidity)

describe("BasicToken", () => {
  const [wallet, walletTo] = new MockProvider().getWallets()
  let token: Contract

  beforeEach(async () => {
    token = await deployContract(wallet, BasicToken, [1000])
  })
})
```

因此，我们使用 `Waffle` 的 `deployContract` 方法来部署我们的代币。 对于参数来说，我们应该传递 `wallet`、我们合约的编译的 json 文件和默认余额。

`Waffle` 也允许我们创建一个 `wallet`，这使得部署合约变得很容易。

您可以在这里阅读更多关于 `wallet` 的信息，并且您可以在这里阅读更多关于部署功能的信息。

让我们写一个简单的测试来检查钱包的余额。 由于我们在部署合约期间提交了 1000 的价值，所以钱包余额必须是 1000 个代币，我们可以在第一次测试中检查。

```
it("Assigns initial balance", async () => {
  expect(await token.balanceOf(wallet.address)).to.equal(1000)
})
```

要运行测试，请使用 `yarn test`

第 4.3 步：发出事件 文档链接 {#step-4-3}

在本教程中，我想向您展示 `Waffle` 最有用的匹配器，所以让我们从第一个开始。

`Waffle` 允许我们测试在哪里发出了哪些事件。

在本教程中，我将测试我们合约的 `transfer` 方法。

在这个测试中，我会用一个钱包向另一个钱包转账，然后检查 `Transfer` 事件是否被调用。

```
it("Transfer emits event", async () => {
  await expect(token.transfer(walletTo.address, 7))
    .to.emit(token, "Transfer")
    .withArgs(wallet.address, walletTo.address, 7)
})
```

另外，这个匹配器的一大优点是，通过将 `with Args` 添加到我们的测试中，我们可以检查调用该事件时使用了哪些参数。

这将可以确保我们的函数被正确地调用！

第 4.4 步：使用消息回滚 文档链接 {#step-4-4}

`Waffle` 允许我们测试使用了哪些消息对其进行回滚。

我们将使用 `revertWith` 匹配器来对它进行检查。

我们可以编写一个测试，我们将在测试中进行一笔大于我们钱包金额的转帐。 然后，我们将检查是否有交易回滚所使用的确切消息！

```
it("Can not transfer above the amount", async () => {
  await expect(token.transfer(walletTo.address, 1007)).to.be.revertedWith(
    "VM Exception while processing transaction: revert ERC20: transfer amount
exceeds balance"
  )
})
```

第 4.5 步：改变代币余额 文档链接 {#step-4-5}

`Waffle` 允许我们检查钱包余额的变化！

我们可以使用 `changeTokenBalance` 匹配器来检查余额变化或使用 `changeTokenBalances` 检查多个帐户。

匹配器可以接受 `数字`、`字符串` 和 `BigNumbers` 作为余额变化，但是地址应该指定为钱包或合约。

我们来编写下一个测试：

```
it("Send transaction changes receiver balance", async () => {
  await expect(() =>
    wallet.sendTransaction({ to: walletTo.address, gasPrice: 0, value: 200 })
  ).to.changeBalance(walletTo, 200)
})
```

以上是对单个钱包的测试。

下一个测试针对多个钱包。

```
it("Send transaction changes sender and receiver balances", async () => {
  await expect(() =>
    wallet.sendTransaction({ to: walletTo.address, gasPrice: 0, value: 200 })
  ).to.changeBalances([wallet, walletTo], [-200, 200])
})
```

该交易应作为回调（我们需要在调用之前检查余额）或作为交易响应传递。

恭喜您！ {#congratulations}

恭喜您！ 您通过这个教程，成功完成了测试。 您已经迈出了使用 Waffle 测试智能合约的第一大步。

您可以在这里找到本教程的代码。

更多关于 `Waffle` 的文档可在这里获取 。

title: "The Graph：修复Web3数据查询"
description: 区块链就像一个数据库，但是没有SQL。 所有数据就在那里，但是没有办法访问它。 让我告诉您如何使用The Graph和GraphQL解决这个问题。
author: Markus Waas
lang: zh
sidebar: true
tags:

- "solidity"
- "智能合约"
- "查询"
- "The Graph"
- "create-eth-app"
- "react"
  skill: 中级
  published: 2020-09-06
  source: soliditydeveloper.com
  sourceUrl: https://soliditydeveloper.com/thegraph

这一次，我们将更仔细地看一下 The Graph，它在去年基本上成为了开发去中心化应用的标准堆栈的一部分。 让我们先看看我们会如何用传统的方式做事...

没有 The Graph... {#without-the-graph}

为了说明起见，让我们举一个简单的例子。 我们都喜欢游戏，所以想象一个用户下注的简单游戏：

```
pragma solidity 0.7.1;

contract Game {
    uint256 totalGamesPlayerWon = 0;
    uint256 totalGamesPlayerLost = 0;
    event BetPlaced(address player, uint256 value, bool hasWon);

    function placeBet() external payable {
```

```
        bool hasWon = evaluateBetForPlayer(msg.sender);

        if (hasWon) {
            (bool success, ) = msg.sender.call{ value: msg.value * 2 }('');
            require(success, "Transfer failed");
            totalGamesPlayerWon++;
        } else {
            totalGamesPlayerLost++;
        }

        emit BetPlaced(msg.sender, msg.value, hasWon);
    }
}
```

现在假设在我们的去中心化应用中，我们想要显示总输/赢游戏数，并在有人再次玩游戏时对其进行更新。 该方法将是：

1. 获取 `totalGamesPlayerWon`。
2. 获取 `totalGamesPlayerLost`。
3. 订阅 `BetPlaced` 事件。

我们可以侦听如右侧所示的Web3 中的事件，但它需要处理相当多的情况。

```
GameContract.events.BetPlaced({
    fromBlock: 0
}, function(error, event) { console.log(event); })
.on('data', function(event) {
    // event fired
})
.on('changed', function(event) {
    // event was removed again
})
.on('error', function(error, receipt) {
    // tx rejected
});
```

现在，对于我们的简单示例来说，这在某种程度上还是不错的。 但是假设我们现在只想显示当前玩家输/赢的赌注金额。 嗯，我们运气不好，您最好部署一份新的合约来存储这些值，并将它们提取出来。 现在想象一下一个更复杂的智能合约和去中心化应用，事情可能很快就会变得一团糟。

一个不简单的查询

您可以看到这并不是最优的：

- 不适用于已经部署的合约。
- 存储这些值需要额外的 gas 成本。
- 需要另一个调用来获取以太坊节点的数据。

是不够好的

现在让我们看看更好的解决方案。

让我向您介绍一下 GraphQL {#let-me-introduce-to-you-graphql}

首先我们来谈谈 GraphQL，它最初是由 Facebook 设计和实现的。 您可能熟悉传统的 Rest API 模型。现在设想一下，您可以编写一个查询来精确查找您想要的数据：

GraphQL API与REST API

```
{
  hero {
    name

  }
}
```

```
{
  "hero": {
    "name": "Luke Skywalker"
  }
}
```

这两张图片基本上抓住了 GraphQL 的精髓。 通过右边的查询，我们可以精确地定义我们想要的数据，这样我们就可以在一个请求中得到所有的东西，而不仅仅是我们需要的东西。 GraphQL 服务器处理所有所需数据的获取，因此前端用户端使用起来非常简单。 如果您感兴趣，这是一个很好的解释，说明服务器是如何处理查询的。

现在有了这些知识，让我们最终进入区块链空间和 The Graph。

什么是 The Graph? {#what-is-the-graph}

The Graph

如何创建子图 {#how-to-create-a-subgraph}

1. 清单(subgraph.yaml)
2. 模式(schema.graphql)
3. 映射(mapping.ts)

清单(subgraph.yaml) {#manifest}

- 要为哪些智能合约建立索引（地址、网络、ABI...）
- 侦听哪些事件
- 其他要侦听的东西，如函数调用或区块
- 所调用的映射函数（参见下面的 mapping.ts）

为方便起见，您可能还想使用像 mustache 这样的模板工具。 然后您可以创建一个 subgraph.template.yaml，并根据最新的部署插入相应的地址。 有关更高级的示例设置，请参阅这个 Aave subgraph repo 示例。

```yaml
specVersion: 0.0.1
description: Placing Bets on Ethereum
repository: - GitHub link -
schema:
  file: ./schema.graphql
dataSources:
  - kind: ethereum/contract
    name: GameContract
    network: mainnet
    source:
      address: '0x2E6454...cf77eC'
      abi: GameContract
      startBlock: 6175244
    mapping:
      kind: ethereum/events
      apiVersion: 0.0.1
      language: wasm/assemblyscript
      entities:
        - GameContract
      abis:
        - name: GameContract
          file: ../build/contracts/GameContract.json
      eventHandlers:
        - event: PlacedBet(address,uint256,bool)
          handler: handleNewBet
      file: ./src/mapping.ts
```

模式(schema.graphql) {#schema}

您还可以使用实体作为类型来定义关系。 在我们的示例中，我们定义了从玩家到投注的一对多关系。 感叹号(!)

- 字节
- ID
- 字符串
- 布尔值
- 整数
- 大整数
- 大十进制数

The Graph 中的映射文件定义了将传入事件转换为实体的函数。 它是用 AssemblyScript（TypeScript 的子集）编写的。 这意味着它可以编译成 WASM (WebAssembly)，以更高效和可移植的方式执行映射。

```
type Bet @entity {
  id: ID!
  player: Player!
  playerHasWon: Boolean!
  time: Int!
}
```

```
type Player @entity {
  id: ID!
  totalPlayedCount: Int
  hasWonCount: Int
  hasLostCount: Int
  bets: [Bet]!
}
```

映射(mapping.ts) {#mapping}

您需要定义 subgraph.yaml 文件中命名的每个函数，因此在我们的例子中，我们只需要一个：handleNewBet。 我们首先尝试将发送者地址中的 Player 实体加载为 id。 如果它不存在，我们创建一个新实体并用起始值填充它。

然后我们创建一个新的 Bet 实体。 它的 id 将是 event.transaction.hash.toHex()+"-" +event.logIndex.toString()，确保值始终唯一。 仅使用哈希是不够的，因为有人可能会通过智能合约在一笔交易中多次调用 placeBet 函数。

最后，我们可以更新 Player 实体的所有数据。 数组不能直接推送，需要按如下所示进行更新。 我们使用 id 来引用投注。

完整的文档可以在这里看到：https://thegraph.com/docs/define-a-subgraph#writing-mappings。 您还可以将日志输出添加到映射文件，请参阅此处。

使用 Apollo Boost 之类的工具，您可以轻松地将 The Graph 集成到您的 React 去中心化应用（或 Apollo-Vue）中。 特别是当使用 React hooks 和 Apollo 这样的工具时，获取数据与在组件中写入单个 GraphQl 查询一样简单。

```
import { Bet, Player } from "../generated/schema"
import { PlacedBet } from "../generated/GameContract/GameContract"

export function handleNewBet(event: PlacedBet): void {
  let player = Player.load(event.transaction.from.toHex())

  if (player == null) {
    // create if doesn't exist yet
    player = new Player(event.transaction.from.toHex())
    player.bets = new Array<string>(0)
    player.totalPlayedCount = 0
    player.hasWonCount = 0
    player.hasLostCount = 0
  }

  let bet = new Bet(
    event.transaction.hash.toHex() + "-" + event.logIndex.toString()
  )
  bet.player = player.id
  bet.playerHasWon = event.params.hasWon
  bet.time = event.block.timestamp
  bet.save()

  player.totalPlayedCount++
  if (event.params.hasWon) {
    player.hasWonCount++
```

```
  } else {
    player.hasLostCount++
  }

  // update array like this
  let bets = player.bets
  bets.push(bet.id)
  player.bets = bets

  player.save()
}
```

在前端使用它 {#using-it-in-the-frontend}

现在我们可以编写一个像这样的查询。 这个查询将帮助我们获取如下数据：

```
// See all subgraphs: https://thegraph.com/explorer/
const client = new ApolloClient({
  uri: "{{ subgraphUrl }}",
})

ReactDOM.render(
  <ApolloProvider client={client}>
    <App />
  </ApolloProvider>,
  document.getElementById("root")
)
```

- 当前用户已经赢得游戏多少次
- 当前用户已经输了游戏多少次
- 他之前所有赌注的时间戳清单

```
const myGraphQlQuery = gql`
    players(where: { id: $currentUser }) {
      totalPlayedCount
      hasWonCount
      hasLostCount
      bets {
        time
      }
    }
`

const { loading, error, data } = useQuery(myGraphQlQuery)

React.useEffect(() => {
  if (!loading && !error && data) {
    console.log({ data })
  }
}, [loading, error, data])
```

Magic

最简单的方法是使用托管服务。 按照此处的说明来部署子图。

The Graph 服务器 {#the-graph-server}

Graph Explorer：托管服务 {#graph-explorer-the-hosted-service}

The Graph-Explorer

运行您自己的节点 {#running-your-own-node}

GraphQL 也支持新传入事件的流。 The Graph 还没有完全支持这一点，但很快就会发布。

去中心化的未来 {#the-decentralized-future}

然而，一个缺失的方面仍然是去中心化。 The Graph 计划最终成为一个完全去中心化的协议。

- https://thegraph.com/blog/the-graph-network-in-depth-part-1
- https://thegraph.com/blog/the-graph-network-in-depth-part-2

1. 用户将为查询支持索引器费用。
2. 索引器将权益质押图形通证(GRT)。

---

title: 通证集成检查清单
description: 与通证交互时需要考虑的事项检查清单
author: "Trailofbits"
lang: zh
sidebar: true
tags:

- "solidity"
- "智能合约"
- "安全性"
- "通证"
  skill: 中级
  published: 2020-08-13
  source: 构建安全的合约
  sourceUrl: https://github.com/crytic/building-secure-contracts/blob/master/development-guidelines/token_integration.md

---

与任意通证交互时遵循此检查清单。 确保您了解与每一项相关的风险，并证明这些规则的任何例外都是合理的。

为方便起见，所有 Slither实用程序都可以直接在通证地址上运行，例如：

使用 Slither 教程

```
slither-check-erc 0xdac17f958d2ee523a2206206994597c13d831ec7 TetherToken
```

要遵循此检查清单，您需要从 Slither 获得通证的以下输出：

```
- slither-check-erc [target] [contractName] [optional: --erc ERC_NUMBER]
- slither [target] --print human-summary
- slither [target] --print contract-summary
- slither-prop . --contract ContractName # requires configuration, and use of Echidna and Manticore
```

一般考虑因素 {#general-considerations}

- **合约包含安全审查。** 避免与缺少安全审查的合约发生互动。 检查评估的时间长度（也称为“努力程度”）、安全公司的声誉以及评估结果的数量和严重程度。
- **您已经联系了开发者。**您可能需要提醒他们的团队发生意外。 在blockchain-security上寻找适当的联系人。
- **他们有一个用于重要公告的安全邮件列表。** 当发现关键问题或升级发生时，他们的团队应该向（像您一样的）用户提供建议。

符合 ERC 标准 {#erc-conformity}

Slither 包含一个实用程序slither-check-erc，用于检查通证是否符合许多相关的 ERC 标准。 使用 slither-check-erc 来检查：

- **Transfer 和 transferFrom 返回一个布尔值。** 一些通证不会在这些函数上返回布尔值。 因此，他们在合约中的调用可能会失败。
- **如果使用了名称、小数和符号函数，就会出现。**因为这些函数在 ERC20 标准中是可选的，所以也有可能不会出现。
- **Decimals 返回 uint8。**一些通证会错误地返回 uint256。 如果是这种情况，请确保返回的值低于 255。
- **该通证缓解了已知的ERC20 条件竞争攻击。** ERC20 标准有一个已知的 ERC20 条件竞争攻击，必须加以修复，从而防止攻击者窃取通证。
- **该通证不是 ERC777 通证，在 transfer 和 transferFrom 中没有外部函数调用。** transfer 函数中的外部调用可能导致重入漏洞。

Slither 包括一个实用程序slither-prop，它可以生成单元测试和安全属性，可以发现许多常见的 ERC 缺陷。 使用 slither-prop 来检查：

- **合约通过了所有单元测试和 slither-prop 的安全属性测试。**运行生成的单元测试，然后用Echidna和Manticore检查属性。

最后，有一些特征是难以自动识别的。 手动检查这些条件：

- **Transfer 和 transferFrom 不收取费用。**通缩通证可能会导致意想不到的行为。
- **从通证中获得的潜在利息也被考虑在内。** 一些通证将利息分配给通证持有者。 如果不加以考虑，这种利息可能会被困于合约中。

合约构成 {#contract-composition}

- **合约避免了不必要的复杂性。**通证应该是一个简单的合约；一个具有复杂代码的通证需要更高的审查标准。 使用 Slither 的human-summary printer来识别复杂的代码。
- **合约使用了 SafeMath。**不使用 SafeMath 的合约需要更高的审查标准。 手动检查合约是否使用了 SafeMath。
- **合约中只有几个与通证无关的函数。**与通证无关的函数增加了合约中出现问题的可能性。 使用 Slither 的human-summary printer来广泛审查合约中使用的代码。
- **该通证只有一个地址。**具有多个余额更新入口点的通证可能会破坏基于地址的内部记账（例如，`balances[token_address][msg.sender]`可能不反映实际余额）。

用户特权 {#owner-privileges}

- **通证无法升级。**可升级的合约可能会随着时间的推移改变其规则。 使用 Slither 的 human-summary printer来确定合约是否可以升级。
- **所有者的铸币能力有限。**恶意的或受到影响的所有者可能会滥用铸币能力。 使用 Slither 的 human-summary printer来审查铸币能力，并考虑手动审查代码。
- **通证不可暂停。**恶意的或受到影响的所有者可以依靠可暂停通证捕获合约。 手动识别可暂停的代码。
- **所有者不能将合约列入黑名单。**恶意的或受到影响的所有者可以使用黑名单捕获依赖通证的合约。 手动识别黑名单特征。
- **大家都知道通证背后的团队，他们对滥用负责。**匿名开发团队，或居住在法律管辖范围外的开发团队，他们开发的合约应该需要更高的审查标准。

通证稀缺性 {#token-scarcity}

对通证稀缺问题的审查需要人工审查。 检查这些情况：

- **没有用户拥有大部分的供应。**如果少数用户拥有大部分的通证，他们可以根据通证的再分配影响操作。
- **总供应量充足。**总供应量低的通证很容易被操纵。
- **通证位于多个交易所。**如果所有的通证都在一个交易所，那么交易所的危害就会影响到依赖通证的合约。
- **用户了解大额资金或闪电贷款的相关风险。**依靠通证余额的合约必须仔细考虑到拥有大额资金的攻击者或通过闪电贷款的攻击。
- **通证不允许闪电铸币。** 闪电铸币会导致余额和总供应量的大幅波动，这就需要在通证的运作中进行严格和全面的溢出检查。

---

title: 通过Solidity智能合约转移和批准ERC-20通证
description: 如何通过Solidity语言使用智能合约与通证进行交互
author: "jdourlens"
tags:

- "智能合约"
- "通证"
- "solidity"
- "入门指南"
- "erc-20"
  skill: 中级
  lang: zh
  sidebar: true
  published: 2020-04-07
  source: 以太坊开发团队
  sourceUrl: https://ethereumdev.io/transfers-and-approval-or-erc20-tokens-from-a-solidity-smart-contract/
  address: "0x19dE91Af973F404EDF5B4c093983a7c6E3EC8ccE"

在之前的教程中，我们学习了对以太坊区块链上基于 Solidity 的 ERC-20 通证的剖析。 在本文中，我们将看到如何借助 Solidity 语言使用智能合约来与通证进行交互。

对于此智能合约，我们将创建一个真正虚拟的去中心化交易所，用户可以在这里使用我们新部署的ERC-20 通证进行以太坊交易。

对于此教程，我们将使用我们在上一个教程中编写的代码作为基础。 我们的 DEX 将在它的构造函数中实例化一个合约中的实例，并进行以下操作：

- 将通证换成 ETH
- 把 ETH 换成通证

我们将通过添加简单的 ERC20 代码库来开启去中心化交易所代码：

```
pragma solidity ^0.8.0;

interface IERC20 {

    function totalSupply() external view returns (uint256);
    function balanceOf(address account) external view returns (uint256);
    function allowance(address owner, address spender) external view returns
(uint256);

    function transfer(address recipient, uint256 amount) external returns
(bool);
    function approve(address spender, uint256 amount) external returns (bool);
    function transferFrom(address sender, address recipient, uint256 amount)
external returns (bool);


    event Transfer(address indexed from, address indexed to, uint256 value);
    event Approval(address indexed owner, address indexed spender, uint256
value);
}


contract ERC20Basic is IERC20 {

    string public constant name = "ERC20Basic";
    string public constant symbol = "ERC";
    uint8 public constant decimals = 18;


    mapping(address => uint256) balances;

    mapping(address => mapping (address => uint256)) allowed;

    uint256 totalSupply_ = 10 ether;


   constructor() {
    balances[msg.sender] = totalSupply_;
    }

    function totalSupply() public override view returns (uint256) {
    return totalSupply_;
    }
```

```
    function balanceOf(address tokenOwner) public override view returns
(uint256) {
        return balances[tokenOwner];
    }

    function transfer(address receiver, uint256 numTokens) public override
returns (bool) {
        require(numTokens <= balances[msg.sender]);
        balances[msg.sender] = balances[msg.sender]-numTokens;
        balances[receiver] = balances[receiver]+numTokens;
        emit Transfer(msg.sender, receiver, numTokens);
        return true;
    }

    function approve(address delegate, uint256 numTokens) public override
returns (bool) {
        allowed[msg.sender][delegate] = numTokens;
        emit Approval(msg.sender, delegate, numTokens);
        return true;
    }

    function allowance(address owner, address delegate) public override view
returns (uint) {
        return allowed[owner][delegate];
    }

    function transferFrom(address owner, address buyer, uint256 numTokens)
public override returns (bool) {
        require(numTokens <= balances[owner]);
        require(numTokens <= allowed[owner][msg.sender]);

        balances[owner] = balances[owner]-numTokens;
        allowed[owner][msg.sender] = allowed[owner][msg.sender]-numTokens;
        balances[buyer] = balances[buyer]+numTokens;
        emit Transfer(owner, buyer, numTokens);
        return true;
    }
}
```

我们新的 DEX 智能合约将部署 ERC-20 并生成总量供应：

```
contract DEX {

    IERC20 public token;

    event Bought(uint256 amount);
    event Sold(uint256 amount);

    constructor() {
        token = new ERC20Basic();
    }

    function buy() payable public {
```

```
        // TODO
    }

    function sell(uint256 amount) public {
        // TODO
    }

}
```

所以我们现在有了我们的 DEX，它所有的代币储备都已准备好。 合约有两个函数：

- `buy`：用户可以发送 ETH 并在交易所中获得通证
- `sell`：用户可以决定发送通证换回 ETH

buy 函数 {#the-buy-function}

让我们编写 buy 函数的代码。 我们首先需要检查消息中包含的 ETH 数量，并验证合约中是否拥有足够的通证，以及消息中是否有一些 ETH。 如果合约拥有足够的通证，它将会向用户发送通证数量，并发出 `Bought` 事件。

请注意，如果我们在出错的情况下调用所需的函数，则发送的 ETH 将会直接还原并退回给用户。

为简单起见，我们只需将 1 个代币换成 1 个 Wei。

```
function buy() payable public {
    uint256 amountTobuy = msg.value;
    uint256 dexBalance = token.balanceOf(address(this));
    require(amountTobuy > 0, "You need to send some ether");
    require(amountTobuy <= dexBalance, "Not enough tokens in the reserve");
    token.transfer(msg.sender, amountTobuy);
    emit Bought(amountTobuy);
}

Text
XPath: /pre[3]/code
```

在购买成功的情况下，我们应会在交易中看到两个事件：通证 `Transfer` 和 `Bought` 事件。

交易中的两个事件：Transfer和Bought

sell 函数 {#the-sell-function}

负责卖出的函数将首先要求用户事先通过调用 approve 函数来批准金额。 批准转账 ERC20Basic 代币需要 instantiated 被 DEX 的用户调用。 这可以通过首先调用 DEX 合约的 `token()` 函数来检索 DEX 部署名为 `token` 的 ERC20Basic 合约的地址来实现。 然后我们在会话中创建该合约的一个实例并调用它的 `approve` 函数。 然后我们可以调用 DEX 的 `sell` 函数并将我们的代币换回以太币。 例如，这是看起来如何在交互的 brownie session:

```
#### Python in interactive brownie console...

# 部署 DEX
dex = DEX.deploy({'from':account1})
```

```
# 调用buy函数将ETH换成代币
# 1e18 是 1 个以 wei 计价的ETH
dex.buy({'from': account2, 1e18})

# 获取 ERC20 代币的部署地址
# 在 DEX 合约创建期间部署的
# dex.token() 返回token的部署地址
token = ERC20Basic.at(dex.token())

# 调用token的approve函数
# 批准dex地址为spender
# 以及允许花费你的多少代币
token.approve(dex.address, 3e18, {'from':account2})
```

然后当调用 sell 函数时，我们会检查从调用者地址到合约地址的转账是否成功，然后将 ETH 发送回调用者地址。

```
function sell(uint256 amount) public {
    require(amount > 0, "You need to sell at least some tokens");
    uint256 allowance = token.allowance(msg.sender, address(this));
    require(allowance >= amount, "Check the token allowance");
    token.transferFrom(msg.sender, address(this), amount);
    payable(msg.sender).transfer(amount);
    emit Sold(amount);
}
```

如果一切正常，您应会在交易中看到两个事件（`Transfer`和`Sold`），并且通证余额和以太坊余额已更新。

交易中的两个事件：Transfer和Sold

从这个教程中，我们看到了如何检查 ERC-20 通证的余额和余量，以及如何使用接口调用 ERC20 智能合约的`Transfer`和`TransferFrom`。

一旦您进行交易，我们有一个 JavaScript 教程来等待并获取有关对您的合约进行的交易的详细信息，以及一个教程来解码由代币转账或任何其他事件生成的事件，只要您有 ABI。

下面是教程的完整代码：

```
pragma solidity ^0.8.0;

interface IERC20 {

    function totalSupply() external view returns (uint256);
    function balanceOf(address account) external view returns (uint256);
    function allowance(address owner, address spender) external view returns
(uint256);

    function transfer(address recipient, uint256 amount) external returns
(bool);
    function approve(address spender, uint256 amount) external returns (bool);
    function transferFrom(address sender, address recipient, uint256 amount)
external returns (bool);
```

```
    event Transfer(address indexed from, address indexed to, uint256 value);
    event Approval(address indexed owner, address indexed spender, uint256
value);
}


contract ERC20Basic is IERC20 {

    string public constant name = "ERC20Basic";
    string public constant symbol = "ERC";
    uint8 public constant decimals = 18;


    mapping(address => uint256) balances;

    mapping(address => mapping (address => uint256)) allowed;

    uint256 totalSupply_ = 10 ether;


   constructor() {
    balances[msg.sender] = totalSupply_;
    }

    function totalSupply() public override view returns (uint256) {
    return totalSupply_;
    }

    function balanceOf(address tokenOwner) public override view returns
(uint256) {
        return balances[tokenOwner];
    }

    function transfer(address receiver, uint256 numTokens) public override
returns (bool) {
        require(numTokens <= balances[msg.sender]);
        balances[msg.sender] = balances[msg.sender]-numTokens;
        balances[receiver] = balances[receiver]+numTokens;
        emit Transfer(msg.sender, receiver, numTokens);
        return true;
    }

    function approve(address delegate, uint256 numTokens) public override
returns (bool) {
        allowed[msg.sender][delegate] = numTokens;
        emit Approval(msg.sender, delegate, numTokens);
        return true;
    }

    function allowance(address owner, address delegate) public override view
returns (uint) {
        return allowed[owner][delegate];
    }
```

```
    function transferFrom(address owner, address buyer, uint256 numTokens)
public override returns (bool) {
        require(numTokens <= balances[owner]);
        require(numTokens <= allowed[owner][msg.sender]);

        balances[owner] = balances[owner]-numTokens;
        allowed[owner][msg.sender] = allowed[owner][msg.sender]-numTokens;
        balances[buyer] = balances[buyer]+numTokens;
        emit Transfer(owner, buyer, numTokens);
        return true;
    }
}


contract DEX {

    event Bought(uint256 amount);
    event Sold(uint256 amount);


    IERC20 public token;

    constructor() {
        token = new ERC20Basic();
    }

    function buy() payable public {
        uint256 amountTobuy = msg.value;
        uint256 dexBalance = token.balanceOf(address(this));
        require(amountTobuy > 0, "You need to send some ether");
        require(amountTobuy <= dexBalance, "Not enough tokens in the reserve");
        token.transfer(msg.sender, amountTobuy);
        emit Bought(amountTobuy);
    }

    function sell(uint256 amount) public {
        require(amount > 0, "You need to sell at least some tokens");
        uint256 allowance = token.allowance(msg.sender, address(this));
        require(allowance >= amount, "Check the token allowance");
        token.transferFrom(msg.sender, address(this), amount);
        payable(msg.sender).transfer(amount);
        emit Sold(amount);
    }

}
```

---

title: 了解ERC-20通证智能合约
description: 介绍如何在以太坊测试网络上部署您第一个智能合约
author: "jdourlens"
tags:

- "智能合约"
- "通证"
- "solidity"

- "入门指南"
- "erc-20"
  skill: 初学者
  lang: zh
  sidebar: true
  published: 2020-04-05
  source: EthereumDev
  sourceUrl: https://ethereumdev.io/understand-the-erc20-token-smart-contract/
  address: "0x19dE91Af973F404EDF5B4c093983a7c6E3EC8ccE"

ERC-20是以太坊上最重要的智能合约标准之一。它已经成为以太坊区块链上用于可替换通证实现的所有智能合约的技术标准。

ERC-20 定义了所有可替换的以太坊通证都应该遵守的通用规则列表。 因此，该通证标准使所有类型的开发者能够准确预测新通证在更大的以太坊系统中将如何工作。 这简化了开发者的任务，因为他们可以继续他们的工作，知道只要通证遵循规则，每次发布新的通证时就不需要重做每个新项目。

这里以接口的形式介绍了 ERC-20 必须实现的函数。 如果您不知道什么是接口：请查看我们关于使用 Solidity 进行 OOP 编程的文章。

```
pragma solidity ^0.6.0;

interface IERC20 {

    function totalSupply() external view returns (uint256);
    function balanceOf(address account) external view returns (uint256);
    function allowance(address owner, address spender) external view returns
(uint256);

    function transfer(address recipient, uint256 amount) external returns
(bool);
    function approve(address spender, uint256 amount) external returns (bool);
    function transferFrom(address sender, address recipient, uint256 amount)
external returns (bool);


    event Transfer(address indexed from, address indexed to, uint256 value);
    event Approval(address indexed owner, address indexed spender, uint256
value);
}
```

下面逐行解释每个函数的用途。 在这之后，我们将展示一个 ERC-20 通证的简单实现。

取值器 {#getters}

```
function totalSupply() external view returns (uint256);
```

返回存在的通证数量。 此函数是一个取值器，不会修改合约的状态。 请记住，Solidity 中没有浮点数。 因此，大多数通证都会采用 18 位小数，并且会返回总供应量和其他结果，如下所示：1 个通证 100000000000000000。 您需要在处理通证时格外注意，并不是每个通证都有 18 位小数。

```
function balanceOf(address account) external view returns (uint256);
```

返回地址拥有的通证数量(`account`)。 此函数是一个取值器，不会修改合约的状态。

```
function allowance(address owner, address spender) external view returns
(uint256);
```

ERC-20 标准使一个地址能够允许另一个地址从中检索通证。 此取值器返回允许`spender`代表`owner`花费的剩余通证数量。 此函数是一个取值器，不会修改合约的状态，并且默认应返回 0。

函数 {#functions}

```
function transfer(address recipient, uint256 amount) external returns (bool);
```

将通证的`amount`从函数调用者地址(`msg.sender`)移动到接收者地址。 此函数发出稍后定义的`Transfer`事件。 如果可进行转账，它将返回 true。

```
function approve(address spender, uint256 amount) external returns (bool);
```

设置允许`spender`从函数调用方(`msg.sender`)余额转账的`allowance`的数额。 此函数发出 Approval 事件。 此函数返回是否成功设置了余量。

```
function transferFrom(address sender, address recipient, uint256 amount)
external returns (bool);
```

使用余量机制将通证的`amount`从`sender`移动到`recipient`。 然后从调用者的余量中扣除该数额。 此函数发出`Transfer`事件。

事件 {#events}

```
event Transfer(address indexed from, address indexed to, uint256 value);
```

将通证（值）的数量从`from`地址发送到`to`地址时会发出此事件。

在铸造新代币的情况下，转账通常是`from` 0x00..0000 地址，而在销毁代币的情况下，转账是`to` 0x00..0000。

```
event Approval(address indexed owner, address indexed spender, uint256 value);
```

当`owner`批准要由`spender`使用的通证数量(`value`)时，将发出此事件。

ERC-20 通证的基本实现 {#a-basic-implementation-of-erc-20-tokens}

下面是 ERC-20 通证的最简单代码：

```
pragma solidity ^0.8.0;

interface IERC20 {
```

```
    function totalSupply() external view returns (uint256);
    function balanceOf(address account) external view returns (uint256);
    function allowance(address owner, address spender) external view returns
(uint256);

    function transfer(address recipient, uint256 amount) external returns
(bool);
    function approve(address spender, uint256 amount) external returns (bool);
    function transferFrom(address sender, address recipient, uint256 amount)
external returns (bool);


    event Transfer(address indexed from, address indexed to, uint256 value);
    event Approval(address indexed owner, address indexed spender, uint256
value);
}


contract ERC20Basic is IERC20 {

    string public constant name = "ERC20Basic";
    string public constant symbol = "ERC";
    uint8 public constant decimals = 18;


    mapping(address => uint256) balances;

    mapping(address => mapping (address => uint256)) allowed;

    uint256 totalSupply_ = 10 ether;


   constructor() {
    balances[msg.sender] = totalSupply_;
    }

    function totalSupply() public override view returns (uint256) {
    return totalSupply_;
    }

    function balanceOf(address tokenOwner) public override view returns
(uint256) {
        return balances[tokenOwner];
    }

    function transfer(address receiver, uint256 numTokens) public override
returns (bool) {
        require(numTokens <= balances[msg.sender]);
        balances[msg.sender] = balances[msg.sender]-numTokens;
        balances[receiver] = balances[receiver]+numTokens;
        emit Transfer(msg.sender, receiver, numTokens);
        return true;
    }
```

```
    function approve(address delegate, uint256 numTokens) public override
returns (bool) {
        allowed[msg.sender][delegate] = numTokens;
        emit Approval(msg.sender, delegate, numTokens);
        return true;
    }

    function allowance(address owner, address delegate) public override view
returns (uint) {
        return allowed[owner][delegate];
    }

    function transferFrom(address owner, address buyer, uint256 numTokens)
public override returns (bool) {
        require(numTokens <= balances[owner]);
        require(numTokens <= allowed[owner][msg.sender]);

        balances[owner] = balances[owner]-numTokens;
        allowed[owner][msg.sender] = allowed[owner][msg.sender]-numTokens;
        balances[buyer] = balances[buyer]+numTokens;
        emit Transfer(owner, buyer, numTokens);
        return true;
    }
}
```

ERC-20 通证标准的另一个优秀实现是[OpenZepelin ERC-20 实现](#)。

title: "Uniswap-v2 合约概览"
description: Uniswap-v2 合约是如何工作的？ 为什么要如此编写？
author: Ori Pomerantz
sidebar: true
tags:

- "solidity"
- "uniswap"
  skill: 中级
  published: 2021-05-01
  lang: zh

简介 {#introduction}

Uniswap v2 可以在任何两个 ERC-20 代币之间创建一个兑换市场。 在这篇文章中， 我们将了解实现此协议的合约的源代码，看看为什么要 这样写代码。

Uniswap 是做什么的？ {#what-does-uniswap-do}

一般来说有两类用户：流动资金提供者和交易者。

*流动资金提供者*为资金池提供两种可以兑换的代币（我们称之为 **Token0** 和 **Token1**）。 作为回报，他们会收到第三种代币，代表对资金池的 部分所有权，这个池叫做*流动代币*。

*交易者*将一种代币发送到资金池，并从流动资金提供者的资金池中接收另一种代币（例如，发送 **Token0** 并获得 **Token1**）。 兑换汇率由 **Token0** 和 **Token1** 的相对数量决定。 此外，资金池将收取汇率的一小部分作为流动资金池的奖励。

当流动资金提供者想要收回他们的代币资产时，他们可以消耗资金池代币并收回他们的代币， 其中包括他们在兑换过程中奖励的份额。

点击这里查看更完整的描述。

为什么选择 v2？ 而不是 v3？ {#why-v2}

编写此教程时，Uniswap v3 已差不多准备就绪。 然而，此次升级 比原来的版本复杂得多。 比较容易的方法是先学习 v2，然后再学习 v3。

核心合约与外围合约 {#contract-types}

Uniswap v2 可以分为两个部分，一个为核心部分，另一个为外围部分。 这种分法可以使拥有资产因而*必须*确保安全的核心合约更加简洁，且更易于审核。 而所有交易者需要的其它功能可以通过外围合约提供。

数据和控制流程 {#flows}

执行 Uniswap 的三个主要操作时，会出现以下数据和控制流程：

1. 兑换不同代币
2. 将资金添加到市场中提供流动性，并获得兑换中奖励的流动池 ERC-20 代币
3. 消耗流动池 ERC-20 代币并收回交易所允许交易者兑换的 ERC-20 代币

兑换 {#swap-flow}

这是交易者最常用的流程：

调用者 {#caller}

1. 向外围帐户提供兑换额度。
2. 调用外围合约中的一个兑换函数。外围合约通常会有多种兑换函数，调用哪一个取决于是否涉及以太币、 交易者是否需要指定存入的代币金额，或指定提取的代币数量等）。 每个兑换函数都接受一个 `path`，即要执行的一系列兑换。

在外围合约 (UniswapV2Router02.sol) 中 {#in-the-periphery-contract-uniswapv2router02-sol}

3. 确定兑换路径中，每次兑换所需交易的代币数额。
4. 沿路径迭代。 对于路径上的每次兑换，首先发送输入代币，然后调用交易所的 `swap` 函数。 在大多数情况下，代币输出的目的地址是路径中下一个配对交易。 在最终的兑换中，该地址是 交易者提供的地址。

在核心合约 (UniswapV2Pair.sol) 中 {#in-the-core-contract-uniswapv2pairsol-2}

5. 验证核心合约没有被欺骗，可在兑换后保持足够的流动资金。
6. 检查除了现有的储备金额外，还有多少额外的代币。 此数额是我们收到的要用于兑换的输入代币数量。
7. 将输出代币发送到目的地址。
8. 调用 `_update` 来更新储备金额

回到外围合约 (UniswapV2Router02.sol) {#back-in-the-periphery-contract-uniswapv2router02-sol}

9. 执行所需的必要清理工作（例如，消耗包装以太币代币以返回以太币给交易者）

增加流动资金 {#add-liquidity-flow}

调用者 {#caller-2}

1. 向外围账户提交准备加入流动资金池的资金额度。
2. 调用外围合约的一个 addLiquidity 函数。

在外围合约 (UniswapV2Router02.sol) 中 {#in-the-periphery-contract-uniswapv2router02sol-2}

3. 必要时创建一个新的配对交易
4. 如果存在现有配对交易，请计算要增加的代币数量。 两个代币应该有相同值，所以新代币与现有代币的比率是相同的。
5. 检查金额是否可接受（调用者可以指定一个最低金额，低于此金额他们不能增加流动资金）
6. 调用核心合约。

在核心合约 (UniswapV2Pair.sol) 中 {#in-the-core-contract-uniswapv2pairsol-2}

7. 生成流动池代币并将其发送给调用者
8. 调用 `_update` 来更新储备金额

撤回流动资金 {#remove-liquidity-flow}

调用者 {#caller-3}

1. 向外围帐户提供一个流动池代币的额度，作为兑换底层代币所需的消耗。
2. 调用外围合约的一个 removeLiquidity 函数。

在外围合约 (UniswapV2Router02.sol) 中 {#in-the-periphery-contract-uniswapv2router02sol-3}

3. 将流动池代币发送到该配对交易

在核心合约 (UniswapV2Pair.sol) 中 {#in-the-core-contract-uniswapv2pairsol-3}

4. 按照消耗代币的比例发送兑换后的代币到目标地址。 例如，如果 流动池里有 1000 个 A 代币，500 个 B 代币和 90 个流动池代币，而我们被要求消耗 9 个 流动池代币，那么，我们将消耗 10% 的流动池代币，然后将返还用户 100 个 A 代币和 50 个 B 代币。
5. 消耗流动池代币
6. 调用`_update`来更新储备金额

核心合约 {#core-contracts}

这些是持有流动资金的安全合约。

UniswapV2Pair.sol {#UniswapV2Pair}

本合约 实现了 用于兑换代币的实际资金池。 这是 Uniswap 的核心功能。

```
pragma solidity =0.5.16;

import './interfaces/IUniswapV2Pair.sol';
import './UniswapV2ERC20.sol';
import './libraries/Math.sol';
import './libraries/UQ112x112.sol';
import './interfaces/IERC20.sol';
import './interfaces/IUniswapV2Factory.sol';
import './interfaces/IUniswapV2Callee.sol';
```

这些都是合约需要知道的接口，因为合约实现了它们 （`IUniswapV2Pair` 和 `UniswapV2ERC20`），或因为合约调用了实现它们的合约。

```
contract UniswapV2Pair is IUniswapV2Pair, UniswapV2ERC20 {
```

此合约继承自 `UniswapV2ERC20`，为流动池代币提供 ERC-20 代币功能。

```
    using SafeMath  for uint;
```

SafeMath 库用于避免整数上溢和 下溢。 这很重要，否则最终可能会出现这样的情况：本该是 `-1` 的值， 结果却成了 `2^256-1`。

```
    using UQ112x112 for uint224;
```

流动池合约中的许多计算都需要分数。 但是，以太坊虚拟机本身不支持分数。 Uniswap 找到的解决方案是使用 224 位数值，整数值为 112 位，分数部分 为 112 位。 因此，`1.0` 用 `2^112` 表示，`1.5` 用 `2^112 + 2^111` 表示，以此类推。

关于这个函数库的更详细内容在文档的稍后部分。

变量 {#pair-vars}

```
    uint public constant MINIMUM_LIQUIDITY = 10**3;
```

为了避免分母为零的情况，最低数量的流动池代币总是存在的 （但为账户零所拥有）。 该数字，即 **MINIMUM_LIQUIDITY**，为 1000。

```
    bytes4 private constant SELECTOR =
bytes4(keccak256(bytes('transfer(address,uint256)')));
```

这是 ERC-20 传输函数的应用程序二进制接口选择程序。 它用于在两个代币账户中转移 ERC-20 代币。

```
    address public factory;
```

这就是由工厂合约创造的资金池地址。 每个资金池都是两个 ERC-20 代币之间的交换， 工厂是连接所有这些代币资金池的中心点。

```
    address public token0;
    address public token1;
```

这两个地址是流动池可以兑换的 两类 ERC-20 代币的合约地址。

```
    uint112 private reserve0;           // uses single storage slot, accessible
via getReserves
    uint112 private reserve1;           // uses single storage slot, accessible
via getReserves
```

每个代币类型都有储备的资源库。 我们假定两者代表相同数量的值， 因此每个 token0 的价值都等同于 reserve1/reserve0 token1。

```
    uint32  private blockTimestampLast; // uses single storage slot, accessible
via getReserves
```

发生兑换的最后一个区块的时间戳，用来追踪一段时间内的汇率。

以太坊合约的最大燃料费用之一是储存，储存的燃料消耗从合约的一个调用一直持续到另一个。 每个储存单元长度为 256 比特。 因此，reserve0、reserve1 和 blockTimestampLast 三个变量被分配在这样的 单个存储值可以包含所有三个 (112+112+32=256) 的方式。

```
    uint public price0CumulativeLast;
    uint public price1CumulativeLast;
```

这些变量包含了每个代币的累计成本（每个代币以另一个代币计算）。 可以用来计算 一段时间内的平均汇率。

```
    uint public kLast; // reserve0 * reserve1, as of immediately after the most
recent liquidity event
```

在配对交易中，决定 token0 和 token1 之间汇率的方式是在交易中 保留两个储备常量的乘数。 即 `kLast` 这个值。 当流动资金提供者存入或提取代币时，它就会发生变化，由于兑换市场的费用为 0.3%，它会略有增加。

下面是一个示例。 请注意，为了简单起见，表格中的数字仅保留了小数点后三位，我们忽略了 0.3% 交易费，因此数字并不准确。

事件	**reserve0**	**reserve1**	**reserve0 * reserve1**	**平均汇率 (token1 / token0)**
初始设置	1,000.000	1,000.000	1,000,000	
交易者 A 用 50 个 token0 兑换 47.619 个 token1	1,050.000	952.381	1,000,000	0.952
交易者 B 用 10 个 token0 兑换 8.984 个 token1	1,060.000	943.396	1,000,000	0.898
交易者 C 用 40 个 token0 兑换 34.305 个 token1	1,100.000	909.090	1,000,000	0.858

事件	reserve0	reserve1	reserve0 * reserve1	平均汇率 (token1 / token0)
交易者 D 用 100 个 token1 兑换 100.01 个 token0	990.990	1,009.090	1,000,000	0.917
交易者 E 用 10 个 token0 兑换 10.079 个 token1	1,000.990	999.010	1,000,000	1.008

由于交易者提供了更多 token0，token1 的相对价值增加了，反之亦然，这取决于供求。

锁定 {#pair-lock}

```
    uint private unlocked = 1;
```

有一类基于 重入攻击的安全问题。 Uniswap 需要转让不同数值的 ERC-20 代币，这意味着调用的 ERC-20 合约可能会导致调用合约的 Uniswap 市场遭受攻击。 使用 `unlocked` 变量， 我们可以防止函数在运行时被调用(在相同的交易内)。

```
    modifier lock() {
```

此函数是一个 modifier 函数，用于以某种方式改变正常函数的行为。

```
        require(unlocked == 1, 'UniswapV2: LOCKED');
        unlocked = 0;
```

如果 `unlocked` 变量值为 1，将其设置为 0。 如果已经是 0，则撤销调用，返回失败。

```
        _;
```

在修饰符中， `_;` 是原始函数调用（含所有参数）。 这里表明仅在 `unlocked` 变量值为 1 时 才能调用函数，而当函数运行时， `unlocked` 值为 0。

```
        unlocked = 1;
    }
```

当主函数返回后，释放锁定。

其他 函数 {#pair-misc}

```
    function getReserves() public view returns (uint112 _reserve0, uint112
_reserve1, uint32 _blockTimestampLast) {
        _reserve0 = reserve0;
        _reserve1 = reserve1;
        _blockTimestampLast = blockTimestampLast;
    }
```

此函数返回给调用者当前的兑换状态。 请注意，Solidity 函数可以返回多个 值。

```
    function _safeTransfer(address token, address to, uint value) private {
        (bool success, bytes memory data) =
token.call(abi.encodeWithSelector(SELECTOR, to, value));
```

此内部函数可以从交易所转账一定数额的 ERC20 代币给其他账户。 `SELECTOR` 指定 我们调用的函数是 `transfer(address,uint)` (见上面的定义)。

为了避免必须为代币函数导入接口，我们需要使用其中一个 ABI 函数 来“手动”创建调用。

```
        require(success && (data.length == 0 || abi.decode(data, (bool))),
'UniswapV2: TRANSFER_FAILED');
    }
```

ERC-20 的转移调用有两种方式可能失败：

1. 回退 如果对外部合约的调用恢复，则布尔返回值为 `false`
2. 正常结束但报告失败。 在这种情况下，返回值的缓冲为非零长度，将其解码为布尔值时，其值为 `false`

一旦出现这两种情况，转移调用就会回退。

事件 {#pair-events}

```
    event Mint(address indexed sender, uint amount0, uint amount1);
    event Burn(address indexed sender, uint amount0, uint amount1, address
indexed to);
```

当流动资金提供者存入流动资金 (`Mint`) 或提取流动资金 (`Burn`) 时，会发出这两个事件。 在 这两种情况下，存入或提取出的 token0 和 token1 的金额是事件的一部分， 以及调用合约的账户地址 (`Sender`)。 在提取资金时，事件中还包括获得代币的目标地址 (`to`) 这个地址可能与发送合约的账户地址不同。

```
    event Swap(
        address indexed sender,
        uint amount0In,
        uint amount1In,
        uint amount0Out,
        uint amount1Out,
        address indexed to
    );
```

当交易者用一种代币交换另一种代币时，会激发此事件。 同样，代币发送者和兑换后代币的存入目的账户可能不一样。 每种代币都可以发送到交易所，或者从交易所接收。

```
    event Sync(uint112 reserve0, uint112 reserve1);
```

最后，每次存入或提取代币时都会发出 `Sync`，无论出于何种原因，提供最新的储备信息 （从而提供汇率）。

设置函数 {#pair-setup}

这些函数应在建立新的配对交易时调用。

```
    constructor() public {
        factory = msg.sender;
    }
```

构造函数确保我们能够跟踪产生配对的工厂合约的地址。 `initialize` 函数和工厂合约执行费（如果有）需要此信息

```
    // called once by the factory at time of deployment
    function initialize(address _token0, address _token1) external {
        require(msg.sender == factory, 'UniswapV2: FORBIDDEN'); // sufficient
check
        token0 = _token0;
        token1 = _token1;
    }
```

这个函数允许工厂（而且只允许工厂）指定配对中进行兑换的两种 ERC-20 代币。

内部更新函数 {#pair-update-internal}

_update

```
    // update reserves and, on the first call per block, price accumulators
    function _update(uint balance0, uint balance1, uint112 _reserve0, uint112
_reserve1) private {
```

每次存入或提取代币时，会调用此函数。

```
        require(balance0 <= uint112(-1) && balance1 <= uint112(-1), 'UniswapV2:
OVERFLOW');
```

如果 balance0 或 balance1 (uint256) 高于 uint112(-1) (=2^112-1) (因此它溢出 & 在转换为 uint112 时返回 0) 拒绝 继续 _update 以防止溢出。 一般的代币可以细分成 10^18 个单元，这意味着 代币每次的兑换限制金额大约为 5.1*10^15。 迄今为止，这并不是一个问题。

```
        uint32 blockTimestamp = uint32(block.timestamp % 2**32);
        uint32 timeElapsed = blockTimestamp - blockTimestampLast; // overflow is
desired
        if (timeElapsed > 0 && _reserve0 != 0 && _reserve1 != 0) {
```

如果流逝的时间值不是零，这意味着本交易是此区块上的第一笔兑换交易。 在这种情况下，我们需要更新累积成本值。

```
            // * never overflows, and + overflow is desired
            price0CumulativeLast +=
uint(UQ112x112.encode(_reserve1).uqdiv(_reserve0)) * timeElapsed;
            price1CumulativeLast +=
uint(UQ112x112.encode(_reserve0).uqdiv(_reserve1)) * timeElapsed;
        }
```

每个累积成本值都用最新成本值（另一个代币的储备金额/本代币的储备金额）乘以以秒为单位的流逝时间加以更新。 要获得平均兑换价格，需要读取两个累积成本值，并除以它们之间的时间差。 例如，假设下面这些事件序列：

事件	reserve0	reserve1	时间戳	边际汇率 (reserve1 / reserve0)	price0CumulativeLast
初始设置	1,000.000	1,000.000	5,000	1.000	0
交易者 A 存入 50 个代币 0 获得 47.619 个代币 1	1,050.000	952.381	5,020	0.907	20
交易者 B 存入 10 个代币 0 获得 8.984 个代币 1	1,060.000	943.396	5,030	0.89	20+10*0.907 = 29.07
交易者 C 存入 40 个代币 0 获得 34.305 个代币 1	1,100.000	909.090	5,100	0.826	29.07+70*0.890 = 91.37
交易者 D 存入 100 个代币 0 获得 109.01 个代币 1	990.990	1,009.090	5,110	1.018	91.37+10*0.826 = 99.63
交易者 E 存入 10 个代币 0 获得 10.079 个代币 1	1,000.990	999.010	5,150	0.998	99.63+40*1.1018 = 143.702

比如说我们想要计算时间戳 5,030 到 5,150 之间**代币 0** 的平均价格。 `price0Cumulative` 的差值 为 143.702-29.07=114.632。 此为两分钟（120 秒）间的平均值。 因此，平均价格为 114.632/120 = 0.955。

此价格计算是我们需要知道原有资金储备规模的原因。

```
        reserve0 = uint112(balance0);
        reserve1 = uint112(balance1);
        blockTimestampLast = blockTimestamp;
        emit Sync(reserve0, reserve1);
    }
```

最后，更新全局变量并发布一个 `Sync` 事件。

_mintFee

```
    // if fee is on, mint liquidity equivalent to 1/6th of the growth in sqrt(k)
    function _mintFee(uint112 _reserve0, uint112 _reserve1) private returns
(bool feeOn) {
```

在 Uniswap 2.0 的合约中规定交易者为使用兑换市场支付 0.30% 的费用。 这笔费用的大部分（交易的 0.25%）支付给流动资金提供者。 余下的 0.5% 可以支付给流动资金提供者或由工厂合约指定的账户地址作为协议费，可以用于支付 Uniswap 团队的开发费用。

为了减少计算次数（因此减少燃料费用），这笔费用只在流动资金被添加或移除时才计算，而不是在每次兑换交易时计算。

```
        address feeTo = IUniswapV2Factory(factory).feeTo();
        feeOn = feeTo != address(0);
```

读取工厂的费用支付地址。 如果返回值为零，则代表没有协议费， 也不需要来计算这笔费用。

```
        uint _kLast = kLast; // gas savings
```

`kLast` 状态变量位于内存中，所以在合约的不同调用中都有一个值。 虽然函数内存每次在函数调用后都会清空，但由于访问存储的费用要比访问内存要高得多， 所以我们使用内存的内部变量来代表存储变量的值，以降低燃料费用。

```
        if (feeOn) {
            if (_kLast != 0) {
```

流动资金提供者仅仅因为提供流动性代币而得到所属的费用。 但是协议 费用要求发行新的流动性代币，并提供给 `feeTo` 的账户地址。

```
                uint rootK = Math.sqrt(uint(_reserve0).mul(_reserve1));
                uint rootKLast = Math.sqrt(_kLast);
                if (rootK > rootKLast) {
```

如果有新的流动性变化需要收取协议费。 你可以在 本文后面看到平方根函数。

```
                    uint numerator = totalSupply.mul(rootK.sub(rootKLast));
                    uint denominator = rootK.mul(5).add(rootKLast);
                    uint liquidity = numerator / denominator;
```

这种复杂的费用计算方法在白皮书第 5 页中作了解释。 在计算 `kLast` 的间隔期间，流动性没有变化（因为每次计算 都是在流动性发生实际变化时发生），所以 `reserve0 * reserve1` 的变化 一定是从交易费用中产生（没有交易费用的话 `reserve0 * reserve1` 值为常量）。

```
                    if (liquidity > 0) _mint(feeTo, liquidity);
                }
            }
```

使用 `UniswapV2ERC20._mint` 函数产生更多的流动池代币并发送到 `feeTo` 地址。

```
        } else if (_kLast != 0) {
            kLast = 0;
        }
    }
```

如果不需收费则将 `klast` 设为 0（如果 klast 不为 0）。 编写该合约时，有一个燃料返还功能，用于鼓励合约将其不需要的存储释放，从而减少以太坊上状态变量的整体存储大小。 此段代码在可行时返还。

外部可访问函数 {#pair-external}

请注意，虽然这些函数*可以*被任意交易或合约调用，其设计目的是用于外部合约调用。 如果直接调用，您无法骗过配对交易， 可能因错误而丢失价值。

铸币

```
    // this low-level function should be called from a contract which performs
important safety checks
    function mint(address to) external lock returns (uint liquidity) {
```

当流动资金提供者为资金池增加流动资金时，将会调用此函数。 它将产生额外的流动池 代币作为奖励。 在 外围合约中增加流动性后调用这个函数，以确保二者在同一交易中（因此其他人都不能提交向合法所有者要求新流动资金的交易）。

```
        (uint112 _reserve0, uint112 _reserve1,) = getReserves(); // gas savings
```

这是 Solidity 函数中读取多个返回值的方式。 这里我们忽略了最后 返回的值，即区块时间戳，因为不需要它。

```
        uint balance0 = IERC20(token0).balanceOf(address(this));
        uint balance1 = IERC20(token1).balanceOf(address(this));
        uint amount0 = balance0.sub(_reserve0);
        uint amount1 = balance1.sub(_reserve1);
```

获取当前余额并查看每个代币类型中添加的数量。

```
        bool feeOn = _mintFee(_reserve0, _reserve1);
```

如果有协议费用的话，计算需要收取的费用，并相应地产生流动池代币。 因为输入 `_mintFee` 函数的参数是原有的储备价值，相应费用的计算只是基于费用 导致的流动池变化。

```
        uint _totalSupply = totalSupply; // gas savings, must be defined here
since totalSupply can update in _mintFee
        if (_totalSupply == 0) {
            liquidity = Math.sqrt(amount0.mul(amount1)).sub(MINIMUM_LIQUIDITY);
           _mint(address(0), MINIMUM_LIQUIDITY); // permanently lock the first
MINIMUM_LIQUIDITY tokens
```

如果这是第一笔存款，会创建数量为 `MINIMUM_LIQUIDITY` 的代币并将它们发送到地址 0 进行锁定。 这些代币 无法被获取，也就是说流动池永远不会为空（避免之后的计算中 出现除零错误）。 `MINIMUM_LIQUIDITY` 的值是 1000，因为考虑到大多数 ERC-20 被细分成 1 个代币的 10^-18 个单位，而以太币则被分为 wei，为 1 个代币价值的 10^-15。 成本不高。

在首次存款时，我们不知道两个代币的相对价值，所以假定两种代币都具有相同的价值，只需要两者数量的乘积并取一下方根。

我们可以相信这一点，因为提供同等价值、避免套利符合存款人的利益。 比方说，这两种代币的价值是相同的，但我们的存款人存入的 **Token1** 是 **Token0** 的四倍。 通过配对交易，交易者可以认为 **Token0** 的价值 比较高。

事件	reserve0	reserve1	reserve0 * reserve1	流动池价值 (reserve0 + reserve1)
初始设置	8	32	256	40
交易者存入 8 个 **Token0** 代币，获得 16 个 **Token1** 代币	16	16	256	32

正如您可以看到的，交易者额外获得了 8 个代币，这是由于流动池价值下降造成的，损害了拥有流动池的存款人。

```
        } else {
            liquidity = Math.min(amount0.mul(_totalSupply) / _reserve0,
amount1.mul(_totalSupply) / _reserve1);
```

对于随后每一笔存款，我们都知道了两种资产之间的汇率。我们期望流动资金提供者提供 等值的两种代币。 如果他们没有，我们根据他们提供的较低价值代币来支付他们的流动池代币以做惩罚。

无论是最初存入还是后续存入，流动池的代币金额均等于 `reserve0*reserve1` 的 平方根，而流动池代币的价值不变（除非存入的资金为不等值的代币类型， 那么就会分派“罚金”）。 这里有另一个例子，两种代币具有相同价值，有三个良性的存款和一个恶性的存款 （即只存入一种类型的代币，所以不会产生任何流动池代币）。

事件	reserve0	reserve1	reserve0 * reserve1	流动池价值 (reserve0 + reserve1)	存入资金而产生的流动池代币	流动池代币总值	每个流动池代币的值
初始设置	8.000	8.000	64	16.000	8	8	2.000
每种代币存入 4 个	12.000	12.000	144	24.000	4	12	2.000
每种代币存入 2 个	14.000	14.000	196	28.000	2	14	2.000
不等							

值的存款	18.000	14.000	252	32.000	0	14	~2.286
套利	~15.874	~15.874	252	~31.748	0	14	~2.267

```
        }
        require(liquidity > 0, 'UniswapV2: INSUFFICIENT_LIQUIDITY_MINTED');
        _mint(to, liquidity);
```

使用 `UniswapV2ERC20._mint` 函数产生更多流动池代币并发送到正确的账户地址。

```
        _update(balance0, balance1, _reserve0, _reserve1);
        if (feeOn) kLast = uint(reserve0).mul(reserve1); // reserve0 and
reserve1 are up-to-date
        emit Mint(msg.sender, amount0, amount1);
    }
```

更新相应的状态变量（`reserve0`、`reserve1`，必要时还包含 `kLast`）并激发相应事件。

销毁

```
    // this low-level function should be called from a contract which performs
important safety checks
    function burn(address to) external lock returns (uint amount0, uint amount1)
{
```

当流动资金被提取且相应的流动池代币需要被销毁时，将调用此函数。 还需要从一个外围账户调用。

```
        (uint112 _reserve0, uint112 _reserve1,) = getReserves(); // gas savings
        address _token0 = token0;                                // gas savings
        address _token1 = token1;                                // gas savings
        uint balance0 = IERC20(_token0).balanceOf(address(this));
        uint balance1 = IERC20(_token1).balanceOf(address(this));
        uint liquidity = balanceOf[address(this)];
```

外围合约在调用函数之前，首先将要销毁的流动资金转到本合约中。 这样 我们知道有多少流动资金需要销毁，并可以确保它被销毁。

```
        bool feeOn = _mintFee(_reserve0, _reserve1);
        uint _totalSupply = totalSupply; // gas savings, must be defined here
since totalSupply can update in _mintFee
        amount0 = liquidity.mul(balance0) / _totalSupply; // using balances
ensures pro-rata distribution
        amount1 = liquidity.mul(balance1) / _totalSupply; // using balances
ensures pro-rata distribution
        require(amount0 > 0 && amount1 > 0, 'UniswapV2:
INSUFFICIENT_LIQUIDITY_BURNED');
```

流动资金提供者获得等值数量的两种代币。 这样不会改变兑换汇率。

```
        _burn(address(this), liquidity);
        _safeTransfer(_token0, to, amount0);
        _safeTransfer(_token1, to, amount1);
        balance0 = IERC20(_token0).balanceOf(address(this));
        balance1 = IERC20(_token1).balanceOf(address(this));

        _update(balance0, balance1, _reserve0, _reserve1);
        if (feeOn) kLast = uint(reserve0).mul(reserve1); // reserve0 and
reserve1 are up-to-date
        emit Burn(msg.sender, amount0, amount1, to);
    }
```

`burn` 函数的其余部分是上述 `mint` 函数的镜像。

兑换

```
    // this low-level function should be called from a contract which performs
important safety checks
    function swap(uint amount0Out, uint amount1Out, address to, bytes calldata
data) external lock {
```

此函数也应该从外围合约调用。

```
        require(amount0Out > 0 || amount1Out > 0, 'UniswapV2:
INSUFFICIENT_OUTPUT_AMOUNT');
        (uint112 _reserve0, uint112 _reserve1,) = getReserves(); // gas savings
        require(amount0Out < _reserve0 && amount1Out < _reserve1, 'UniswapV2:
INSUFFICIENT_LIQUIDITY');

        uint balance0;
        uint balance1;
        { // scope for _token{0,1}, avoids stack too deep errors
```

本地变量可以存储在内存中，或者如果变量数目不太多，直接存储进堆栈。 如果我们可以限制变量数量，那么建议使用堆栈以减少燃料消耗。 更多详情见 以太坊黄皮书（以前的以太坊规范）p. 26“方程式 298”。

```
            address _token0 = token0;
            address _token1 = token1;
            require(to != _token0 && to != _token1, 'UniswapV2: INVALID_TO');
            if (amount0Out > 0) _safeTransfer(_token0, to, amount0Out); //
optimistically transfer tokens
            if (amount1Out > 0) _safeTransfer(_token1, to, amount1Out); //
optimistically transfer tokens
```

这种转移应该是会成功的，因为在转移之前我们确信所有条件都得到满足。 在以太坊中这样操作是可以的， 原因在于如果调用条件没有得到满足，我们可以恢复操作及造成的改变。

```
            if (data.length > 0) IUniswapV2Callee(to).uniswapV2Call(msg.sender,
amount0Out, amount1Out, data);
```

如果收到请求，则通知接收者要进行兑换。

```
            balance0 = IERC20(_token0).balanceOf(address(this));
            balance1 = IERC20(_token1).balanceOf(address(this));
        }
```

获取当前余额。 外围合约在调用交换函数之前，需要向合约发送要兑换的代币。 这个功能可以使得合约检查它没有受到欺骗，这个检查*必须*通过核心合约调用（因为本功能可能被除我们外围合约之外的其它单位调用）。

```
        uint amount0In = balance0 > _reserve0 - amount0Out ? balance0 -
(_reserve0 - amount0Out) : 0;
        uint amount1In = balance1 > _reserve1 - amount1Out ? balance1 -
(_reserve1 - amount1Out) : 0;
        require(amount0In > 0 || amount1In > 0, 'UniswapV2:
INSUFFICIENT_INPUT_AMOUNT');
        { // scope for reserve{0,1}Adjusted, avoids stack too deep errors
            uint balance0Adjusted = balance0.mul(1000).sub(amount0In.mul(3));
            uint balance1Adjusted = balance1.mul(1000).sub(amount1In.mul(3));
            require(balance0Adjusted.mul(balance1Adjusted) >=
uint(_reserve0).mul(_reserve1).mul(1000**2), 'UniswapV2: K');
```

这是一项健全性检查，确保我们不会因兑换而损失代币。 在任何情况下交换都不应减少 `reserve0*reserve1`。 这也是我们确保在掉期中发送 0.3% 费用的地方； 在检查 K 值之前，我们将两个余额乘以 1000 减去金额乘以 3，这意味着在将其 K 值与当前值进行比较之前，从余额中扣除 0.3% (3/1000 = 0.003 = 0.3%) 储备 K 值。

```
        }

        _update(balance0, balance1, _reserve0, _reserve1);
        emit Swap(msg.sender, amount0In, amount1In, amount0Out, amount1Out, to);
    }
```

更新 `reserve0` 和 `reserve1` 的值，并在必要时更新价格累积值和时间戳并激发相应事件。

同步或跳过 {#sync-or-skim}

实际余额有可能与配对交易所认为的储备金余额没有同步。 没有合约的认同，就无法撤回代币，但存款却不同。 帐户 可以将代币转移到交易所，而无需调用 `mint` 或 `swap`。

在这种情况下，有两种解决办法：

- `sync`，将储备金更新为当前余额
- `skim`，撤回额外的金额。 请注意任何账户都可以调用 `skim` 函数，因为无法知道是谁 存入的代币。 此信息是在一个事件中发布的，但这些事件无法从区块链中访问。

```
    // force balances to match reserves
    function skim(address to) external lock {
        address _token0 = token0; // gas savings
        address _token1 = token1; // gas savings
        _safeTransfer(_token0, to,
IERC20(_token0).balanceOf(address(this)).sub(reserve0));
```

```
        _safeTransfer(_token1, to,
IERC20(_token1).balanceOf(address(this)).sub(reserve1));
    }



    // force reserves to match balances
    function sync() external lock {
        _update(IERC20(token0).balanceOf(address(this)),
IERC20(token1).balanceOf(address(this)), reserve0, reserve1);
    }
}
```

UniswapV2Factory.sol {#UniswapV2Factory}

此合约实现配对 兑换。

```
pragma solidity =0.5.16;

import './interfaces/IUniswapV2Factory.sol';
import './UniswapV2Pair.sol';

contract UniswapV2Factory is IUniswapV2Factory {
    address public feeTo;
    address public feeToSetter;
```

这些状态变量是执行协议费用所必需的（请见白皮书的第 5 页）。 `feeTo` 地址用于累加协议费用的流动池代币，而 `feeToSetter` 是允许更改 `feeTo` 为 不同地址的地址值。

```
    mapping(address => mapping(address => address)) public getPair;
    address[] public allPairs;
```

这些变量用以跟踪配对，即两种代币之间的兑换。

第一个变量， `getPair` 是一个映射，根据兑换的两个 ERC-20 代币 来识别配对交易合约。 ERC-20 代币通过实现合约的地址来识别，所以关键字和值都是地址。 为了获取 配对交易的地址，以便能够从 `tokenA` 转换为 `tokenB`，可以使用 `getPair [<tokenA address><tokenB address>]` （或反之）。

第二个变量， `allPairs` 是一个数组，其中包括该工厂创建的所有 配对交易的地址。 在以太坊中，您无法循环访问映射内容， 或获取所有关键字的列表，所以，这个变量是唯一能够知道此工厂 管理哪个兑换的方法。

注意: 您不能循环访问所有关键字的原因是合约数据 存储*十分昂贵*，所以我们越少用越好，且越少改变越好。 您可以创建支持循环访问的映射， 但它们需要额外存储关键字列表。 但在大多数应用程序中并不需要。

```
    event PairCreated(address indexed token0, address indexed token1, address
pair, uint);
```

当新的配对交易创建时，将激发此事件。 它包括代币地址、 配对交易地址以及工厂管理的兑换交易总数。

```
    constructor(address _feeToSetter) public {
        feeToSetter = _feeToSetter;
    }
```

构造函数做的唯一事情是指定 `feeToSetter`。 工厂开始时没有 费用，只有 `feeSetter` 可以更改这种情况。

```
    function allPairsLength() external view returns (uint) {
        return allPairs.length;
    }
```

此函数返回交易配对的数量。

```
    function createPair(address tokenA, address tokenB) external returns
(address pair) {
```

这是工厂的主要函数，可以在两个 ERC-20 代币之间创建配对交易。 注意， 任何人都可以调用此函数。 并不需要 Uniswap 许可就能创建新的配对 兑换。

```
        require(tokenA != tokenB, 'UniswapV2: IDENTICAL_ADDRESSES');
        (address token0, address token1) = tokenA < tokenB ? (tokenA, tokenB) :
(tokenB, tokenA);
```

我们希望新兑换的地址可以确定， 这样它可以在链下预计算 （这对于第二层的交易 来说比较有用）。 为了做到这一点，我们需要代币地址始终按顺序排列，无论收到代币地址的顺序如何， 都需要在这里排序。

```
        require(token0 != address(0), 'UniswapV2: ZERO_ADDRESS');
        require(getPair[token0][token1] == address(0), 'UniswapV2:
PAIR_EXISTS'); // single check is sufficient
```

大流动资金池优于小流动资金池，因为其价格比较稳定。 对于每一对代币， 我们不想有多个流动资金池。 如果已经有一个配对交易，则无需为相同的代币对 创建另一个配对交易。

```
        bytes memory bytecode = type(UniswapV2Pair).creationCode;
```

为了创建一个新的合约，我们需要获得创建代码（包括构造函数和写入 用于存储实际合约以太坊虚拟机字节码的代码）。 在 Solidity 语言中，通常使用 `addr = new <name of contract>(<constructor parameters>)` 的格式语句，然后编译器就可以完成所有的工作，不过为了获取一个确定的合约地址，需要使用 CREATE2 操作码。 当这个代码编写出来时，Solidity 还不支持操作码，因此需要手动获取 代码。 目前这已经不再是问题，因为 Solidity 现已支持 CREATE2。

```
        bytes32 salt = keccak256(abi.encodePacked(token0, token1));
        assembly {
            pair := create2(0, add(bytecode, 32), mload(bytecode), salt)
        }
```

当 Solidity 不支持操作码时，我们可以通过内联汇编来调用。

```
        IUniswapV2Pair(pair).initialize(token0, token1);
```

调用 `initialize` 函数来告诉新兑换交易可以兑换哪两种代币。

```
        getPair[token0][token1] = pair;
        getPair[token1][token0] = pair; // populate mapping in the reverse
direction
        allPairs.push(pair);
        emit PairCreated(token0, token1, pair, allPairs.length);
    }
```

在状态变量中保存新的配对信息，并激发一个事件来告知外界新的配对交易合约已生成。

```
    function setFeeTo(address _feeTo) external {
        require(msg.sender == feeToSetter, 'UniswapV2: FORBIDDEN');
        feeTo = _feeTo;
    }

    function setFeeToSetter(address _feeToSetter) external {
        require(msg.sender == feeToSetter, 'UniswapV2: FORBIDDEN');
        feeToSetter = _feeToSetter;
    }
}
```

这两个函数，允许 `setFeeTo` 管理费用的接收者（如有），并将 `setFeeToSetter` 更改为一个新 地址。

UniswapV2ERC20.sol {#UniswapV2ERC20}

本合约实现了 ERC-20 流动代币。 这与 OpenWhisk ERC-20 合约相似，因此 这里仅解释不同的部分，`permit` 的功能。

以太坊上的交易需要消耗以太币 (ETH)，相当于实际货币。 如果您有 ERC-20 代币但没有以太币，就无法发送 交易，因而不能用代币做任何事情。 避免该问题的一个解决方案是 元交易。 代币的所有者签署一个交易，许可他人将代币从链上取出，并通过网络将其发送给 接收人。 接收人拥有以太币，可以代表所有者提交许可。

```
    bytes32 public DOMAIN_SEPARATOR;
    // keccak256("Permit(address owner,address spender,uint256 value,uint256
nonce,uint256 deadline)");
    bytes32 public constant PERMIT_TYPEHASH =
0x6e71edae12b1b97f4d1f60370fef10105fa2faae0126114a169c64845d6126c9;
```

此哈希值是这种交易类型的标识。 在这里 我们唯一支持的是带有这些参数的 `Permit`。

```
    mapping(address => uint) public nonces;
```

接收人无法伪造数字签名。 但是，可以两次发送相同的交易 （这是一种重放攻击形式）。 为防止这种情况，我们使用 一个随机数。 如果新 `Permit` 的随机数不是上一次的使用的随机数加一， 我们便判定它无效。

```
    constructor() public {
        uint chainId;
        assembly {
            chainId := chainid
        }
```

这是获取链标识符的代码。 它使用名为 Yul 的以太坊虚拟机编译语言。 请注意，在当前版本 Yul 中，您必须使用 `chainid()`， 而非 `chainid`。

```
        DOMAIN_SEPARATOR = keccak256(
            abi.encode(
                keccak256('EIP712Domain(string name,string version,uint256
chainId,address verifyingContract)'),
                keccak256(bytes(name)),
                keccak256(bytes('1')),
                chainId,
                address(this)
            )
        );
    }
```

计算 EIP-712 的域分隔符。

```
    function permit(address owner, address spender, uint value, uint deadline,
uint8 v, bytes32 r, bytes32 s) external {
```

这是实现批准功能的函数。 它接收相关字段的参数，以及数字签名 的三个标量值（v、r 和 s）。

```
        require(deadline >= block.timestamp, 'UniswapV2: EXPIRED');
```

截止日期后请勿接受交易。

```
        bytes32 digest = keccak256(
            abi.encodePacked(
                '\x19\x01',
                DOMAIN_SEPARATOR,
                keccak256(abi.encode(PERMIT_TYPEHASH, owner, spender, value,
nonces[owner]++, deadline))
            )
        );
```

`abi.encodePacked(...)` 是我们预计将收到的信息。 我们知道随机数应该是什么，所以不需要 将它作为一个参数

以太坊签名算法预计获得 256 位用于签名，所以我们使用 `keccak256` 哈希函数。

```
        address recoveredAddress = ecrecover(digest, v, r, s);
```

从摘要和签名中，我们可以用 ecrecover 函数计算出签名的地址。

```
        require(recoveredAddress != address(0) && recoveredAddress == owner,
'UniswapV2: INVALID_SIGNATURE');
        _approve(owner, spender, value);
    }
```

如果一切正常，则将其视为 ERC-20 批准。

外围合约 {#periphery-contracts}

外围合约是用于 Uniswap 的 API（应用程序接口）。 它们可用于来自 其他合约或去中心化应用程序的外部调用。 你可以直接调用核心合约，但这更为复杂， 如果您犯了错误，则可能会丢失值。 核心合约只包含确保它们不会遭受欺骗的测试，不会对其他调用者进行健全性检查。 它们在外围，因此可以根据需要进行更新。

UniswapV2Router01.sol {#UniswapV2Router01}

该合约 存有问题，不应该再使用。 幸运的是， 外围合约无状态记录，也不拥有任何资产，所以很容易废弃。建议 使用 `UniswapV2Router02` 来替代。

UniswapV2Router02.sol {#UniswapV2Router02}

在大多数情况下，您会通过该合约使用 Uniswap。 有关使用说明，您可以在这里找到。

```
pragma solidity =0.6.6;

import '@uniswap/v2-core/contracts/interfaces/IUniswapV2Factory.sol';
import '@uniswap/lib/contracts/libraries/TransferHelper.sol';

import './interfaces/IUniswapV2Router02.sol';
import './libraries/UniswapV2Library.sol';
import './libraries/SafeMath.sol';
import './interfaces/IERC20.sol';
import './interfaces/IWETH.sol';
```

其中大部分我们都曾遇到过，或相当明显。 一个例外是 `IWETH.sol`。 Uniswapv2 允许兑换 任意一对 ERC-20 代币，但以太币 (ETH) 本身并不是 ERC-20 代币。 它早于该标准出现，并采用独特的机制转换。 为了 在适用于 ERC-20 代币的合约中使用以太币，人们制定出包装以太币 (WETH) 合约。 您 发送以太币到该合约，它会为您铸造相同金额的包装以太币。 或者您可以销毁包装以太币，然后换回以太币。

```
contract UniswapV2Router02 is IUniswapV2Router02 {
    using SafeMath for uint;

    address public immutable override factory;
    address public immutable override WETH;
```

路由需要知道使用哪个工厂，以及对于需要包装以太币的交易，要使用什么包装以太币合约。 这些变量值是 不可修改的，意味着它们 只能在构造函数中设置。 这使得用户可以相信没有人能够改变它们，比如指向有风险 的合约。

```
    modifier ensure(uint deadline) {
        require(deadline >= block.timestamp, 'UniswapV2Router: EXPIRED');
        _;
    }
```

此修改函数确保有时间限制的交易（如果可以，请在 Y 之前执行 X）不会在时限后发生。

```
    constructor(address _factory, address _WETH) public {
        factory = _factory;
        WETH = _WETH;
    }
```

构造函数仅用于设置不可变的状态变量。

```
    receive() external payable {
        assert(msg.sender == WETH); // only accept ETH via fallback from the
WETH contract
    }
```

当我们将代币从包装以太币合约换回以太币时，需要调用此函数。 只有我们使用的包装以太币合约才能授权 完成此操作。

增加流动资金 {#add-liquidity}

这些函数添加代币进行配对交易，从而增大了流动资金池。

```

    // **** ADD LIQUIDITY ****
    function _addLiquidity(
```

此函数用于计算应存入 配对交易的 A 代币和 B 代币的金额。

```
        address tokenA,
        address tokenB,
```

这些是 ERC-20 代币合约的地址。

```
        uint amountADesired,
        uint amountBDesired,
```

这些是流动资金提供者想要存入的代币数额。 它们也是要存入的 A 和 B 的最大数额。

```
        uint amountAMin,
        uint amountBMin
```

这些是可接受的最低存款数额。 如果大于这些金额的交易无法完成， 则会回退。 如果不想要此功能，将它们设定为零即可。

流动资金提供者指定最低限额的目的，是想要将交易限制在 与当前汇率接近的汇率。 如果汇率波动太大， 可能意味着基础价值可能发生改变，他们需要人工决定做什么。

例如，想象汇率是一比一时，流动资金提供者 指定了这些值：

参数	值
amountADesired	1000
amountBDesired	1000
amountAMin	900
amountBMin	800

只要汇率保持在 0.9 至 1.25 之间，交易就会进行。 如果汇率超出这个范围，交易将被取消。

采取这种预防措施的原因是交易不是立即执行的，提交这些交易之后，最终 要等到矿工会将它们包含在区块中才算执行完（除非交易的燃料价格非常低，在这种情况下， 需要提交另一笔具有相同随机数和更高燃料价格的交易，以覆盖前一笔交易）。 在 提交交易和写入区块之间发生的事情是无法控制的。

```
    ) internal virtual returns (uint amountA, uint amountB) {
```

该函数返回流动资金提供者应存入的金额，其比率等于当前 储备金之间的比率。

```
        // create the pair if it doesn't exist yet
        if (IUniswapV2Factory(factory).getPair(tokenA, tokenB) == address(0)) {
            IUniswapV2Factory(factory).createPair(tokenA, tokenB);
        }
```

如果还没有此代币对的兑换交易，则创建一个。

```
        (uint reserveA, uint reserveB) = UniswapV2Library.getReserves(factory,
tokenA, tokenB);
```

获取配对中的当前储备金。

```
        if (reserveA == 0 && reserveB == 0) {
            (amountA, amountB) = (amountADesired, amountBDesired);
```

如果当前储备金为空，那么这是一笔新的配对交易。 存入的金额应与 流动资金提供者想要提供的金额完全相同。

```
        } else {
            uint amountBOptimal = UniswapV2Library.quote(amountADesired,
reserveA, reserveB);
```

如果我们需要知道要多大的金额，我们可使用 此函数获得最佳金额。 我们想要与当前储备相同的比率。

```
            if (amountBOptimal <= amountBDesired) {
                require(amountBOptimal >= amountBMin, 'UniswapV2Router:
INSUFFICIENT_B_AMOUNT');
                (amountA, amountB) = (amountADesired, amountBOptimal);
```

如果最佳金额 `amountBOptimal` 小于流动资金提供者想要存入的金额，这意味着代币 B 目前比流动资金存款人所认为的价值更高，所以需要更少的数额。

```
            } else {
                uint amountAOptimal = UniswapV2Library.quote(amountBDesired,
reserveB, reserveA);
                assert(amountAOptimal <= amountADesired);
                require(amountAOptimal >= amountAMin, 'UniswapV2Router:
INSUFFICIENT_A_AMOUNT');
                (amountA, amountB) = (amountAOptimal, amountBDesired);
```

如果 B 代币的最佳数额大于所需的 B 代币数额，这意味着 B 代币目前的价值 低于流动资金存款人的估计，所以需要更高的金额。 然而，所需的金额是最大值，意味着我们无法存入更多数量的 B 代币。 可以选择的另一种方法是，我们计算所需 B 代币数额对应的最佳 A 代币数额。

把数值汇总起来，我们就会得到这张图表。 假定您正在试图存入 1000 个 A 代币（蓝线）和 1000 个 B 代币（红线）。 X 轴是汇率，A/B。 如果 x=1，它们的价值相等，并且你每次可以存入 1000 个 A 代币和 1000 个 B 代币。 如果 x=2，A 的价值是 B 的两倍（每个 A 代币可换两个 B 代币），所以您可以存 1000 个 B 代币， 但只能存 500 个 A 代币。 如果是 x=0.5，情况就会逆转，即可存 1000 个 A 代币或 500 个 B 代币。

图表

```
            }
        }
    }
```

您可以将流动资金直接存入核心合约（使用 UniswapV2Pair:::mint），但核心合约 只是检查合约自己没有遭受欺骗。因此，如果汇率在 提交交易至执行交易之间发生变化，您将面临损失资金价值的风险。 如果使用外围合约，它会计算您应该存入 的金额并会立即存入，所以汇率不会改变，您不会损失资金价值。

```
    function addLiquidity(
        address tokenA,
        address tokenB,
        uint amountADesired,
        uint amountBDesired,
        uint amountAMin,
        uint amountBMin,
        address to,
        uint deadline
```

此函数可以在交易中调用，用于存入流动资金。 大多数参数与上述 `_addLiquidity` 中相同，但有两个例外：

. `to` 是会获取新流动池代币的地址，这些代币铸造用于显示流动资金提供者在池中所占比率 `deadline` 是交易的时间限制

```
    ) external virtual override ensure(deadline) returns (uint amountA, uint
amountB, uint liquidity) {
        (amountA, amountB) = _addLiquidity(tokenA, tokenB, amountADesired,
amountBDesired, amountAMin, amountBMin);
        address pair = UniswapV2Library.pairFor(factory, tokenA, tokenB);
```

我们计算实际存入的金额，然后找到流动资金池的账户地址。 为了节省燃料，我们不用 询问工厂，但可以使用库函数 `pairFor`（参见如下程序库）

```
        TransferHelper.safeTransferFrom(tokenA, msg.sender, pair, amountA);
        TransferHelper.safeTransferFrom(tokenB, msg.sender, pair, amountB);
```

将正确数额的代币从用户账户转到配对交易。

```
        liquidity = IUniswapV2Pair(pair).mint(to);
    }
```

反过来，将流动资金池的部分所有权赋予 `to` 地址的流动性代币。 核心 合约的 `mint` 函数可以查看到它有多少额外的代币（与 上次流动性发生变化时所有的数额进行比较），并相应地铸造流动性代币。

```
    function addLiquidityETH(
        address token,
        uint amountTokenDesired,
```

当流动资金提供者想要向代币/以太币配对交易提供流动资金时，存在一些差别。 合约 为流动资金提供者处理以太币的包装。 用户不需要指定想要存入多少以太币， 因为用户可以直接随交易发送（金额可以在 `msg.value` 中查到）。

```
        uint amountTokenMin,
        uint amountETHMin,
        address to,
        uint deadline
    ) external virtual override payable ensure(deadline) returns (uint
amountToken, uint amountETH, uint liquidity) {
        (amountToken, amountETH) = _addLiquidity(
            token,
            WETH,
            amountTokenDesired,
            msg.value,
            amountTokenMin,
            amountETHMin
        );
        address pair = UniswapV2Library.pairFor(factory, token, WETH);
        TransferHelper.safeTransferFrom(token, msg.sender, pair, amountToken);
        IWETH(WETH).deposit{value: amountETH}();
        assert(IWETH(WETH).transfer(pair, amountETH));
```

为了将以太币存入合约，首先将其包装成包装以太币，然后将包装以太币转入配对。 请注意 转账将打包进 `assert` 中。 这意味着如果转账失败，此合约调用也会失败， 因此包装不会真的发生。

```
        liquidity = IUniswapV2Pair(pair).mint(to);
        // refund dust eth, if any
        if (msg.value > amountETH) TransferHelper.safeTransferETH(msg.sender,
msg.value - amountETH);
    }
```

用户已经向我们发送了以太币，如果还有任何额外的资金剩余（因为其他代币 比用户认定的价值更低），我们需要签发退款。

撤回流动资金 {#remove-liquidity}

下面的函数将撤回流动资金并还给流动资金提供者。

```
    // **** REMOVE LIQUIDITY ****
    function removeLiquidity(
        address tokenA,
        address tokenB,
        uint liquidity,
        uint amountAMin,
        uint amountBMin,
        address to,
        uint deadline
    ) public virtual override ensure(deadline) returns (uint amountA, uint
amountB) {
```

最简单的流动资金撤回案例。 流动资金提供者同意 接受每种代币有一个最低数额，必须在截止时间之前完成。

```
        address pair = UniswapV2Library.pairFor(factory, tokenA, tokenB);
        IUniswapV2Pair(pair).transferFrom(msg.sender, pair, liquidity); // send
liquidity to pair
        (uint amount0, uint amount1) = IUniswapV2Pair(pair).burn(to);
```

核心合约的 `burn` 函数处理返还给用户的代币。

```
        (address token0,) = UniswapV2Library.sortTokens(tokenA, tokenB);
```

某个函数返回多个值时，如果我们只对其中几个值感兴趣，以下便是 我们只获取那些值的方式。 从消耗燃料的角度来说，这样比读取那些从来不用的值更加经济。

```
        (amountA, amountB) = tokenA == token0 ? (amount0, amount1) : (amount1,
amount0);
```

将按从核心合约返回代币的路径（按代币地址降序）调整为 以用户期望的方式（对应于 `tokenA` 和 `tokenB`）。

```
        require(amountA >= amountAMin, 'UniswapV2Router:
INSUFFICIENT_A_AMOUNT');
        require(amountB >= amountBMin, 'UniswapV2Router:
INSUFFICIENT_B_AMOUNT');
    }
```

可以首先进行代币转让，然后再核实转让是否合法，因为如果不合法，我们可以恢复 所有的状态更改。

```
    function removeLiquidityETH(
        address token,
        uint liquidity,
        uint amountTokenMin,
        uint amountETHMin,
        address to,
        uint deadline
```

```
    ) public virtual override ensure(deadline) returns (uint amountToken, uint
amountETH) {
        (amountToken, amountETH) = removeLiquidity(
            token,
            WETH,
            liquidity,
            amountTokenMin,
            amountETHMin,
            address(this),
            deadline
        );
        TransferHelper.safeTransfer(token, to, amountToken);
        IWETH(WETH).withdraw(amountETH);
        TransferHelper.safeTransferETH(to, amountETH);
    }
```

撤回以太币流动资金的方式几乎是一样的，区别在于我们首先会收到包装以太币代币，然后将它们兑换为 以太币，最后再退还给流动资金提供者。

```
    function removeLiquidityWithPermit(
        address tokenA,
        address tokenB,
        uint liquidity,
        uint amountAMin,
        uint amountBMin,
        address to,
        uint deadline,
        bool approveMax, uint8 v, bytes32 r, bytes32 s
    ) external virtual override returns (uint amountA, uint amountB) {
        address pair = UniswapV2Library.pairFor(factory, tokenA, tokenB);
        uint value = approveMax ? uint(-1) : liquidity;
        IUniswapV2Pair(pair).permit(msg.sender, address(this), value, deadline,
v, r, s);
        (amountA, amountB) = removeLiquidity(tokenA, tokenB, liquidity,
amountAMin, amountBMin, to, deadline);
    }


    function removeLiquidityETHWithPermit(
        address token,
        uint liquidity,
        uint amountTokenMin,
        uint amountETHMin,
        address to,
        uint deadline,
        bool approveMax, uint8 v, bytes32 r, bytes32 s
    ) external virtual override returns (uint amountToken, uint amountETH) {
        address pair = UniswapV2Library.pairFor(factory, token, WETH);
        uint value = approveMax ? uint(-1) : liquidity;
        IUniswapV2Pair(pair).permit(msg.sender, address(this), value, deadline,
v, r, s);
        (amountToken, amountETH) = removeLiquidityETH(token, liquidity,
amountTokenMin, amountETHMin, to, deadline);
    }
```

这些函数转发元交易，通过许可证机制使没有以太币的用户能够从流动池中提取资金。

```
    // **** REMOVE LIQUIDITY (supporting fee-on-transfer tokens) ****
    function removeLiquidityETHSupportingFeeOnTransferTokens(
        address token,
        uint liquidity,
        uint amountTokenMin,
        uint amountETHMin,
        address to,
        uint deadline
    ) public virtual override ensure(deadline) returns (uint amountETH) {
        (, amountETH) = removeLiquidity(
            token,
            WETH,
            liquidity,
            amountTokenMin,
            amountETHMin,
            address(this),
            deadline
        );
        TransferHelper.safeTransfer(token, to,
IERC20(token).balanceOf(address(this)));
        IWETH(WETH).withdraw(amountETH);
        TransferHelper.safeTransferETH(to, amountETH);
    }
```

此函数可以用于在传输或存储时收取费用的代币。 当代币合约中有这种费用时，我们不能依靠 `removeLiquidity` 函数来告诉我们可以撤回多少代币。因此，我们需要先撤回然后查询代币金额。

```
    function removeLiquidityETHWithPermitSupportingFeeOnTransferTokens(
        address token,
        uint liquidity,
        uint amountTokenMin,
        uint amountETHMin,
        address to,
        uint deadline,
        bool approveMax, uint8 v, bytes32 r, bytes32 s
    ) external virtual override returns (uint amountETH) {
        address pair = UniswapV2Library.pairFor(factory, token, WETH);
        uint value = approveMax ? uint(-1) : liquidity;
        IUniswapV2Pair(pair).permit(msg.sender, address(this), value, deadline,
v, r, s);
        amountETH = removeLiquidityETHSupportingFeeOnTransferTokens(
            token, liquidity, amountTokenMin, amountETHMin, to, deadline
        );
    }
```

最后这个函数将存储费用计入元交易。

交易 {#trade}

```
    // **** SWAP ****
    // requires the initial amount to have already been sent to the first pair
    function _swap(uint[] memory amounts, address[] memory path, address _to)
internal virtual {
```

呈现给交易者的函数可以调用此函数 以执行内部处理。

```
        for (uint i; i < path.length - 1; i++) {
```

在撰写此教程时，已有 388,160 个 ERC-20 代币。 如果每个代币对 都有配对交易，配对交易数将超过 1500 亿次。 目前为止， 整个链上只拥有该帐户数量的 0.1%。 实际上，兑换 函数支持路径概念。 交易者可以将 A 兑换成 B、B 兑换成 C、C 兑换成 D，因此 不需要直接的 A-D 配对交易。

这些市场上的价格往往是同步的，因为当它们没有同步时， 就会为套利创造机会。 设想一下，例如有三种代币，A、B 和 C。有三次配对交易， 每一对一次。

1. 初始情况
2. 交易者出售 24.695 A 代币，获得 25.305 B 代币。
3. 交易者卖出了 24.695 B 代币以得到 25.305 C 代币，大约获得 0.61 B 代币的利润。
4. 交易者卖出了 24.695 C 代币以得到 25.305 A 代币，大约获得 0.61 C 代币的利润。 交易者还拥有剩下的 0.61 A 代币（交易者最终拥有的 25.305 A 代币，减去原始投资 24.695 A 代币）。

步骤	A-B 兑换	B-C 兑换	A-C 兑换
1	A:1000 B:1050 A/B=1.05	B:1000 C:1050 B/C=1.05	A:1050 C:1000 C/A=1.05
2	A:1024.695 B:1024.695 A/B=1	B:1000 C:1050 B/C=1.05	A:1050 C:1000 C/A=1.05
3	A:1024.695 B:1024.695 A/B=1	B:1024.695 C:1024.695 B/C=1	A:1050 C:1000 C/A=1.05
4	A:1024.695 B:1024.695 A/B=1	B:1024.695 C:1024.695 B/C=1	A:1024.695 C:1024.695 C/A=1

```
            (address input, address output) = (path[i], path[i + 1]);
            (address token0,) = UniswapV2Library.sortTokens(input, output);
            uint amountOut = amounts[i + 1];
```

获取我们当前处理的配对，排序后（以便与配对一起使用）获得预期的输出金额。

```
            (uint amount0Out, uint amount1Out) = input == token0 ? (uint(0),
amountOut) : (amountOut, uint(0));
```

获得预期的金额后，按配对交易所需方式排序。

```
            address to = i < path.length - 2 ? UniswapV2Library.pairFor(factory,
output, path[i + 2]) : _to;
```

这是最后一次兑换吗？ 如果是，将收到用于交易的代币发送到目的地址。 如果不是，则将代币发送到下一个配对交易。

```
            IUniswapV2Pair(UniswapV2Library.pairFor(factory, input,
output)).swap(
                amount0Out, amount1Out, to, new bytes(0)
            );
        }
    }
```

真正调用配对交易来兑换代币。 我们不需要回调函数来了解交易信息， 因此没有在该字段中发送任何字节。

```
    function swapExactTokensForTokens(
```

交易者直接使用此函数来兑换代币。

```
        uint amountIn,
        uint amountOutMin,
        address[] calldata path,
```

此参数包含 ERC-20 合约的地址。 如上文所述，此参数是一个数组，因为可能 需要通过多次配对交易来从现有资产获取到想要的资产。

Solidity 中的函数参数可以存入 `memory` 或者 `calldata`。 如果此函数是合约的一个入口点， 即直接由用户（通过交易）或从另一个合约调用，那么参数的值 可以直接从调用数据中获取。 如果函数是通过内部调用，如上述 `_swap`，则参数 必须存储在 `memory` 中。 从所调用合约的角度来看， `calldata` 为只读变量。

对于标量类型，如 `uint` 或者 `address`，编译器可以帮助我们处理存储方式的选择，但对于数组， 由于它们需要更多的存储空间也消耗更多的燃料，我们需要指定要使用的存储类型。

```
        address to,
        uint deadline
    ) external virtual override ensure(deadline) returns (uint[] memory amounts)
{
```

返回值总是返回内存中。

```
        amounts = UniswapV2Library.getAmountsOut(factory, amountIn, path);
        require(amounts[amounts.length - 1] >= amountOutMin, 'UniswapV2Router:
INSUFFICIENT_OUTPUT_AMOUNT');
```

计算每次兑换时要购买的代币金额。 如果结果低于交易者愿意接受的最低限度， 则撤销交易。

```
        TransferHelper.safeTransferFrom(
            path[0], msg.sender, UniswapV2Library.pairFor(factory, path[0],
path[1]), amounts[0]
        );
        _swap(amounts, path, to);
    }
```

最后，将初始的 ERC-20 代币转到第一个配对交易的帐户中，然后调用 `_swap`。 所有这些 都发生在同一次交易中，因此配对交易知道任何意料之外的代币都是此次转账的一部分。

```
    function swapTokensForExactTokens(
        uint amountOut,
        uint amountInMax,
        address[] calldata path,
        address to,
        uint deadline
    ) external virtual override ensure(deadline) returns (uint[] memory amounts)
{
        amounts = UniswapV2Library.getAmountsIn(factory, amountOut, path);
        require(amounts[0] <= amountInMax, 'UniswapV2Router:
EXCESSIVE_INPUT_AMOUNT');
        TransferHelper.safeTransferFrom(
            path[0], msg.sender, UniswapV2Library.pairFor(factory, path[0],
path[1]), amounts[0]
        );
        _swap(amounts, path, to);
    }
```

前一个函数，`swapTokensForTokens`，使交易者可以指定愿意 给出代币的准确数量和愿意接受代币的最低数量。 此函数可以撤销兑换， 使交易者能够指定想要的输出代币数额，以及愿意支付的输入代币的最大数额。

在这两种情况下，交易者必须首先给予此外围合约一定的额度，用于转账。

```
    function swapExactETHForTokens(uint amountOutMin, address[] calldata path,
address to, uint deadline)
        external
        virtual
        override
        payable
        ensure(deadline)
        returns (uint[] memory amounts)
    {
        require(path[0] == WETH, 'UniswapV2Router: INVALID_PATH');
        amounts = UniswapV2Library.getAmountsOut(factory, msg.value, path);
        require(amounts[amounts.length - 1] >= amountOutMin, 'UniswapV2Router:
INSUFFICIENT_OUTPUT_AMOUNT');
        IWETH(WETH).deposit{value: amounts[0]}();
        assert(IWETH(WETH).transfer(UniswapV2Library.pairFor(factory, path[0],
path[1]), amounts[0]));
        _swap(amounts, path, to);
    }


    function swapTokensForExactETH(uint amountOut, uint amountInMax, address[]
calldata path, address to, uint deadline)
        external
        virtual
        override
        ensure(deadline)
        returns (uint[] memory amounts)
```

```
    {
        require(path[path.length - 1] == WETH, 'UniswapV2Router: INVALID_PATH');
        amounts = UniswapV2Library.getAmountsIn(factory, amountOut, path);
        require(amounts[0] <= amountInMax, 'UniswapV2Router:
EXCESSIVE_INPUT_AMOUNT');
        TransferHelper.safeTransferFrom(
            path[0], msg.sender, UniswapV2Library.pairFor(factory, path[0],
path[1]), amounts[0]
        );
        _swap(amounts, path, address(this));
        IWETH(WETH).withdraw(amounts[amounts.length - 1]);
        TransferHelper.safeTransferETH(to, amounts[amounts.length - 1]);
    }



    function swapExactTokensForETH(uint amountIn, uint amountOutMin, address[]
calldata path, address to, uint deadline)
        external
        virtual
        override
        ensure(deadline)
        returns (uint[] memory amounts)
    {
        require(path[path.length - 1] == WETH, 'UniswapV2Router: INVALID_PATH');
        amounts = UniswapV2Library.getAmountsOut(factory, amountIn, path);
        require(amounts[amounts.length - 1] >= amountOutMin, 'UniswapV2Router:
INSUFFICIENT_OUTPUT_AMOUNT');
        TransferHelper.safeTransferFrom(
            path[0], msg.sender, UniswapV2Library.pairFor(factory, path[0],
path[1]), amounts[0]
        );
        _swap(amounts, path, address(this));
        IWETH(WETH).withdraw(amounts[amounts.length - 1]);
        TransferHelper.safeTransferETH(to, amounts[amounts.length - 1]);
    }


    function swapETHForExactTokens(uint amountOut, address[] calldata path,
address to, uint deadline)
        external
        virtual
        override
        payable
        ensure(deadline)
        returns (uint[] memory amounts)
    {
        require(path[0] == WETH, 'UniswapV2Router: INVALID_PATH');
        amounts = UniswapV2Library.getAmountsIn(factory, amountOut, path);
        require(amounts[0] <= msg.value, 'UniswapV2Router:
EXCESSIVE_INPUT_AMOUNT');
        IWETH(WETH).deposit{value: amounts[0]}();
        assert(IWETH(WETH).transfer(UniswapV2Library.pairFor(factory, path[0],
path[1]), amounts[0]));
        _swap(amounts, path, to);
```

```
        // refund dust eth, if any
        if (msg.value > amounts[0]) TransferHelper.safeTransferETH(msg.sender,
msg.value - amounts[0]);
    }
```

这四种转换方式都涉及到以太币和代币之间的交易。 唯一不同的是，我们要么从交易者处收到以太币，并使用以太币铸造包装以太币，或者从路径上最后一次交易收到包装以太币， 消耗后将得到的以太币再发送给交易者。

```
    // **** SWAP (supporting fee-on-transfer tokens) ****
    // requires the initial amount to have already been sent to the first pair
    function _swapSupportingFeeOnTransferTokens(address[] memory path, address
_to) internal virtual {
```

此内部函数用于兑换代币，但有转账或存储费用，以解决 （此问题）。

```
        for (uint i; i < path.length - 1; i++) {
            (address input, address output) = (path[i], path[i + 1]);
            (address token0,) = UniswapV2Library.sortTokens(input, output);
            IUniswapV2Pair pair =
IUniswapV2Pair(UniswapV2Library.pairFor(factory, input, output));
            uint amountInput;
            uint amountOutput;
            { // scope to avoid stack too deep errors
            (uint reserve0, uint reserve1,) = pair.getReserves();
            (uint reserveInput, uint reserveOutput) = input == token0 ?
(reserve0, reserve1) : (reserve1, reserve0);
            amountInput =
IERC20(input).balanceOf(address(pair)).sub(reserveInput);
            amountOutput = UniswapV2Library.getAmountOut(amountInput,
reserveInput, reserveOutput);
```

由于有转账费用，我们不能依靠 `getAmountsOut` 函数来告诉我们 每次转账完成后获得的金额（之前我们可以调用原来的 `_swap`）。 相反，我们必须先完成转账然后再查看 我们收回的代币数量。

注意：理论上我们可以使用此函数而非 `_swap`，但在某些情况下（例如， 如果因为在最后无法满足所需最低数额而导致转账撤销），最终会消耗更多 燃料。 转账需要收费的代币很少见，所以，尽管我们需要接纳它们，但不需要让所有的兑换都假定 至少需要兑换一种需要收取转账费用的代币。

```
            }
            (uint amount0Out, uint amount1Out) = input == token0 ? (uint(0),
amountOutput) : (amountOutput, uint(0));
            address to = i < path.length - 2 ? UniswapV2Library.pairFor(factory,
output, path[i + 2]) : _to;
            pair.swap(amount0Out, amount1Out, to, new bytes(0));
        }
    }


    function swapExactTokensForTokensSupportingFeeOnTransferTokens(
        uint amountIn,
        uint amountOutMin,
        address[] calldata path,
```

```
        address to,
        uint deadline
    ) external virtual override ensure(deadline) {
        TransferHelper.safeTransferFrom(
            path[0], msg.sender, UniswapV2Library.pairFor(factory, path[0],
path[1]), amountIn
        );
        uint balanceBefore = IERC20(path[path.length - 1]).balanceOf(to);
        _swapSupportingFeeOnTransferTokens(path, to);
        require(
            IERC20(path[path.length - 1]).balanceOf(to).sub(balanceBefore) >=
amountOutMin,
            'UniswapV2Router: INSUFFICIENT_OUTPUT_AMOUNT'
        );
    }


    function swapExactETHForTokensSupportingFeeOnTransferTokens(
        uint amountOutMin,
        address[] calldata path,
        address to,
        uint deadline
    )
        external
        virtual
        override
        payable
        ensure(deadline)
    {
        require(path[0] == WETH, 'UniswapV2Router: INVALID_PATH');
        uint amountIn = msg.value;
        IWETH(WETH).deposit{value: amountIn}();
        assert(IWETH(WETH).transfer(UniswapV2Library.pairFor(factory, path[0],
path[1]), amountIn));
        uint balanceBefore = IERC20(path[path.length - 1]).balanceOf(to);
        _swapSupportingFeeOnTransferTokens(path, to);
        require(
            IERC20(path[path.length - 1]).balanceOf(to).sub(balanceBefore) >=
amountOutMin,
            'UniswapV2Router: INSUFFICIENT_OUTPUT_AMOUNT'
        );
    }


    function swapExactTokensForETHSupportingFeeOnTransferTokens(
        uint amountIn,
        uint amountOutMin,
        address[] calldata path,
        address to,
        uint deadline
    )
        external
        virtual
        override
        ensure(deadline)
```

```
    {
        require(path[path.length - 1] == WETH, 'UniswapV2Router: INVALID_PATH');
        TransferHelper.safeTransferFrom(
            path[0], msg.sender, UniswapV2Library.pairFor(factory, path[0],
path[1]), amountIn
        );
        _swapSupportingFeeOnTransferTokens(path, address(this));
        uint amountOut = IERC20(WETH).balanceOf(address(this));
        require(amountOut >= amountOutMin, 'UniswapV2Router:
INSUFFICIENT_OUTPUT_AMOUNT');
        IWETH(WETH).withdraw(amountOut);
        TransferHelper.safeTransferETH(to, amountOut);
    }
```

这些方式与用于普通代币的相同，区别在于它们调用的是`_swapSupportingFeeOnTransferTokens`。

```
    // **** LIBRARY FUNCTIONS ****
    function quote(uint amountA, uint reserveA, uint reserveB) public pure
virtual override returns (uint amountB) {
        return UniswapV2Library.quote(amountA, reserveA, reserveB);
    }

    function getAmountOut(uint amountIn, uint reserveIn, uint reserveOut)
        public
        pure
        virtual
        override
        returns (uint amountOut)
    {
        return UniswapV2Library.getAmountOut(amountIn, reserveIn, reserveOut);
    }

    function getAmountIn(uint amountOut, uint reserveIn, uint reserveOut)
        public
        pure
        virtual
        override
        returns (uint amountIn)
    {
        return UniswapV2Library.getAmountIn(amountOut, reserveIn, reserveOut);
    }

    function getAmountsOut(uint amountIn, address[] memory path)
        public
        view
        virtual
        override
        returns (uint[] memory amounts)
    {
        return UniswapV2Library.getAmountsOut(factory, amountIn, path);
    }

    function getAmountsIn(uint amountOut, address[] memory path)
        public
        view
```

```
        virtual
        override
        returns (uint[] memory amounts)
    {
        return UniswapV2Library.getAmountsIn(factory, amountOut, path);
    }
}
```

这些函数仅仅是调用 UniswapV2Library 函数的代理。

UniswapV2Migrator.sol {#UniswapV2Migrator}

这个合约用于将交易从旧版 v1 迁移至 v2。 目前版本已经迁移，便不再相关。

程序库 {#libraries}

SafeMath 库是一个文档很完备的程序库，这里 便无需赘述了。

数学 {#Math}

此库包含一些 Solidity 代码通常不需要的数学函数，因而它们不是 Solidity 语言的一部分。

```
pragma solidity =0.5.16;

// a library for performing various math operations

library Math {
    function min(uint x, uint y) internal pure returns (uint z) {
        z = x < y ? x : y;
    }

    // babylonian method
(https://wikipedia.org/wiki/Methods_of_computing_square_roots#Babylonian_method)
    function sqrt(uint y) internal pure returns (uint z) {
        if (y > 3) {
            z = y;
            uint x = y / 2 + 1;
```

首先赋予 x 一个大于平方根的估值（这是我们需要把 1-3 当作特殊情况处理的原因）。

```
            while (x < z) {
                z = x;
                x = (y / x + x) / 2;
```

获取一个更接近的估值，即前一个估值与我们试图找到的方根值的平均数除以 前一个估值。 重复计算，直到新的估值不再低于现有估值。 欲了解更多详情， 请参见此处。

```
            }
        } else if (y != 0) {
            z = 1;
```

我们永远不需要零的平方根。 1、2 和 3 的平方根大致为 1（我们使用的是 整数，所以忽略分数）。

```
        }
    }
}
```

定点小数 (UQ112x112) {#FixedPoint}

该库处理小数，这些小数通常不属于以太坊计算的一部分。 为此，它将数值 *x* 编码为 *x*2^112*。 这使我们能够使用原来的加法和减法操作码，无需更改。

```
pragma solidity =0.5.16;

// a library for handling binary fixed point numbers
(https://wikipedia.org/wiki/Q_(number_format))

// range: [0, 2**112 - 1]
// resolution: 1 / 2**112

library UQ112x112 {
    uint224 constant Q112 = 2**112;
```

`Q112` 是 1 的编码。

```
    // encode a uint112 as a UQ112x112
    function encode(uint112 y) internal pure returns (uint224 z) {
        z = uint224(y) * Q112; // never overflows
    }
```

因为 y 是 `uint112`，所以最多可以是 2^112-1。 该数值还可以编码为 `UQ112x112`。

```
    // divide a UQ112x112 by a uint112, returning a UQ112x112
    function uqdiv(uint224 x, uint112 y) internal pure returns (uint224 z) {
        z = x / uint224(y);
    }
}
```

如果我们需要两个 `UQ112x112` 值相除，结果不需要再乘以 2^112。 因此， 我们为分母取一个整数。 我们需要使用类似的技巧来做乘法，但不需要将 `UQ112x112` 的值相乘。

UniswapV2Library {#uniswapV2library}

此库仅被外围合约使用

```
pragma solidity >=0.5.0;

import '@uniswap/v2-core/contracts/interfaces/IUniswapV2Pair.sol';

import "./SafeMath.sol";

library UniswapV2Library {
    using SafeMath for uint;
```

```
    // returns sorted token addresses, used to handle return values from pairs
sorted in this order
    function sortTokens(address tokenA, address tokenB) internal pure returns
(address token0, address token1) {
        require(tokenA != tokenB, 'UniswapV2Library: IDENTICAL_ADDRESSES');
        (token0, token1) = tokenA < tokenB ? (tokenA, tokenB) : (tokenB,
tokenA);
        require(token0 != address(0), 'UniswapV2Library: ZERO_ADDRESS');
    }
```

按地址对这两个代币排序，所以我们将能够获得相应的配对交易地址。 这很有必要， 否则有两种可能性，一种是用于参数 A,B，而另一种是用于 参数 B,A，导致两次交易而非一个。

```
    // calculates the CREATE2 address for a pair without making any external
calls
    function pairFor(address factory, address tokenA, address tokenB) internal
pure returns (address pair) {
        (address token0, address token1) = sortTokens(tokenA, tokenB);
        pair = address(uint(keccak256(abi.encodePacked(
                hex'ff',
                factory,
                keccak256(abi.encodePacked(token0, token1)),

 hex'96e8ac4277198ff8b6f785478aa9a39f403cb768dd02cbee326c3e7da348845f' // init
code hash
            ))));
    }
```

此函数计算两种代币的配对交易地址。 此合约使用 CREATE2 操作码创建，如果我们知道所使用的参数， 我们可以使用相同的算法计算地址。 这比查询工厂便宜得多，而且

```
    // fetches and sorts the reserves for a pair
    function getReserves(address factory, address tokenA, address tokenB)
internal view returns (uint reserveA, uint reserveB) {
        (address token0,) = sortTokens(tokenA, tokenB);
        (uint reserve0, uint reserve1,) = IUniswapV2Pair(pairFor(factory,
tokenA, tokenB)).getReserves();
        (reserveA, reserveB) = tokenA == token0 ? (reserve0, reserve1) :
(reserve1, reserve0);
    }
```

此函数返回配对交易所拥有的两种代币的储备金。 请注意，它可以任意顺序接收代币， 并将其排序，以便内部使用。

```
    // given some amount of an asset and pair reserves, returns an equivalent
amount of the other asset
    function quote(uint amountA, uint reserveA, uint reserveB) internal pure
returns (uint amountB) {
        require(amountA > 0, 'UniswapV2Library: INSUFFICIENT_AMOUNT');
        require(reserveA > 0 && reserveB > 0, 'UniswapV2Library:
INSUFFICIENT_LIQUIDITY');
        amountB = amountA.mul(reserveB) / reserveA;
    }
```

如果不涉及交易费用的话，此函数将返回给您代币 A 兑换得到的代币 B。 此计算 考虑到转账可能会改变汇率。

```
    // given an input amount of an asset and pair reserves, returns the maximum
output amount of the other asset
    function getAmountOut(uint amountIn, uint reserveIn, uint reserveOut)
internal pure returns (uint amountOut) {
```

如果使用配对交易没有手续费，上述 `quote` 函数非常有效。 然而，如果有 0.3% 的 手续费，您实际得到的金额就会低于此值。 此函数可以计算缴纳交易费用后的金额。

```
        require(amountIn > 0, 'UniswapV2Library: INSUFFICIENT_INPUT_AMOUNT');
        require(reserveIn > 0 && reserveOut > 0, 'UniswapV2Library:
INSUFFICIENT_LIQUIDITY');
        uint amountInWithFee = amountIn.mul(997);
        uint numerator = amountInWithFee.mul(reserveOut);
        uint denominator = reserveIn.mul(1000).add(amountInWithFee);
        amountOut = numerator / denominator;
    }
```

Solidity 本身不能进行小数计算，所以不能简单地将金额乘以 0.997。 作为替代方法， 我们将分子乘以 997，分母乘以 1000，也能取得相同的效果。

```
    // given an output amount of an asset and pair reserves, returns a required
input amount of the other asset
    function getAmountIn(uint amountOut, uint reserveIn, uint reserveOut)
internal pure returns (uint amountIn) {
        require(amountOut > 0, 'UniswapV2Library: INSUFFICIENT_OUTPUT_AMOUNT');
        require(reserveIn > 0 && reserveOut > 0, 'UniswapV2Library:
INSUFFICIENT_LIQUIDITY');
        uint numerator = reserveIn.mul(amountOut).mul(1000);
        uint denominator = reserveOut.sub(amountOut).mul(997);
        amountIn = (numerator / denominator).add(1);
    }
```

此函数大致完成相同的功能，但它会获取输出数额并提供输入代币的数量。

```
    // performs chained getAmountOut calculations on any number of pairs
    function getAmountsOut(address factory, uint amountIn, address[] memory
path) internal view returns (uint[] memory amounts) {
        require(path.length >= 2, 'UniswapV2Library: INVALID_PATH');
        amounts = new uint[](path.length);
        amounts[0] = amountIn;
        for (uint i; i < path.length - 1; i++) {
            (uint reserveIn, uint reserveOut) = getReserves(factory, path[i],
path[i + 1]);
            amounts[i + 1] = getAmountOut(amounts[i], reserveIn, reserveOut);
        }
    }

    // performs chained getAmountIn calculations on any number of pairs
```

```
    function getAmountsIn(address factory, uint amountOut, address[] memory
path) internal view returns (uint[] memory amounts) {
        require(path.length >= 2, 'UniswapV2Library: INVALID_PATH');
        amounts = new uint[](path.length);
        amounts[amounts.length - 1] = amountOut;
        for (uint i = path.length - 1; i > 0; i--) {
            (uint reserveIn, uint reserveOut) = getReserves(factory, path[i -
1], path[i]);
            amounts[i - 1] = getAmountIn(amounts[i], reserveIn, reserveOut);
        }
    }
}
```

在需要进行数次配对交易时，可以通过这两个函数获得相应数值。

转账帮助 {#transfer-helper}

此库添加了围绕 ERC-20 和以太坊转账的成功检查，并以同样的方式处理回退和返回 `false` 值。

```
// SPDX-License-Identifier: GPL-3.0-or-later

pragma solidity >=0.6.0;

// helper methods for interacting with ERC20 tokens and sending ETH that do not
consistently return true/false
library TransferHelper {
    function safeApprove(
        address token,
        address to,
        uint256 value
    ) internal {
        // bytes4(keccak256(bytes('approve(address,uint256)')));
        (bool success, bytes memory data) =
token.call(abi.encodeWithSelector(0x095ea7b3, to, value));
```

我们可以通过以下两种方式调用不同的合约：

- 使用一个接口定义创建函数调用
- 使用 应用程序二进制接口 (ABI)“手动” 创建调用。 这是代码作者的决定。

```
        require(
            success && (data.length == 0 || abi.decode(data, (bool))),
            'TransferHelper::safeApprove: approve failed'
        );
    }
```

为了与之前的 ERC-20 标准创建的代币反向兼容，ERC-20 调用 失败可能有两种情况：回退（在这种情况下 `success` 即是 `false`），或者调用成功但返回 `false` 值（在这种情况下有输出数据，将其解码为布尔值，会得到 `false`）。

```
    function safeTransfer(
```

```
        address token,
        address to,
        uint256 value
    ) internal {
        // bytes4(keccak256(bytes('transfer(address,uint256)')));
        (bool success, bytes memory data) =
token.call(abi.encodeWithSelector(0xa9059cbb, to, value));
        require(
            success && (data.length == 0 || abi.decode(data, (bool))),
            'TransferHelper::safeTransfer: transfer failed'
        );
    }
```

此函数实现了 ERC-20 的转账功能， 可使一个帐户花掉由不同帐户所提供的额度。

```
    function safeTransferFrom(
        address token,
        address from,
        address to,
        uint256 value
    ) internal {
        // bytes4(keccak256(bytes('transferFrom(address,address,uint256)')));
        (bool success, bytes memory data) =
token.call(abi.encodeWithSelector(0x23b872dd, from, to, value));
        require(
            success && (data.length == 0 || abi.decode(data, (bool))),
            'TransferHelper::transferFrom: transferFrom failed'
        );
    }
```

此函数实现了 ERC-20 的 transferFrom 功能， 可使一个帐户花掉由不同帐户所提供的额度。

```
    function safeTransferETH(address to, uint256 value) internal {
        (bool success, ) = to.call{value: value}(new bytes(0));
        require(success, 'TransferHelper::safeTransferETH: ETH transfer
failed');
    }
}
```

此函数将以太币转至一个帐户。 任何对不同合约的调用都可以尝试发送以太币。 因为我们 实际上不需要调用任何函数，就不需要在调用中发送数据。

结论 {#conclusion}

本篇文章较长，约有 50 页。 如果您已读到此处，恭喜您！ 希望您现在已经了解 编写真实应用程序（相对于短小的示例程序）的考虑因素，并且能够更好地为您自己的 用例编写合约。

现在去写点实用的东西吧，希望您能给我们惊喜。

title: 使用WebSocket
description: 有关使用WebSocket和Alchemy进行JSON-RPC请求并订阅事件的指南。
author: "Elan Halpern"
lang: zh
sidebar: true
tags:

- "alchemy"
- "websockets"
- "查询"
- "入门指南"
- "订阅"
- "javascript"
  skill: 初学者
  source: Alchemy文档
  sourceUrl: https://docs.alchemyapi.io/guides/using-websockets
  published: 2020-12-01

这是有关使用 WebSocket 和 Alchemy 向以太坊区块链发出请求的入门级指南。

WebSocket 与 HTTP {#websockets-vs-http}

与 HTTP 不同的是，使用 WebSocket，您无需要需要特定信息时持续进行请求。 WebSocket 为您维护网络连接（如果操作正确）并侦听变化。

与任何网络连接一样，您不应该假设 WebSocket 将永远保持打开状态而不会中断，但是正确地手动处理丢弃的连接和重新连接可能很难做到正确。 WebSocket 的另一个缺点是，您在响应中不会得到 HTTP 状态码，而只会得到错误消息。

Alchemy Web3自动添加对 WebSocket 失败的处理并重试，而无需进行配置。

试试看 {#try-it-out}

测试 WebSocket 最简单的方式是安装命令行工具来进行 WebSocket 请求，例如wscat。 使用 wscat，您可以发送如下请求：

注意：如果您有 Alchemy 帐户，则可以将 `demo` 替换成自己的 API 密钥。 在此处注册免费的 Alchemy 帐户！

```
$ wscat -c wss://eth-mainnet.ws.alchemyapi.io/ws/demo

>  {"jsonrpc":  "2.0", "id": 0, "method":  "eth_gasPrice"}

<  {"jsonrpc":  "2.0", "result":  "0xb2d05e00", "id": 0}
```

如何使用 WebSocket {#how-to-use-websockets}

首先，使用应用的 WebSocket URL 打开 WebSocket 您可以在您的仪表板中打开应用的页面并点击“查看密钥”来找到您的应用的 WebSocket URL。 请注意，您的应用的 WebSocket URL 与其 HTTP 请求的 URL 不同，但两者都可以通过点击“查看密钥” 找到。

在您的Alchemy仪表板中的何处可找到您的WebSocket URL

Alchemy API 参考中列出的的任何 API 都可以通过 WebSocket 使用。 为此，请使用与 HTTP POST 请求正文相同的有效载荷，而不是通过 WebSocket 发送该有效负载。

使用 Web3 {#with-web3}

在使用像 Web3 这样的客户端库时过渡到 WebSocket 是很简单的。 在实例化您的 Web3 客户端时，只需传递 WebSocket URL 而不是 HTTP URL。 例如：

```
const web3 = new Web3("wss://eth-mainnet.ws.alchemyapi.io/ws/your-api-key")

web3.eth.getBlockNumber().then(console.log) // -> 7946893
```

订阅 API {#subscription-api}

当通过 WebSocket 连接时，您可以使用两个额外的方法：`eth_subscribe`和`eth_unsubscribe`。 这些方法将允许您侦听特定事件并立即收到通知。

`eth_subscribe` {#eth-subscribe}

为指定的事件创建新的订阅。 详细了解`eth_subscribe`。

参数 {#parameters}

1. 订阅类型
2. 可选参数

第一个参数指定要侦听的事件类型。 第二个参数包含其他选项，具体取决于第一个参数。 不同的描述类型，其选项和事件有效载荷描述如下。

返回 {#returns}

订阅 ID：此 ID 将附加到任何收到的事件 并且也可用于通过`eth_unsubscribe`取消订阅。

订阅事件 {#subscription-events}

当订阅处于活动状态时，您将收到包含以下字段的对象事件：

- `jsonrpc`：始终为“2.0”
- `method`：始终为“eth_subscription”
- `params`：具有以下字段的对象：
  - `subscription`：由创建此订阅的`eth_subscription`调用返回的订阅 ID。
  - `result`：其内容因订阅类型而异的对象。

订阅类型 {#subscription-types}

1. `alchemy_newFullPendingTransactions`

返回被添加到待定状态的所有交易的交易信息。 此订阅类型订阅待处理交易，类似于标准 Web3 调用 <`web3.eth.subscribe("pendingTransactions")`，但不同之处是它发出*完整的交易信息*，而不只是交易哈希。

示例：

```
> {"jsonrpc": "2.0", "id": 1, "method": "eth_subscribe", "params":
["alchemy_newFullPendingTransactions"]}

< {"id":1,"result":"0x9a52eeddc2b289f985c0e23a7d8427c8","jsonrpc":"2.0"}
< {
      "jsonrpc":"2.0",
      "method":"eth_subscription",
      "params":{
          "result":{
          "blockHash":null,
          "blockNumber":null,
          "from":"0xa36452fc31f6f482ad823cd1cf5515177d57667f",
          "gas":"0x1adb0",
          "gasPrice":"0x7735c4d40",

 "hash":"0x50bff0736c713458c92dd1848d12f3354149be1363123dae35e94e0f2a9d56bf",
"input":"0xa9059cbb000000000000000000000000d0707963952f2fba59dd06f2b425ace40b49
2fe0000000000000000000000000000000000000000000015b1111266cfca100000",
          "nonce":"0x0",
          "to":"0xea38eaa3c86c8f9b751533ba2e562deb9acded40",
          "transactionIndex":null,
          "value":"0x0",
          "v":"0x26",

 "r":"0x195c2c1ed126088e12d290aa93541677d3e3b1d10f137e11f86b1b9227f01e3b",

 "s":"0x60fc4edbf1527832a2a36dbc1e63ed6193a6eee654472fbebbf88ef1750b5344"},
          "subscription":"0x9a52eeddc2b289f985c0e23a7d8427c8"
      }
  }
```

2. `newHeads`

每当向链中添加新标头时（包括在链重组期间）都会发出事件。

发生链重组时，此订阅将发出一个事件，其中包含新链的所有新标头。 具体地说，这意味着您可能会看到多个高度相同的标头，当发生这种情况时，应该将后一个标头视为重组后的正确标头。

示例：

```
> {"jsonrpc": "2.0", "id": 1, "method": "eth_subscribe", "params":
["newHeads"]}

< {"jsonrpc":"2.0","id":2,"result":"0x9ce59a13059e417087c02d3236a0b1cc"}
< {
  "jsonrpc": "2.0",
  "method": "eth_subscription",
  "params": {
      "result": {
          "difficulty": "0x15d9223a23aa",
          "extraData":
 "0xd983010305844765746887676f312e342e328777696e646f7773",
          "gasLimit": "0x47e7c4",
          "gasUsed": "0x38658",
          "logsBloom":
```

```
"0x0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000",
          "miner":  "0xf8b483dba2c3b7176a3da549ad41a48bb3121069",
          "nonce":  "0x084149998194cc5f",
          "number":  "0x1348c9",
          "parentHash":
 "0x7736fab79e05dc611604d22470dadad26f56fe494421b5b333de816ce1f25701",
          "receiptRoot":
 "0x2fab35823ad00c7bb388595cb46652fe7886e00660a01e867824d3dceb1c8d36",
          "sha3Uncles":
 "0x1dcc4de8dec75d7aab85b567b6ccd41ad312451b948a7413f0a142fd40d49347",
          "stateRoot":
 "0xb3346685172db67de536d8765c43c31009d0eb3bd9c501c9be3229203f15f378",
          "timestamp":  "0x56ffeff8",
          "transactionsRoot":
 "0x0167ffa60e3ebc0b080cdb95f7c0087dd6c0e61413140e39d94d3468d7c9689f"
      },
  "subscription":  "0x9ce59a13059e417087c02d3236a0b1cc"
  }
}
```


3. `logs`

发出日志，这些日志是符合指定筛选条件的新添加区块的一部分。

当发生链重组时，作为旧链上区块的一部分的日志将再次发出，并将属性`removed`设置为`true`。 此外，作为新链上区块的一部分的日志被发出，这意味着在重组的情况下可以多次看到同一交易的日志。

参数

1. 具有以下字段的对象：
   - `adddress` (可选)：表示地址的字符串或此类字符串数组。
     - 只会发出从这些地址之一创建的日志。
   - `topics`：主题说明符的数组。
     - 每个主题说明符都是`null`、表示主题的字符串或字符串数组。
     - 数组中非`null`的每个位置将发出的日志仅限制为在该位置具有给定主题之一的那些位置。

主题规范的一些示例：

- `[]`：任何允许的主题。
- `[A]`：A 位置第一个位置（以及后面任何位置）。
- `[null, B]`：任何内容位置第一个位置的，B 位于第二个位置（以及后面任何位置）。
- `[A, B]`：A 位于第一个位置的，B 位于第二个位置（以及后面任何位置）。
- `[[A、B]、[A、B]]`：（A 或 B）位于第一个位置，（A 或 B）位于第二个位置（以及后面任何位置）。

示例：

```
> {"jsonrpc": "2.0", "id": 1, "method": "eth_subscribe", "params":
["logs", {"address": "0x8320fe7702b96808f7bbc0d4a888ed1468216cfd", "topics":
["0xd78a0cb8bb633d06981248b816e7bd33c2a35a6089241d099fa519e361cab902"]}]}

< {"jsonrpc":"2.0","id":2,"result":"0x4a8a4c0517381924f9838102c5a4dcb7"}
< {
  "jsonrpc": "2.0",
  "method": "eth_subscription",
  "params": {
      "subscription": "0x4a8a4c0517381924f9838102c5a4dcb7",
      "result": {
          "address": "0x8320fe7702b96808f7bbc0d4a888ed1468216cfd",
          "blockHash":
 "0x61cdb2a09ab99abf791d474f20c2ea89bf8de2923a2d42bb49944c8c993cbf04",
          "blockNumber": "0x29e87",
          "data":
"0x0000000000000000000000000000000000000000000000000000000000000010000000000000000
0000000000000000000000000000000000000000000000000000000000000003",
          "logIndex":"0x0",
          "topics":
["0xd78a0cb8bb633d06981248b816e7bd33c2a35a6089241d099fa519e361cab902"],
          "transactionHash":
 "0xe044554a0a55067caafd07f8020ab9f2af60bdfe337e395ecd84b4877a3d1ab4",
          "transactionIndex": "0x0"
      }
  }
}
```

`eth_unsubscribe` {#eth-unsubscribe}

取消一个现有的订阅，以便不再发送任何事件。

参数

1. 先前从 `eth_comment` 调用返回的订阅 ID。

返回

如果订阅成功取消则为 `true`，或者，如果没有具有给定 ID 的订阅，则为 `false`。

示例：

请求

```
curl https://eth-mainnet.alchemyapi.io/v2/your-api-key
-X POST
-H "Content-Type: application/json"
-d '{"id": 1, "method": "eth_unsubscribe", "params":
["0x9cef478923ff08bf67fde6c64013158d"]}'
```

结果

```
{
  "jsonrpc": "2.0",
  "id": 1,
  "result": true
}
```

免费注册 Alchemy，查看我们的文档，如需了解最新消息，请在Twitter上关注我们。

title: "Waffle：动态模拟和测试合约调用"
description: 使用动态模拟和测试合约调用的高级 Waffle 教程
author: "Daniel Izdebski"
tags:

- "waffle"
- "智能合约"
- "solidity"
- "测试"
- "模拟"

  skill: 中等难度
  lang: zh
  sidebar: true
  published: 2020-11-14

本教程是关于什么的？ {#what-is-this-tutorial-about}

在本教程中，您将学习如何：

- 使用动态模拟
- 测试智能合约之间的交互

本文假定：

- 您已经知道如何使用 `Solidity` 编写一个简单的智能合约
- 您熟悉 `JavaScript` 和 `TypeScript`
- 您已经完成其他 `Waffle` 教程或对其略知一二。

动态模拟 {#dynamic-mocking}

为什么动态模拟很有用？ 它允许我们编写单元测试，而不是集成测试。 这是什么意思呢？ 这意味着我们不必担心智能合约的依赖项，因此我们可以完全孤立地测试所有合约。 让我演示一下如何才能实现。

1. 项目 {#1-project}

在开始之前，我们需要准备一个简单的 node.js 项目：

```
$ mkdir dynamic-mocking
$ cd dynamic-mocking
$ mkdir contracts src

$ yarn init
# or if you're using npm
$ npm init
```

让我们从添加 typescript 和测试依赖项开始 -mocha & chai：

```
$ yarn add --dev @types/chai @types/mocha chai mocha ts-node typescript
# or if you're using npm
$ npm install @types/chai @types/mocha chai mocha ts-node typescript --save-dev
```

现在让我们添加 `Waffle` 和 `Ethers`：

```
$ yarn add --dev ethereum-waffle ethers
# or if you're using npm
$ npm install ethereum-waffle ethers --save-dev
```

您的项目结构现在应该如下所示：

```
.
├── contracts
├── package.json
└── test
```

2. 智能合约 {#2-smart-contract}

要开始动态模拟，我们需要一个包含依赖项的智能合约。 别担心，我会帮助您的！

这是一个用 `Solidity` 编写的简单智能合约，其唯一目的是检查我们是否有很多钱。 它使用 ERC20 代币来检查我们是否有足够的代币。 将其放在 `./contracts/AmIRichAlready.sol` 中。

```
pragma solidity ^0.6.2;

interface IERC20 {
    function balanceOf(address account) external view returns (uint256);
}

contract AmIRichAlready {
    IERC20 private tokenContract;
    uint public richness = 1000000 * 10 ** 18;

    constructor (IERC20 _tokenContract) public {
        tokenContract = _tokenContract;
    }

    function check() public view returns (bool) {
        uint balance = tokenContract.balanceOf(msg.sender);
        return balance > richness;
    }
}
```

因为我们想使用动态模拟，所以我们不需要整个 ERC20，这就是为什么我们使用只有一个函数的 IERC20 接口的原因。

现在是构建这个合约的时候了！ 为此，我们将使用 `Waffle`。 首先，我们将创建一个简单的 `waffle.json` 配置文件，它指定了编译选项。

```
{
  "compilerType": "solcjs",
  "compilerVersion": "0.6.2",
  "sourceDirectory": "./contracts",
  "outputDirectory": "./build"
}
```

现在我们已经准备好使用 Waffer 构建合约：

```
$ npx waffle
```

很简单，对吗？ 在 `build/` 文件夹中，出现了与合约和接口相对应的两个文件。 我们稍后将使用它们进行测试。

3. 测试 {#3-testing}

让我们创建一个名为 `AmIRichAlready.test.ts` 的文件来进行实际测试。 首先，我们必须处理导入。 我们稍后将需要它们：

```
import { expect, use } from "chai"
import { Contract, utils, Wallet } from "ethers"
import {
  deployContract,
  deployMockContract,
  MockProvider,
  solidity,
} from "ethereum-waffle"
```

除了 JS 依赖项之外，我们需要导入我们创建的合约和接口。

```
import IERC20 from "../build/IERC20.json"
import AmIRichAlready from "../build/AmIRichAlready.json"
```

Waffle 使用 `chai` 进行测试。 但是，在我们使用它之前，我们必须将 Waffle 的匹配器注入到 chai 本身：

```
use(solidity)
```

我们需要实现 `beforeEach()` 函数，它将在每次测试前重置合约的状态。 让我们先想想我们需要什么。 要部署一个合约，我们需要两样东西：一个钱包和一个已部署的 ERC20 合约，以便将其作为 `AmIRichAlready` 合约的参数传递。

首先，我们创建一个钱包：

```
const [wallet] = new MockProvider().getWallets()
```

然后我们需要部署一个 ERC20 合约。 这里是棘手的部分 -- 我们只有一个接口。 这就是使用 Waffle 来帮助我们的部分。 Waffle 有一个神奇的 `deployMockContract()` 函数，它只使用接口的 *abi* 来创建合约。

```
const mockERC20 = await deployMockContract(wallet, IERC20.abi)
```

现在有了钱包和部署的 ERC20，我们就可以继续部署 `AmIRichAlready` 合约。

```
const contract = await deployContract(wallet, AmIRichAlready, [
  mockERC20.address,
])
```

做完这些，我们的 `beforeEach()` 函数就完成了。 到目前为止，您的 `AmIRichAlready.test.ts` 文件看起来应如下所示：

```
import { expect, use } from "chai"
import { Contract, utils, Wallet } from "ethers"
import {
  deployContract,
  deployMockContract,
  MockProvider,
  solidity,
} from "ethereum-waffle"

import IERC20 from "../build/IERC20.json"
import AmIRichAlready from "../build/AmIRichAlready.json"

use(solidity)

describe("Am I Rich Already", () => {
  let mockERC20: Contract
  let contract: Contract
  let wallet: Wallet

  beforeEach(async () => {
    ;[wallet] = new MockProvider().getWallets()
    mockERC20 = await deployMockContract(wallet, IERC20.abi)
    contract = await deployContract(wallet, AmIRichAlready, [mockERC20.address])
  })
})
```

让我们为 `AmIRichAlready` 合约编写第一个测试。 您认为我们的测试应该是关于什么的？ 没错，您是对的！ 我们应该检查我们是否有很多钱：）

但是等一下。 我们的模拟合约怎么知道要返回哪些值呢？ 我们还没有实现 `balanceOf()` 函数的任何逻辑。 同样，Waffle 在这里也可以提供帮助。 我们的模拟合约现在有了一些新花招：

```
await mockERC20.mock.<nameOfMethod>.returns(<value>)
await mockERC20.mock.<nameOfMethod>.withArgs(<arguments>).returns(<value>)
```

有了这些知识，我们终于可以编写出我们的第一个测试啦：

```
it("returns false if the wallet has less than 1000000 tokens", async () => {
  await mockERC20.mock.balanceOf.returns(utils.parseEther("999999"))
  expect(await contract.check()).to.be.equal(false)
})
```

让我们把这个测试分解成几个部分：

1. 将我们的模拟 ERC20 合约设置为始终返回 99999 个代币的余额。
2. 检查 `contract.check()` 方法是否返回 `false`。

我们已经准备好启动这个家伙了：

一次测试通过

所以这个测试是有效的，但是，还是有一些改进的余地。 `balanceOf()` 函数将始终返回 99999。 我们可以通过指定一个钱包来改进它，该函数应该为它返回一些东西 -- 就像一个真正的合约。

```
it("returns false if the wallet has less than 1000001 tokens", async () => {
  await mockERC20.mock.balanceOf
    .withArgs(wallet.address)
    .returns(utils.parseEther("999999"))
  expect(await contract.check()).to.be.equal(false)
})
```

到目前为止，我们只测试了我们不够有钱的情况。 让我们来测试一下相反的情况：

```
it("returns true if the wallet has at least 1000001 tokens", async () => {
  await mockERC20.mock.balanceOf
    .withArgs(wallet.address)
    .returns(utils.parseEther("1000001"))
  expect(await contract.check()).to.be.equal(true)
})
```

您运行测试...

两次测试通过

...您已经到这儿啦！ 我们的合约似乎按计划进行 :)

测试合约调用 {#testing-contract-calls}

让我们总结一下到目前为止所做的事情。 我们已经测试了我们的 `AmIRichAlready` 合约的功能，它似乎正常工作。 这意味着我们已经完成了，对吧？ 并非如此！ Waffle 允许我们进一步测试合约。 但是具体怎么做呢？ 那么，在 Waffle 的武器库中，有一个 `calledOnContract()` 和 `calledOnContractWith()` 匹配器。 他们允许我们检查，我们的合约是否调用了 ERC20 模拟合约。 下面是使用其中一个匹配器的基本测试：

```
it("checks if contract called balanceOf on the ERC20 token", async () => {
  await mockERC20.mock.balanceOf.returns(utils.parseEther("999999"))
  await contract.check()
  expect("balanceOf").to.be.calledOnContract(mockERC20)
})
```

我们可以更进一步，用我告诉您的另一个匹配器改进这个测试：

```
it("checks if contract called balanceOf with certain wallet on the ERC20 token",
async () => {
  await mockERC20.mock.balanceOf
    .withArgs(wallet.address)
    .returns(utils.parseEther("999999"))
  await contract.check()
  expect("balanceOf").to.be.calledOnContractWith(mockERC20, [wallet.address])
})
```

让我们检查测试是否正确：

三次测试通过

太好了，所有测试都通过了。

用 Waffle 测试智能合约调用非常容易。 而这是最精彩的部分。 这些匹配器对正常合约和模拟合约都有效！ 这是因为 Waffle 记录并过滤 EVM 调用，而不是像其他技术的流行测试库那样注入代码。

最后 {#the-finish-line}

恭喜！ 现在您知道如何使用 Waffle 来测试合约调用和动态模拟智能合约了。 还有更多有趣的功能等着我们去发现。 我建议您深入研究 Waffle 文档。

Waffle 的文档可在此处获得。

本教程的源代码可以在此处找到。

您可能还对这些教程感兴趣：

- 使用 Waffle 测试智能合约

---

title: "Waffle使用hardhat和ethers设置hello world的教程"
description: 使用hardhat和ethers.js制作您的第一个Waffle项目
author: "MiZiet"
tags:

- "waffle"
- "智能合约"
- "solidity"
- "测试"
- "hardhat"
- "ethers.js"
  skill: 入门级
  lang: zh
  sidebar: true
  published: 2020-10-16

在这个 Waffle 教程中，我们将学习如何使用 hardhat 和 ethers.js 设置一个简单的“Hello world”智能合约项目。 然后我们将学习如何向我们的智能合约中添加一个新功能，以及如何使用 Waffle 测试它。

让我们从创建新项目开始：

```
yarn init
```

或

```
npm init
```

并安装所需的软件包：

```
yarn add -D hardhat
@nomiclabs/hardhat-ethers ethers
@nomiclabs/hardhat-waffle ethereum-waffle chai
```

或

```
npm install -D hardhat
@nomiclabs/hardhat-ethers ethers
@nomiclabs/hardhat-waffle ethereum-waffle chai
```

下一步是通过运行 `npx hardhat` 创建一个示例 hardhat 项目。

```
888     888                        888 888                888
888     888                        888 888                888
888     888                        888 888                888
8888888888  8888b.  888d888 .d88888 88888b.   8888b.  888888
888     888     "88b 888P"  d88" 888 888 "88b     "88b 888
888     888 .d888888 888    888  888 888  888 .d888888 888
888     888 888  888 888    Y88b 888 888  888 888  888 Y88b.
888     888 "Y888888 888     "Y88888 888  888 "Y888888  "Y888

👷 Welcome to Hardhat v2.0.3 👷‍♂️

? What do you want to do? …
Bachan
Mise en contexte. Est ce des lignes de code, dans ce cas ce n'est pas
traduisible
❯ Create a sample project
Create an empty hardhat.config.js
Quit
```

选择 `Create a sample project`（创建示例项目）

我们的项目结构应如下所示：

```
MyWaffleProject
├── contracts
|   └── Greeter.sol
├── node_modules
├── scripts
|   └── sample-script.js
├── test
|   └── sample-test.js
├── .gitattributs
├── .gitignore
├── hardhat.config.js
└── package.json
```

现在让我们来谈谈其中一些文件： {#now-lets-talk}

- Greeter.sol - 我们的智能合约是用 solidity 编写的；

```
contract Greeter {
string greeting;

constructor(string memory _greeting) public {
console.log("Deploying a Greeter with greeting:", _greeting);
greeting = _greeting;
}

function greet() public view returns (string memory) {
return greeting;
}

function setGreeting(string memory _greeting) public {
console.log("Changing greeting from '%s' to '%s'", greeting, _greeting);
greeting = _greeting;
}
}
```

我们的智能合约可以分为三个部分：

1. 构造函数 - 我们在其中声明一个字符串类型变量，名为 `greeting`，
2. 函数 greet - 调用时返回 `greeting` 的函数，
3. 函数 setGreeting - 允许我们更改 `greeting` 值的函数。

- sample-test.js - 我们的测试文件

```
describe("Greeter", function () {
  it("Should return the new greeting once it's changed", async function () {
    const Greeter = await ethers.getContractFactory("Greeter")
    const greeter = await Greeter.deploy("Hello, world!")

    await greeter.deployed()
    expect(await greeter.greet()).to.equal("Hello, world!")

    await greeter.setGreeting("Hola, mundo!")
    expect(await greeter.greet()).to.equal("Hola, mundo!")
  })
})
```

下一步是编译我们的合约并运行测试： {#compiling-and-testing}

Waffle 测试使用 Mocha（测试框架）与 Chai（一个断言库）。 您只需运行 `npx hardhat test` 并等待以下消息出现。

```
✓ Should return the new greeting once it's changed（更改后应立即返回新问候语）
```

到目前为止，一切看起来都很好，让我们为我们的项目增加一些复杂性吧 {#adding-complexity}

想象一下这种情况，有人添加了一个空字符串作为问候语。 这不是一种热情的问候，对吗？
让我们确保这一点不会发生：

当有人传递空字符串时，我们想使用 solidity 的 `revert`。 一件好事是，我们可以使用 Waffle 的 chai 匹配器 `to.be.revertedWith()` 轻松测试此功能。

```
it("Should revert when passing an empty string", async () => {
  const Greeter = await ethers.getContractFactory("Greeter")
  const greeter = await Greeter.deploy("Hello, world!")

  await greeter.deployed()
  await expect(greeter.setGreeting("")).to.be.revertedWith(
    "Greeting should not be empty"
  )
})
```

看起来我们的新测试没通过：

```
Deploying a Greeter with greeting: Hello, world!
Changing greeting from 'Hello, world!' to 'Hola, mundo!'
    ✓ Should return the new greeting once it's changed (1514ms)
Deploying a Greeter with greeting: Hello, world!
Changing greeting from 'Hello, world!' to ''
    1) Should revert when passing an empty string


  1 passing (2s)
  1 failing
```

让我们在智能合约中实现这个功能：

```
require(bytes(_greeting).length > 0, "Greeting should not be empty");
```

现在，我们的 setGreeting 函数如下所示：

```
function setGreeting(string memory _greeting) public {
require(bytes(_greeting).length > 0, "Greeting should not be empty");
console.log("Changing greeting from '%s' to '%s'", greeting, _greeting);
greeting = _greeting;
}
```

让我们再次运行测试：

```
✓ Should return the new greeting once it's changed (1467ms)
✓ Should revert when passing an empty string (276ms)

2 passing (2s)
```

恭喜！ 您做到了:)

结论 {#conclusion}

我们使用 Waffle、Hardhat 和 ethers.js 制作了一个简单的项目。 我们学习了如何设置项目、添加测试和实现新功能。

欲了解更多用于测试您的智能合约的优秀 chai 匹配器，请查看官方 Waffle 文档。

title: 后端应用程序接口库
description: 以太坊客户端应用程序接口的介绍，使您能够从您的应用程序中与区块链进行交互。
lang: zh

sidebar: true

为了使软件应用程序能够与以太坊区块链进行交互（例如：读取区块链数据或发送交易信息到网络），软件必须连接到以太坊节点。

为此目的，每个以太坊客户端都执行 JSON-RPC 规范，所以应用程序可以依赖统一的端点集。

如果您想使用特定的编程语言去连接以太坊的节点，您可自行选择，但是在社区中已有几个方便的库，可以更方便地实现应用程序与以太坊的连接。 通过这些库，开发者可以方便地写下直观的一行函数来初始化（后端的）JSON RPC 请求并用于与以太坊进行交互。

前置要求 {#prerequisites}

了解以太坊堆栈和以太坊客户端可能会对您有所帮助。

为什么要使用库？ {#why-use-a-library}

这些库降低了与一个以太坊节点交互的复杂性。 它们还提供实用的函数（例如：将 ETH 转化为 Gwei），而作为开发者，您可以花费更少的时间来处理以太坊客户端的复杂问题，从而将更多的时间集中于处理您的应用程序的独特功能。

可用的库 {#available-libraries}

Alchemy - ***以太坊开发平台***

- alchemy.com
- 相关文档
- GitHub
- Discord

BlockCypher - ***以太坊网络 API。***

- blockcypher.com
- 相关文档

Infura - ***以太坊应用程序接口即服务。***

- infura.io
- 相关文档
- GitHub

Cloudflare 以太坊网关。

- cloudflare-eth.com

Figment 的数据中心 - ***以太坊主网和测试网的 Web3 应用程序接口服务***

- 数据中心
- 相关文档

Nodesmith - ***可对以太坊主网和测试网进行 JSON-RPC 应用程序接口访问。***

- nodesmith.io
- 相关文档

Ethercluster - ***运行您自己的支持以太坊和以太坊经典的以太坊应用程序接口服务。***

- ethercluster.com

Chainstack - ***共享及专用的以太坊节点即服务。***

- chainstack.com
- 相关文档

QuickNode - ***区块链基础设施即服务。***

- quicknode.com
- 相关文档
- Discord

Python Tooling - ***用于通过 Python 进行以太坊交互的各种库。***

- py.ethereum.org
- web3.py GitHub
- web3.py 聊天

web3j - ***以太坊的 Java/Android/Kotlin/Scala 集成库。***

- GitHub
- 相关文档
- Gitter

Rivet - ***由开源软件提供支持的以太坊和以太坊经典应用程序接口服务。***

- rivet.cloud
- 相关文档
- GitHub

Nethereum - ***区块链的开源 .NET 集成库。***

- GitHub
- 相关文档
- Discord

QuikNode - ***终极区块链开发平台。***

- Tatum
- GitHub
- 相关文档
- Discord

Watchdata - ***提供对以太坊区块链简单和可靠的应用程序接口访问。***

- Watchdata
- 相关文档
- Discord

Zmok - ***以速度为导向的以太坊节点作为 JSON-RPC/WebSockets API 接口。***

- zmok.io
- GitHub
- 相关文档
- Discord

延伸阅读 {#further-reading}

还有哪些社区资源帮助过您？ 编辑并添加本页面！

相关主题 {#related-topics}

- 节点和客户端
- 开发框架

相关教程 {#related-tutorials}

- 设置 Web3js 以在 JavaScript 中使用以太坊区块链 *– 在项目中设置 web3.js 的说明。*
- 从 JavaScript 中调用智能合约 *– 使用 DAI 代币，从而使用 JavaScript 调用合约函数。*

---

title: 以太坊改进提议 (EIP)
description: 了解以太坊改进提议 (EIP) 的基本信息。
lang: zh

sidebar: true

隆重介绍以太坊改进提议（EIPs） {#introduction-to-ethereum-improvement-proposals-eips}

什么是 EIP？ {#what-are-eips}

以太坊改进提议 (EIP)是为以太坊潜在新功能或流程提出建议的标准。 EIP 包含提议改进的技术规范，并作为社区的“真相来源”。 在 EIP 的过程中，将会讨论和制定以太坊的网络升级和应用标准。

以太坊社区中的任何人都可以创造一个 EIP。 关于如何编写 EIP 的指南将纳入 EIP 1。 EIP 应提供功能及其原理的简明技术细节和提出的理由。 EIP 作者负责在社区内建立共识并记录反对意见。 提交一个高水平的 EIP 需要很强的技术能力，历史上，大多数 EIP 作者已经成为应用程序或者协议开发者。

为什么 EIP 很重要？ {#why-do-eips-matter}

EIP 作为一个中心角色，记载以太坊的变化并且记载在以太坊中。 它们是人们提议、辩论和适应变化的途径。 有各种不同类型的 EIP，其中包括会影响共识并需要网络升级的用于底层协议更改的核心 EIP，以及用于应用程序标准的 ERC。 例如，创建 token 的标准，ERC20 或 ERC721 允许应用程序使用相同的规则处理所有 token，使得创建互操作应用程序更加容易。

每个网络升级都包含一组 EIP，需要网络上每个 以太坊客户端来执行。 这意味着要在以太坊主网上与其他客户达成共识， 客户端开发者需要确保他们都实现了所需的 EIP。

在提供变更技术规格的同时，EIP 是以太坊社区治理的基本单位。任何人都可以自由提议，然后由社区不同的参与者讨论，决定是否将其发展为通用标准或者加入到网络升级中。 因为非核心 EIP 不必被所有应用程序采纳（例如，您可以创建一个非 ERC20 代币)，但核心 EIP 必须被广泛采用（因为所有节点都必须升级，才能保证其属于同一网络的一部分），与非核心 EIP 相比，核心 EIP 需要在社区内达成更广泛的共识。

EIP 历史 {#history-of-eips}

Ethereum Improvement Proposals (EIP) Github 存储库 于 2015 年 10 月创建。 EIP 进程基于比特币改进提议 (BIP) 进程。它本身基于 Python 增强提议 (PEP) 进程。

EIP 编辑人员的任务是审查 EIP 的技术可靠性、正确的拼写/语法和代码风格。 Martin Becze、Vitalik Buterin、Gavin Wood 和其他一些人是 2015 年至 2016 年末最初的 EIP 编辑者。 当前 EIP 编辑者是：

- Alex Beregszaszi（EWASM/以太坊基金会）
- Greg Colvin（社区）
- Casey Detrio（EWASM/以太坊基金会）
- Matt Garnett（Quilt）
- Hudson James（以太坊基金会）
- Nick Johnson (ENS)
- Nick Savers（社区）
- Micah Zoltu（社区）

EIP 编辑们与 Ethereum Cat Herders 和 Ethererum Magicians 的社区成员共同决定实施哪个 EIP，共同负责促进 EIP 实施以及将 EIP 推入到“最终”或“已撤销”阶段。

完整的标准化过程和图表在 EIP-1 中描述

查看更多 {#learn-more}

如果您有兴趣阅读更多关于 EIP 的信息，请查看 EIP 网站 ，您可以在那里找到更多信息，包括：

- 各种不同类型的 EIP
- 已创建的 EIP 列表
- EIP 状态和它们的含义

参与 {#participate}

任何人都可以创建 EIP 或 ERC，尽管你应该阅读 EIP-1，其中概述了 EIP 过程、什么是 EIP、EIP 的类型、EIP 文件应该包含什么、EIP 格式和模板、EIP 编辑名单以及在创建 EIP 之前你需要了解的所有内容。 你的新 EIP 应该定义新的功能，这些功能并不复杂，但也不是超级小众的，可以在以太坊生态系统的项目中使用。 最难的部分是推进，作为作者，您需要推动人们围绕 EIP，收集反馈，写文章描述您的 EIP 解决的问题，并与项目合作，实施您的 EIP。

如果您有兴趣跟随或分享您对 EIP 的贡献，请查看 Ethereum Magicans 论坛，在那里与社区讨论 EIP。

另请参见：

- 如何创建 EIP

引用 {#references}

网页内容部分来自 Hudson Jameson 以太坊协议开发治理和网络升级协调

---

title: 以太坊能源消耗
description: 你需要了解以太坊能源消耗的基本信息
lang: zh

sidebar: true

以太坊的能源消耗 {#introduction}

根据工作量证明，以太坊当前的能源消耗太高，且不具可持续性。 在不牺牲安全和去中心化的情况下解决能源消耗问题是一项重大的技术挑战，多年来一直是研究和开发的重点。 让我们探索为什么构建以太坊会对环境产生很大的影响，以及网络即将升级为权益证明会如何有效地改变这一点。

能源保障网络 {#energy-secures-the-network}

以太坊区块链上的交易由矿工进行验证。 矿工将交易捆绑在一起形成有序区块，并将它们添加到以太坊区块链中。 新区块会传播到所有其他独立管理交易的节点操作者，并验证它们是否有效。 任何不真实之处都会显示出不同节点之间的不一致。 最真实区块被添加到区块链，成为历史上不可或缺的一部分。

只有存在于采矿相关的成本，以及无法预测哪个特定节点将提交下个区块时，才允许任何矿工添加新区块。 为了满足这些条件，必须出示工作量证明 (PoW)。 要有资格提交交易区块，矿工必须比任何其他矿工更快地解决任意的计算难题。 解决此问题会使矿工之间产生竞争，并导致能源消耗成本。 要成功地欺骗区块链，不诚实的矿工就必须始终如一地赢得工作量证明竞赛，这很不可能实现，同时也很昂贵。

以太坊自创世块以来就一直使用工作量证明。 从工作量证明转为权益证明一直是以太坊的一项基本目标。 然而，开发一个符合以太坊安全性和去中心化核心原则的权益证明系统并非易事。 需要在密码学、密码经济学和机制设计方面进行大量研究和突破，才能实现转换。

工作量证明能源消耗 {#proof-of-work}

工作量证明是保护网络和对区块链强制实施真实更改的可靠方法，但由于一些原因，这种方法也存在问题。 由于开采区块的权利需要解决任意计算难题，矿工可以通过投资更强大的硬件来增加成功的几率。 这些诱因导致了矿工间的装备竞争，他们会购买越来越多耗电的采矿设备。 以太坊的工作量证明协议目前的年化功耗总量大约等于芬兰的年化功耗总量[^1]，而其碳足迹则与瑞士相似[^1]。

权益证明 {#proof-of-stake}

正在以 **权益证明 (PoS)** 链的形式为以太坊打造一个更加绿色的未来。 在权益证明下，无需解决任意问题。 无需解决问题大大减少了保障网络安全所需的能源消耗。 矿工被执行相同功能的验证者所取代，区别在于他们没有以计算工作的形式预先花费他们的资产，而是将 ETH 作为抵押品来防止不诚实的行为。 如果验证者较为懒惰（在应该履行某些验证者职责时离线），他们质押的 ETH 可能会慢慢流失，而可证明的不诚实行为会导致质押资产遭受“惩罚”。 这有力地鼓励人们积极和诚实地参与到保障网络安全中。

类似于工作量证明，一个恶意实体将至少需要网络中 ETH 质押总量的 51% 来执行 51% 攻击。 然而，使用工作量证明时，失败攻击的潜在损失仅仅是产生采矿所需哈希值的成本，而与之不同的是，使用权益证明时，攻击可能造成的损失是作为抵押品的全部 ETH。 这种阻碍结构允许通过权益证明实现网络安全，同时消除了在任意计算上消耗能量的需要。 权益证明下的详细网络安全说明可在这里和这里找到。

合并 {#the-merge}

有一个名为信标链的功能性权益证明链自 2020 年 12 月以来一直在运行，该证明链展示了权益证明协议的可行性。 合并是指以太坊抛弃工作量证明，转而全面采用权益证明的时间点。 合并预计在 2022 年第 3/4 季度进行。 关于合并的更多信息。

权益证明能源消耗 {#proof-of-stake-energy}

除了建立对权益证明机制的信心外，信标链还使人们能够估计出以太坊在合并后的能源使用情况。 最近的估计表明，合并为权益证明可能会导致能源使用总量减少 99.5%，因为权益证明的能效比工作量证明高约 2000x。 以太坊的能源消耗大致相当于网络上每个节点运行家用计算机的费用。

图片

根据 2021 年 5 月数据图表中所用的 tx 估计工作量证明能耗，在撰写本文时，该来源建议能耗不超过 175.56 Kwh

我们将这些数据与 Visa 等服务进行比较。 100,000 次 Visa 交易使用的能量为 149kWh[^2]。 假设已实施区块分片，以太坊当前的交易率（每秒 15 笔交易）将至少增加 64 倍（分片数量），不考虑汇总的额外优化。 对于分片后带汇总的以太坊，合并后的真实估计值为 25,000 - 100,000 次交易。 我们可以利用这一信息估计每 100 000 笔交易的最高和最低能源消耗。

- 每秒 25,000 笔交易。
- 完成 100,000 笔交易需要 `100,000 / 25,000 = 4` 秒。

我们还可以估算以太坊每秒钟的能源消耗，保守估计有 10,000 个活跃验证者正在保护网络（信标链上有超过 250,000 名验证者，但许多验证者可以在单个节点上操作。 目前估计有 3,000-4,000 个节点，所以 10,000 个是合并后的保守估计）：

`1.44kwh 的每日用量 * 10,000 个网络节点 = 14,400kwh`/天。 每天有 86,400 秒，所以 `14,400/86,400=0.1666667 千瓦时`/秒。

如果我们将其乘以处理 100,000 次交易所需的时间：`0.1667 * 4 = 17.366666701 千瓦时`

这是 Visa 相同数量交易所用能耗的大约 0.4%，或者与以太坊当前的工作量证明网络相比，能源消耗减少为原来的约 1/225。

如果按照每秒最大交易次数重复计算，得出平均每秒消耗能量 0.1667 kWh，约为 Visa 能源消耗的 0.1%，或减少至其 1/894。

注意：根据交易数量进行比较并不完全准确，因为以太坊的能源使用是以时间为基础的。 无论进行 1 次交易还是 1,000 次交易，以太坊的能源使用量在 1 分钟内都是相同的。

我们还必须要考虑到，以太坊不仅限于简单的金融交易，它还是一个为智能合约和去中心化应用程序构建的完整平台。

更加绿色节能的以太坊 {#green-ethereum}

虽然从历史上看，以太坊的能源消耗一直很大，但开发人员在时间和能力上进行了大量的投资，从耗能区块验证转换为节能区块验证。 据 Bankless 称，减少工作量证明能源消耗的最好办法就是简单地“关闭它”，这是以太坊承诺采取的办法。

如果您认为这些数据不正确或可以更加精确，请提出问题或 PR。 这些是 ethereum.org 团队使用可公开访问的信息和当前以太坊路线图做出的估计。 这些说法并不代表以太坊基金会的正式承诺。

延伸阅读 {#further-reading}

- A country's worth of power, no more – *Carl Beekhuizen，2021 年 5 月 18 日*
- Ethereum Emissions: A Bottom-up Estimate _ Kyle McDonald_
- Ethereum Energy Consumption Index – *Digiconomist*
- ETHMerge.com — *@InsideTheSim*

相关主题 {#related-topics}

- 以太坊的展望
- 信标链
- 合并
- 区块分片

脚注和来源 {#footnotes-and-sources}

1. 以太坊工作量证明能源消耗 {#fn-1}

按国家企业划分的能源消耗量。 以太坊（年化 TWh）

2. Visa 能耗 {#fn-2}

截至 2020 年，与 VISA 网络相比，比特币网络每笔交易的平均能耗，Statista

Visa 2020 年第四季度财务报告

title: 在以太坊主网络上的企业
description: 有关公共以太坊区块链上的企业应用的指南、文章和工具
lang: zh

sidebar: true

企业以太坊主网络 {#ethereum-for-enterprise}

区块链应用程序有助于企业：

- 增加信任，降低企业之间协调的成本
- 提高业务网络的帐户属性及运营效率
- 发现新的商业模式和创造价值的机会
- 提高组织未来的竞争力

企业级的区块链应用可以建立在以太坊主网或基于以太坊技术的私有链上。 详见企业级私有以太坊。

以太坊主网与私有链 {#private-vs-public}

以太坊主网是唯一的。 建立在主网上的应用程序能够相互调用，类似于在互联网上建立的应用程序能够相互连接，充分发挥区块链去中心化的潜力。

许多企业和财团已经为基于以太坊技术的特定应用程序部署了私人的、需要许可的区块链。

主要区别 {#key-differences}

- 区块链安全/不可变性——区块链能否抵制被篡改是由其协商一致性的算法决定的。 以太网主网的安全是由世界各地的个人和矿工管理的数千个独立节点间的交互保证的。 私有链通常有少数几个节点，由一个或几个组织控制。 这些节点可以被严格地控制，但少数节点重写链上信息或进行欺诈性交易的行为必须受到惩罚。
- 性能——由于私有的以太坊可能使用具有特殊硬件要求和不同共识算法的高性能节点，例如 POA 等。它们可能在基准层（第一层）实现较高的交易吞吐量。 在以太网主网上，使用 Layer 2 解决方案 可以实现高吞吐量。
- 成本——经营私有链的成本主要是建立和管理这条链所花费的精力及运行它的服务器。 虽然与以太网连接没有成本，但每笔交易都有 gas 成本，必须在 Ether 支付。 目前正在开发交易转发器（又名 gas 站），以消除最终用户甚至企业在交易中直接使用 Ether 的必要性。 一些分析显示，在以太坊主网上运行应用程序的总成本可能低于运行私有链。
- 节点权限——只有授权的节点可以加入私有链。 任何人都可以在以太坊主链上设置一个节点。
- 隐私——访问写入私有链的数据可以通过限制访问网络来控制，并更加细粒度地进行访问控制和私有交易。 任何人都可以查看写入到主网一层的所有数据，所以敏感信息应该脱链存储、传输或者加密。 一些实现了以上设想的设计模式已经出现（如 Baseline、Aztec），以及能够保持数据分割的二层解决方案。

为什么要在以太坊主网上开发 {#why-build-on-ethereum-mainnet}

企业从 2016 年左右开始使用区块链技术，当时已经有了 Hyperledger、Quorum 和 Corda 项目。 最初的重点主要放在私有企业级区块链上。但从 2019 年开始，人们对商业应用程序在公共与私有区块链上的思考发生了转变。 福雷斯特进行的调查显示，“调查答卷人... 看看这个潜力，75% 的人说他们可能会在未来使用公链，将近三分之一的人说他们很可能”。 EY’s Paul Brody 曾经谈到有关在以太坊主链上进行构建的优点。其中（根据不同的应用）包括了更强的安全性/不可更改性、透明、低成本及方便与其他主链上的应用交互（网络效应）。 企业之间分享一个共同的参照基准，可以避免不必要地产生无数孤立的节点，它们不能相互沟通、分享或同步信息。

公链另一个引人注意的发展是第二层。 第二层主要是一种可伸缩性技术类别，它提高了公共链上的吞吐量。 但第二层解决方案也可以解决一些其他问题。这些问题正是当初企业开发者在过去选择私有链的原因。

“Baseline 协议”是一个关键项目，它定义了一项协议，使企业之间能够进行保密和复杂的合作而不会在链上留下任何敏感数据。 它在 2020 年取得了重大的进展。

企业开发者资源 {#enterprise-developer-resources}

组织 {#organizations}

不同组织为使以太坊企业级应用合作更便利作出了一些努力：

- 企业以太坊联盟（Enterprise Ethereum Alliance, EEA） EEA 旨在让各组织能够在其日常业务中采用和使用以太坊技术。 它让以太坊的生态系统可以找到新的商业机会，推进工业界的采用，并让参与者之间可以相互学习与合作。 EEA 的主网工作组是在公共以太坊主机上建设的企业代表的协调中心，以及希望支持他们的以太坊社区成员。
- 以太坊 OASIS 开放项目（Ethereum OASIS Open Project） 以太坊 OASIS 开放项目是一个 OASIS 开放项目，它为不同的利益攸关方提供了一个中立的论坛，以高质量地规范以太坊长期稳定性、交互性并使其易于集成。 该项目打算制定明确、开放标准、高质量的文档和共同的测试套件，以促进以太坊协议的改进。
- Baseline 项目 Baseline 协议是一个开放源码倡议，它结合了加密方面，发送消息和区块链的进步，通过公共以太坊主网以低成本的方式提供安全和私人的业务流程。 该协议使企业之间能够进行保密和复杂的合作，而不会在链上留下任何敏感的数据。 Baseline 项目是以太坊 OASIS 开放项目的一个次级项目，由 Baseline 技术指导委员会协调。

产品和服务 {#products-and-services}

- Alchemy _ 提供了 API 服务和工具，用于构建和监控以太坊上的应用程序_
- Blockapps *通过部署企业级以太坊协议、工具和 API 形成 STRATO 平台*
- Chainstack *主网和测试网以太坊基础设施托管在公共& 分开的客户云中*
- ConsenSys_ 为以太坊上的开发提供了一系列的工具和产品，同时还包括了咨询和自定义开发服务_
- Envision Blockchain_ 向专注于以太坊主网的企业提供咨询和开发服务_
- EY OpsChain *通过在您信任的商业伙伴网络中签发 RFQ、合同、订单和发票提供了一个采购流程*
- Hyperledger Besu _ 遵守 Apache 2.0 开源许可并用 Java 编写的开源以太坊客户端_
- Infura *对以太坊和 IPFS 网络的可调用的 API 访问*
- Provide *为 Enterprise Web3 应用程序提供基础架构和 API*
- Unibright *一个拥有 20 多年业务流程和整合经验的区块链专家、架构师、开发人员和咨询人员团队*

工具和库 {#tooling-and-libraries}

- Alathio *以太坊数据分析平台*
- Epirus _ Web3 实验室的开发、部署和监控区块链应用程序的平台_
- Ernst & Young 的“Nightfall” *私有的交易工具包*
- EthSigner *与 Web3 应用提供商一起使用的交易签名应用程序*
- Tenderly *一个提供实时分析、警报和监测并支持私有网络的数据平台。*
- Truffle Suite *区块链开发套件（Truffle、Ganache、Drizzle）*

可扩展性解决方案 {#scalability-solutions}

第二层 是一套在以太坊上（一层）运行的技术或系统，它与一层具有相同的安全属性，但提供了更大的交易处理能力（吞吐量），更低交易费（操作费用），和比第一层更快的交易确认速度。 第二层扩容解决方案由第一层保护，但它们使区块链应用程序相比第一层能够处理更多的用户、操作或数据。 其中许多解决方案利用最近在加密和零知识证明（ZK）方面取得的进展，最大限度地提高了性能和安全性。

在第二层上开发应用可以帮助 解决一些当初导致企业开发者在过去选择私有链的问题，同时保留了使用以太坊主网的好处。

已准备工作或即将完成的 L2 解决办法的例子包括：

- Optimistic rollups（链上数据、欺诈证明）
  - Optimism
  - Offchain Labs Arbitrum Rollup
  - Offchain Labs Arbitrum Rollup
- ZK rollups（链上数据，ZK 有效性证明）
  - Loopring
  - Starkware
  - Matter Labs zkSync
  - Aztec 2.0
- Validium（链下数据，ZK 有效性证明）
  - Starkware
  - Matter Labs zkPorter
- Plasma（链下数据，欺诈证明）
  - OMG Network
  - Gazelle
  - Matic Network
  - LeapDAO
- 状态通道
  - Connext
  - Kchannels
  - Perun
  - Raiden
- 侧链
  - Skale
  - POA Network
- 组合多个类别属性的混合解决方案
  - Offchain Labs Arbitrum SCSC
  - Celer

以太主网上的企业级应用 {#enterprise-live-on-mainnet}

以下是已经部署到以太主网上的一些企业级应用

支付 {#payments}

- Brave 浏览器 *让用户关注广告，用户可以通过 BAT 代币支付出版商以支持他们。*
- hCaptcha *防止机器人的 CAPTCHA 系统，这个系统将用户标记机器学习数据的费用支付给网站操作者。 现在由 Cloudfllar 部署。*
- Audius *一个直接连接音乐粉丝和艺术家的流媒体服务，并且允许艺术家由他们的粉丝全额支付，直接和即时支付给每个流媒体*

金融 {#finance}

- Santander Bank *债券发行和结算*
- Societe Generale *保证金发行*
- Cadence *FAT 商标发行与代币化*
- Sila *银行和 ACH 支付的基础设施即服务*

- Tinlake *应收款融资通过代币化的真实资产，如发票、抵押或串流使用费*
- Kratos *商品交易和贸易融资平台，将商品交易者连接起来，并使他们能够直接在线交易和从贷方获得资本*
- Fasset *支持可持续基础设施的平台*

数据公证 {#notarization-of-data}

- BBVA *最终贷款的详细信息被散列并记录在主网上*
- Slusk *数据完整性可以通过定期将索引数据的哈希写入到主网*
- ANSA *意大利最大的新闻机构与假新闻作斗争，并使读者能够通过在主网上录制这些新闻故事来验证其来源*
- Verizon *在以太坊上记录新闻稿，以确保公司责任和信用*
- Breitling *记录以太坊上的手表的出处和修理历史*

供应链 {#supply-chain}

- CargoX *提单和单证传输提供者*
- Morphosus.network _供应链自动化平台，它实现了私有链与以太坊主网公证数据的混合，加拿大食品、石油天然气经销有限公司和阿根廷宠物食品供应商 Vitalcan 等目前在使用 _
- Minespider *供应链跟踪*
- ShipChain *公开的以太坊侧链，同时也是企业级的可见可信任的供应链，特别是以多式联运物流为目的*
- Follow Our Fibre *直观跟踪供应链*
- EY OpsChain Network Procurement *通过在您信任的商业伙伴网络中签发 RFQ、合同、订单和发票提供了一个采购流程*
- Treum *使用区块链技术为供应链带来透明度、可追溯性和可交易性*

所有权与证明 {#credentials}

- 犹他州 *在以太坊签发数码结婚证书*
- 两所意大利高中 *在以太坊主网上颁发的数字文凭*
- 圣加仑大学 *验证瑞士一所大学学位的试点项目*
- Malta _ Hyland_ 在主网上记录所有教育证书
- 波汉科学技术大学 *南韩大学向其新毕业生提供区块链储存的文凭*
- OpenCerts *在新加坡发布区块链教育凭据*
- BlockCerts *开发了一个开放的区块链凭据标准*
- SkillTree *在线技能培训和认证，可以通过过期触发器或依赖其他技能来配置*

工具 {#utilities}

- GridPlus *电费支付*

如果您想要向这个列表中继续添加，请参阅贡献说明。

title: 企业级私有以太坊
description: 在私有以太坊区块链上开发企业级应用的资源
lang: zh

sidebar: true

企业级私有以太坊 {#private-ethereum-for-enterprise}

企业级的区块链应用可以建立在以太坊主网或基于以太坊技术的私有链上。 有关在公共以太坊主网上构建的更多信息，请参见企业级以太坊主网应用

企业级私有以太坊开发者资源 {#developer-resources-private-enterprise-ethereum}

组织 {#organisations}

不同的组织进行了一些合作，以使以太坊企业更友好：

- 企业以太坊联盟（Enterprise Ethereum Alliance, EEA） 旨在让各组织能够在其日常业务中采用和使用以太坊技术。 我们让以太坊的生态系统可以找到新的商业机会，推进工业界的采用，并让参与者之间可以相互学习与合作。
- Hyperledger *Hyperledger 是一种开源的旨在促进跨行业的区块链技术。 这是一项全球合作，由 Linux 基金会主持，包括了金融、银行、物联网、供应链、制造和科技等领域的主要机构。 基金会在上面有一些基于以太坊技术栈的项目，如 Besu 和 Burrow.*

协议和基础架构 {#protocol-and-infrastructure}

- Chainstack *跨云和跨协议的平台即服务使企业能够快速构建、部署和管理去中心化网络和服务*
- Clearmatics Autonity *实现了 p2p 协议并提供了客户端软件和基础架构*
- Hyperledger Besu _ 一个基于 Apache2.0 协议和用 Java 编写的开源以太坊客户端，包含了多种共识算法，如 PoW 和 PoA（IBFT、IBFT 2.0、Etherhash 和 Clique）。 它的全面许可计划是专门为了在联盟链场景中使用而设计的。_
- Hyperledger Burrow *模块化区块链客户端，包含经过许可的、部分根据以太坊虚拟机 (EVM) 规范而开发的智能合约解释器*
- Kaleido *用于构建和运行跨云、混合式企业生态系统的全栈平台*
- Quorum *一个基于以太坊的开源企业级区块链平台，具有高级企业级功能，能够提供隐私、权限和性能指标*

---

title: 以太坊基金会
description: 了解以太坊基金会（EF），一个致力于支持以太坊和相关技术的非营利组织。
sidebar: false

lang: zh

关于以太坊基金会 {#about-the-ethereum-foundation}

以太坊基金会 (EF) 是一个非营利性组织，致力于支持 以太坊 以及相关技术。

EF 不是一家公司，甚至不是传统的非营利组织。 他们的作用不是控制或领导以太坊，也不是为与以太坊相关的关键技术开发提供资金的唯一组织。 EF 只是巨大的生态系统 的一部分。

以太基基金会倡议 {#ethereum-foundation-initiatives}

生态系统支持方案 {#ecosystem-support-program}

生态系统支持计划 旨在向广大以太坊社区内的项目和实体提供财政和非财政支持来加速生态系统的成长。 生态系统支持计划是对原始以太坊赠款计划的扩展，该计划主要侧重于财务支持。

在esp.ethereum.foundation可以了解更多关于生态系统支持计划，曾经的赠款接收者和申请过程。 你也可以查看 生态系统支持计划博客 或关注 @EF_ESP 获取他们的最新消息和公告。

测试版 {#devcon}

从 2014 年开始，以太坊基金会开始组织“测试版“——所有以太坊开发人员、研究人员、思想家和开发人员的年度会议。

从会议开始以来，您每年都可以在 archive.devcon.org 访问会议的视频内容。

在devcon.org可以学到更多，查看Devcon Blog，或者关注@efdevcon 获得最新通知。

Fellowship 计划 {#fellowship-program}

以太坊基金会 Fellowship 计划 是一个试点项目，其重点是在新兴经济体寻找机会。 今天，发展中国家占全球人口的 80% 以上。 在全世界每天出生的大约 40 万名婴儿中，36 万多人出生在发展中国家。 发展中国家的人民不仅代表着我们作为一个民族的未来，而且还代表着未来十亿人口的积极使用者。

Fellowship 计划为以太坊基金提供了与改变世界的人合作的机会。 其项目很有可能通过使用以太坊对新兴经济体产生影响。 该方案具有独特的经验，包括辅导、专门知识和接触以太坊生态系统领袖。

在 ellowship.ethereum.foundation 或查看 最近在 Fellowship 发表的博客

关于更多关于基金会的内容和他们的工作，请访问以太坊基金会，或者查看以太坊基金会博客 来了解最新的新闻或者通告。

title: 以太坊词汇表
description: 以太坊相关的技术和非技术术语不完全清单
lang: zh
sidebar: true

sidebarDepth: 2

词汇表 {#ethereum-glossary}

{#section-numbers}

51% 攻击 {#51-attack}

一种对去中心化网络的攻击方式，一个群体获得了大多数节点的控制权。 这将使他们能够通过逆转交易和加倍花费 ETH 和其他 token 来欺诈区块链。

A {#section-a}

account 帐户 {#account}

帐户是一个对象，它包含地址、余额、nonce，并且存储了状态和代码（皆可为空）。 一个帐户可以是合约帐户，也可以是外部帐户（EOA）。

以太坊帐户

address 地址 {#address}

通常来说，地址代表一个外部帐户或合约，可以在区块链上接收（目标地址）或发送（源地址）交易。 地址是 ECDSA 公钥的 Keccak 哈希中最右的 160 位数。

应用程序二进制接口 (ABI) {#abi}

在以太坊生态系统中与 合约 交互的标准方式。 包括链外与合约 - 合约交互。

ABI=应用程序二进制接口（是两个二进制程序模块之间的接口）

应用程序接口 {#api}

应用程序编程界面（API）是一套如何使用软件的定义。 API 位于应用程序和 Web 服务器之间，并便利它们之间的数据传输。

assert 断言 {#assert}

在 Solidity 语言里，`assert(false)` 会被编译成 `0xfe`，这是一个无效的操作码，它会消耗完剩下的 gas 并复原所有的变更。 当一个 `assert()` 语句失效，表明出现了非常严重和没有预期的问题，您将需要修复代码。 您应该使用 `assert()` 以避免永远不应该发生的情况。

智能合约安全性

认证 {#attestation}

验证者对信标链或分片 区块投票。 验证者必须对区块做证明，示意他们认同区块提议的状态。

B {#section-b}

信标链 {#beacon-chain}

Eth2 升级的一部分，它将成为以太坊网络的协调者。 它会给以太坊引入权益证明和验证者 。 它最终将并入主网。

信标链

big-endian 大端模式 {#big-endian}

是指数据的高字节位保存在内存的低地址中。 与小端模式相反，小端模式是低位字节保存在内存的低地址中。

block 区块 {#block}

一个关于其所包含交易的所需信息（区块头）的集合，以及称为 ommer的一组其他区块头。 区块由以太坊网络中的矿工添加上链。

区块

区块链 {#blockchain}

以太坊网络中由工作量证明验证的区块序列，每个区块与其父块相连，可一直追溯到创世纪块。 它没有区块大小限制，而使用 gas 上限来调整区块大小。

什么是区块链?

bytecode 字节码 {#bytecode}

为软件解释器或虚拟机的高效执行而设计的抽象指令集。 与人类可读的源代码不同，字节码以数字格式表示。

Byzantium fork 拜占庭分叉 {#byzantium-fork}

指的是大都会 阶段的头两次硬分叉 。 它纳入了 EIP-649 “大都会延迟难度炸弹 和减少区块奖励”，具体内容是冰河时代被延迟 1 年，区块奖励从 5 个 ETH 减少为 3 个。

C {#section-c}

compiling 编译 {#compiling}

把用高级编程语言 (例如，Solidity) 写的代码转换为低级语言 (例如，EVM “字节码”)。

编译智能合约

committee 委员会 {#committee}

由至少 128 个验证者组成的集合，这个集合由信标链随机分配到信标区块和分片区块。

consensus 共识 {#consensus}

当网络中的许多节点，通常是大部分节点，都拥有相同的本地验证的最长区块时，称为共识。 不要与共识规则混淆。

consensus rules 共识规则 {#consensus-rules}

全节点与其他节点保持共识的区块验证规则。 不要与共识混淆。

Constantinople fork 君士坦丁堡分叉 {#constantinople-fork}

这是大都会阶段的第二部分，一开始计划在 2018 年中进行。 预期除了其他的变更，还会纳入一个提案是关于转为采用一个工作量证明/权益证明混合的共识算法。

contract account 合约帐户 {#contract-account}

一个包含代码的帐户，只要接收到来自其他帐户（EOA 或合约帐户) 交易就会执行合约代码。

contract creation transaction 创建合约交易 {#contract-creation-transaction}

一种为了注册一个合约并将其记录在以太坊区块链上的特殊交易，其中接收者地址 “为零” 。

crosslink 交叉链接 {#crosslink}

交叉链接提供一个分片状态的总结。 这是在实现分片的权益证明系统里分片链通过信标链互相通信的方式。

权益证明（PoS)

D {#section-d}

Decentralized Autonomous Organization (DAO) 去中心化自治组织 {#dao}

不以等级体系管理来运作的公司或其他组织。 DAO 可能还指一份名为“The DAO”的合约，于 2016 年 4 月 30 日发布，后来在 2016 年 6 月遭到攻击；这件事最终引发了一次硬分叉（代码名称为 DAO），分叉的区块是 1,192,000，这次分叉回滚了被攻击的 DAO 合约，并导致产生了以太坊和 Ethereum Classic 两个相互竞争的系统。

去中心化自治组织 (DAO)

DApp {#dapp}

去中心化应用。 最基本的定义是，它是一个智能合约和一个网络用户界面。 从更广义来说，Dapp 是一个建在开放、去中心化、点对点的基础设施服务上的网络应用。 此外，很多 Dapp 还包括去中心化存储和/或一个通信协议以及平台。

Dapp 介绍

去中心化交易所 (DEX) {#dex}

是 dapp 的一种类型，让人们可以在网络上与对等点交换代币。 你需要有 Eth 才可以使用 Dex（以支付交易费），但它们不像中心化交易所那样受地理限制——任何人都可以参与。

去中心化交易所

行为 {#deed}

行为

DeFi 去中心化金融 {#defi}

去中心化金融的缩写，旨在通过区块链提供金融服务的一类 dapp，无需中介，任何人只需要联网就可以参与。

去中心化金融 (DeFi)

难度 {#difficulty}

是一个全网络的设置，控制产生一个工作量证明所需的计算。

难度炸弹 {#difficulty-bomb}

计划中工作量证明系统里出块难度会呈指数增长，这样的设置旨在使转换到权益证明共识时，减少分叉的变化。

digital signature 数字签名 {#digital-signatures}

用户使用私钥为文件产生的一串短数据，这样任何有对应公钥、签名和文档的人都能验证 (1) 文档由该特定私钥的所有者“签名”，以及 (2) 文档签名后不能再被修改。

E {#section-e}

elliptic curve digital signature algorithm (ECDSA) 椭圆曲线数字签名算法 {#ecdsa}

以太坊使用的一种加密算法，用于确保资金只能被其所有者使用。 这是创建公钥和私钥的首选方法。 相关账户 地址 生成和 交易 验证。

周期 {#epoch}

在信标链协作系统里，每 32 个slot（6.4 分钟）组成的时间间隔称作为一个 epoch。 为了确保安全，每个 epoch 对会对验证者委员会进行混洗。 每个 epoch 里都有为链做最终确定的一次机会。

权益证明（PoS)

以太坊改进提案 (EIP) {#eip}

为以太坊社区提供信息的一种设计文档，说明一个提议的新功能或它的流程或环境（请参阅 ERC）。

EIPS 简介

以太坊域名服务 (ENS) {#ens}

以太坊域名服务 (ENS)

在 ens.domains 上阅读更多信息

熵 {#entropy}

在加密学里，缺乏可预测性或某种水平的随机性。 在生成私密信息时，例如私钥，算法经常需要依赖于提供高熵的信源以确保其输出的不可预测性。

外部帐户 (EOA) {#eoa}

以太坊网络中由人类用户创建或为人类用户使用的帐户。

以太坊征求意见 (ERC) {#erc}

ERC 是部分试图定义以太坊具体使用标准的 EIP 贴上的标签。

EIPS 简介

Ethash {#ethash}

一个 Ethereum 1.0 的工作量证明算法。

更多信息请参考 eth.wiki

ETH {#ether}

以太坊生态系统中使用的原生加密货币，用来支付执行事务时的 gas 开销。 也被写作 ETH 或符号形式 Ξ，这是希腊字母 Xi 的大写。

我们数字未来的货币

事件 {#events}

EVM 使用的日志工具。 Dapps 可以监听事件，并在用户界面使用它们来触发 JavaScript 回调。

事件和日志

以太机虚拟机 (EVM) {#evm}

可执行字节码的基于堆栈的虚拟机。 在以太坊，执行模型指定了在给定一系列字节码指令和一小组环境数据的情况下如何改变系统状态。 这是通过虚拟状态机的正式模型指定的。

以太坊虚拟机（EVM）

EVM assembly language EVM 汇编语言 {#evm-assembly-language}

人类可读的以太坊虚拟机字节码的一种形式。

F {#section-f}

回退函数 {#fallback-function}

在缺失数据或无法匹配函数名称时的默认函数调用。

水龙头 {#faucet}

通过智能合约，免费提供测试网可用的测试以太币的服务

测试网水龙头

终局性 {#finality}

终局性是保证在指定时间之前的一组交易不会被改变且无法撤销。

工作量证明的终局性

权益证明的终局性

Finney {#finney}

ETH 的一种计量单位。 1 Finney = 10^{15} wei。 10^{3} Finney = 1 ETH。

分叉 {#fork}

在协议层面的一次改动，可能会产生一条竞争链，或造成在挖矿中对未来区块路径的暂时性分歧。

欺诈证明 {#fraud-proof}

二层解决方案的安全模型，为了提高交易速度，交易 合并 在单笔交易中提交到以太坊中。 交易默认假定为是有效的，但如果怀疑有欺诈行为，可以对它们提出质疑。 然后，对交易进行欺诈证明，以确定是否发生欺诈。 这种方法提升了交易量，同时保证安全性。 一些 rollup 使用的是 欺诈证明。

Optimistic rollup

边境 {#frontier}

以太坊的初始测试开发阶段，从 2015 年 7 月持续到 2016 年 3 月。

G {#section-g}

gas {#gas}

以太坊网络中为执行智能合约所消耗的虚拟 “燃油”。 以太坊虚拟机使用一种记账方法来衡量 gas 用量，以限制算力资源（参见图灵完备）的消耗。

Gas 和费用

矿工报酬限额 {#gas-limit}

一个交易或一个区块允许消耗的最大 gas 量。

创世区块 {#genesis-block}

区块链上的第一个区块，用于初始化特定的区块链以及原生加密货币。

geth 客户端 {#geth}

Go Ethereum。 以太坊协议最好的实现之一，使用 Go 语言编写。

更多信息请参考 geth.ethereum.org

gwei {#gwei}

Gigawei 的缩写，ether 的一个货币单位，通常用于计算 gas 价格。 1 gwei = 10^9 wei。 10^9 gwei = 1 ETH。

H {#section-h}

硬分叉 {#hard-fork}

区块链中的永久性分歧；硬分叉也叫硬分叉改变。 通常发生在拒绝升级的节点无法验证遵循更新共识机制的已升级节点所创建的区块时。 请不要与分叉、软分叉、软件分叉或 Git 分叉相混淆。

哈希值 {#hash}

由可变长度的输入，通过哈希函数（散列）生成固定长度的数字指纹。 （参见 keccak-256）

HD 钱包 {#hd-wallet}

使用分层确定性创建密钥和转账的钱包。

在 github.com 上阅读更多信息

HD 钱包种子 {#hd-wallet-seed}

用来生成 HD 钱包中主私钥与主链码的短种子值。 可以用助记词表示，以便大家复制、备份及恢复私钥。

家园 {#homestead}

以太坊的第二个开发阶段，于 2016 年 3 月在第 1,150,000 个区块上启动。

I {#section-i}

索引 {#index}

一个网络结构，旨在通过为其存储源提供一个有效的路径，从 区块链 中优化查询信息。

可交换客户端地址协议 (ICAP) {#icap}

以太坊地址编码，与国际银行帐户编号 (IBAN) 编码部分兼容，为以太坊地址提供多用途的、可校验的和可操作的编码。 ICAP 地址使用一个新的 IBAN 伪国家代码-XE，全称是“eXended Ethereum”，代表非管辖货币（如 XBT、XRP、XCP）。

冰河世纪 {#ice-age}

块高为 200,000 的 硬分叉， 增加了指数级难度增加（又称为难度炸弹），为过渡到权益证明做准备。

集成开发环境 (IDE) {#ide}

将代码编辑器、编译器、运行时和调试器合并的用户界面。

集成开发环境 (IDE)

不可变部署代码问题 {#immutable-deployed-code-problem}

合约（或库）的代码一旦部署，就不可变更。 传统标准软件开发习惯于能够修复可能的缺陷并增加新的功能，但这是对智能合约的开发是一个挑战。

部署智能合约

内部交易 {#internal-transaction}

从一个合约帐户发往另外一个合约帐户或者外部帐户的交易。

K {#section-k}

密钥导出函数 (KDF) {#kdf}

也称为密钥延伸算法，通过密钥库文件格式来防止暴力破解，防止攻击者预先计算派生密钥的字典或 “彩虹表”，通过重复计算密令的哈希来实现。

智能合约安全性

keccak-256 {#keccak-256}

以太坊协议使用的加密哈希函数。 Keccak-256 是从 SHA-3 规范演化出来的。

keystore file 密钥存储文件 {#keystore-file}

一个 JSON 编码文件，其中包含单个（随机生成）私钥，用密码加密以获取额外安全性。

L {#section-l}

layer 2 二层 {#layer-2}

在以太坊协议之上的聚焦与分层优化的开发实现。 这些改进关系到 交易 速度、更便宜的 交易费以及交易隐私。

二层

LevelDB {#level-db}

开源的轻量级 k-v 存储库，可以在不同的平台中使用。

library 存储库 {#library}

特殊类型的合约， 没有 payable 函数，没有 fallback 函数，也没有数据存储。 因此，它不能接收或持有以太坊，也不能储存数据。 一个存储库代表以前部署的代码，其他合约只能进行只读调用。

智能合约库

轻量级客户端 {#lightweight-client}

一个以太坊客户端，不存储区块链的本地副本，不验证区块和交易。 它提供一个钱包的功能，可以创建和广播交易。

M {#section-m}

主网 {#mainnet}

"主网"的缩写，这是以太坊区块链的主网。 真正的的以太坊，真正的价值和真正的共识。 在讨论二层扩容解决方案时，主网也被称为一层。 （另见 测试网）

Merkle Patricia trie 前缀树 {#merkle-patricia-tree}

用于以太坊有效存储密钥对的数据结构。

消息 {#message}

一个内部交易， 永不会被序列化，且仅在 EVM 中发送。

message call 消息调用 {#message-call}

将一个消息从一个帐户传递到另一个帐户的行为。 如果目标帐户与 EVM 代码相关联，VM 将从消息依赖的状态开始执行。

Metropolis {#metropolis}

以太坊的第三个开发阶段，于 2017 年 10 月启动。

矿工 {#miner}

通过不断做哈希运算，找到新区块的有效工作量证明<的网络节点。

采矿

N {#section-n}

网络 {#network}

指以太坊网络，一个对等网络向每个以太坊节点（网络参与者）推广交易和块。

网络

非同质化代币 (NFT) {#nft}

也叫 “所有权证书” 或 “权证”，是由 ERC721 议案提出的代币标准。 NFT 能够被追溯也可以交易，每个代币是唯一且独一无二的，不像 ERC20 代币，每个 NFT 都是无法互换的。 NFT 能够代表数字或物理资产的所有权。

非同质化代币 (NFT)

ERC-721 非同质化代币标准

节点 {#node}

参与点对点网络的软件客户端。

节点和客户端

nonce {#nonce}

在密码学中，nonce 指的是在加密过程中只能使用一次的值。 以太坊中有两类 nonce：帐户 nonce 值，用来统计每个帐户的交易数，防止重放攻击；工作量证明 nonce，区块中用于满足工作量证明的随机值。

O {#section-o}

叔块 {#ommer}

当一个矿工找到一个有效的块，另一个矿工可能已经发布了一个存在竞争的块，这个块将首先添加到区块链中。 这个有效但是之前的区块可以被更新的区块归为*叔块*，此时可以领取部分区块奖励。 对于父区块的相邻区块来说，“ommer”一词是首选的不分性别的词，但有时也被称为“uncle”。

Optimistic rollup {#optimistic-rollup}

Rollup 交易使用 欺诈证明 提供增加二层的交易吞吐量，同时使用主网（一层）提供的安全保障。 与 Plasma（一个类似的二层的解决方案）不同， Optimistic rollup 可以处理更复杂的交易类型 -- EVM 中任何可能的事情。 与 Zk Rollup 相比，他们确实有延迟问题，因为交易可以通过欺诈证据被质疑。

Optimistic Rollup

P {#section-p}

奇偶校验 {#parity}

以太坊客户端软件最棒的实现之一。

以太坊 Plasma 扩容解决方案 {#plasma}

二层使用欺诈性证明的扩容解决方案，例如 Optimistic rollup。 Plasma 仅限于简单的交易，例如基本令牌传输和交换。

以太坊 Plasma 扩容解决方案

private key (secret key) 私钥（密钥） {#private-key}

用户证明对某一帐户或合约的所有权的字符序列，用来生成数字签名（见 公钥、地址、ECDSA）。

权益证明 (PoS) {#pos}

在区块链中实现分布式共识的方法。 PoS 要求用户证明自己拥有一定数量的加密货币（在网络中"质押"），以便能够参与交易的验证。

权益证明

工作量证明 (PoW) {#pow}

需要大量计算才能得出的数据（证明）。 在以太坊中， 矿工 必须找到符合整个网络范围的 算力难度 目标的 Ethash 算法的数值。

工作量证明

公钥 {#public-key}

通过私钥的单向函数派生的数字，可以公开共享，任何人都可以使用它来验证用其对应私钥进行的数字签名。

R {#section-r}

收据 {#receipt}

收据是由以太坊客户端返回的数据，用来表示特定交易的结果，数据包含交易哈希值、打包的区块号、实际 gas 消耗量，如果该交易用来部署合约，则还会返回该合约地址交易。

重放攻击 {#re-entrancy-attack}

攻击者合约调用受害者合约函数，使得在调用执行过程中受害者合约会循环调用攻击者合约。 这可能导致通过跳过受害者合约的余额更新或提款金额计算的部分来盗窃资金。

重入攻击

奖励 {#reward}

以太坊网络给予找到相应工作量证明的矿工的奖励，该奖励用以太币计价并纳入每个新区块中。

递归长度前缀编码 (RLP) {#rlp}

以太坊开发者设计的编码标准，用于编码和序列化具有任意复杂性和长度的对象（数据结构）。

rollup {#rollups}

一种二层扩容解决方案，将多笔交易分批提交到以太坊主链的单笔交易中。 这就可以降低 gas 成本，增加交易吞吐量。 有些 Optimistic 和 ZK Rollup 使用两种不同的安全方法来提供扩容能力。

Rollup

S {#section-s}

宁静 {#serenity}

以太坊的第四个也是最后一个发展阶段，又称以太坊 2.0。

以太坊 2.0（Eth2）

Secure Hash Algorithm (SHA) 安全哈希算法 {#sha}

国家标准和技术研究所（NIST）推出的加密散列函数库。

分片/分片链 {#shard}

由信标链管理并由验证者保护的权益证明链。 作为 Eth2 分片链升级的一部分，将有 64 个加入该网络。 分片链为二层解决方案提供额外数据，例如为 optimistic rollup 和 ZK rollup 提供额外的交易吞吐量。

分片链

侧链 {#sidechain}

一种使用不同链、通常更快的 共识规则的扩容解决方案。 。 需要桥来连接这些侧链到 主网。 Rollup 也使用侧链，但是他们可以与主网进行协作。

侧链

单例 {#singleton}

一种计算机编程术语，指一个对象只能存在一个实例。

插槽 {#slot}

一个时间段（12 秒），在这个时间段内，验证者可以在权益证明系统中提出一个新的信标链和分片链区块。 时隙有可能为空。 32 个时隙构成一个周期。

权益证明 (PoS)

智能合约 {#smart-contract}

在以太坊计算基础框架上执行的程序。

智能合约简介

Solidity {#solidity}

一种语法类似 JavaScript、C++ 或 Java 的程序化（命令式）编程语言， 是用于编写以太坊智能合约的最流行也最常用的编程语言。 由 Gavin Wood 博士创建。

Solidity

Solidity 内联汇编 {#solidity-inline-assembly}

EVM Solidity 的内联汇编语言。 Solidity 对内联汇编的支持使得写入某些操作变得更加容易。

伪龙 {#spurious-dragon}

以太坊区块链在区块 2,675 的一个硬分叉，解决拒绝服务攻击和状态清除的问题（见 Tangerine Whistle）。 另外，还有一个重放攻击保护机制（见 nonce）。

稳定币 {#stablecoin}

一种 ERC-20 代币，其价值与另一种资产的价值锚定。 稳定币的锚定资产包括法币（如美元），贵金属（如黄金）以及其他加密货币（如比特币）。

ETH 不是以太坊唯一的加密货币

权益质押 {#staking}

存入一定量的 ether（质押金）成为验证者并参与维护以太坊网络。 验证者在 proof-of-stake（权益证明）共识模型中检查 交易 并且提议 区块。 质押能够为符合网络利益的行为提供经济激励。 你将会因为履行验证者义务而获得奖励，反之将损失一定量的 ETH。

质押您的以太币并成为以太坊验证者

状态通道 {#state-channels}

一种二层解决方案，在参与者之间设置一个通道，他们可以在通道中自由交易且成本低廉。 只有开设和关闭通道的交易才被提交到 主网。 这使得交易吞吐量大幅提升，但依赖于已知的参与人数和锁定资金。

状态通道

szabo {#szabo}

ether 的一种计量单位。 1 szabo = 10^{12} wei，10^{6} szabo = 1 ETH。

T {#section-t}

Tangerine Whistle {#tangerine-whistle}

以太坊区块链的一次硬分叉，发生在区块高度 2,463,000 处，更改了某些 I/O 密集型操作的 gas 计算，并清除 DoS 攻击造成的累积状态，该攻击利用了相关操作的低 gas 消耗。

测试网 {#testnet}

"测试网络"的简称，用于模拟以太坊主网行为的网络（参阅 mainnet 主网）。

测试网

代币标准 {#token-standard}

由 ERC-20 提案引入，为同质化代币提供了标准化的 智能合约 结构。 链下解决方案，使用有效性证明来提高交易吞吐量。

ERC-20 代币标准

交易 {#transaction}

由一个原始帐户签署并以一个特定地址为目标的提交到以太坊区块链上的数据， 该交易包含交易 gas 上限等元数据。

交易

transaction fee 交易费 {#transaction-fee}

每当你使用以太坊网络时需要支付的费用。 包括从你的钱包发送资金或与 dapp 交互，例如兑换代币或购买一件藏品。 可以将其看作是服务费。 这笔费用取决于网络的使用率。 这是因为 矿工（负责处理你的交易的人）很可能以手续费高低来对交易进行排序——因此拥堵情况下使得价格上涨。

从技术层面来说，你的交易费用与相应交易需要的 gas 数量相关。

降低交易费用目前非常受关注。 参阅二层

图灵完备 {#turing-complete}

在可计算性理论里，如果一系列操作数据的规则（如指令集、编程语言、细胞自动机）可以用来模拟单带图灵机，那么它是图灵完备的。这个名字来源于英国数学家和计算机科学家艾伦·图灵。

V {#section-v}

验证者 {#validator}

权益证明系统中的节点，负责存储数据、处理交易并且在区块链中添加新区块。 要激活一名验证者，需要质押 32 ETH。

权益证明

质押

有效性证明 {#validity-proof}

某些二层解决方案的安全模型，以提高速度，交易 合并为交易批并在单次交易中提交到以太坊。 交易计算在链下进行，然后提交给主链，并附带有效性证明。 这种方法提升了交易量，同时保证安全性。 一些 rollup 使用的是欺诈证明。

零知识汇总

Validium {#validium}

二层解决方案，使用有效性证明来提高交易吞吐量。 与 Zk Rollup 不同，Validium 的数据没有存储在一层主网中。

Validium

Vyper {#vyper}

高层次的编程语言与，语法与 Python 类似。 旨在更接近纯函数式语言。 由 Vitalik Buterin 创造。

Vyper

W {#section-w}

钱包 {#wallet}

保存用户密钥的软件， 可以用来访问并管理你的以太坊帐户并与智能合约交互。 密钥不一定要存储在钱包中，为了提高安全性，可以离线存储（如写在一张记忆卡或者纸上）。 虽然称其为 “钱包”，但它并不存储代币本身。

以太坊钱包

Web3 {#web3}

万维网的第三个版本。 由 Gavin Wood 博士首次提议，代表了对网络应用的新愿景与新焦点：从中心化所有并管理的应用转移到在去中心化协议)上构建的应用。

Web2 与 Web3 对比

wei {#wei}

ETH 的最小计量单位。 10^{18} wei = 1 ETH。

Z {#section-z}

零地址 {#zero-address}

一个特殊的以太坊地址，地址的所有字节都为 0，作为 “创建合约交易” 的目标地址。

零知识汇总 {#zk-rollup}

使用有效性证明的[rollup](#rollups)交易在使用主网（一层）安全性的同时提高二层交易吞吐量。 虽然他们无法处理复杂的交易类型，如 Optimistic rollup，但没有延迟问题，因为提交交易时可以证明有效。

零知识汇总

来源 {#sources}

部分由位于 Andreas M. Antonopoulos，Gavin Wood 的 Mastering Ethereum 在 CC-BY-SA 下提供

完善本页面 {#contribute-to-this-page}

我们错过了什么吗？ 有什么不正确吗？ 在 GitHub 上为这个词汇表贡献力量以帮助我们改进！

了解更多关于如何贡献的信息

title: 以太坊治理
description: 简介：如何做出关于以太坊的决策。
lang: zh

sidebar: true

以太坊治理简介 {#introduction}

如果没有人拥有以太坊，那么关于以太坊过去和未来变化的决定是如何做出的？ 以太坊治理是指允许做出这些决定的过程。

什么是治理？ {#what-is-governance}

治理是指允许做出决定的制度系统。 在典型的组织架构中，执行小组或董事会拥有最终决策的发言权。 或者股东们投票决定改革提案。 在政治制度中，当选的官员可以颁布立法，代表其选民的期望。

去中心化治理 {#decentralized-governance}

没有任何人拥有或者控制以太坊协议，但是为了确保网络的长久繁荣，仍然需要对变革做出决定。 由于缺少所有权，这使得传统组织治理不再适合。

以太坊治理 {#ethereum-governance}

以太坊治理是指进行协议更改流程。 需要特别指出的是，这个流程与人们和应用程序如何使用该协议无关 - 以太坊无需许可。 世界上任何人都可以在任意地方参与链上活动。 没有规定谁可以或者不可以构建应用程序或者发送交易。 但是，需要按流程对这些应用程序赖以运行的核心协议提议修改。 因为太多人依赖以太坊的稳定性，所以核心协议的变更的协调门槛非常高，包括社区和技术进程，确保了以太坊的任何变化都安全且得到了社区的广泛支持。

链上治理与链下治理 {#on-chain-vs-off-chain}

区块链技术支持新治理功能，称为“链上治理”。 链上治理是指拟定的协议修改由利益相关方投票决定。 通常由区块链上治理代币的持有者进行投票。 有了某种形式的链上治理，拟定的协议修改已采用代码编写，如果利益相关方批准修改，将自动实施修改。

与之相对的是链下治理，即任何协议变更决议都通过非正式社区讨论过程决定。如获批准，将在代码中加以执行。

以太坊的链下治理吸引了非常广泛的利益相关方参与该流程。

虽然在协议层上，以太坊的治理在链下进行，但许多基于以太坊构建的使用案例（例如 DAO）采用链上治理。

关于 DAO 的更多信息

谁参与这项工作？ {#who-is-involved}

以太坊社区有各种利益相关者，每个人在治理进程中各司其职。 从距离协议最远的利益相关方开始，我们有：

- **ETH 持有者**: 这些人拥有任意数量的 ETH。 关于 ETH 的更多信息。
- **应用程序用户**：这些人员与以太坊区块链上的应用程序交互。
- **应用程序/工具开发者**：这些人编写在以太坊区块链上运行的应用程序是（比如 DeFi、NFT 等）或构建以太坊交互的工具（例如钱包、测试套件等）。 关于 dapps 的更多信息。
- **节点运营商**：这些人运行区块和交易广播节点，拒绝遇到的任何无效交易或区块。 关于节点的更多信息。
- **EIP 作者**：这些人提议修改以太坊协议，具体形式为以太坊改进提议 (EIP)。 关于 EIP 的更多信息。
- **矿工/验证者**：这些人运行可以向以太坊区块链添加新区块的节点。
- **协议开发者**（又名 “核心开发者”）：这些人需要维护各种以太坊实现（例如执行层的 go-ethereum、Nethermind、Besu、Erigon，或共识层的 Prysm、Lighthouse、Nimbus、Teku、Lodestar)。 关于以太坊客户端的更多信息。

注：任何个人都可以属于多个组（如：协议开发者可以支持 EIP，运行信标链验证器，并使用 DeFi 应用程序）。 不过为了明确概念，这样最容易区分。

什么是 EIP？ {#what-is-an-eip}

以太坊治理中使用的一个重要流程是**以太坊改进提议 (EIP)**。 EIP 是指明以太坊潜在新功能或流程的一套标准。 以太坊社区内的任何人都可以创建 EIP。 例如，EIP-721（标准化 NFT 的 EIP）的作者都没有直接参与以太坊的协议开发。

关于 EIP 的更多信息

正式流程 {#formal-process}

对以太坊协议进行修改的正式流程如下：

1. **提出核心 EIP**：如 EIP-1 所述，正式提议对以太坊进行修改的第一步是在核心 EIP 进行详细说明。 一旦被接受，这将作为协议开发者要执行的 EIP 正式规范。
2. **向协议开发者展示 EIP**：一旦拥有已对其收集社区意见的核心 EIP，您应该将它展示给协议开发者。 您可以通过提议在 AllCoreDevs 调用中进行讨论来实现此目的。 很可能已经在 Ethereum Magician 论坛或 Ethereum R&D Discord 中进行过异步讨论。

> 此阶段的潜在成果包括：
>
> - 该 EIP 将被考虑用于未来网络升级
> - 要求技术变更
> - 如果不是优先事项或相对于发展工作而言改进不够明显，则可能会被否决。

3. 收到所有相关利益相关方的反馈意见后，**进入最终提议：**您很可能需要更改初始建议，以改善其安全性，或更好地满足不同用户的需求。 一旦 EIP 中包含了您认为必要的所有更改，您需要再次将其展示给协议开发者。 然后进入流程的下一步，或者出现新问题，需要对您的建议进行另一轮迭代。
4. **将 EIP 包含在网络升级中**：假定该 EIP 已经过审批、测试和实施，将被安排作为网络升级的一部分。 鉴于网络升级的协调成本很高（每个人都需要同步升级），EIP 通常被捆绑在一起升级。
5. **网络升级已激活**：网络升级激活后，EIP 将在以太坊网络上运行。 *注意：网络升级通常在测试网上激活后才会在以太坊主网上激活。*

这一流程虽然极度简化，但概述了在以太坊激活协议修改的重要阶段。 现在看看该流程中的非正式因素。

非正式流程 {#informal-process}

了解先前的工作 {#prior-work}

EIP 倡导者应首先熟悉先前的工作和建议，然后认真考虑创建一个可部署到以太坊主网上的 EIP。 这样，该 EIP 很有可能带入一些以前没有被拒绝过的新事物。 研究此问题的三个主要地方是 EIP 存储库、Ethereum Magicans 和 ethresear.ch。

工作组 {#working-groups}

如果未经编辑或修改，EIP 的初稿不太可能在以太坊主网上执行。 一般而言，EIP 倡导者将与一部分协议开发者合作，指明、实施、测试、迭代和最后确定提议。 根据历史经验，这些工作组需要几个月（有时甚至需要几年！）。 同样的，EIP 倡导者应在收集最终用户反馈意见和减轻部署风险的工作早期就让相关应用程序/工具开发者参与。

社区共识 {#community-consensus}

虽然有些 EIP 是差别细微的直接技术改进，但有些则比较复杂和内在的利益权衡，将以不同方式影响不同的利益相关方。 这意味着某些 EIP 在社区内的争议性超过其他 EIP。

在如何处理争议性建议方面没有明确的指导手册。 因为协议开发者无法强制人们执行网络升级，如果 EIP 的争议性超出对更广泛社区的好处，他们一般会避免执行 EIP 方案。

EIP 倡导者应征求所有相关利益相关方的反馈意见。 如果您发现自己是一个有争议的的 EIP 的倡导者，你应该尝试解决反对方的问题，以便围绕该 EIP 建立共识。 考虑到以太坊社区的规模和多样性，无法使用固定形式（如：投币选举）来达成社区共识，并且 EIP 倡导者将适应他们提案的情况。

除了以太坊网络的安全性外，协议开发者历来对应用程序/工具开发者和应用程序用户的价值给予了高度重视，因为正是因为他们使用和发展以太坊，才使以太坊生态系统对其他利益相关方形成吸引力。 此外，EIP 需要在所有客户实端实施执行，这些项目由不同的团队管理。 在这个过程中，通常意味着让多个协议开发者团队相信，某个特定的改变有价值，它有助于最终用户或解决安全问题。

处理意见不一致的问题 {#disagreements}

许多利益相关方的动机和信仰不同，这意味着分歧很正常。

通常，在公共论坛上通过长时间讨论来处理分歧，以了解问题的根源并允许任何人发表意见。 通常会有一方让步，或大家达成一致。 如果某一方的愿望非常强烈，强迫实施特定变化可能导致区块链分叉。 区块链分叉是一些利益相关方抗议实施协议更改，导致不同、互不兼容的协议版本运行，从而出现两个不同的区块链。

DAO 分叉 {#dao-fork}

分叉是指需要对网络进行重大技术升级或改变，并改变协议“规则”。 以太坊客户端 必须更新他们的软件以执行新的分叉规则。

DAO 分叉是为了解决 2016 DAO 攻击。当时，一份不安全 DAO 合约导致黑客盗走了超过三百万个 ETH。 分叉将资金从错误合约转移到新合约，允许在黑客攻击中丢失资金的任何人收回这些资金。

这一行为由以太坊社区投票决定。 任何 ETH 持有人都能够通过在投票平台上交易来进行投票。 分叉的决定获得了 85% 以上的票数。

注意，虽然的确是为了回滚被黑状态而导致分叉，但在决定分叉时，投票的权重是值得商榷的，原因如下：

- 投票率非常低
- 大多数人不知道正在投票
- 投票仅代表 ETH 持有人，而非系统中任何其他参与者

部分社区用户拒绝分叉，主要是因为他们认为该 DAO 事件不是协议缺陷。 他们随后成立了 以太坊经典 。

如今，以太坊社区已经采取了不干预合约漏洞或资金损失的政策，以保持系统的可信中立性。

观看关于 DAO 黑客事件的更多信息：

分叉的用途 {#forking-utility}

以太坊/Ethereum Classic 分叉是一个良好分叉的优秀案例。 我们有两组人，他们在一些核心价值观上的分歧很大，因而认为值得冒险去坚持他们的具体行动方案。

面对重大的政治、哲学或经济分歧时能够分叉，这一点对于以太坊的成功治理意义重大。 如果无法分叉，替代方案就是持续内斗，让最终组成 Ethereum Classic 的人们被迫不情愿地参与，并且对于未来如何成功的看法也越来越不一致。

信标链开发 {#beacon-chain}

以太坊的治理过程往往以速度和效率换取开放性和包容性。 为了加快信标链的发展，它与以太坊网络的工作量证明分开启动，并遵循自己的治理惯例。

虽然规范和实施开发一直是完全开源的，但并没有使用正式的程序提出上述更新。 这使研究人员和执行人员能够更快地确定和商定各种变化。

当信标链与以太坊执行层合并时，提议修改的治理程序将得到协调。 实施合并的过程已经在进行中。

关于合并的更多信息

如何参与其中？ {#get-involved}

- 提出 EIP
- 讨论目前的提案
- 参与 R&D 讨论
- 加入以太坊 R&D Discord
- 运行节点
- 参与开发
- 核心开发者学徒计划

延伸阅读 {#further-reading}

以太坊的治理并没有严格定义。 各社区参与者对此有着不同的看法。 以下是其中一部分：

- Notes on Blockchain Governance - *Vitalik Buterin*
- Governance on Ethereum – *ETHHub*
- How does Ethereum Governance work? – *Cryptotesters*
- How Ethereum governance works – *Micah Zoltu*
- What is an Ethereum core developer? - *Hudson Jameson*
- Governance, Part 2: Plutocracy Is Still Bad - *Vitalik Buterin*
- Moving beyond coin voting governance - *Vitalik Buterin*

---

title: 以太坊的历史和分叉
description: 以太坊区块链的历史，包括了主要的里程碑、版本发布和分叉。
lang: zh
sidebar: true

sidebarDepth: 1

以太坊的历史 {#the-history-of-ethereum}

所有主要里程碑、分叉和更新以太坊区块链的时间表

分叉是需要对网络进行重大技术升级或更改时的变化——它们通常源自【以太坊改进建议（EIP）】(/eips/)，并且更改以太坊协议的“规则”。

当传统的中心化软件需要升级时，公司会为终端用户发布新版本。 因为没有中心化所有权，区块链以不同的方式运作。 【以太坊客户端】(/developers/docs/nodes-and-clients/) 必须升级他们的软件来执行新的分叉规则。 直链区块创造者（POW 中的矿工，POS 中的验证者）和节点必须创造区块和按照新规则进行验证。 关于共识机制的更多信息

这些规则更改可能会在网络中造成暂时的分叉。 新区块的产生，可以来自新规则，也可以来自旧规则。 分叉通常提前商定，以便让客户端能够采用 Unison 的升级，升级后的分叉链成为主链。 然而，在极少数情况下，对分叉的不同意见可能导致网络永久硬分叉——最为著名的是【DAO fork】(#dao-fork) 的分叉，产生了 Ethereum Classic（ETC-ETH）。

2021 {#2021}

(*进行中*) 牵牛星升级 {#altair}

Altair 升级是信标链的第一次预定升级。 预计将于 2021 年启用。 它将增加对“同步委员会”的支持，可以启用轻型客户端，并将不活动和削价的惩罚提高到其全部价值。

- 阅读 Altair 升级说明

伦敦 {#london}

<Emoji text="📅" size={1} mr={"0.5rem"} mb={"0.5rem"} /> `世界标准时间 2021 年 8 月 5 日下午 12 点 33 分 42 秒`

<Emoji text=":bricks:" size={1} mr={"0.5rem"} mb={"0.5rem"} /> 区块号：12,965,000

<Emoji text=":money_bag:" size={1} mr={"0.5rem"} mb={"0.5rem"} />ETH 价格：2621 美金

<Emoji text="🖥" size={1} mr={"0.5rem"} mb={"0.5rem"} /> ethereum.org on waybackmachine

概览 {#london-summary}

伦敦升级版引入了 EIP-1559，对交易费市场进行了改革，同时还对 gas 费用退款的处理方式和冰河世纪日程进行了修改。

- 您是去中心化应用开发者吗？ 请务必升级您的库和工具包。
- 请阅读以太坊基金会公告
- 请阅读以太坊 Cat Herder 的解释

- EIP-1559 – *improves the transaction fee market*
- EIP-3198 – *returns the* `BASEFEE` *from a block*
- EIP-3529 - *reduces gas refunds for EVM operations*
- EIP-3541 - *prevents deploying contracts starting with* `0xEF`
- EIP-3554 – *delays the Ice Age until December 2021*

柏林升级 {#berlin}

<Emoji text="📅" size={1} mr={"0.5rem"} mb={"0.5rem"} /> `Apr-15-2021 10:07:03 AM +UTC`

<Emoji text=":bricks:" size={1} mr={"0.5rem"} mb={"0.5rem"} /> 区块高度：12,965,000

<Emoji text=":money_bag:" size={1} mr={"0.5rem"} mb={"0.5rem"} />以太坊价格：2621 美金

<Emoji text="💻" size={1} mr={"0.5rem"} mb={"0.5rem"} /> ethereum.org on waybackmachine

概览 {#berlin-summary}

柏林升级优化了某些 EVM 行动的 Gas 成本，并增加了对多种交易类型的支持。

- 请阅读以太坊基金会公告
- 请阅读以太坊 Cat Herder 的解释

- EIP-2565 – *lowers ModExp gas cost*
- EIP-2718 – *enables easier support for multiple transaction types*
- EIP-2929 – *gas cost increases for state access opcodes*
- EIP-2930 – *adds optional access lists*

2020 {#2020}

信标链的起源 {#beacon-chain-genesis}

<Emoji text="📅" size={1} mr={"0.5rem"} mb={"0.5rem"} /> `Dec-01-2020 12:00:35 PM +UTC`

<Emoji text=":bricks:" size={1} mr={"0.5rem"} mb={"0.5rem"} /> 信标链区块号：1

<Emoji text=":money_bag:" size={1} mr={"0.5rem"} mb={"0.5rem"} /> ETH 价格：586.23 美元

<Emoji text="💻" size={1} mr={"0.5rem"} mb={"0.5rem"} /> ethereum.org

概览 {#beacon-chain-genesis-summary}

信标链需要 16384 个 ETH 并且每个节点拥有 32 个 ETH，以保证网络安全。 这发生在 2020 年 11 月 27 日，意味着信标链在 2020 年 12 月 1 日开始产生区块。 这是实现 Eth2.0 愿景的重要一步。

请阅读以太坊基金会公告

信标链

质押合约部署 {#staking-deposit-contract}

<Emoji text="📅" size={1} mr={"0.5rem"} mb={"0.5rem"} /> `Oct-14-2020 09:22:52 AM +UTC`

<Emoji text=":bricks:" size={1} mr={"0.5rem"} mb={"0.5rem"} /> 区块号：11052984

<Emoji text=":money_bag:" size={1} mr={"0.5rem"} mb={"0.5rem"} /> ETH 价格：379.04 美元

<Emoji text="💻" size={1} mr={"0.5rem"} mb={"0.5rem"} /> waybackmachine 上 ethereum.org

概览 {#deposit-contract-summary}

质押存款合约将质押引入以太坊生态系统。 虽然这只是一个主网 合约，但它直接影响了 信标链的发布，而这是Eth2 升级的重要部分。

请阅读以太坊基金会公告

权益质押

缪尔冰川升级 {#muir-glacier}

<Emoji text="📅" size={1} mr={"0.5rem"} mb={"0.5rem"} /> `Jan-02-2020 08:30:49 AM +UTC`

<Emoji text=":bricks:" size={1} mr={"0.5rem"} mb={"0.5rem"} /> 区块号：9200000

<Emoji text=":money_bag:" size={1} mr={"0.5rem"} mb={"0.5rem"} /> ETH 价格：127.18 美元

<Emoji text="🖥" size={1} mr={"0.5rem"} mb={"0.5rem"} /> waybackmachine 上 ethereum.org

概览 {#muir-glacier-summary}

缪尔冰川分叉将 难度炸弹 的启动延迟。 增加Pow 的区块难度可能会增加发送交易和使用数据库的等待时间，从而降低以太坊的可用性。

- 请阅读以太坊基金会公告
- 请阅读 Ethereum Cat Herder 的解释

- EIP-2384 – *delays the difficulty bomb for another 4,000,000 blocks, or ~611 days.*

2019 {#2019}

伊斯坦布尔升级 {#istanbul}

<Emoji text="📅" size={1} mr={"0.5rem"} mb={"0.5rem"} /> `Dec-08-2019 12:25:09 AM +UTC`

<Emoji text=":bricks:" size={1} mr={"0.5rem"} mb={"0.5rem"} /> 区块号：9069000

<Emoji text=":money_bag:" size={1} mr={"0.5rem"} mb={"0.5rem"} /> ETH 价格：151.06 美元

<Emoji text="🖥" size={1} mr={"0.5rem"} mb={"0.5rem"} /> waybackmachine 上 ethereum.org

概览 {#istanbul-summary}

伊斯坦布尔分叉：

- 优化 EVM 数据存储操作的 gas 耗用量计量方式。
- 提高拒绝服务（DoS）攻击的弹性。
- 使基于 SNARK 和 STARK 的第二层方案性能更佳。
- 使以太坊和 Zcash 能够互操作。
- 让合约能够引入更有创造性的功能。

请阅读以太坊基金会公告

- EIP-152 – *allow Ethereum to work with privacy-preserving currency like Zcash.*

- EIP-1108 – *cheaper cryptography to improve gas costs.*
- EIP-1344 – *protects Ethereum against replay attacks by adding* `CHAINID` *opcode.*
- EIP-1884 – *optimising opcode gas prices based on consumption.*
- EIP-2028 – *reduces the cost of CallData to allow more data in blocks – good for Layer 2 scaling.*
- EIP-2200 – *other opcode gas price alterations.*

君士坦丁堡升级 {#constantinople}

<Emoji text="📅" size={1} mr={"0.5rem"} mb={"0.5rem"} /> `Feb-28-2019 07:52:04 PM +UTC`

<Emoji text=":bricks:" size={1} mr={"0.5rem"} mb={"0.5rem"} /> 区块号：7280000

<Emoji text=":money_bag:" size={1} mr={"0.5rem"} mb={"0.5rem"} /> ETH 价格：136.29 美元

<Emoji text="🖥" size={1} mr={"0.5rem"} mb={"0.5rem"} /> waybackmachine 上 ethereum.org

概览 {#constantinople-summary}

君士坦丁堡分叉：

- 保证在 POS 实现前区块不会被冻结。
- 优化 EVM 数据存储操作的 gas 耗用量计量方式。
- 添加了与尚未创建的地址进行交互的能力。

请阅读以太坊基金会公告

- EIP-145 – *optimises cost of certain on-chain actions.*
- EIP-1014 – *allows you to interact with addresses that have yet to be created.*
- EIP-1052 – *optimises cost of certain on-chain actions.*
- EIP-1234 – *makes sure the blockchain doesn't freeze before proof-of-stake.*

2017 {#2017}

Byzantium {#byzantium}

<Emoji text="📅" size={1} mr={"0.5rem"} mb={"0.5rem"} /> `Oct-16-2017 05:22:11 AM +UTC`

<Emoji text=":bricks:" size={1} mr={"0.5rem"} mb={"0.5rem"} /> 区块号：4370000

<Emoji text=":money_bag:" size={1} mr={"0.5rem"} mb={"0.5rem"} /> ETH 价格：334.23 美元

<Emoji text="🖥" size={1} mr={"0.5rem"} mb={"0.5rem"} /> waybackmachine 上 ethereum.org

概览 {#byzantium-summary}

拜占庭分叉：

- 将区块挖矿奖励从 5 ETH 减少到 3 ETH。
- 将难度炸弹 升级延迟一年。
- 增加了调用其他合约的能力。
- 增加了某些密码学方法，允许2 层缩放。

请阅读以太坊基金会公告

- EIP-140 – *adds* `REVERT` *opcode.*
- EIP-658 – *status field added to transaction receipts to indicate success or failure.*
- EIP-196 – *adds elliptic curve and scalar multiplication to allow for ZK-Snarks.*
- EIP-197 – *adds elliptic curve and scalar multiplication to allow for ZK-Snarks.*
- EIP-198 – *enables RSA signature verification.*
- EIP-211 – *adds support for variable length return values.*
- EIP-214 – *adds* `STATICCALL` *opcode, allowing non-state-changing calls to other contracts.*
- EIP-100 – *changes difficulty adjustment formula.*
- EIP-649 – *delays difficulty bomb by 1 year and reduces block reward from 5 to 3 ETH.*

2016 {#2016}

Spurious Dragon {#spurious-dragon}

<Emoji text="📅" size={1} mr={"0.5rem"} mb={"0.5rem"} /> `Nov-22-2016 04:15:44 PM +UTC`

<Emoji text=":bricks:" size={1} mr={"0.5rem"} mb={"0.5rem"} /> 区块号：2675000

<Emoji text=":money_bag:" size={1} mr={"0.5rem"} mb={"0.5rem"} /> ETH 价格：9.84 美元

<Emoji text="💻" size={1} mr={"0.5rem"} mb={"0.5rem"} /> waybackmachine 上 ethereum.org

概览 {#spurious-dragon-summary}

对拒绝服务（DoS）攻击（2016 年 9 月/10 月）的第二个反应是 Spurious Dragon 分叉，包括：

- 调整操作码的价格以防止今后对网络的攻击。
- 启用区块链状态的“区块链减重”。
- 添加重放攻击保护。

请阅读以太坊基金会公告

- EIP-155 – *prevents transactions from one Ethereum chain from being rebroadcasted on an alternative chain, for example a testnet transaction being replayed on the main Ethereum chain.*
- EIP-160 – *adjusts prices of* `EXP` *opcode – makes it more difficult to slow down the network via computationally expensive contract operations.*
- EIP-161 – *allows for removal of empty accounts added via the DOS attacks.*
- EIP-170 – *changes the maximum code size that a contract on the blockchain can have – to 24576 bytes.*

Tangerine Whistle {#tangerine-whistle}

<Emoji text="📅" size={1} mr={"0.5rem"} mb={"0.5rem"} /> `Oct-18-2016 01:19:31 PM +UTC`

<Emoji text=":bricks:" size={1} mr={"0.5rem"} mb={"0.5rem"} /> 区块号：2463000

<Emoji text=":money_bag:" size={1} mr={"0.5rem"} mb={"0.5rem"} /> ETH 价格：12.50 美元

<Emoji text="💻" size={1} mr={"0.5rem"} mb={"0.5rem"} /> waybackmachine 上 ethereum.org

概览 {#tangerine-whistle-summary}

Tangerine Whistle 分叉是第一次对拒绝提供服务（DoS）网络攻击作出的反应（2016 年 9 月/10 月），包括：

- 处理与价格低廉的操作代码有关的紧急网络健康问题。

请阅读以太坊基金会公告

- EIP-150 – *increases gas costs of opcodes that can be used in spam attacks.*
- EIP-158 – *reduces state size by removing a large number of empty accounts that were put in the state at very low cost due to flaws in earlier versions of the Ethereum protocol.*

DAO 分叉 {#dao-fork}

<Emoji text="📅" size={1} mr={"0.5rem"} mb={"0.5rem"} /> `Jul-20-2016 01:20:40 PM +UTC`

<Emoji text=":bricks:" size={1} mr={"0.5rem"} mb={"0.5rem"} />区块号：1920000

<Emoji text=":money_bag:" size={1} mr={"0.5rem"} mb={"0.5rem"} /> ETH 价格：12.54 美元

<Emoji text="🖥" size={1} mr={"0.5rem"} mb={"0.5rem"} /> waybackmachine 上 ethereum.org

概览 {#dao-fork-summary}

DAO 分叉是为了解决 2016 DAO 攻击，当时一个不安全 DAO 合约被黑客盗走了超过 3 百万个 ETH。此分叉将资金从有问题的合约转移到一个新的合约，只有一个功能：提款。 任何损失资金的人都可以在他们的钱包中为每 100 个 DAO 代币提取 1 个 ETH。

这一决定是由以太坊社区投票决定的。 任何 ETH 持有人都能够通过一个交易在一个投票平台进行投票。 分叉的决定获得了 85% 以上的票。

一些矿工拒绝分叉，因为 DAO 事件并不是协议中的一个缺陷。 他们随后成为了 Ethereum Classic。

请阅读以太坊基金会公告

Homestead {#homestead}

<Emoji text="📅" size={1} mr={"0.5rem"} mb={"0.5rem"} /> `Mar-14-2016 06:49:53 PM +UTC`

<Emoji text=":bricks:" size={1} mr={"0.5rem"} mb={"0.5rem"} /> 区块号：1150000

<Emoji text=":money_bag:" size={1} mr={"0.5rem"} mb={"0.5rem"} /> ETH 价格：12.50 美元

<Emoji text="🖥" size={1} mr={"0.5rem"} mb={"0.5rem"} /> waybackmachine 上 ethereum.org

概览 {#homestead-summary}

展望未来的 Homestead 分叉。 其中包括若干协议修改和网络变更，使以太坊能够进一步进行网络升级。

请阅读以太坊基金会公告

- EIP-2 – *makes edits to contract creation process.*

- EIP-7 – *adds new opcode:* `DELEGATECALL`
- EIP-8 – *introduces devp2p forward compatibility requirements*

2015 {#2015}

Frontier thawing {#frontier-thawing}

<Emoji text="📅" size={1} mr={"0.5rem"} mb={"0.5rem"} /> `Sep-07-2015 09:33:09 PM +UTC`

<Emoji text=":bricks:" size={1} mr={"0.5rem"} mb={"0.5rem"} /> 区块号：200000

<Emoji text=":money_bag:" size={1} mr={"0.5rem"} mb={"0.5rem"} /> ETH 价格：1.24 美元

<Emoji text="🖥" size={1} mr={"0.5rem"} mb={"0.5rem"} /> waybackmachine 上 ethereum.org

概览 {#frontier-thawing-summary}

Frontier thawing 分叉取消了每个区块 5,000 gas 的限制，并将默认的 gas 价格设置为 51 gwei。 这开启了交易功能——交易需要 21,000 gas。

请阅读以太坊基金会公告

Frontier {#frontier}

<Emoji text="📅" size={1} mr={"0.5rem"} mb={"0.5rem"} /> `Jul-30-2015 03:26:13 PM +UTC`

<Emoji text=":bricks:" size={1} mr={"0.5rem"} mb={"0.5rem"} /> 区块号：0

<Emoji text=":money_bag:" size={1} mr={"0.5rem"} mb={"0.5rem"} /> ETP 价格：不适用

<Emoji text="🖥" size={1} mr={"0.5rem"} mb={"0.5rem"} /> waybackmachine 上 ethereum.org

概览 {#frontier-summary}

边境是以太坊的最初版本，但在上面能做的事情很少。 它是在 Olympic 测试阶段之后进行的。 它面向技术用户，特别是开发者。 每个区块有一个 gas 限制为 5,000。 这个“缓冲”期使矿工能够开始工作，并使早期采用者能够安装他们的客户端而不必“匆忙”。

请阅读以太坊基金会公告

2014 {#2014}

公开募资 {#ether-sale}

<Emoji text="📅" size={1} mr={"0.5rem"} mb={"0.5rem"} /> 2014 年 7 月 22 日 - 9 月 2 日

<Emoji text="🖥" size={1} mr={"0.5rem"} mb={"0.5rem"} /> waybackmachine 上 ethereum.org

以太币预售为期 42 天。 你可以使用比特币进行购买。

请阅读以太坊基金会公告

黄皮书发布 {#yellowpaper}

<Emoji text="📅" size={1} mr={"0.5rem"} mb={"0.5rem"} /> 2014 年 4 月 1 日

<Emoji text="💻" size={1} mr={"0.5rem"} mb={"0.5rem"} /> waybackmachine 上 ethereum.org

Gavin Wood 博士撰写的黄皮书，是关于以太坊协议的技术定义。

查看黄皮书

2013 {#2013}

白皮书发布 {#whitepaper}

<Emoji text="📅" size={1} mr={"0.5rem"} mb={"0.5rem"} /> 2013 年 11 月 27 日

<Emoji text="💻" size={1} mr={"0.5rem"} mb={"0.5rem"} /> waybackmachine 上 ethereum.org

该项目在 2015 年启动。但早在 2013 年，以太坊的创始人 Vitalik Buterin 就发表了介绍性文章。

白皮书

---

title: 非同质化代币 (NFT)
description: 以太坊上的 NFT 简介
lang: zh
template: use-cases
emoji: ":frame_with_picture:"
sidebar: true
sidebarDepth: 2
image: ../../../../assets/infrastructure_transparent.png
alt: 通过全息图显示的 ETH 徽标。
summaryPoint1: 一种可以用以太坊资产表示任何独特事物的方法。
summaryPoint2: NFT 相比较以往，更能赋予内容创作者更多的权利。

summaryPoint3: 由以太坊区块链上的智能合约提供支持

NFT 目前在数字艺术品和收藏品世界中掀起风潮。 得益于新加密货币受众带来巨额销售，数字艺术家的生活正在发生变化。 一些名人也纷纷加入，因为他们发现了与粉丝们联系的新契机。 但数字艺术品只是使用 NFT 的一种方式。 实际上，它们可以代表任何独特资产的所有权，如数字世界或物理王国里持有某项物品的凭据。

如果 Andy Warhol 出生在 90 年代后期，他很可能会将 Campbell's Soup 制作成一个 NFT。 Kanye 在以太坊上运行 Yeezys 只是时间问题。 某一天，您可以用 NFT 来证明自己对汽车的所有权。

什么是 NFT？ {#what-are-nfts}

NFT 是我们用以代表独特物品所有权的代币。 NFT 让我们把诸如艺术品、收藏品、甚至房地产等物品代币化。 他们一次只有一个正式主人，并且受到以太坊区块链的保护 - 没有人可以修改所有权记录或者根据现有的 NFT 复制粘贴一份新的。

NFT 代表非同质化代币。 非同质化是一个经济术语，您可以用它来描述家具、歌曲文件或您的电脑等物品。 这些东西不能与其他物品互换，因为它们具有独特属性。

另一方面，同质化物品可以互换，这取决于它们的价值而非独特属性。 例如，ETH 或美元具有同质化属性，因为 1 ETH/1 USD 可以兑换成另外的 1 ETH/1 USD。

资产互联网 {#internet-of-assets}

NFT 和以太坊解决了当前互联网上存在的一些问题。 随着一切都变得更加数字化，我们需要复制实体物品的属性，如稀缺性、独特性和所有权证明。 更不用说数字物品常常只是在其产品范围内有用。 例如，您不能转售已购买的 iTunes mp3，又或者说，即便有市场，您也无法将一家公司的忠诚度积分兑换成其他平台的信用额度。

与我们今天大多数人使用的互联网相比，下面是使用 NFT 互联网的情况...

对比 {#nft-comparison}

NFT 互联网	当今互联网
NFT 在数字世界里是独一无二的，没有两个相同的 NFT。	文件副本（如.mp3 或 .jpg）与原始文件并无二致。
每个 NFT 必须有一个所有者，而且是一条公开记录，任何人都可以轻松核实。	数字物品的所有权记录存储在由机构控制的服务器上 - 您必须相信他们的话。
NFT 与使用以太坊构建的任何内容都兼容。 一场活动的 NFT 门票可以在任意以太坊市场上进行交易，兑换完全不同的 NFT。 您可以用一件艺术品换取一张门票！	拥有数字产品的公司必须构建自己的基础设施。 例如，一个售卖活动数字门票的应用程序必须建立自己的门票交易所。
内容创建人可以在任何地方出售他们的作品，并可以进入全球市场。	创建人依靠所用平台的基础设施和分布。 这些通常受到使用条款和地理限制的制约。
创建人可以保留对自己作品的所有权，并直接要求收取转售版税。	音乐媒体服务等平台扣留了大部分销售利润。
人们可以通过一些出人意料的方式使用物品。 例如，您可以将数字作品用作去中心化的贷款的抵押物。	

NFT 示例 {#nft-examples}

NFT 世界相对较新。 理论上，NFT 的应用范围是任何需要证明所有权的独特事物。 以下是现今一些 NFT 示例，有助于了解情况。

- 独特的数字艺术品
- 独一无二的限量版时尚系列跑鞋
- 游戏内物品
- 文章
- 数字收藏品
- 域名
- 可以参加某项活动的门票或优惠券

ethereum.org 示例 {#ethereum-org-examples}

我们用 NFT 来回馈我们的贡献者，我们甚至有自己的 NFT 域名。

POAP（出勤证明协议） {#poaps}

如果您对 ethereum.org 做了贡献，即可认领 POAP NFT。 这些是证明您参加过活动的收藏品。 一些秘密聚会使用 POAP 作为一种活动门票。 关于贡献的更多信息。

ethereum.org POAP

ethereum.eth {#ethereum-dot-eth}

该网站有一个由 NFT 提供支持的备用域名：**ethereum.eth**。 我们的 `.org` 地址由域名系统 (DNS) 提供商集中管理，而 ethereum `.eth` 则通过以太坊域名服务 (ENS) 在以太坊上注册。 并由我们所有与管理。 查看我们的 ENS 记录

关于 ENS 的更多信息

NFT 的工作原理是什么？ {#how-nfts-work}

NFT 不同于 DAI 或 LINK 等 ERC-20 代币，因为每个代币都完全独一无二、不可分割。 NFT 支持分配或声明任何独特数字数据的所有权，可通过使用以太坊的区块链作为公共账本进行追踪。 NFT 由代表数字或非数字资产的数字对象铸成。 例如，NFT 可以代表：

- 数字艺术品：
  - GIF
  - 收藏品
  - 音乐
  - 视频
- 实物：
  - 车内饰品
  - 真实世界活动的门票
  - 代币化发票
  - 法律文档
  - 签名
- 有很多很多的选择，可以发挥创意。

NFT 一次只能有一个所有者。 通过唯一的 ID 和其他代币无法复制的元数据管理所有权。 NFT 通过智能合约铸造，智能合约分配 NFT 的所有权并管理它们的可转让性。 有人创建或铸造 NFT 时，他们会执行存储在符合不同标准的智能合约中的代码，如 ERC-721。 此信息会添加到正在管理 NFT 的区块链中。 从高层次来看，铸造过程具有以下步骤：

- 新建区块
- 验证信息
- 将信息录入区块链

NFT 有一些特殊属性：

- 铸造的每个代币都有直接链接到以太坊地址的唯一标识符。
- 不能直接与其代币 1:1 互换。 例如，1 个 ETH 与另一个 ETH 完全相同。 NFT 却并非如此。
- 每个代币都有主人，这个信息很容易验证。
- 它们依托于以太坊，可以在任何基于以太坊的 NFT 市场上买卖。

换句话说，如果您*拥有*一个 NFT：

- 您可以轻松地证明自己拥有它。
    - 证明您拥有 NFT 和证明您的帐户中有 ETH 十分类似。
    - 例如，假设您购买了一个 NFT，此唯一代币的所有权通过公共地址转移到您的钱包里。
    - 该代币证明您的数字文件副本是原件。
    - 您的私钥是原始的所有权证明。
    - 内容创建人的公钥作为该特定数字艺术品的真实性证书。
        - 创建人的公钥基本上是代币历史的永久组成部分。 创建人的公钥可以证明您持有的代币是由特定个人制作，从而促进其市场价值（与假货相比）。
    - 证明您拥有 NFT 的另一种方式是通过签署信息来证明您拥有地址背后的私钥。
        - 如上所述，您的私钥是原始所有权证明。 由此可以得知，地址背后的私钥掌控着 NFT。
        - 通过已签名消息，无需向任何人透露私钥即可证明您拥有私钥，进而证明您拥有 NFT！
- 没有人能够以任何方式操纵它。
- 您可以出售它，在某些情况下，这可以让原创建人赚到转售版税。
- 或者，您可以一直持有它，知道您的资产由以太坊上的钱包担保，便可安心无虞。

并且如果您*创建了*一个 NFT：

- 您可以轻松地证明自己是创建人。
- 由您确定其稀有性。
- 每次出售您都可以赚取版税。
- 您可以在任何 NFT 市场或点对点出售。 您没有被限制在任何平台上，也不需要任何中介。

稀缺性 {#scarcity}

NFT 的创建人需要决定其资产的稀缺性。

例如，拿体育活动的门票来说。 就像活动组织方可以选择出售多少门票一样，NFT 的创建人可以决定有多少份复制品。 有的是完整复制品，如 5000 张普通门票。 有时会制作一些非常相似但略有不同的门票，如指定座位的门票。 在另一种情况下，创建人可能想创造一种 NFT，其中只有一个被当成特殊的稀有收藏品。

在这些情况下，每个 NFT 仍然有一个独特的标识符（像传统“票据”上的条形码），只有一个所有者。 NFT 的预期稀缺性很重要，这取决于创建人。 创建人可能打算使每个 NFT 完全独特以营造稀缺性，或者有理由生产几千个复制品。 记住，这些信息都是公开的。

版税 {#royalties}

某些 NFT 会自动向其创建人支付版税。 这个概念还有待完善，但不失为最强大的概念之一。 EulerBeats Originals 的原始所有者在 NFT 每次出售时都会获得 8% 的版税。 而一些平台，如 Foundation 和 Zora，则支持为其艺术家提供版税。

这是完全自动化的，他们的作品会在人们手上流转，因此创建人可以坐赚版税。 目前，弄清楚版税是一个体力活，而且缺乏准确性 - 很多创建人都没有得到应得的报酬。 如果您的 NFT 已编入版税，您就不会错过。

NFT 有哪些用武之地？ {#nft-use-cases}

以下详细描述了一些以太坊上发展不错的 NFT 使用案例和愿景。

- 数字内容
- 游戏物品

- 域名
- 实体物品
- 投资和抵押品

为创建人带来更大的收益 {#nfts-for-creators}

目前，NFT 的最大用途是在数字内容领域。 这是因为现在该行业已走向崩坏。 内容创建人眼看着自己的利润和潜在收入被平台吞噬。

艺术家在社交网络上发表作品，为平台赚钱，而平台向艺术家的粉丝出售广告。 他们得到曝光度的回报，但曝光度并不能带来真金白银。

NFT 为新的创建人经济提供动力，创建人不会将其内容的所有权交给他们用来发布的平台。 所有权融入到内容本身。

当他们出售内容时，资金直接转给他们。 如果新所有者随后出售 NFT，原创建人甚至可以自动收到版税。 这在每次出售时都有保证，因为创建人的地址是代币元数据的一部分 - 元数据无法修改。

探索、购买或创建您自己的 NFT 艺术/收藏品...

探索 NFT 艺术

复制/粘贴问题 {#nfts-copy-paste}

反对者们常常提及这样一件事实，他们一般会拿出一个 NFT 作品截图的图片来证明 NFT 是“愚蠢至极”的。 “看，现在我免费得到这个图像了！”他们得意地说。

是的。 但是，在谷歌上搜索毕加索的《格尔尼卡》图片，是否会让您骄傲地成为这件在艺术史上价值数百万美元的画作的新主人呢？

归根结底，拥有真正的东西才能享有市场赋予它的价值。 一段内容被屏幕抓取、分享和普遍使用的次数越多，它的价值就越大。

拥有可供鉴定的真迹总会比赝品更有价值。

提升游戏的潜力 {#nft-gaming}

NFT 已经看到了来自游戏开发者的极大兴趣。 NFT 可以为游戏中的物品提供所有权记录，促进游戏中的经济发展，并为玩家带来很多好处。

在许多常规游戏中，您可以购买物品供您在游戏中使用。 但如果该物品是一个 NFT，您可以在游戏完成后将其出售来收回本钱。 如果该物品增值了的话，您甚至还可以额外赚一笔。

对于游戏开发者（作为 NFT 发行者）来说，商品在开放市场上重新销售时，他们都可以赚取版税。 这就创造了一个更加互惠的商业模式，玩家和开发人员都可以从二级 NFT 市场获利。

这也意味着，如果开发商不再维护该游戏，您收集的东西仍然归您所有。

最终，您在游戏磨练中获得的物品可以超过游戏本身。 即使游戏不再维护，物品将始终为您所有。 这意味着游戏内的物品变成数字纪念品，并且在游戏外也有价值。

Decentraland 是一款虚拟现实游戏，甚至允许玩家购买代表虚拟土地的 NFT，您可以按需使用这些土地。

查看由 NFT 提供支持的以太坊游戏。

探索 NFT 游戏

让以太坊地址更好记 {#nft-domains}

以太坊域名服务使用 NFT 为您的以太坊地址提供一个好记的名字，如 `mywallet.eth`。 这意味着您可以要求某人通过 `mywallet.eth` 而不是 `0x123456789.....` 来向您发送 ETH。

这与网站域名相似，使得 IP 地址更好记。 跟域名一样，ENS 名称有价值，通常取决于长度和相关性。 有了 ENS，就不再需要域名注册机构来促进所有权转移。 相反，您可以在 NFT 市场上交易您的 ENS 名称。

您的 ENS 名称可以：

- 接收加密货币和其他 NFT。
- 指向一个去中心化网站，例如 ethereum.eth。 关于网站去中心化的更多信息
- 存储任何信息，包括电子邮件地址和 Twitter 用户名等个人资料信息。

实体物品 {#nft-physical-items}

实物物品的代币化还没有像数字物品那样发达。 但是有许多项目在探索房地产、独特的时尚物品等实物的代币化。

由于 NFT 本质上是契约，有一天您可以用以太坊买车或买房，并收到作为 NFT 回报的契约（在同一交易中）。 随着事情变得越来越高科技，不难想象，在这个世界上，您的以太坊钱包会成为车子或房子的钥匙 - 加密的所有权证明可以用来开锁。

随着像汽车和财产这类贵重资产得以在以太坊上标识，您可以在去中心化贷款中使用 NFT 作为抵押品。 如果您没有现金或数字货币，但拥有有价值的实物物品，这尤为有用。 关于 DeFi 的更多信息

NFT 和 DeFi {#nfts-and-defi}

NFT 世界和去中心化金融 (DeFi) 世界正在以一些有趣的方式开始合作。

NFT 担保贷款 {#nft-backed-loans}

有一些 DeFi 应用程序允许通过抵押借款。 例如，您抵押了 10 个 ETH，所以可以借到 5000 个 DAI（一个稳定币）。 这保证了贷款人得到回报 - 如果借款人不偿还 DAI，抵押品就会发送给贷款人。 然而，并非每个人都有足够的数字货币来作为抵押品。

一些项目开始探索使用 NFT 作为抵押品。 想象一下，您当年买了一种稀有的 CryptoPunk NFT - 以今天的价格，它们可以卖到 1000 美元。 以此作为抵押品，您可以按同样的规则获得贷款。 如果不退还 DAI，您的 CryptoPunk 将作为抵押品发送给贷款人。 这最终可以适用于任何 NFT 化的物品。

而这在以太坊上并不难，因为两个体系（NFT 和 DeFi）采用相同的基础架构。

分割所有权 {#fractional-ownership}

NFT 创建人也可以为他们的 NFT 创建“股份”。 这使投资者和粉丝们有机会拥有 NFT 的一部分，而不必购买全部。 这为 NFT 的铸造者和收藏家们增加了更多机会。

- 部分产权化的 NFT 可以在 DEX 上像 Uniswap 一样交易，而不仅是 NFT 市场。 这意味着有更多的买家和卖家。
- NFT 的总价格可以根据其部分的价格来确定。
- 您有更多的机会拥有您关注的物品并从中获利。 拥有 NFT 的人更难被定价。

这仍处于试验中，但您可以在以下交易所了解关于分割 NFT 所有权的更多信息。

- NIFTEX
- NFTX

从理论上讲，这将释放更多可能性，比如拥有一件毕加索的作品。 您将成为毕加索 NFT 的股东，这意味着您将在受益分红等方面拥有发言权。 很有可能在不久的将来，拥有 NFT 的部分所有权就能让您进入一个去中心化自治组织 (DAO) 来管理该资产。

这些是由以太坊提供支持的组织，允许陌生人（如某个资产的全球股东）在不一定要信任其他人的情况下安全地加以协调。 这是因为未经团体批准就无法花费一分钱。

正如我们所提到的，这是一个新兴空间。 NFT、DAO、分割的代币都以不同的速度发展。 但他们所有的基础设施都存在，并且可以轻松地系统工作，因为他们都使用同样的语言：以太坊。 因此，不妨关注一下这个领域。

关于 DAO 的更多信息

以太坊与 NFT {#ethereum-and-nfts}

以太坊使 NFT 得以运行的原因如下：

- 交易历史和代币元数据可以公开核实 - 证明所有权历史很简单。
- 一旦确认交易，几乎不可能操纵这些数据来“窃取”所有权。
- NFT 的交易可以点对点进行，无需利用平台中介，因为中介会大量克扣作为补偿。
- 所有以太坊产品共享相同的“后端”。 换句话说，所有以太坊产品都可以轻松地相互理解 - 这使得 NFT 可以跨产品移植。 您可以在一个产品上购买 NFT，然后在另一个产品上轻松出售。 作为创建人，您可以同时在多个产品上展示您的 NFT - 每个产品都拥有最新的所有权信息。
- 以太坊永远不会倒闭，这意味着您的代币将永远可以出售。

NFT 的环境影响 {#environmental-impact-nfts}

NFT 越来越受欢迎，这意味着它们受到的监管也日益增强 - 特别是碳足迹。

我们要阐明以下几点：

- NFT 并没有直接增加以太坊的碳足迹。
- 以太坊保证资金和资产安全的方式目前需要耗费大量能源，但这一点将得到改进。
- 一旦改进，以太坊的碳足迹将减少 99.95%，使其比许多现有行业更节能。

为了进一步说明，我们将需要采取更多技术手段，所以请耐心等待...

不要将其归咎于 NFT {#nft-qualities}

整个 NFT 生态系统行之有效，得归功于以太坊的去中心化特性与安全性。

去中心化意味着您和其他任何人都可以证明您拥有某件东西。 所有这些都不需要信任或将监管权授予可以随意强加规则的第三方。 这也意味着您的 NFT 可在许多不同的产品和市场间流通。

安全性意味着任何人都不能复制/粘贴或偷窃您的 NFT。

得益于以太坊的这些特性，现在可以通过数字方式拥有独特的物品并为您的内容获得合理价格。 但这样做需要付出代价。 像比特币和以太坊这样的区块链现在为能源密集型，因为需要大量的能源来维护这些特性。 如果能够简单通过重写以太坊的历史来窃取 NFT 或者加密货币，整个系统就会崩溃。

铸造 NFT 所需要做的工作 {#minting-nfts}

当您铸造 NFT 时，会发生以下情况：

- 需要将其确认为链上资产。
- 帐户所有者的帐户余额必须更新以包含该资产。 从而以便之后进行交易或验证为自己“所有”。
- 确认上述内容的交易需要被添加到区块中，并在链上“永存”。
- 该区块需要由网络中的所有人确认为“正确”。 这种共识消除了对中介机构的需求，因为网络同意您的 NFT 存在并属于您。 这是在链上的，任何人都可以检查。 这是以太坊帮助 NFT 创建人最大限度地赚取收入的方法之一。

所有这些任务都由矿工完成。 他们让整个网络上的其他人了解您的 NFT 并知道谁拥有它。 这意味着采矿需要有足够的难度，否则任何人都可以声称他们拥有您刚刚铸造的 NFT，并利用欺诈手段转移所有权。 有很多激励措施可以确保矿工诚实行事。

关于挖矿的更多信息

通过挖矿来保护您的 NFT {#securing-nfts}

挖矿的难度取决于以下事实：在链上创造新区块需要消耗大量算力。 重要的是，链上会不断创建新区块，而不仅仅在需要时创建。 每隔 12 秒左右创建一个区块。

这对于以太坊的防篡改机制来说至关重要，这是使 NFT 成为现实的特性之一。 区块越多越安全。 如果您的 NFT 是在第 600 个区块中创建的，而黑客试图通过修改其数据来窃取您的 NFT，那么所有后续区块的数字指纹将发生变化。 这意味着任何运行以太坊软件的人都能够立即发现并阻止其发生。

然而，这也意味着我们需要持续不断地消耗算力。 这也意味着，一个包含 0 个 NFT 交易的区块仍会产生大致相同的碳足迹，因为创建它仍会消耗算力。 其他非 NFT 交易将填充区块。

区块链现在属于能源密集度型 {#blockchains-intensive}

是的，通过挖矿创造区块会产生碳足迹，这对比特币等链来说也是一个问题，但直接责任并不在 NFT。

大量挖矿使用可再生能源或边远地区未开发的能源。 还有一种说法，NFT 和加密货币所颠覆的行业本身也有巨大的碳足迹。 但我们不能仅仅因为现有产业的糟糕表现就放弃自身努力。

我们也是如此。 以太坊正在不断发展，使自身（及藉此的 NFT）更加节能。 而这一直是我们计划的一部分。

我们在这里不是为采矿的环境足迹辩护，相反，我们想解释事情如何向好的方向发展。

更绿色的未来... {#a-greener-future}

自以太坊出现以来，挖矿的能源消耗一直是开发者和研究人员的重点关注领域。 而我们的愿景一直是尽快取代它。 关于以太坊的更多愿景

这一愿景正在变为现实。

更加绿色节能的以太坊 {#greener-ethereum}

以太坊目前正在经历一系列升级，它将用权益质押取代采矿。 这将移除作为安全机制的算力，并将以太坊的碳足迹减少约 99.95%[^1]。 在这个世界上，质押者投入资金而不是算力来确保网络安全。

以太坊的能源消耗将变为一台家用电脑的运行成本乘以网络中的节点数量。 如果网络中有 10,000 个节点，家用计算机的运行成本约为每年 525 千瓦时。 整个网络每年需要 5,250,000 千瓦时[^1]

我们可以用这个来对比未来的以太坊 和 Visa 这样的全球服务。 100,000 次 Visa 交易使用的能量为 149 千瓦时[^2]。 在 PoS 中，相同数量的交易会消耗 17.4 千瓦时的能量，约占总能量的 11%[^3]。 这还没有考虑到共识层与分片链同时进行的许多优化工作，比如 rollups。 100,000 笔交易的能耗可能只有 0.16666667 千瓦时。

重要的是提高了能源效率，同时保留了以太坊的去中心化和安全性。 其他许多区块链可能已经使用了某种形式的权益质押，但是它们是由少数几个利益相关者来保护的，以太坊拥有的则是数以千计的支持者。 越是去中心化，系统就越安全。

关于能源估算的更多信息

为了更好理解，我们给出了以太坊 PoS 与 Visa 能源消耗的基本比较， 实际上，根据交易次数进行比较并不太准确。 以太坊的能量输出基于时间。 如果以太坊在一分钟内进行多次交易，能源输出将保持不变。

同样重要的是要记住，以太坊做的不仅仅是金融交易，它是一个应用程序平台，因此我们不妨拿它和包括 Visa、AWS 等在内的更多企业/行业进行更为公平的比较。

时间轴 {#timelines}

这个过程已经开始了。 信标链，在 2020 年 12 月上线。 这允许质押者能够参与网络，为权益质押奠定了基础。 与能效有关的下一步措施便是把当前的链合并，从当前由矿工保护的链，合并到不需要挖矿的信标链中。 目前还不能明确具体的时间节点，但估计这将在 2022 年的某个时候发生。 这个过程被称为合并（以前称为对接）。 关于合并的更多信息。

以太坊升级的更多信息

开发 NFT {#build-with-nfts}

大多数 NFT 使用同一套标准建立，那就是 ERC-721。 然而，还有其他的标准，您可能也想研究一下。 ERC-1155 标准允许半同质化代币，这在游戏领域特别有用。 最近还提出了 EIP-2309，这使得制作 NFT 更高效。 这个标准让您在一次交易中尽情铸币！

延伸阅读 {#further-reading}

- 加密艺术数据– Richard Chen，自动更新
- OpenSea：NFT 圣经- *Devin Fizner，2020 年 1 月 10 日*
- NFT 初学者指南 - *Linda Xie，2020 年 1 月*
- 您需要了解的元宇宙 - *Foundation 团队，foundation.app*
- CryptoArtists 没有危害地球
- 举国之力，仅此而已- *Carl Beekhuizen，2021 年 5 月 18 日*

脚注和来源 {#footnotes-and-sources}

这解释了我们如何得出上述能源估算。 这些估算值适用于整个网络，而不仅仅是保留给创建、购买或出售 NFT 的流程。

1. 降低 99.95% 的采矿能源 {#fn-1}

根据如下数据源可以计算出从通过挖矿保护的系统转变到质押保护的系统，能耗降低了 99.98%：

- 以太坊挖矿每年消耗 44.49 太瓦时电能 - Digiconomist
- 普通的台式电脑（即运行权益证明所需的一切），每小时消耗 0.06 千瓦时的能源 - 硅谷电力图表（有些估算值略高，为 0.15 千瓦时）。

在撰写本报告时，共有来自 16 405 个独特的地址 140,592 个验证者。 其中，87,897 名验证者被认为在家中进行权益质押。

假设普通人使用 100 瓦的台式机在家里权益质押，平均运行 5.4 个验证者客户端。

在家里运行的 87,897 个验证者产生了 16,300 个用户，消耗了约 1.64 兆瓦的能源。

其余的验证者是由交易所和权益质押服务商等运行。 可以假设他们每 5.5 个验证者消耗 100 瓦。 为保守意见，这个数据严重高估。

因此，总的来说，以太坊在权益证明上的消耗量约为 2.62 兆瓦，相当于一个美国小镇的能耗量。

Digiconomist 预估以太坊矿工当前每年消耗 44.94 太瓦时的总能耗，与此相比，以太坊能耗至少降低了 99.95%。

2. Visa 能耗 {#fn-2}

10 万笔 Visa 交易的成本是 149 千瓦时 - 截至 2020 年，比特币网络每笔交易的平均能耗与 VISA 网络对比，Statista。

到 2020 年 9 月底，他们处理了 140,839,000,000 笔交易 - Visa 2020 年 Q4 财务报告

3. PoS 网络上 100,000 次交易的能源使用情况 {#fn-3}

据估计，可扩展性的升级将使网络能够每秒处理 25,000 至 100,000 笔交易，其中 100,000 是目前的理论最大值。

Vitalik Buterin 谈论在分片上每秒可行的交易量

最起码，分片允许的交易量是现在的 64 倍，当前交易量大约为 15 笔交易。 这是正在引入分片链（额外数据和容量）的数量。 关于分片链的更多信息

这意味着我们可以估算处理 100,000 笔交易需要的时间，以便我们能够将其与上述 Visa 例子对比。

- `15 * 64 = 960` 笔交易/秒。
- `100,000 / 960 = 104.2` 秒，完成 100,000 笔交易。

在 104.2 秒内，以太坊网络将消耗以下能量：

`1.44kwh 每日用量 * 10,000个网络节点 = 14,400 千瓦时/天`

每天有 86,400 秒，所以 `14,400/86,400=0.1666667 千瓦时`/秒。

如果我们将其乘以处理 100,000 次交易所需的时间：`0.1666666667 * 104.2 = 17.366666701 千瓦时`

这只相当于在 Visa 上进行相同次数交易所消耗能量的 **11.6554809866 %**

别忘了，这是基于以太坊每秒能够处理的最小交易量而得出的数据。 如果以太坊达到每秒 100,000 笔交易，这 100,000 笔交易耗费 0.16666667 千瓦时。

换言之，如果 Visa 处理 140,839,000,000 笔交易，能耗为每 100,000 笔交易 149 千瓦时，这将导致 209,850,110 千瓦时的年消耗量。

以太坊的一年能耗量为 5,256,000 千瓦时。 它所能处理的潜在交易量将达到 788,940,000,000 - 3,153,600,000,000 笔。

如果您认为这些数据不正确或可以更加精确，请提出问题或 PR。 这些是 ethereum.org 团队使用可公开访问的信息计划的以太坊路线图做出的估计。 这并不代表以太坊基金会的正式承诺。

title: 以太坊安全和预防欺诈措施
description: 在以太坊上保持安全
lang: zh

sidebar: true

以太坊安全和预防欺诈措施 {#introduction}

随着人们对加密货币的兴趣越来越大，学习使用加密货币时的最佳做法是至关重要的。 加密货币是非常有趣和令人兴奋的，但也存在严重的风险。 如果您前期投入少量的工作，你就可以降低这些风险。

网络安全 101 {#web-security}

使用强密码 {#use-strong-passwords}

超过 80% 的账户被黑客攻击是由于密码薄弱或被盗造成的。 一个很长的字符、数字和符号组合是保证您的账户安全的最好方法。

个人常犯的一个错误是使用两到三个常见的、具有相关性的词典词汇的组合。 像这样的密码是不安全的，因为它们很容易被称为字典攻击的简单黑客技术所攻击。

```
弱密码示例：CuteFluffyKittens!

强密码示例：ymv\*azu.EAC8eyp8umf
```

另一个常见错误是使用容易通过社交工程猜出或发现的密码。 在密码中加入母亲的婚前姓氏、孩子或宠物的名字或出生日期是不安全的，会增加密码被黑客攻击的风险。

强密码设置方法： {#good-password-practices}

- 在密码生成器或你所填写的表格允许的范围内，将密码设得越长越好
- 混合使用大写字母、小写字母、数字和符号
- 不要在密码中使用个人详细资料，如姓氏
- 避免使用字典中的常见词汇

关于创建强密码的更多内容

使用独立的密码 {#use-unique-passwords}

如果密码数据被泄露，强密码也无法提供足够的保护。 网站 Have I Been Pwned 允许您检查您的账户是否参与了存储在其数据库中的任何数据入侵。 如果有，**您应该立刻更改泄露的密码。** 为每个帐户设置独立的密码可以降低当你的一个密码被破解时黑客进入你所有账户的风险。

使用密码管理器 {#use-password-manager}

使用密码管理器可以创建唯一的强密码并记住它们！ 我们**强烈**推荐使用它，而且它们大部分是免费的。

记住为每个账户设置的唯一强密码并不现实。 密码管理器为您的所有密码提供了一个安全、加密的存储空间，您可以通过一个强主密码进行访问。 他们还在注册新服务时生成建议使用的强密码，这样你就不必自行创建密码了。 许多密码管理器也会告诉你是否涉及数据泄露，让你在任何恶意攻击之前更改密码。

使用密码管理器的示例

尝试一个密码管理器： {#try-password-manager}

- Bitwarden
- KeePass
- LastPass
- 1Password

2FA（双重身份验证） {#two-factor-authentication}

为了证明您是本人，有不同的唯一证明可以用来验证。 这些被称为**因素**的三个主要因素是：

- 您知道的信息（例如密码或安全问题）
- 生理特征（如指纹或虹膜/面部识别）
- 你私有的（安全密钥或你手机上的认证程序）

使用**双重身份验证 (2FA)** 为您的在线账户提供额外的*安全因素*，因此仅知道您的密码（您知道的信息）不足以访问一个账户。 最常见的是，双重验证是一个随机的 6 位数代码，被称为**基于时间的一次性密码（TOTP）**，您可以通过身份验证程序应用程序（如 Google authenticator 或 Authy）生成。 这是一种“你私有的”，因为生成随机时间密码的种子文件存储在你的设备上。

注意：使用基于短信的 2FA 很容易遭受 SIM 卡盗用 并不安全。 为了获得最佳安全性，请使用诸如{" "} Google 身份验证器 或 Authy 之类的服务。

安全密钥 {#security-keys}

对于那些想要进阶使用 2FA 的人，请考虑使用安全密钥。 安全密钥是物理硬件身份验证设备，其工作方式与身份验证程序应用程序相同。 使用安全密钥是最安全的 2FA 使用方式。 这些密钥中有许多采用了 FIDO 通用第二因素 (U2F) 标准。 了解更多关于 FIDO U2F 的信息。

查看更多 2FA 相关信息：

卸载浏览器扩展程序 {#uninstall-browser-extensions}

Chrome 扩展程序或 Firefox 插件等浏览器扩展程序可以增强浏览器的功能并提高用户体验，但它们也存在风险。 大多数浏览器扩展程序默认请求获得“读取和更改网站数据”的权限，几乎允许它们对您的数据做任何事情。 Chrome 扩展程序总是自动更新，因此旧版本安全的扩展程序可能会在更新后被加入恶意代码。 大多数浏览器扩展程序都不会试图窃取您的数据，但您应该知道它们可以。

通过以下方式保持安全： {#browser-extension-safety}

- 只安装来自受信任来源的浏览器扩展程序
- 删除不使用的浏览器扩展程序
- 在本地安装 Chrome 扩展程序以停止自动更新（高级）

更多关于浏览器扩展程序的风险

加密货币安全 101 {#crypto-security}

提升您的知识水平 {#level-up-your-knowledge}

人们在网络中被骗的主要原因之一通常是缺乏了解。 例如，如果您不了解以太坊网络是去中心化的，不为任何人所拥有，您就很容易上当受骗。有人假装成客户服务人员，承诺用您的私钥找回您丢失的 ETH。 了解以太坊如何运作是一项值得的投资。

以太坊简介

什么是以太币 (ETH)?

钱包安全 {#wallet-security}

不要把您的私钥给别人 {#protect-private-keys}

无论任何原因，永远不要与他人分享您的私钥！

您钱包的私钥就像您的以太坊钱包的密码。 这是阻止知道您的钱包地址的人榨干您账户中所有资产的唯一方法。

什么是以太坊钱包?

不要将您的助记词/私钥截图 {#screenshot-private-keys}

如果将您的助记词或私钥截图，您有可能将它们同步到云端，并有可能使它们被黑客获取。 从云端获取私钥是黑客最常见的攻击方式。

使用硬件钱包 {#use-hardware-wallet}

硬件钱包为私钥提供离线存储。 它们被认为是存储私钥最安全的钱包选项。

即使黑客控制了你的电脑，将私钥保持在离线状态，可以大大降低被黑客攻击的风险。

尝试以下硬件钱包: {#try-hardware-wallet}

- Ledger
- Trezor

发送交易前仔细检查 {#double-check-transactions}

不小心将加密货币发送到错误的钱包地址是一个常见的错误。 **在以太坊上发送交易是不可逆的。**除非您认识地址所有者并能说服他们把您的资金退回来，否则您将没有办法找回您的资金。

在发送交易之前，请务必确保您发送的地址与接收人的地址完全匹配。 在与智能合约进行交互时，建议在签名前阅读交易消息。

设置智能合约的支出限额 {#spend-limits}

与智能合约进行交互时，不得允许无限制支出限额。 无支出限额可以让智能合约掏空您的钱包。 相反，只将支出限额设置为交易所需金额。

许多以太坊钱包提供限额保护，以防止账户被掏空。

探索具有限额保护的钱包

常见的骗术 {#common-scams}

骗子总是想方设法从您身上骗走您的资金。 完全阻止骗子是不可能的，但我们可以通过了解所使用的大多数技术来降低他们的成功率。 虽然骗术层出不穷，但是他们的本质相同。 不出意外，请记住：

- 始终保持怀疑态度
- 没有人会给您免费或打折 ETH！
- 没有人需要获取您的私钥或个人信息

赠品诈骗 {#giveaway}

加密货币中最常见的骗局之一是赠品诈骗。 赠品诈骗有多种形式，但一般前提是，如果您将 ETH 发送到所提供的钱包地址，您将会收到双倍的 ETH。 *因此，它也称为“买一送一”诈骗。*

这些骗局通常规定在有限时间内索取赠品，以鼓励人们做出错误的决定，并制造一种虚假的紧迫感。

社交媒体黑客攻击 {#social-media-hacks}

最出名的一次发生在 2020 年 7 月，当时很多知名人士和组织的 Twitter 账户被黑。 黑客使用被盗的账户发布了一个比特币赠送活动。 尽管这些欺骗性的推文很快就被发现并删除，但黑客们还是成功骗走了 11 个比特币（截止 2021 年 9 月，这些比特币价值 50 万美元）。

Twitter 上的一个诈骗案例

名人的赠品 {#celebrity-giveaway}

名人的赠品是赠品诈骗的另一种常见形式。 骗子会录制名人的视频采访或会议演讲，并在 YouTube 上进行直播——看起来好像名人正在接受直播视频采访，为加密货币赠送活动背书。

除最常使用的 Vitalik Buterin 以外，此类骗局也使用了网络上许多其他知名人士（例如 Elon Musk 或 Charles Hoskinson）。 在直播中加入一位知名人士会让骗局看起来有一种合法性（这看起来有点牵强，但 Vitalik 参与其中，所以肯定没问题！）。

赠品活动通常是骗局。 如果您把钱转到这些账户，您将永远失去它们。

YouTube 上的一个诈骗案例

技术支持诈骗 {#support-scams}

加密货币是一种相对年轻且常被误解的技术。 利用这一点的一种常见的骗局是技术支持诈骗 (Support scams)，骗子会冒充受欢迎的加密货币钱包、交易所或区块链的支持人员。

很多关于以太坊的讨论都发生在 Discord 上。 骗子通常会通过在公开的 Discord 频道中搜索寻求支持的问题，然后冒充支持人员向询问者发送私人信息以找到他们的目标。 通过建立信任，骗子试图诱使您透露您的私钥或将您的资金发送到他们的钱包。

Discord 上的一个技术支持诈骗案例

一般来说，员工永远不会通过私人的非官方渠道与您交流。 在处理支持问题时，要牢记一些简单的事情：

- 永远不要分享您的私钥、助记词或密码
- 绝不允许任何人远程访问您的电脑
- 切勿通过官方指定以外的渠道沟通

请注意：尽管支持类型的骗局通常发生在 Discord 上，但它们也在任何讨论加密货币聊天应用程序（包括电子邮件）上普遍存在。

网络钓鱼诈骗 {#phishing-scams}

网络钓鱼诈骗是另一种越来越常见的诈骗手段，他们试图窃取您钱包中的资金。

一些网络钓鱼电子邮件要求用户点击链接，将其重新定向到仿冒网站，并要求他们输入助记词、重置密码或发送电子邮件。 还有一些可能会让您在不知情的情况下安装恶意软件来感染您的电脑，并让骗子访问您的电脑文件。

如果你收到一封来历不明的邮件，请记住:

- 永远不要打开您不认识的电子邮件地址中的链接或附件
- 切勿将你的个人信息或密码泄露给任何人
- 删除来历不明的电子邮件

更多关于避免网络钓鱼诈骗的内容

加密货币交易经纪人诈骗 {#broker-scams}

他们自称是专业的加密货币经纪人，他们会收取您的钱，代您投资。 这一提议通常伴随着不切实际的回报承诺。 在骗子收到您的钱后，他们可能会诱导您，要求您拿出更多的资金，这样您就不会错过进一步的投资收益，或者他们也可能就完全消失了。

这些骗子利用 YouTube 上的虚假账户，找到他们的目标，然后开始看似自然地开始有关经纪人的交谈。 这些对话通常会收到很多点赞，以增加真实性，但这些“赞”都是来自机器人账户。

不要相信互联网上的陌生人，让他们代您投资。 您将失去您的加密货币。

YouTube 上的交易经纪人诈骗案例

加密货币矿池骗局 {#mining-pool-scams}

在矿池骗局中会有人主动联系您，并声称您可以通过加入以太坊矿池获得大量回报。 骗子会提出要求，并一直与您保持联系。 基本上，骗子会试图让您相信，用当您加入一个以太坊矿池时，您的加密货币将被用来创建 ETH，您将以 ETH 的形式获得红利。 最终会发生的是，您会发现您的加密货币只能获得微薄的回报。 这只是为了引诱您投入更多的资金。 最终，您所有的钱都会被送到一个未知的地址，而骗子要么会消失，要么在某些情况下会继续保持联系，就像最近发生的一个案例一样。

总之，要提防那些在社交媒体上与您联系并要求您加入矿池的人。 一旦您失去您的加密货币，它就永远回不来了。

有些事情要记住：

- 警惕任何与您联系，告诉您如何使用加密货币赚钱的人
- 做好关于赌注、流动性池或其他加密货币投资方式的调研
- 这种计划即使有，也很少是合法的。 如果是的话，它们会成为主流，您会听说过它们。

有人在矿池骗局中损失 20 万美元

Eth2 代币骗局 {#eth2-token-scam}

在 2022 年合并来临之前，骗子们尝试使用“Eth2”这个新概念来欺骗用户，将他们的以太币兑换为“ETH2”代币。 实际上合并并没有产生任何“ETH2”或者任何其它新代币。 当前您所拥有的以太币在合并之后仍然是一样的以太币，而不用进行任何的兑换。

骗子们可能会以“技术支持”的面貌出现，告诉您需要把手中的 ETH 存到指定地址，这样你就可以得到“ETH2”。 实际上，并没有官方的以太坊技术支持，也没有新代币。 永远不要把您钱包的助记词和陌生人分享。

空投骗局 {#airdrop-scams}

空投骗局通常是先构建一个诈骗项目，并向您的钱包空投一种资产（NFT，代币），然后把诈骗的网站发给您，让您去获取这些空投的资产。 当您试图获取资产时，网站会要求您使用自己的以太坊钱包登录，并“批准”一笔交易。 实际上，这个交易会将您账户的公私密钥都发给骗子。 这种骗局的另一种方式是让您确认一笔可以把资金转移到骗子账户的交易。

关于空投骗局的更多信息

延伸阅读 {#further-reading}

网络安全 {#reading-web-security}

- This is why you shouldn't use texts for two-factor authentication - *The Verge*
- Up to 3 million devices infected by malware-laced Chrome and Edge add-ons - *Dan Goodin*
- How to Create a Strong Password — That You Won't Forget - *AVG*
- What is a security key? - *Coinbase*

加密货币安全 {#reading-crypto-security}

- Protecting Yourself and Your Funds - *MyCrypto*
- 4 Ways to Stay Safe in Crypto - *CoinDesk*
- Security Guide For Dummies And Smart People Too - *MyCrypto*
- Crypto Security: Passwords and Authentication - *Andreas M. Antonopoulos*

防诈骗指南 {#reading-scam-education}

- Staying Safe: Common Scams - *MyCrypto*
- Avoiding Scams *Bitcoin.org*

---

title: JavaScript 应用编程接口库
description: 以太坊 JavaScript 应用程序接口的介绍，使您能够从您的应用程序中与区块链进行交互。
lang: zh

sidebar: true

为了使软件应用程序能够与以太坊区块链进行交互（例如：读取区块链数据或发送交易信息到网络），软件必须连接到以太坊节点。

为此目的，每个以太坊客户端都执行 JSON-RPC 规范，所以应用程序可以依赖统一的端点集。

如果您想要用 JavaScript 连接到一个以太坊节点， 可以使用原生 JavaScript，不过生态系统中存在一些方便的库，使得这个事情变得更加容易。 通过这些库，开发者可以写下直观易懂甚至单行的代码就能初始化与以太坊的互动（背后使用 JSON RPC 请求）。

前置要求 {#prerequisites}

除了了解 JavaScript 外，了解以太坊堆栈和以太坊 客户端也许是有帮助的。

为什么要使用库？ {#why-use-a-library}

这些库降低了大多数与一个以太坊节点交互的复杂度。 它们还提供实用的函数（例如：将 ETH 转化为 Gwei），而作为开发者，您可以花费更少的时间来处理以太坊客户端的复杂问题，从而将更多的时间集中于处理您的应用程序的独特功能。

库的可用功能 {#library-features}

连接到以太坊节点 {#connect-to-ethereum-nodes}

使用提供器，这些库允许您连接到以太坊并读取它的数据，不管是通过 JSON-RPC、INFURA、Etherscan、Alchemy 还是 Metamask。

Ethers 示例

```
// A Web3Provider wraps a standard Web3 provider, which is
// what MetaMask injects as window.ethereum into each page
const provider = new ethers.providers.Web3Provider(window.ethereum)

// The MetaMask plugin also allows signing transactions to
// send ether and pay to change state within the blockchain.
// 为此，我们需要帐户签名者...
const signer = provider.getSigner()
```

Web3js 示例

```
var web3 = new Web3("http://localhost:8545")
// 或
var web3 = new Web3(new Web3.providers.HttpProvider("http://localhost:8545"))

// 更改提供者
web3.setProvider("ws://localhost:8546")
// 或
web3.setProvider(new Web3.providers.WebsocketProvider("ws://localhost:8546"))

// 在 node.js 中使用 IPC 提供者
var net = require("net")
var web3 = new Web3("/Users/myuser/Library/Ethereum/geth). pc", net// mac os
path
// 或
var web3 = new Web3(
  new Web3.providers. pcProvider("/Users/myuser/Library/Ethereum/geth.ipc", net"
) // mac os path
// 在 windows 操作系统上的路径是 "\\\\pipe\\geth.ipc"
// 在 linux 上的路径是 "/users/myuser/.efer/geth.ipc"
```

一旦设置，您将能够查询区块链的以下内容：

- 区块高度
- 燃料评估
- 智能合约事件
- 网络 ID
- 以及更多...

钱包功能 {#wallet-functionality}

这些库给予开发者创建钱包的功能，用于管理密匙和对交易进行签署。

这里提供了一个以太币的例子

```
// 从助记符创建一个钱包实例...
mnemonic =
```

```
  "announce room limb pattern dry unit scale effort smooth jazz weasel alcohol"
walletMnemonic = Wallet.fromMnemonic(mnemonic)

// ...或者从一个私有密匙中创建
walletPrivateKey = new Wallet(walletMnemonic.privateKey)

walletMnemonic.address === walletPrivateKey.address
// true

// 每个签署 API 都是一个异步操作地址
walletMnemonic.getAddress()
// { Promise: '0x71CB05EE1b1F506fF321Da3dac38f25c0c9ce6E1' }

// 每个钱包地址同时也可以是同步的
walletMnemonic.address
// '0x71CB05EE1b1F506fF321Da3dac38f25c0c9ce6E1'

// 内部加密组件
walletMnemonic.privateKey
// '0x1da6847600b0ee25e9ad9a52abbd786dd2502fa4005dd5af9310b7cc7a3b25db'
walletMnemonic.publicKey
//
'0x04b9e72dfd423bcf95b3801ac93f4392be5ff22143f9980eb78b3a860c4843bfd04829ae61cdb
a4b3b1978ac5fc64f5cc2f4350e35a108a9c9a92a81200a60cd64'

// 钱包的助记符（mnemonic）
walletMnemonic.mnemonic
// {
//   locale: 'en',
//   path: 'm/44\'/60\'/0\'/0/0',
//   phrase: 'announce room limb pattern dry unit scale effort smooth jazz weasel
alcohol'
// }

// 注意：使用私有密匙创建的钱包没有
//       从助记符（被原生屏蔽）
walletPrivateKey.mnemonic
// null

// 签署一个信息
walletMnemonic.signMessage("Hello World")
// { Promise:
'0x14280e5885a19f60e536de50097e96e3738c7acae4e9e62d67272d794b8127d31c03d9cd59781
d4ee31fb4e1b893bd9b020ec67dfa65cfb51e2bdadbb1de26d91c' }

tx = {
  to: "0x8ba1f109551bD432803012645Ac136ddd64DBA72",
  value: utils.parseEther("1.0"),
}

// 签署一笔交易
walletMnemonic.signTransaction(tx)
```

```
// { Promise:
'0xf865808080948ba1f109551bd432803012645ac136ddd64dba72880de0b6b3a7640000801ca09
18e294306d177ab7bd664f5e141436563854ebe0a3e523b9690b4922bbb52b8a01181612cec9c431
c4257a79b8c9f0c980a2c49bb5a0e6ac52949163eeb565dfc' }

// 这个连接方法会返回一个新的连接到提供者的钱包实例
wallet = walletMnemonic.connect(provider)

// 查询以太坊网络
wallet.getBalance()
// { Promise: { BigNumber: "42" } }
wallet.getTransactionCount()
// { Promise: 0 }

// 发送 Ether
wallet.sendTransaction(tx)
```

[阅读完整文档]

一旦设置，您将能够：

- 创建帐户
- 发送交易
- 签署交易
- 以及更多...

与智能合约交互的方法 {#interact-with-smart-contract-functions}

JavaScript 客户端库允许您的应用程序通过读取已编译合约的应用程序二进制接口 (ABI) 来调用智能合约函数。

应用程序二进制接口本质上是基于 JSON 格式解释了合约的函数，并且允许您像普通 JavaScript 对象一样使用它。

以下是基于 Solidity 开发的智能合约：

```
contract Test {
    uint a;
    address d = 0x12345678901234567890123456789012;

    function Test(uint testInt)  { a = testInt;}

    event Event(uint indexed b, bytes32 c);

    event Event2(uint indexed b, bytes32 c);

    function foo(uint b, bytes32 c) returns(address) {
        Event(b, c);
        return d;
    }
}
```

可能会产生类似下面的 JSON 数据：

```
[{
    "type":"constructor",
    "payable":false,
    "stateMutability":"nonpayable"
    "inputs":[{"name":"testInt","type":"uint256"}],
  },{
    "type":"function",
    "name":"foo",
    "constant":false,
    "payable":false,
    "stateMutability":"nonpayable",
    "inputs":[{"name":"b","type":"uint256"}, {"name":"c","type":"bytes32"}],
    "outputs":[{"name":"","type":"address"}]
  },{
    "type":"event",
    "name":"Event",
    "inputs":[{"indexed":true,"name":"b","type":"uint256"},
{"indexed":false,"name":"c","type":"bytes32"}],
    "anonymous":false
  },{
    "type":"event",
    "name":"Event2",
    "inputs":[{"indexed":true,"name":"b","type":"uint256"},
{"indexed":false,"name":"c","type":"bytes32"}],
    "anonymous":false
}]
```

这意味着您可以：

- 发送一笔交易到指定的智能合约上，并执行智能合约上的方法
- 调用方法去评估对气体的需求量。这个方法的执行是在以太坊虚拟机中执行的。
- 部署一个合约
- 以及更多...

实用功能 {#utility-functions}

这些实用功能类似快捷键操作，可以让开发者在构建以太坊时更加简单些。

我们默认以太坊的价值单位是 Wei（以太坊的最小价值单位）。 1 ETH = 1,000,000,000,000,000,000 WEI - 这意味着开发者们可以处理计量很大的数字。 使用 `web3.utils.toWei` 可以将 ether 转换为 Wei 。

以 Ether 为单位是这样的：

```
// 获取帐户中的资产（通过地址或者 ENS 名）
balance = await provider.getBalance("ethers.eth")
// { BigNumber: "2337132817842795605" }

// 通常来说开发者可能会需要为用户格式化一下输出
// 用户更喜欢以 ether（而非 wei）表示的价值
ethers.utils.formatEther(balance)
// '2.337132817842795605'
```

- Web3js 实用功能
- Ethers 实用功能

可用的库 {#available-libraries}

Web3.js - ***以太坊 JavaScript API。***

- 相关文档
- GitHub

Ethers.js - ***JavaScript 和 TypeScript 中完整的以太坊钱包实现和实用工具。***

- 相关文档
- GitHub

Graph - ***用于为以太坊和 IPFS 数据建立索引并使用 GraphQL 对其进行查询的协议。***

- 图表
- Graph Explorer
- 相关文档
- GitHub
- Discord

Web3.js - ***针对轻客户端优化的高级响应式 JS 库。***

- GitHub

Web3-wrapper - ***可替代 Web3.js 的 Typescript。***

- 相关文档
- GitHub

Alchemyweb3 - ***用于包裹 Web3.js 的库，带自动重试和增强应用程序接口。***

- 相关文档
- GitHub

Alchemy NFT API - ***用于获取 NFT 数据的 API，包括所有权、元数据属性和 更多。***

- 相关文档
- GitHub

延伸阅读 {#further-reading}

还有哪些社区资源对您有所帮助？ 编辑并添加本页面！

相关主题 {#related-topics}

- 节点和客户端
- 开发框架

相关教程 {#related-tutorials}

- 设置 Web3js 以在 JavaScript 中使用以太坊区块链 *– 在项目中设置 web3.js 的说明。*
- 在 JavaScript 中调用智能合约 *使用 DAI token，从而使用 JavaScript 调用合约函数。*
- 使用 web3 和 Alchemy 发送交易 *– 一步步带您了解如何从后端发送一笔交易。*

title: 信标链
description: 了解信标链 - 引入权益证明的以太坊升级。
lang: zh
template: upgrade
sidebar: true
image: ../../../../../assets/upgrades/core.png
summaryPoint1: 信标链对我们今天使用的以太坊没有造成任何变化。
summaryPoint2: 它将作为共识层协调网络。
summaryPoint3: 它为以太坊生态系统提供了权益证明。

summaryPoint4: 您可能知道，它在技术路线图上属于“阶段 0”。

信标链于 2020 年 12 月 1 日中午标准时间 (UTC) 启动。 要了解更多信息，请浏览此处。 如需参与验证该信标链，您可以质押您的以太币。

信标链有什么作用？ {#what-does-the-beacon-chain-do}

信标链将处理或协调分片和质押人的扩展网络。 但它会与今天的以太坊主网不同。 它无法处理帐户或智能合约。

信标链的角色将随着时间的推移而改变，但它是我们正在努力实现的安全、可持续和可扩展以太坊的基础组件。

信标链的特点 {#beacon-chain-features}

质押介绍 {#introducing-staking}

信标链将在以太坊中引入权益证明共识机制 (PoS)。 这是您帮助维护以太坊安全的一个新方式。 该机制可以看成是一种让以太坊生态更健康的公共产品，并且在此过程中能让您获得更多以太币。 在实际操作中，它将需要您质押以太币，以便激活验证者软件。 作为验证者，您将处理交易并在链中创建新的区块。

质押和成为验证者比挖矿（当前保护以太坊网络安全的方式）更容易。 从长远来看，人们希望这能帮助提高以太坊的安全性。 参与该网络的人越多，网络就会变得越分散，面临攻击时也越安全。

如果您有兴趣成为验证者并帮助保护信标链，请点击了解关于质押的更多信息。

引入该机制这一重要变化也有助于实现另一个升级：分片链。

设置分片链 {#setting-up-for-shard-chains}

主网与信标链合并后，下一个升级便是在权益证明网络中引入分片链。 这些“分片”将通过将网络扩展到 64 个区块链来增加网络的容量并提高交易速度。 由于分片链要求质押才能安全运行，因此信标链是引入分片链重要的第一步。

最终，信标链还将负责随机分配质押人来验证分片链。 这是让质押人难以串通并控制分片的关键机制。 该几率小于万亿分之一。

升级之间的关系 {#relationship-between-upgrades}

以太坊所有升级都是有点关联的。 所以我们来回顾一下信标链对其他升级的影响。

主网和信标链 {#mainnet-and-beacon-chain}

按照最初的设计，信标链与当前的以太坊主网独立运行。 但最终会相互连接。 该计划是将主网“合并”到由信标链控制和协调的权益证明系统中。

合并

分片和信标链 {#shards-and-beacon-chain}

分片链只有在已建立权益证明共识机制的情况下才能安全进入以太坊生态系统。 信标链将引入权益质押，为后续的分片链升级奠定基础。

分片链

与信标链交互 {#interact-with-beacon-chain}

title: 合并
description: 了解“合并”：当以太坊主网加入信标链协调的的权益证明系统时。
lang: zh
template: upgrade
sidebar: true
image: ../../../../../assets/upgrades/merge.png
summaryPoint1: 最终，当前以太坊主网将与信标链权益证明系统“合并”。
summaryPoint2: 这将标志着以太坊工作证明的淘汰以及向权益证明的完全过渡。
summaryPoint3: 计划在推出分片链之前进行这项工作。

summaryPoint4: 我们以前将此称为“对接”。

此次升级代表以太坊正式转向权益证明共识。 能源密集型挖矿工作需求将消失，网络保护机制将通过质押以太币实现。 这是实现以太坊愿景真正令人激动的一步，也就是“更高的可扩展性、安全性和可持续性”。

什么是合并？ {#what-is-the-docking}

请务必记住，信标链最初是与主网（我们今天所使用的链）分开上线的。 以太坊主网继续受工作量证明的保护，即使信标链使用权益证明并行运行也同样如此。 “合并”是指这两个系统最终融合在一起。

想象一下，以太坊是一艘尚未做好星际航行准备的宇宙飞船。 社区通过信标链搭建了一个新引擎和坚固外壳。 到时候，当前的飞船将与这个新系统对接，结合为一艘飞船，朝着遥远的未来和宇宙蓄势待发。

与主网合并 {#docking-mainnet}

准备好后，以太坊主网将与信标链“合并”，成为以太坊的分片链，使用权益证明而非工作量证明。

基于以太坊的历史与现状，主网将为权益证明系统带来执行智能合约的能力，向所有以太币持有人和使用者保证过渡顺利。

合并后 {#after-the-merge}

这将标志着以太坊工作量证明的结束，并开启一个更可持续、更环保的以太坊时代。 届时，以太坊将更接近实现其以太坊愿景中概述的全面性、安全性和可持续性。

值得注意的是，实施合并的目标是简单化，以加快从工作量证明到权益证明的过渡。 开发者正在将精力集中在这一过渡上，并尽量减少可能延迟实现这一目标的附加功能。

这意味着，在合并完成后，一些功能（如提取质押的以太币）将需要稍微等待一段时间方会上线。相关计划包括合并后用于处理这些功能的“清理”升级，预计在合并完成后很快执行。

升级间的关系 {#relationship-between-upgrades}

以太坊所有升级都是有点关联的。 因此，让我们概括一下合并如何与其它升级关联。

合并与信标链 {#docking-and-beacon-chain}

一旦发生合并，将分配质押人来验证以太坊主网。 届时将不再需要挖矿，因此矿工很可能会将他们的收入投入到在新的权益证明系统中进行质押。

信标链

合并与合并后清理 {#merge-and-post-merge-cleanup}

合并之后，一些功能（如撤回质押的以太币）尚不受支持。 计划在合并后不久对这些功能单独进行升级。

请随时了解以太坊基金会研究和开发博客发布的最新信息。 如果感兴趣，请详细了解 Vitalik 在 2021 年 4 月的 ETHGlobal 活动中发表的演讲：合并后会发生什么。

合并与分片链 {#docking-and-shard-chains}

最初，计划是在合并之前研究分片链，以解决可扩展性问题。 然而，随着第二层扩容解决方案的盛行，工作重点已经转移到通过合并将工作量证明转换为权益证明。

这需要社区进行持续评估，以确定是否可能需要多轮分片链来实现无限的可扩展性。

分片链

了解更多 {#read-more}

---

title: 分片链
description: 了解分片链 - 网络的分区可以让以太坊拥有更强的交易能力并使其更容易运行。
lang: zh
template: upgrade
sidebar: true
image: ../../../../../assets/upgrades/newrings.png
summaryPoint1: 分片是一个多阶段升级，以提高以太坊的可扩容性和容量。
summaryPoint2: 分片链为应用程序和卷叠提供更便宜的额外存储层来存储数据。
summaryPoint3: 它们使第二层解决方案能够提供更低的交易费，同时利用以太坊的安全性。

summaryPoint4: 此升级计划跟随主网与信标链的合并。

分片链应该会在 2023 年的某个时间上线，具体取决于合并后的工作进度。 这些分片将增强以太坊存储和访问数据的能力，但不会用于执行代码。

是什么区块分片？ {#what-is-sharding}

是水平分割数据库以分散负载的过程——这是计算机科学中的一个常见概念。 在以太坊的情况下，将通过创建新的链（称为“shards”）来减少网络拥塞并增加每秒的交易量。

除了可扩展性之外，其他原因也很重要。

分片特性 {#features-of-sharding}

每个人都可以运行节点 {#everyone-can-run-a-node}

如果您想保持去中心化，那么是一种很好的扩展方式，另一种选择是通过增加现有数据库的大小来扩展。 这将使网络验证者更难访问以太坊，因为他们需要强大而昂贵的计算机。 对于链，验证者只需要为他们正在验证的存储/运行数据，而不需要存储/运行整个网络（就像今天发生的那样）。 这加快了速度，并大大降低了硬件需求。

更多的网络参与 {#more-network-participation}

将最终让您在个人笔记本电脑或手机上运行以太坊。 因此，更多的人应该能够在中参与或运行客户端。 这将提高安全性，因为网络越分散，攻击面越小。

由于硬件需求较低，区块分片将使您更容易单独运行 客户端 ，而根本不依赖任何中间服务。 如果您可以，请考虑运行多个客户端。 这可以通过进一步减少故障点来帮助网络健康。 运行信标链客户端

首先，您需要在运行信标链客户端的同时运行主网客户端。 启动板将引导您完成硬件要求和过程。 或者使用后端应用程序接口。

分片链版本 1：数据可用性 {#data-availability}

当第一个分片链发布时，它们只会向网络提供额外的数据。 他们不会处理交易或智能合约。 但当它们与卷叠相结合时，仍然会对每秒交易数提供令人难以置信的改进。

卷叠是一种现存的“第二层”技术。 它们允许去中心化应用将交易捆绑或“卷起”到链外的单个交易中，生成加密证明，然后提交到链上。 这减少了交易所需的数据。 将这一点与分片提供的所有额外数据可用性结合起来，您每秒可以得到 100,000TPS。

鉴于第二层扩容解决方案近期的研发进展，促使我们考虑在推出分片链之前进行合并。 这些将是主网向权益证明过渡之后的重点。

关于卷叠的更多信息

分片链版本 2: 代码执行 {#code-execution}

我们一直计划为分片添加额外功能，使其更像现在的以太坊主网。 这将使他们能够存储和执行代码并处理交易，因为每个分片将包含一套独特的智能合约和账户余额。 跨分片通信将允许在分片之间进行交易。

尽管考虑到版本 1 分片会导致每秒增加交易量，但是这是否仍需要实现？ 社区里仍在辩论这个问题，这似乎有几种选择。

分片是否需要代码执行？ {#do-shards-need-code-execution}

Vitalik Buterin 在接受 Bankless 播客采访时提出了 3 个值得讨论的可能选择。

1. 不需要状态执行 {#state-execution-not-needed}

这将意味着，我们不会赋予分片处理智能合约的功能，而是让其成为数据仓库。

2. 对部分分片执行 {#some-execution-shards}

也许存在一个折衷方案，那就是：我们不需要所有的分片（现在计划有 64 个分片）都变得更智能。 我们可以仅将此功能添加到少数分片中，其余保持不变。 这可以加快交付速度。

3. 等到我们能够使用简明非交互式零知识论证 {#wait-for-zk-snarks}

最后，也许我们应该在简明非交互式零知识论证得到验证后再讨论这个问题。 这种技术可以帮助将真正的私人交易带入网络。 很可能他们需要更智能的分片，但这种分片仍在研究和开发中。

其他资源 {#other-sources}

以下文章具有一些类似的思路：

- 阶段 1 已完成：Eth2 成为数据可用性引擎 – *cdetrio, ethresear.ch*

这仍然是一个活跃的讨论点。 一旦我们了解到更多信息，就会更新这些页面。

升级间的关系 {#relationship-between-upgrades}

以太坊所有升级都是有点关联的。 所以让我们回顾一下分批链如何与其他升级相关联。

分片和信标链 {#shards-and-beacon-chain}

信标链包含所有保持分片安全和同步的逻辑。 信标链将协调网络中的质押人，将他们分配给他们需要处理的分片。 它还将通过接收和存储其他分片可访问的分片交易数据来促进分片之间的通信。 这将为分片提供以太坊状态的概况，以保持一切最新。

信标链

分片与合并 {#shards-and-docking}

添加额外分片时，以太坊主网已由信标链使用权益证明提供保护。 这使得一个多产的主网可以建立多个分片链，并由可增强可扩容性的第二层解决方案提供支持。

主网是否会成为唯一能够处理代码执行的“智能”分片还有待观察 – 但无论如何，都可以根据需要重新考虑扩展分片的决定。

合并

了解更多 {#read-more}

---

title: JSON-RPC 应用程序接口
description: 用于以太坊客户端的无状态、轻量级远程过程调用协议。
lang: zh

sidebar: true

为了使软件应用程序能够与以太坊区块链进行交互（读取区块链数据或发送交易信息到网络），软件必须连接到以太坊节点。

为此目的，每个以太坊客户端都执行 JSON-RPC 规范，所以应用可以依赖统一的方法集。

JSON-RPC 是无状态、轻量级远程程序调用协议。 规范主要界定了若干数据结构和围绕其处理的规则。 它与传输无关，因为这些概念可以在同一进程，通过接口、超文本传输协议或许多不同的消息传递环境中使用。 它使用 JSON (RFC 4627) 作为数据格式。

JSON-RPC 资源 {#json-rpc-resources}

- 以太坊 JSON-RPC 规范
- 以太坊 JSON-RPC 规范 Github 代码库

客户端实现 {#client-implementations}

每个客户端在执行 JSON-RPC 规范时可以使用不同的编程语言。 更多与特定编程语言相关的详细信息，请查阅客户端文档。 我们建议查看每个客户端文档以获取最新的应用程序接口支持信息。

便利性库 {#convenience-libraries}

虽然您可以选择通过 JSON 应用程序接口直接与以太坊客户端交互，但是对于去中心化应用程序开发者来说，常常有更容易的选项。 许多 JavaScript 和后端应用程序接口库已经存在，可以在 JSON-RPC 应用程序接口之上提供封装。 通过这些库，开发者可以方便地写下直观的一行函数来初始化（后端的）JSON RPC 请求并用于与以太坊进行交互。

相关主题 {#related-topics}

- 节点和客户端
- JavaScript API
- 后端应用程序接口

---

title: 区块
description: 以太坊区块链中区块的概述 – 其数据结构、存在的意义以及区块如何生成
lang: zh

sidebar: true

区块是指一批交易的组合，并且包含链中上一个区块的哈希。 这将区块连接在一起（成为一个链），因为哈希是从区块数据中加密得出的。 这可以防止欺诈，因为以前的任何区块中的任何改变都会使后续所有区块无效，而且所有哈希都会改变，所有运行区块链的人都会注意到。

前置要求 {#prerequisites}

区块是一个对初学者非常友好的主题。 为了帮助您更好地理解这个页面，我们建议您先阅读帐户、交易和我们的以太坊简介。

为什么要有区块？ {#why-blocks}

为了确保以太坊网络上的所有参与者保持同步状态并就交易的确切历史达成共识，我们将交易分为多个区块。 这意味着一次提交、商定和同步数十个（或数百个）交易。

区块中的交易导致状态变化的图表 *图表来自 Ethereum EVM illustrated*

通过间隔提交，即使交易请求每秒发生数十次，我们仍能给所有网络参与者足够的时间达成共识。 以太坊上的区块大约每十五秒提交一次。

区块如何工作 {#how-blocks-work}

为了保存交易历史，区块被严格排序（创建的每个新区块都包含一个其父块的引用），区块内的交易也严格排序。 除极少数情况外，在任何特定时间，网络上的所有参与者都同意区块的确切数目和历史， 并且正在努力将当前的活动交易请求分批到下一个区块。

一旦某个区块被网络上的某个矿工放在一起（挖矿），它就会传播到网络的其余部分； 所有节点都将此块添加到其区块链的末尾，并且继续挖掘。 目前，以太坊的“工作量证明”协议指定了确切的块组装（挖矿）过程和提交/共识过程。

视觉演示 {#a-visual-demo}

工作量证明协议（POW） {#proof-of-work-protocol}

工作量证明是指：

- 挖矿节点必须花费可变但大量的能源、时间和算力，才能为试图提交给网络的区块生成“合法性证明”。 这可以帮助保护网络免受垃圾邮件/拒绝服务攻击，因为证书的生成成本很高。
- 其他矿工如果听说新区块拥有有效的合法性证书，则必须接受新区块作为区块链上的下一个规范区块。
- 特定矿工生成此证书所需的确切时间是一个随机变量，彼此差异很大。 这确保了两个矿工不太可能同时对提议的下一个区块产生验证；当矿工产生并广播经认证的新区块时，他们几乎可以肯定该区块将被网络接受为区块链上规范的下一个区块，没有冲突（尽管在几乎同时产生两个认证块链的情况下，也有处理冲突的协议）。

关于挖矿的更多信息

区块包含什么？ {#block-anatomy}

- `timestamp` – 开采区块的时间。
- `blockNumber` – 区块链中区块的长度。
- `baseFeePerGas` - 要将交易纳入区块，每个 gas 所需的最低费用。
- `difficulty` – 开采所需的努力。
- `mixHash` – 该区块的唯一标识符。
- `parentHash` – 前一区块的唯一标识符（这是区块形成链的方式）。
- `transactions` –包含在区块中的交易。
- `stateRoot` –系统的整个状态：帐户余额、合约存储、合约代码和帐户随机数。
- `nonce` – 哈希，当与 mixHash 结合使用时，可以证明该块已经通过了工作量证明。

区块时间 {#block-time}

区块时间指的是开采一个新区块所需的时间。 在以太坊中，平均区块时间在 12 到 14 秒之间，并在每个区块后进行评估。 预期区块时间在协议层面被设定为一个常数，当矿工增加计算能力时，用于保护网络的安全。 平均区块时间与预期区块时间进行比较，如果平均区块时间较高，则区块头的难度会降低。 如果平均区块时间较小，则会增加区块头中的难度。

区块大小 {#block-size}

最后一个重要的一点是，区块本身的大小是有界限的。 每个区块的目标大小为 1500 万 gas，但区块的大小将根据网络需求而增加或减少。区块限制最多不超过 3000 万（目标区块大小的 2 倍）。 区块中所有交易所消耗的 gas 总量必须低于区块的 gas 限制。 这很重要，因为它可以确保区块不会任意扩大。 如果区块可以任意大，由于空间和速度方面的要求，性能较差的完整节点将逐渐无法跟上网络。

延伸阅读 {#further-reading}

还有哪些社区资源对您有所帮助？ 请编辑本页面并添加！

相关主题 {#related-topics}

- 挖矿
- 交易
- Gas

---

title: 共识机制
description: 解释分布式系统中的协商一致协议及其在以太坊中扮演的角色。
lang: zh
sidebar: true

incomplete: true

当涉及到像以太坊这样的区块链（本质上是分布式数据库）时，网络节点必须能够就系统的当前状态达成一致。 这是通过协商一致机制实现的。

虽然共识机制与建立 dapp 并无直接关系，但理解两者将有助于阐明与您和您的用户体验相关的概念，如 gas 价格和交易时间。

前置要求 {#prerequisites}

为了让您更好地读懂本页面的内容，建议您先阅读：以太坊（Ethereum）项目介绍

什么是共识？ {#what-is-consensus}

我们所说的共识，是指达成了普遍协议。 比如，一群人去看电影， 如果对电影的选择没有异议，那么共识就达成了。 在极端情况下，即没有对电影选择达成一致时，这群人就会分开。

对于区块链，达成共识意味着网络上至少有 51% 的节点同意网络的下一个全球状态。

什么是共识机制？ {#what-is-a-consensus-mechanism}

共识机制（也称为共识协议或共识算法）允许分布式系统（计算机网络）协同工作并保持安全。

几十年来，我们一直采用这些机制在数据库节点、应用程序服务器和其他企业基础设施之间建立共识。 近年来，有人发明了新的共识机制，使以太坊等加密经济系统能够就网络状态达成一致。

在加密经济体系中建立共识机制也有助于防止某些类型的经济攻击。 从理论上讲，攻击者可以通过控制 51% 的网络而破坏共识。 共识机制旨在使这种“51% 攻击”不可行。 人们设计、实现了不同的机制，以不同的方式解决这个问题。

共识机制类型 {#types-of-consensus-mechanisms}

工作量证明 {#proof-of-work}

与比特币一样，以太坊目前使用**工作量证明 (PoW)** 作为共识机制。

创建区块 {#pow-block-creation}

工作量证明是通过矿工来完成的，矿工们需要竞争以创建包含已处理交易的新区块。 优胜者将与网络中的其他节点分享该新区块，以获得新铸造 ETH。 谁的计算机能够最快解决数学难题，谁就能赢得这场竞赛。这个难题就是生成现有区块与以往区块之间的加密链接。 解决此问题就是“工作量证明”中的工作。

安全性 {#pow-security}

因为用户需要拥有超过网络中 51% 的算力才能够欺骗整条链，因此网络的安全得以保证。 这将需要巨大的设备与能源投入，所需的开支甚至可能超过收益。

关于工作量证明 (PoW) 的更多信息

权益证明 {#proof-of-stake}

以太坊已经计划升级使用**权益证明 (PoS)** 来作为共识协议。

区块创建 {#pos-block-creation}

权益证明由验证者完成，他们通过质押以太币参与到系统中。 系统将随机选择一位验证者来创建新区块，并将它们分享到网络从而获得奖励。 相较于需要用大量算力来完成计算，您只需要轻松地将您的以太币质押到网络。 这将有利于激励健康的网络行为。

安全性 {#pos-security}

您将需要 51% 的以太币来捣毁此区块链，因此权益证明系统相对来说比较安全。 同时您的权益将会因为您的恶意行为被扣除。

关于权益证明 (PoS) 的更多信息

直观指南 {#types-of-consensus-video}

观看有关以太坊所用的不同类型共识机制的更多信息。

Sybil 抗性和链选择 {#sybil-chain}

现在，从技术上讲，工作量证明和权益证明本身并不是共识协议，但为了简单起见，它们通常被称为共识协议。 它们实际上是女巫防御机制和区块作者选择器；它们是决定谁是最新区块作者的一种方式。 这种女巫防御机制加上一种链选择规则，构成了真正共识机制。

女巫防御可衡量一个协议如何抗衡女巫攻击。 女巫攻击是指一个用户或团体假装成许多用户。 抵御这种攻击对去中心化的区块链至关重要，并使矿工和验证者能够在投入资源的基础上获得平等回报。 通过让用户花费大量能源或提供大量抵押品，工作量证明和权益证明可以防止这种情况。 这些保护措施是对女巫攻击的经济威慑。

链选择规则用于决定哪个链才是“正确”的。 以太坊和比特币目前使用“最长链”规则。这意味着，只要是最长的区块链，都会被其他节点视为有效节点并接受协作。 对于工作量证明链，最长链由链上累积的工作量证明总累积难度决定。

工作量证明机制和最长链规则结合称为“Nakamoto 共识”。

信标链使用了名为Casper the Friendly Finality Gadget的共识机制，该机制则基于权益证明。

延伸阅读 {#further-reading}

- 什么是区块链共识算法？
- 什么是 Nakamoto 共识？ 完整的初学者指南
- Casper 的工作原理
- 关于工作量证明区块链的安全性和性能

还有哪些社区资源对您有所帮助？ 请编辑本页面并添加！

相关主题 {#related-topics}

- 工作量证明
- 挖矿
- 权益证明

---

title: 权益证明机制(PoS)
description: 权益证明共识算法以及其在以太坊中作用的解释。
lang: zh
sidebar: true

incomplete: true

以太坊正在将其共识机制从工作量证明(PoW)转变为权益证明(PoS)。 这一直在计划之中，因为这是社区策略中通过升级来实现以太坊扩展的关键部分。 然而正确实现 PoS 是一项巨大的技术挑战,，并不像使用 PoW 在整个网络中达成共识那样简单。

前置要求 {#prerequisites}

为了更好地理解本文，我们建议你首先阅读以下内容 共识机制。

什么是权益证明？ {#what-is-pos}

权益证明是一种区块链网络达成共识的共识机制。

这将要求用户抵押他们的以太币从而成为网络中合法的验证者。 验证者有着与矿工在 工作量证明（pow） 中相同的职责：将交易排序和创建新的区块，以便让所有的节点就网路状态达成一致。

权益证明相较于工作量证明系统有许多改进：

- 提高能效——您不需要大量能源去挖掘区块
- 门槛降低，硬件要求减少——您不需要优秀的硬件从而获得建立新区块的机会
- 更强的去中心化——权益证明可以在网络中提供更多的节点。
- 更有力地支持分片链，这是以太坊网络扩展的关键升级

权益证明、权益质押和验证者 {#pos-staking-validators}

权益证明是一种用于激励验证者接受更多质押的基本机制。 就以太币而言，用户需要质押 32ETH 来获得作为验证者的资格。 验证者被随机选择去创建区块，并且负责检查和确认那些不是由他们创造的区块。 一个用户的权益也被用于激励良好的验证者行为的一种方式。 例如，用户可能会因为离线（验证失败）而损失一部分权益， 或因故意勾结而损失他们的全部权益。

以太坊权益证明是如何运作的？ {#how-does-pos-work}

与工作量证明不同的是，验证者不需要使用大量的算力，因为它们是随机选择的，相互间没有竞争。 他们不需要开采区块，他们只需要在被选中的时候创建区块并且在没有被选中的时候验证他人提交的区块。 此验证被称为证明。 你可以认为证明是说“这个块在我看来没问题”。 验证者因提出新区块和证明他们已经看到的区块而获得奖励。

如果你为恶意区块提供证明，你就会失去你的股权。

信标链 {#the-beacon-chain}

以太坊用权益证明取代工作量证明时，分片链的复杂性会增加。 这是需要验证者来处理交易和创建新区块的独立区块链。 计划中将有 64 个分片链，并且它们都需要对网络的当下状态有一个共同的理解。 所以这需要额外的协调工作，这将由信标链来完成。

信标链从碎片接收状态信息，并可供其他碎片使用，以便网络能够一直保持同步。 信标链也会管理验证者，从注册其股份存储到发布奖励和惩罚。

下面是这个过程的运作方式。

验证是如何工作的？ {#how-does-validation-work}

当您在分片上提交交易时，验证者将负责将您的交易添加到分片区块中。 信标链通过算法选择验证器以提出新的块。

区块认证 {#attestation}

如果一个验证者没有被选中提出一个新的分块，它们将不得不证明另一个验证者的提议，并确认一切都正常。 它是在信标链中记录的证明，而不是交易本身。

至少需要 128 个被称为“委员会”的验证者来证明每个分片块。

委员会有一个提出和验证分片区块的时限， 这被称为“slot”。 每个插槽只能创建一个有效区块，一个“周期”有 32 个插槽。 每个周期过后，委员会都由不同的、随机的参与者解散与改革。 这有助于避免委员会中的不良参与者伤害到分片。

交叉链接 {#rewards-and-penalties}

一旦一个新的碎片区块提案有足够的证明，就会创建一个“交叉链接”；它确认了在信标链中包含方块和你的交易。

一旦出现交叉链接，推荐区块的验证者将获得奖励。

终局 {#finality}

在分布式网络中，当交易是一个不能改变的区块的一部分时，交易具有“终局性”。

要利用权益证明做到这一点，Cassper（一个终局协议）可以让验证者在某些检查点就一个区块的状态达成协议。 只要 2/3 的验证者同意，该区块就会被最终确定。 验证者如果试图稍后通过 51% 的攻击恢复，就会失去他们的全部权益。

正如 Vlad Zammfir 说的那样，这就像一位采矿者参与了 51% 的攻击，导致他们的采矿硬件立即被烧毁。

权益证明和安全性 {#pos-and-security}

权益证明中仍然存在 51% 攻击的威胁，但对于攻击者来说攻击成本越来越高。 要发起 51% 攻击，你需要掌控 51% 以上的以太币股权。 这不仅仅是一笔巨款，还很有可能导致以太币贬值。 破坏你的货币价值的大部分权益是非常容易的。 当然也有更强有力的激励措施来保持网络的安全和健康。

信标链上的权益消减、踢出和其余惩罚、协调来防治其他恶意行为。 验证者还将负责记录这些事件。

优缺点 {#pros-and-cons}

优点	缺点
权益质押让您更容易运行一个节点。 无需大量硬件或能源投资，如果您没有足够的 ETH 抵押，您也可以加入权益质押池。	与工作量证明相比，权益证明仍处于起步阶段，并且没有经过实际应用的测试
权益质押更加去中心化。 它允许更多人参与，并且更多的节点不意味着像挖矿一样增加百分比的回报。	
权益质押可以保证安全的防护。 分片链允许以太坊同时创建多个区块，增加交易输送量。 将以太坊网络置于工作量证明系统内，这会降低网络被攻击所需的算力。	

延伸阅读 {#further-reading}

- Proof of Stake FAQ *Vitalik Buterin*
- What is Proof of Stake *ConsenSys*
- What Proof of Stake Is And Why It Matters *Vitalik Buterin*
- The Beacon Chain Ethereum 2.0 explainer you need to read first *Ethos.dev*
- Why Proof of Stake (Nov 2020) *Vitalik Buterin*
- Proof of Stake: How I Learned to Love Weak Subjectivity *Vitalik Buterin*
- A Proof of Stake Design Philosophy *Vitalik Buterin*

相关主题 {#related-topics}

- 工作量证明

---

title: 工作量证明 (PoW)
description: 工作量证明共识协议的解释及其在以太坊中的作用。
lang: zh
sidebar: true

incomplete: true

类似于比特币，以太坊目前使用的共识协议被称为**工作量证明 (PoW)**。 这种机制允许以太坊网络的节点就以太坊区块链上记录的所有信息的状态达成共识，并防止某些产生经济影响的攻击。

在接下来一年，工作量证明将被逐步淘汰，这有利于**权益证明 (PoS)** 的发展。 向权益证明 (PoS) 的过渡也将从以太坊逐步淘汰挖矿机制。 关于合并的更多信息。

前置要求 {#prerequisites}

为了更好地了解此页面，推荐先阅读交易、区块和共识机制。

什么是工作量证明 (POW) {#what-is-pow}

工作量证明是一种允许去中心化的以太坊网络达成共识或者一致认可帐户余额和交易顺序的机制。 这个机制防止用户“双花”他们的币，同时确保极难攻击或操作以太坊的链。

工作量证明和挖矿 {#pow-and-mining}

工作量证明是一个为矿工设置挖矿难度和规则的基础性算法。 挖矿就是“工作”本身。 挖矿是向区块链中添加有效区块。 这个工作很重要，因为区块链的长度帮助网络识别出有效的以太坊链并且理解当前以太坊的状态。 完成的“工作”越多链越长，区块链的序号就越大，网络就能更加确定当前链和区块的状态。

关于挖矿的更多信息

以太坊的工作量证明机制是如何运作的？ {#how-it-works}

以太坊的交易被处理为区块。 每个区块都具备以下内容：

- 区块的难度，例如：3,324,092,183,262,715
- 混合哈希（mixHash），例如：`0x44bca881b07a6a09f83b130798072441705d9a665c5ac8bdf2f39a3cdf3bee29`
- nonce--例如：`0xd3ee432b4fb3d26b`

区块数据与工作量证明直接关联。

工作量证明机制 {#the-work}

工作量证明协议 (Ethash)，要求矿工经过激烈的试错竞赛，找到一个区块的随机数。 只有具备有效随机数的区块才能被加入区块链中。

通过比赛创建区块时，矿工将反复放置一个数据集，只能通过数学函数从下载和运行完整链（像矿工那样）中获得数据集。 这是为了根据区块所声明的难度，生成一个低于目标随机数的混合哈希。 做到这些的最佳途径就是试错。

难度决定了该哈希的目标。 目标越小，有效的哈希值的集合就越小。 一旦生成哈希，其他矿工和客户就很容易校验哈希。 即使一个交易记录出现改变，整个哈希就会变得完全不同，表示出现欺诈。

哈希使得交易欺诈性的交易更容易发现。 此外，工作量证明作为一个流程，同样有效威慑了对区块链的攻击。

工作量证明和安全性 {#security}

在以太坊主链上工作的矿工会因挖矿获得回报。 对于一部分矿工而言，他们几乎没有理由去开辟一条自己的新链 - 因为这破坏了系统。 区块链依赖于将单一状态来作为真实性来源。 同时，使用者要始终选择最长的或者“最重”的链。

工作量证明机制是为了扩展链。 最长的链最可信，因为它完成的计算工作最多。 在以太坊的 PoW 系统中，几乎无法创建一个新区块以擦除交易信息、创建假区块或者维持第二个链。 这是因为恶意矿工需要比其他所有人更快地计算出区块随机数。

要一直创建恶意但有效的区块，就需要有超过 51% 的全网采矿算力去打败其他人。 这项“工作”需要大量算力才能完成。 而能源支出可能甚至超出攻击所取得的收益。

工作量证明的经济模型 {#economics}

工作量证明还负责在系统中发行新币，并鼓励矿工参与这项工作。

将为成功创建区块的矿工奖励 2 个新铸造的 ETH 和这个区块内所有的交易费用。 矿工通过计算叔块也可以获得 1.75 个 ETH。 叔块是矿工在另一个矿工出块的同时创建的有效块。 叔块通常由于网络延迟导致。

终局性 {#finality}

如果交易属于一个无法更改的区块时，它就在以太坊上具有“确定性”。

由于矿工分散工作，可以同时开采出两个有效区块。 这创建了临时分叉。 最后，在挖出并添加后续区块，使其中一条链更长时，这条链就会成为被接受的链。

但更复杂的是，在临时分叉上被拒绝的交易将会记录在被接受的主链上。 这意味着区块可逆。 因此，确定性是指您在认为交易不可逆之前需要等待的一段时间。 以太坊推荐的时间是等待 6 个区块或者超过 1 分钟。 6 个区块后，您可以相对稳妥地说交易成功了。 当然，您也可以等待更长的时间，这样更有保障。

在设计去中心化应用时，应始终牢记确定性。 如果将交易信息误报给用户，特别当交易价值很高时，这将严重有损用户体验。

请记住，这个时间不包含等待矿工处理交易的时间。

工作量证明能源消耗 {#energy}

对工作证明提出的一项主要批评是保持网络安全所需的能源消耗。 为了维持安全性和去中心化，以太坊工作量证明每年消耗 73.2 太瓦时的能源，这相当于奥地利这样的中等国家的能耗。

优点和缺点 {#pros-and-cons}

优点	缺点
工作量证明并无对错。 你不需要 ETH 来启动，区块允许你从 0ETH 获得正的收益。 在权益证明中，您需要 ETH 去启动挖矿。	工作量证明消耗了大量能源，对环境不利。
工作量证明是一个经过考验和测试的共识机制，多年来一直保持了比特币和以太坊的安全性和去中心化。	如果你想要采矿，你需要花大量的资金去购买专业设备。
与权益证明方式相比，PoW 是比较容易实施的。	因为算力的不断增加，矿池可能会主导挖矿过程，导致中心化和安全风险。

与权益证明对比 {#compared-to-pos}

从更高层面上看，权益证明 (PoS) 和工作量证明 (PoW) 的最终目标相同，即帮助去中心化网络安全地达成共识。 但他们在实现方式和人员上有些不同：

- 权益证明 (PoS) 让质押 ETH 不再需要算力。
- 权益证明 (PoS) 将矿工替换为验证者。 验证者通过质押他们的 ETH 来创建新的区块。

- 验证者不会竞争创建区块，而是由算法随机选择。
- 终局性是很清晰的：在特定的检查节点，如果 2/3 验证者确定了区块，这个区块将被视为最终状态。 验证者必须在区块压上自己全部质押，因此如果他们试图串通更改区块链，他们就会失去他们的全部质押。

关于权益证明 (PoS) 的更多信息

更偏向视频学习者？ {#visual-learner}

延伸阅读 {#further-reading}

- 多数攻击
- 关于结算终局性

视频 {#videos}

- 对工作量证明协议的技术说明

相关主题 {#related-topics}

- 采矿
- 权益证明

---

title: 采矿
description: 关于挖矿在以太坊中如何工作以及如何帮助保持以太坊安全和去中心化的解释。
lang: zh
sidebar: true

incomplete: true

前置要求 {#prerequisites}

为了更好地了解此页面，推荐先阅读 交易、区块和工作量证明。

以太坊挖矿是什么？ {#what-is-ethereum-mining}

挖矿是一个创造区块来添加转账到以太坊区块链上的过程。

与比特币一样，以太坊目前使用的是工作量证明（PoW）作为共识机制。 挖矿是工作量证明的本质。 以太坊矿工——运行软件的计算机，利用他们的时间和算力，处理交易和生产模块。

权益证明将在未来一年内取代挖矿和工作量证明。 您可以从今天开始权益质押 ETH。 关于权益质押的更多信息

为什么存在矿工？ {#why-do-miners-exist}

像以太坊这样的去中心化系统，我们需要确保每个人同意交易的顺序。 矿工通过解决计算难题来解决这一问题，以产生区块，从而使网络免受攻击，从而帮助实现了这一目标。

工作量证明的更多信息

谁可以成为以太坊上的矿工？ {#who-can-become-a-miner}

从技术上讲，任何人都可以使用自己电脑在以太坊网络上挖矿。 然而，并非每个人都能以挖到以太币 (ETH)。 在大多数情况下，为了盈利，矿工必须购买专门的计算机硬件。 虽然确实任何人都可以在他们的电脑上运行挖矿软件，但一般的电脑不太可能赚取足够的区块奖励来支付挖矿的相关费用。

挖矿成本 {#cost-of-mining}

- 建造和维护矿机所需硬件的潜在成本
- 矿机的电力成本
- 如果您已加入矿池，矿池通常对其产生的每个区块收取固定比例的费用
- 支持矿机的潜在成本（通风、能源监测、电线等）

要进一步探索挖矿利润，请使用挖矿收益计算器，例如 Etherscan 提供的挖矿收益计算器。

如何开采以太坊交易 {#how-ethereum-transactions-are-mined}

1. 用户编写和通过一些帐户私钥来签署交易请求。
2. 用户通过一些节点将自己的交易请求广播到整个以太坊网络。
3. 在听到新的转账请求时，每个以太坊网络节点会添加这笔交易到本地的内存池，这些内存池包括他们收到的没有被添加到区块链以承认的所有转账请求。
4. 在这个时候，一个挖矿节点将几十或上百个交易请求汇总到潜在区块中，从而尽量多收取交易手续费，同时保证不超出区块 gas 限制。 采矿节点将：
    1. 验证每个交易请求的有效性（例如没有人试图将以太币从他们没有签名的帐户中转移出来，请求是否有格式错误等），然后执行请求的本地代码，改变本地副本 EVM 的状态。 矿工获得每个交易请求的转账的手续费到他们的帐户。
    2. 一旦在本地 EVM 副本上验证并执行了块中的所有转账请求，就开始为潜在块生成工作证明“合法性证书”。
5. 最终，矿工将完成为包含我们特定交易请求的区块生成的证书。 然后，矿工广播完成的区块，其中包括证书和校验新 EVM 状态。
6. 其他节点将收到新的区块。 他们将验证证书，执行区块上所有的转账（包括最初由用户广播的交易），然后校验新 EVM 状态，之后执行所有满足 EVM 校验和的转账。 只有这样，这些节点才会将该块附加到区块链的尾部，并接受新的 EVM 状态作为新的规范状态。
7. 每个节点将从其未完成的本地内存池的转账请求中删除新区块中已经存在的转账请求。
8. 加入网络的新节点将按顺序下载所有块，包括未被打包的交易块。 初始化本地 EVM 副本（作为空白状态的 EVM 开始），在本地 EVM 副本上执行每个块中的每个转账，校验各块的校验和。

每个交易都只会被开采（包含在新区块中并首次传播）一次，但在推进规范 EVM 状态的过程中由每个参与者执行和验证。 这凸显出区块链的核心思想之一：**不信任，请校验。**

直观演示 {#a-visual-demo}

跟随 Austin 了解挖矿和工作量证明区块链。

延伸阅读 {#further-reading}

- 挖矿是什么意思？ *EthHub*

相关工具 {#related-tools}

- 顶级以太坊矿工
- Etherscan 挖矿收益计算器
- Minerstar 挖矿收益计算器

相关主题 {#related-topics}

- Gas
- EVM
- 工作量证明

---

title: dapp 介绍
description:
lang: zh

sidebar: true

去中心化应用 (dapp) 是在去中心化网络上构建的应用程序，结合了智能合约和前端用户界面。 请注意，以太坊智能合约具有可访问性和透明性——就像开放 API 一样 ——所以你的 dapp 里甚至可以包含其他人写过的智能合约。

基本要求 {#prerequisites}

在学习 dapp 之前，您应该了解区块链基础知识，并了解以太坊网络及其去中心化方式。

dapp 的定义 {#definition-of-a-dapp}

一个 dapp 的后端代码在一个去中心化 P2P 网络上运行。 与此相对应的，是在中心化服务器上运行后端代码的应用程序。

dapp 可以用任何语言编写（就像是一个 app）。它有前端代码和用户界面，能调用其后端。 此外，它的前端可以托管在去中心化存储上，例如 IPFS。

- **去中心化** - dapp 在以太坊上运行，这是一个开放的公共去中心化平台，没有任何一个人或团体可以控制
- **确定性**，无论执行的环境如何，都执行相同的功能。
- **图灵完备** - dapp 可以根据所需资源执行任何操作
- **隔离性** - 它们在称为 EVM 的虚拟环境中执行。即使智能合约出现问题，也不会妨碍区块链网络的正常运行

智能合约 {#on-smart-contracts}

要引入 dapp，我们需要引入智能合约 —— dapp 的后端，因为缺少更好的术语。 有关详细概述，请访问我们的智能合约部分。

智能合约是一种在以太坊网络上的计算机程序，它严格按照事先编写的代码来运行。 智能合约一旦部署到以太坊网络中，就无法更改。 Dapps 可以是去中心化的，就是由于它们受智能合约的既定逻辑控制，而不是个人或公司。 这也意味着你需要非常仔细地设计合约，并进行全面测试。

Dapp 开发的好处 {#benefits-of-dapp-development}

- **零停机时间** – 一旦将某 dapp 的智能合约部署到区块链上，整个网络都能为那些希望与合约互动的客户提供服务。 因此，恶意参与者无法针对单个 dapp 发起 DoS 攻击。
- **隐私** – 您不需要提供真实世界的身份来部署或与 dapp 进行交互。
- **抵制审查** – 网络上没有任何一个实体可以阻止用户提交交易、部署 dapp 或读取区块链上的数据。
- **数据完整性** – 由于采用了加密基元，存储在区块链上的数据是不可更改和无可争议的。 恶意行为者无法伪造已经公开的交易或其他数据。
- **无需信任关系的计算/可验证的行为** – 智能合约可以被分析，并保证以可预测的方式执行，而无需信任中央权威机构。 这在传统模式下是不存在的，比如我们使用网上银行系统时，我们要相信金融机构不会滥用我们的金融数据，不会篡改记录，也不会被黑客攻击。

Dapp 开发的缺陷 {#drawbacks-of-dapp-development}

- **维护** – dapp 可能更难维护，因为发布到区块链的代码和数据更难修改。 一旦部署了 dapp（或 dapp 存储的底层数据），开发者就很难对其进行更新——即使在旧版本中发现了 bug 或安全风险。
- **性能开销** – 巨大的性能开销，而且难以扩展更多性能。 为了达到以太坊所追求的安全、完整、透明和可靠的水平，每个节点都会运行和存储每一笔交易。 除此之外，工作量证明也需要时间。 粗略计算，开销会达到目前标准计算的 1,000,000 倍左右。
- **网络拥塞** – 至少在当前模型中，如果一个 dapp 使用了太多的计算资源，整个网络都会承担影响。 目前，该网络每秒只能处理约 10 笔交易；如果交易发送的速度超过这个速度，未确认的交易池会迅速膨胀。
- **用户体验** – 设计用户友好的体验可能更难。普通终端用户可能会发现，很难以真正安全的方式设置与区块链互动所需的工具堆栈。
- **集中化** — 无论如何，建立在以太坊基础层之上的用户友好型和开发人员友好型解决方案最终看起来都像集中式服务。 例如，这种服务可以在服务器端存储密钥或其他敏感信息，使用中心化服务器为前端服务，或在写到区块链之前在中心化服务器上运行重要的业务逻辑。 这消除了区块链与传统模式相比的许多（并不是全部）优势。

更偏向视频学习者？ {#visual-learner}

Dapp 工具 {#dapp-tools}

Scaffold-ETH _ - 使用能自适应智能合约的前端，快速尝试 Solidity。_

- GitHub
- 示例 dapp

创建 Eth App - *使用一个命令创建支持以太坊的应用程序。*

- GitHub

一键去中心化 - *用于从 ABI 生成去中心化应用程序前端的免费和开放源代码软件工具。*

- oneclickdapp.com
- GitHub

Etherflow _ - 供以太坊开发者测试节点，并在浏览器中编写和调试 RPC 调用的免费和开放源代码软件工具。_

- etherflow.quiknode.io
- GitHub

延伸阅读 {#further-reading}

- The Architecture of a Web 3.0 application - *Preethi Kasireddy*
- A 2021 guide to decentralized applications - *LimeChain*
- What Are Decentralized Apps? - *Gemini*

还有哪些社区资源对您有所帮助？ 请编辑本页面并添加！

相关主题 {#related-topics}

- 以太坊堆栈简介
- 开发框架

---

title: 如何加入？
description: 如何加入以太坊社区。
sidebar: true

lang: zh

如何加入？ {#get-involved}

以太坊社区中的成员具有许多不同的背景和技能。 无论您是开发人员、艺术家还是会计师，都可以加入。 以下建议列表可能会帮助您开始加入。

开发人员 {#developers}

- 访问 ethereum.org/developers/，了解和尝试使用以太坊
- 在 Gitcoin 上查找悬赏，解决一个或大或小的技术问题，赚取加密货币！
- 参加您附近的一个 ETHGlobal 黑客马拉松活动！
- 查看与您的专业领域或所选编程语言相关的项目
- 观看或参与核心开发人员电话会议
- 生态系统支持计划的愿望清单 - 适用于工具、文档和基础设施领域，以太坊生态系统支持计划正在积极寻求这些领域的资助申请
- Web3Bridge - 加入有抱负的 web3 社区，一起积极寻找、培训和支持整个非洲的数百名开发人员和社区成员

研究人员 & 学者 {#researchers-and-academics}

您是否有数学、加密或经济学方面的背景？ 您可能会对以太坊生态系统中正在进行的一些前沿工作感兴趣

- Challenges.ethereum.org - 一系列高额研究奖金，您可以从中赚取 >100,000 美元
- ethrear.ch - 以太坊的主要研究论坛，也是世界上最具影响力的加密经济学论坛
- 生态系统支持计划的愿望清单 - 适用于多个研究领域，以太坊生态系统支持计划正在积极寻求这些研究领域的资助申请

非技术技能组合 {#non-technical}

如果您不是开发人员，可能很难知道从哪里入手以太坊。 这里有几项建议和为特定专业背景提供的资源。

在您的城市组织一场小聚会 {#meetups}

- 不知道如何开始？ BUIDL 网络可以提供帮助。

撰写关于以太坊的内容 {#write-content}

- 以太坊需要能用通俗易懂的语言阐述以太坊价值的优秀写手
- 还没有准备好发布自己的文章？ 可以考虑为 EthHub 等社区资源上的现有内容贡献一份力量，或者为 ethereum.org 提出新内容！

主动提出为社区电话会议做会议纪要 {#take-notes}

- 现在有许多开源社区电话会议，有人记录会议纪要将是一个极大的帮助。 如果您感兴趣，请加入 Ethereum Cat Herders discord，并做简短的自我介绍！

将以太坊的相关文章翻译成您的母语 {#translate-ethereum}

- ethereum.org 具有一个翻译计划，可将网站和其他资源翻译成多种语言
- 点击这里了解如何参与

运行节点 {#run-a-node}

与数以千计的节点运营商一起，帮助以太坊进一步去中心化。

- 关于如何运行节点的更多信息

质押您的以太币 {#staking}

通过质押您的以太币，您可以在帮助保护以太坊网络安全的同时获得奖励。

- 关于权益质押的更多信息

资助项目 {#suport-projects}

以太坊生态系统的使命是资助公共物品和有影响力的项目。 只需很少的捐款，您就可以表达您的支持，并让重要的工作得以实现。

- Gitcoin
- clr.fund

金融专业人士 & 会计师 {#financial-professionals}

- 以太坊是“去中心化金融”生态系统的发源地，该系统是一个提供替代金融系统的协议和应用程序网络。 如果您是金融专业人士，请访问 DeFi Pulse 或 DeFiPrime，查看一些去中心化金融应用程序。
- 会计师？ 以太坊上的资产（以太币、令牌、去中心化金融等）带来了许多新的会计问题。 您可以从查看一些项目开始，帮助加密货币用户克服他们的记账 & 会计挑战，如 Rotki

产品经理 {#product-managers}

- 以太坊生态系统需要有才之士！ 许多公司正在招聘产品经理。 如果您想从为开源项目做贡献开始，请联系 Ethereum Cat Herders 或 MetaCartel

市场营销 {#marketing}

- 以太坊生态系统中提供了许多市场营销和通信相关的职位！

以太坊招聘职位 {#ethereum-jobs}

想在以太坊找到一份工作吗？

- ethereum.org 招聘职位
- 以太坊基金会工作委员会 (Lever)
- 以太坊基金会工作委员会 (BambooHR)
- Cryptocurrency 招聘职位
- Crypto.招聘职位
- ConsenSys 招聘职位
- Crypto 招聘职位列表
- Bankless 招聘平台
- useWeb3 招聘职位
- Web3 招聘职位
- Web3 Army

加入去中心化自治组织 {#decentralized-autonomous-organizations-daos}

"DAO" 是去中心化自治组织。 这些团队利用以太坊技术促进组织和协作。 例如，会籍管理、提案投票或管理集合资产。 虽然去中心化自治组织仍处于试验阶段，但它们为您提供了机会，去寻找您认同的团队和合作者以及扩大您对以太坊社区的影响。 关于去中心化自治组织的更多信息

- LexDAO @lex_DAO - *法律工程*
- Machi X @MachiXOfficial - *艺术社区*
- MetaCartel @Meta_Cartel - *去中心化自治组织培养基地*
- MetaCartel Ventures @VENTURE_DAO - *从事种子前加密工程的组织*
- MetaGame @MetaFam - *为真实生活而生的 MMORPG 游戏机制*
- MetaFactory @TheMetaFactory - *数字化实体服装品牌*
- MolochDAO @MolochDAO - *专注于资助以太坊发展的社区*
- MΓΔ (Meta Gamma Delta) @metagammadelta - *女性主导的项目*
- Raid Guild @RaidGuild - *由 Web3 构建者组成的集体企业*
- DAOSquare @DAOSquare - *在非技术领域推广去中心化自治组织这一概念，帮助人们通过去中心化自治组织创造价值。*
- dOrg @dOrg_tech - *以去中心化自治组织形式运行的自由职业者 Web3 开发团队*
- HausDAO @nowdaoit - *DAOhaus 社区治理*
- Developer DAO @developer_dao - *相信互联网为集体所有的构建者社区*
- DXdao @DXdao - *去中心化开发& dApps & 的治理 协议*

---

title: 区块浏览器
description: 这是对于区块浏览器的介绍，让您进入区块链数据世界的大门，在这里您可以查询交易、帐户、合约以及更多以太坊区块链相关数据。
lang: zh
sidebar: true

sidebarDepth: 3

区块浏览器是您进入以太坊数据世界的大门。 您可以通过区块浏览器来查看实时的区块、交易、矿工、帐户以及其他的链上相关活动。

前提条件 {#prerequisites}

您应该理解以太坊的基本概念，以便您能够理解区块浏览器向您展示的数据。 开始介绍以太坊。

服务 {#services}

- Etherscan – *中文、韩文、俄文和日文*
- Etherchain
- Ethplorer -_ 支持中文、西班牙语、法语、土耳其语、俄语、韩语和越南语_
- Blockchair --*也有西班牙文、法文、意大利文、荷兰文、葡萄牙文、俄文、中文和波斯文*
- Blockscout--区块链浏览器
- OKLink

数据 {#data}

以太坊从设计上是透明的，所有信息都可以验证。 区块浏览器为获取这些信息提供了一个界面。 这适用于以太坊的主网络和测试网，如果您需要这些数据的话。

以下是您可以从区块浏览器获得的数据类型概要。

区块 {#blocks}

每隔 12 秒就会有新的区块添加到以太坊（这个时间可能会波动），因此有几乎恒定的数据流被添加到区块浏览器中。 区块里面包含着许多您可能会觉得有用的重要数据：

常规数据

- 区块高度 – 创建当前区块时区块链的区块数量和长度（以区块为单位）。
- 时间戳 – 矿工开采该区块的时间。
- 交易 – 一个区块包含的交易数量。
- 矿工 – 挖掘出当前区块的矿工地址。
- 奖励 – 为添加该区块授予矿工的 ETH 数量（标准 2ETH 奖励 + 区块中包含的交易的任何交易费用）。
- 难度 – 与开采区块相关的难度。
- 大小 – 区块中数据的大小（以字节为单位）。
- 消耗的 Gas – 区块中交易消耗的 Gas 的总单位。
- Gas 限制 – 区块中的交易设置的总 Gas 限制。
- 额外数据 – 矿工已 附加在区块中的任何额外数据。

高级数据

- 哈希值 – 代表区块头的加密哈希值（区块的唯一标识符）。
- 父哈希 – 前一个区块的哈希值。
- Sha3Uncles – 个给定的父节点所有叔块的散列组合。
- StateRoot – Merkle trie 的根哈希，存储系统的整个状态。
- Nonce – 一个用于显示矿工对区块的工作量证明的值。

叔块

当两个矿工几乎同时创建区块时，就会创建叔块 – 在这些节点中只有一个区块有效。 叔块不会打包到区块链上，但是矿工仍然会因为其所做的工作量得到奖励。

区块浏览器提供关于叔块的信息，如：

- 叔块数字。

- 发生时间。
- 它们被创建时的块高度。
- 谁开采它。
- ETH 奖励。

Gas {#gas}

区块浏览器不仅会为您提供有关交易和区块中 Gas 使用量的数据，而且还会为您提供有关网络当前 Gas 价格的信息。 这将帮助您了解网络使用情况，提交安全的交易和不超支的 Gas。 寻找可以帮助您将这些信息获取到您自己的产品界面的 API。 Gas 特定数据涵盖：

- 安全但缓慢的交易所需的估计 Gas 单位（+估计价格和持续时间）。
- 平均交易所需的估计 Gas 单位（+估计价格和持续时间）。
- 快速交易所需的估计 Gas 单位（+估计价格和持续时间）。
- 基于 gas 价格的平均确认时间。
- 消耗 gas 的合约 – 换句话说，在网络上有大量使用的流行应用。
- 正在消费 Gas 的帐户 – 换句话说，就是频繁使用网络的用户。

交易 {#transactions}

区块浏览器已成为人们跟踪交易进度的常见应用， 这是因为您可以获得的详细程度提供了更多的确定性。 交易数据包含：

标准数据

- 交易哈希 – 交易被提交时生成的哈希值。
- 状态 – 指示交易是等待、失败还是成功。
- 区块 – 包含交易的区块。
- 时间戳 – 矿工开采交易的时间。
- 发件人 – 提交交易的帐户的地址。
- 收件人 – 与交易进行交互的收件人或智能合约的地址。
- 被转移的代币 – 作为交易的一部分被转移的代币列表。
- 价值 – 被转移的全部 ETH 的价值。
- 交易费用 – 支付给矿工处理交易的金额（以 gas 价格\ *消耗的 gas 计算）。

高级数据

- Gas 限制 – 此交易可以消耗的最大 gas 单位数。
- 消耗的 Gas – 交易消耗的实际 gas 单位数量。
- Gas 价格 – 每 gas 单位设定的价格。
- Nonce – `发件人`地址的交易号（请记住，它从 0 开始，所以`100`的现时数实际上是该帐户提交的第 101 个交易。
- 输入数据 – 交易所需的任何额外资料。

帐户 {#accounts}

您可以访问一个帐户的很多数据。 这就是为什么建议用多个帐户来操作，这样您的资产和价值传输才不会轻易地被追踪。 还有一些解决方案正在开发之中，以使交易和帐户活动更加具备隐私保护。 以下是帐户可用的数据：

用户帐户

- 帐户地址 – 您可以用来发送资金的公共地址。
- ETH 余额 – 与该帐户相关联的 ETH 金额。

- 总 ETH 值 – ETH 的值。
- 代币 – 与帐户关联的代币及其价值。
- 交易历史 – 该帐户发送和接受的交易历史列表。

智能合约

智能合约帐户拥有用户帐户将拥有的所有数据，但一些区块浏览器甚至还会显示一些代码信息。 比如如下实例：

- 合约创建者 – 将合约部署到主网的外部帐户地址。
- 合约创建交易 – 在主网包含此次合约部署的交易。
- 源代码 – solidity 和 vyper 的智能合约源代码。
- 合约 ABI – 合约的应用二进制接口 - 合约调用和接收的数据。
- 合约创建代码 – 合约编译后的字节码 - 当您写完 solidity 和 vyper 并且编译后生成。
- 合约事件 – 合约调用方法的历史记录， 基本上用于查看合约如何使用以及被调用的频率。

代币 {#tokens}

代币合约是合约的一种类型，所以他们会有类似于合约的数据。 但是因为它们有价值并且可以交易，所以它们还有其他数据点：

- 类型 – 它们是 ERC-20、ERC-721 或其他代币标准。
- 价格 – 如果是 ERC-20，它们将具有当前的市场价值。
- 市值 – 如果它们是 ERC-20，则将具有市值（由价格\ *总供应量计算）。
- 总供应量 – 流通的代币数量。
- 持有人 – 持有代币的地址数。
- 转账 – 代币在帐户之间转账的次数。
- 交易历史记录 – 包括代币在内的所有交易的历史记录。
- 合约地址 – 部署到主网的代币的地址。
- 小数位 – ERC-20 代币是可分割的，并具有小数位。

网络 {#network}

当然，有些数据说明了网络的健康状况。 这些是专门针对以太坊的工作量证明共识机制的。 当以太坊过渡到权益证明阶段时，某些数据将冗余

- 难度 – 当前的采矿难度。
- 哈希率 – 对试图解决当前以太坊区块或任何给定区块的以太坊矿工正在生成多少哈希的估计。
- 总交易数 – 自以太坊创建以来的交易数。
- 每秒交易数 – 每秒钟内可处理的交易次数。
- ETH 价格 – 1 ETH 的当前价值。
- 以太坊总供应量 – 流通中的以太坊数量 – 请记住，以区块奖励的形式创建每个区块都会创建新的 ETH。
- 市值 – 以价格\ *供应量计算。

共识层数据 {#consensus-layer-data}

以太坊扩容升级仍在开发中，但区块浏览器为您提供的数据点值得一看。 事实上，所有这些数据现在都可用于测试网。

如果您不熟悉以太坊共识层路线图，请查看以太坊升级概述 。

Epoch {#epoch}

信标链将创建多个验证委员会。出于安全考虑，在每个周期（每 6.4 分钟）结束时会随机分配验证者到委员会中。 Epoch 数据包含：

- Epoch 号：
- 最终状态 – 周期是否完成（是/否）。
- 时间 - epoch 结束的时间。
- 区块认证 - epoch 中的区块认证次数（slot 内区块的投票）。
- 存款 - 特定 epoch 包含的 ETH 存款的数量（验证者必须质押 ETH 才能成为验证者）。
- 惩罚 - 对区块的提出者或证明人的惩罚数。
- 投票参与 - 用于证明区块的抵押 ETH 数量。
- 验证者 - 在特定 epoch 活跃的验证者数量。
- 平均验证者余额 - 活动验证者的平均余额。
- Slot - epoch 中包含的 slot 数（slot 包括一个有效区块）。

Slot {#slot}

Slot 是区块创建的机会，每个 slot 可用的数据包括：

- Epoch - slot 有效的 epoch。
- Slot 号。
- 状态 - slot 的状态（提议/缺失）。
- 时间 - slot 的时间戳。
- 提议者 - 提议 slot 的验证者。
- 区块根 - 信标块的哈希树根。
- 父根 - 以前块的哈希。
- 状态根 - 信标状态的哈希树根。
- 签名。
- Randao reveal。
- Graffiti - 区块提议者可以在其区块提议中包含 32 个字节长的消息。
- 执行数据。
  - 区块哈希。
  - 存款余额。
  - 存款根。
- 区块认证 – 此 Slot 中该区块的认证数量。
- 存款者 - 此 Slot 内的存款数量。
- 自愿退出 - Slot 中退出的验证者数量。
- 惩罚 - 对区块的提出者或证明人的惩罚数。
- 投票 - 验证者在此 Slot 为该区块投票。

区块 {#blocks-1}

在共识层中，区块的工作方式不同，因为矿工被验证者取代，而信标链将时隙和周期引入了以太坊。 这意味着新的数据！

- 提议者 - 在算法上选择提议新区块的验证者。
- Epoch - 提出该区块的 Epoch。
- Slot - 区块被提议的 Slot。
- 区块认证 - Slot 中包含的区块认证数量， 区块认证就像投票一样，表明该区块已准备好进入信标链。

验证者 {#validators}

验证者负责提出区块并在 Slot 中对其进行证明。

- 验证者编号 - 代表验证者的唯一编号。
- 当前余额 - 验证者包含奖励在内的余额。
- 有效余额 - 验证者用于质押的余额。
- 收入 - 验证者收到的奖励或惩罚。
- 状态 - 验证者当前是否在线和活动。
- 区块认证有效性 - 将验证者的区块认证纳入链中平均需要的时间。
- 激活资格 - 验证者可用来验证的日期（和 epoch）。
- 活动开始时间 - 验证者开始激活的日期（和 epoch）。
- 已提交区块 - 验证者已经提交的区块数量。
- 区块认证数量 - 验证者已经提供的区块认证数量。
- 存款 - 来自地址、交易哈希、区块号码、时间戳、金额和由验证者进行的质押存款状态。

区块认证 {#attestations}

区块认证就像是投票标记“是”将区块加入到链上， 这些数据关联到区块认证的历史以及哪个验证者已经证明。

- Slot - 区块认证发生的 Slot。
- 委员会索引 - 给定 Slot 的委员会索引。
- 聚合位 - 代表所有参与验证者在区块认证中的聚合证明。
- 验证者 - 提供认证的验证者。
- 信标链根 - 指向验证者正在证明的区块。
- 来源 – 指向最新的更正后的 epoch。
- 目标 - 指向最新 epoch 边界。
- 签名。

网络 {#network-1}

共识层顶级数据包括以下内容：

- 当前 epoch。
- 当前 Slot。
- 活跃的验证者 - 活跃的验证者数量。
- 等待验证者 - 等待激活的验证者数量。
- 已质押的 ETH - 质押到网络的 ETH 数量。
- 平均余额 - 验证者的平均 ETH 余额。

区块浏览器 {#block-explorers}

- Etherscan - 一个您可以用来获取以太坊主网、Ropsten Testnet、Kovan Testnet、Rinkeby Testnet 和 Goerli Testnet 数据的区块浏览器。
- Blockscout - 主要关注以下网络：
  - xDai - MakerDAO 的 DAI 稳定币与 POA 的侧链和代币桥技术的巧妙结合。
  - POA - 由一组可信验证者保护的侧链和自治网络。 网络上的所有验证者均为可信公众人，其信息可公开获得。
  - POA Sokol 测试网。
  - ARTIS - 以太坊兼容的区块链。
  - LUKSO L14 - L14 充当第一个测试网络，以允许 LUKSO 社区在通用基础结构上进行构建和测试。
  - qDai
- Etherchain - 以太坊主网的区块浏览器。
- Ethplorer - 一个专注于以太坊主网和 Kovan 测试网上代币信息的区块浏览器。
- Blockchair - 最具隐私保护的以太坊浏览器。 也用于排序和过滤（内存池）数据。

信标链（共识层）区块浏览器 {#beacon-chain-block-explorers}

- https://beaconcha.in/
- https://beaconscan.com/
- https://ethscan.org/（beaconcha.in 的分叉）

延伸阅读 {#further-reading}

还有哪些社区资源对您有所帮助？ 编辑并添加本页面！

相关主题 {#related-topics}

- 挖矿
- 交易
- 帐户
- 网络

---

title: 数据和分析学
description: 如何获取在链分析和数据以用于您的去中心化应用程序。
lang: zh

sidebar: true

介绍 {#Introduction}

随着网络利用率不断提高，链上数据中将有越来越多的宝贵信息。 随着数据量的迅速增加，计算和汇总此信息以报告或驱动一个去中心化应用程序可能变成很费时间和体力的过程。

利用现有数据提供商可以加快发展，产生更准确的结果，并减少正在进行的维护工作。 这将使一个团队能够集中精力处理试图在项目中提供的核心功能。

前提条件 {#prerequisites}

您应该理解区块浏览器 的基本概念，以便更好地理解在数据分析环境中使用它们。 此外，熟悉索引概念，以了解它们给系统设计带来的好处。

就架构基础而言，也要从理论上了解应用程序接口和 REST 是什么。

图表 {#the-graph}

Graph 网络是用于组织区块链数据的去中心化索引协议。 通过 Graph 网络，开发者可以建立完全在公共基础设施上运行的无服务器应用程序，而不是建立和管理链外和集中的数据存储来聚合链上数据。

使用 GraphQL，开发人员可以查询任何管理的开放应用程序接口（被称为子图表），以获得必要的信息来驱动他们的应用程序。 通过查询这些索引子图表，报告和去中心化应用程序不仅能够提高性能和可扩展性，而且能够通过网络共识准确地构建。 当网络中新增改进和/或子图表时，您的项目可以快速迭代，利用这些增强功能。

区块浏览器 {#block-explorers}

许多区块浏览器提供 RESTfify 应用程序接口网关，它将使开发者可以看见区块上的实时数据、交易、矿工，帐户和其他链上活动。

开发者然后可以处理和转换此数据，让他们的用户有独特的洞察力并与区块链交互。

延伸阅读 {#further-reading}

- Graph 网络概览
- Graph 查询实战
- EtherScan 上的应用程序接口代码示例

---

title: 开发网络
description: 对以太坊应用的开发网络环境与开发工具的概览。
lang: zh

sidebar: true

当使用智能合约来开发一个以太坊应用时，您可能想要在部署之前在本地查看它是如何工作的。

这和在本地运行一个本地网页服务器相似。为了测试您的去中心化应用程序，您可以使用开发网络创建一个本地的区块链。 这些以太坊开发网络提供了能够比公共测试网更快的迭代功能（例如您不需要从测试网获取以太币）。

前置要求 {#prerequisites}

您应该先理解以太坊堆栈和以太坊网络基础知识才能进入开发网络。

什么是开发网络？ {#what-is-a-development-network}

实质上开发网络是指哪些对本地开发特殊设计的以太坊客户端（会对以太坊进行部署应用）。

为什么不在本地运行一个标准的以太坊节点?

您*可以* 运行一个节点（例如 Geth、Erigon 或 Nethermind），但由于开发网络是为开发目的而建立的，它们往往会打包一些快捷方便的功能，例如：

- 为本地区块链提供数据，这个功能很重要（例如使用以太币余额的帐户）
- 在接受到每个交易时就立即按顺序和没有延迟地挖掘区块。

- 增强调试和日志功能

可用工具 {#available-projects}

注意：大多数开发框架包含一个内置的开发网络。 我们建议从一个框架开始设置您的本地开发环境。

Ganache {#ganache}

快速构建一个个人的以太坊区块链，您可以用它来运行测试，执行命令，并在控制链的运行方式时检查状态。

Ganache 提供了一个桌面应用程序 (Ganache UI) 以及一个命令行工具 (`ganache-cli`)。 它是 Truffle 工具套装的一部分。

- 网站
- GitHub
- 相关文档

Hardhat 网络 {#hardhat-network}

一个专门用于开发的本地以太坊网络。 它允许您部署您的合约，运行您的测试并调试您的代码.

Hardhat 网络内置了安全帽，安全帽是专业人员的以太坊开发环境。

- 网站
- GitHub

延伸阅读 {#further-reading}

还有哪些社区资源对您有所帮助？ 编辑并添加本页面！

相关主题 {#related-topics}

- 开发框架
- 构建您的本地开发环境

---

title: 以太坊堆栈简介
description: 一次从不同层面对以太坊堆栈原理的浏览，并且了解如何让不同的以太坊堆栈融为一体。
lang: zh

sidebar: true

就像其他任何一种堆栈结构，完整的“以太栈”会基于不同的目的在不同的项目之间变换。

然而，以太坊的核心技术是提供一种心智模型，这种模型帮助解决了以太坊区块如何在不同的应用之间的交互的问题。 理解堆栈的层级将有助于您理解可以将以太坊融入软件项目的不同方法。

级别 1：以太坊虚拟机 {#ethereum-virtual-machine}

以太坊虚拟机 (EVM) 是以太坊智能合约的运行环境。 以太坊区块链上所有智能合约的状态都通过执行交易来改变。

与任何虚拟机一样，以太坊虚拟机在执行代码和执行机器（以太坊节点）之间创建一个抽象级别。 目前，在世界各地散布着成千上万的正在执行的以太坊虚拟机节点。

在后台，那些以太坊虚拟机会使用操作码执行一些特殊的任务。 这些 140 个独特的执行码使得以太坊虚拟机是图灵完备的。这意味着以太坊虚拟机只要有足够的资源就能够计算出任何东西。

作为去中心化应用程序的开发者，除了了解以太坊虚拟机的存在之外，您不需要了解更多关于以太坊虚拟机的信息，并且可以在以太坊上畅通无阻地授权所有应用程序。

级别 2：智能合约 {#smart-contracts}

智能合约是在以太坊区块链上运行的可执行程序。

智能合约使用了特定的编程语言来编译到以太坊虚拟机字节码（调用操作码的低级机器说明）。

智能合约不仅是开放源码库，而且它们基本上是运行 24/7 的开放应用程序接口服务，不能被取消。 智能合约提供了为用户和应用程序（去中心化应用程序）之间交互的公开方法，无需许可。 任何应用程序都可能会与已部署的智能合约集成组成功能，如添加数据源或支持代币交换。 任何人都可以在以太坊上部署智能合约，以便添加自定义功能来满足其应用程序的需要。

作为一个去中心化应用程序开发者，如果您需要在以太坊区块链上添加自定义功能，需要通过写智能合约来实现。 您可能会发现您可以仅仅通过与现有智能合约进行整合来满足您项目的大部分或全部的需要。例如，如果您想要支持支付稳定币或启用分散交换代币。

级别 3：以太坊节点 {#ethereum-nodes}

为了让应用程序与以太坊区块链交互，它必须连接到 以太坊节点。 正在连接到一个节点，您可以读取区块链数据和/或将交易发送到网络。

以太坊节点是运行着软件的电脑，这些软件也就是一个以太坊客户端。 客户端是一种以太坊的实现，它可以验证每个区块中的所有交易，从而确保网络安全和数据准确。 **以太坊节点是以太坊区块链**。 他们集体存储以太坊区块链的状态，并根据改变区块链状态的交易达成共识。

通过将应用程序连接到以太坊节点（通过 JSON RPC 应用程序接口），应用程序可以从区块链（例如用户帐户余额）读取数据，并向网络广播新交易（例如在用户帐户之间传输以太币或执行智能合约的功能）。

级别 4：以太坊客户端应用程序接口 {#ethereum-client-apis}

许多方便的库（由以太坊开源社区建立和维护）允许您的终端用户应用程序连接到以太坊区块链并进行通信。

如果您的面向用户应用程序是一个 web 应用程序，您可以直接选择在您的前端使用 `npm 安装` 一个 JavaScript 应用程序接口。 或许您会选择使用 Python 或 Java 的应用程序接口在后端实现此功能。

虽然这些应用程序接口不是栈必须的一部分，但它们抽象并消减了与以太坊节点直接互动的大部分复杂性。 它们还提供好用的函数（例如：将 ETH 转化为 Gwei），而作为开发者，您可以花费更少的时间处理以太坊客户端的复杂问题，从而将更多的时间集中于处理您的应用程序的独特功能。

级别 5：终端用户应用程序 {#end-user-applications}

在栈的顶层是面向用户的应用程序。 这些是您今天经常用和构建的标准应用程序：主要是网络程序和移动应用程序。

开发这些用户界面的方式基本上保持不变。 用户常常不需要知道他们所使用的应用程序是使用区块链构建的。

准备好选择技术栈了吗？ {#ready-to-choose-your-stack}

查看我们的指南为您的以太坊应用程序 设置本地开发环境。

延伸阅读 {#further-reading}

- 一个 Web 3.0 应用的架构 - *Preethi Kasireddy*

还有哪些社区资源对您有所帮助？ 编辑并添加本页面！

title: 以太坊虚拟机 (EVM)
description: 介绍以太坊虚拟机及其与状态、交易和智能合约的关系。
lang: zh

sidebar: true

EVM 的物理实例不能像人们指向云或海浪那样描述，它是真实*存在*并由数以千计运行以太坊客户端的计算机共同维护的一个实体。

以太坊协议本身的存在仅仅是为了保持这种特殊状态机的连续、不间断和永久的运行。 这是所有以太坊帐户和智能合约赖以存在的环境。 在链中任何给定的区块上，以太坊只有一个“规范”状态，而 EVM 定义了从一个区块计算产生下一个区块新的有效状态的规则。

前置要求 {#prerequisites}

对计算机科学中常见术语的基本了解，如字节、内存和堆栈是理解 EVM 的前提。 熟悉诸如 哈希函数、工作量证明和 Merkle 树等密码学/区块链概念对理解也很有帮助。

从账本到状态机 {#from-ledger-to-state-machine}

通常使用“分布式账簿”的类比来描述像比特币这样的区块链，它使用密码学的基本工具来实现去中心化的货币。 加密货币的行为类似于“正常”货币，这是因为有规则支配着人们可以做什么和不可以做什么来修改账簿。 例如，比特币地址不能花费比之前收到的更多的比特币。 这些规则是比特币和许多其他区块链上所有交易的基础。

虽然以太坊有自己的本机加密货币 (ETH)，遵循几乎完全相同的直观规则，但它也支持更强大的功能：智能合约。 对于此更复杂的功能，需要一个更复杂的类比。 以太坊不是分布式账本，而是分布式状态机器。 以太坊的状态是一个大型数据结构，它不仅保存所有帐户和余额，而且还保存一个*机器状态*，它可以根据预定义的一组规则在不同的区块之间进行更改，并且可以执行任意的机器代码。 在区块中更改状态的具体规则由 EVM 定义。

EVM 组成结构图 *图表来自 Ethereum EVM illustrated*

以太坊状态转换函数 {#the-ethereum-state-transition-function}

EVM 的行为就像一个数学函数：在给定输入的情况下，它会产生确定性的输出。 因此，将以太坊更正式地描述为具有**状态转换函数**非常有帮助：

```
Y(S, T)= S'
```

给定一个旧的有效状态 `（S）` > 和一组新的有效交易 `（T）`，以太坊状态转换函数 `Y（S，T）` 产生新的有效输出状态 `S'`

状态 {#state}

在以太坊的上下文中，状态是一个巨大的数据结构，称为调整后的 Merkle Patricia Trie，使所有帐户通过哈希链接，并可回溯到存储在区块链上的单个根哈希。

交易 {#transactions}

交易是来自帐户的密码学签名指令。 交易分为两种：一种是消息调用交易，另一种是合约创建交易。

合约创建交易会创建一个新的合约帐户，其中包含已编译的 智能合约 字节码。 每当另一个帐户对该合约进行消息调用时，它都会执行其字节码。

EVM 说明 {#evm-instructions}

EVM 作为一个堆栈机运行，其栈的深度为 1024 个项。 每个项目都是 256 位字，为了便于使用，选择了 256 位加密技术（如 Keccak-256 哈希或 secp256k1 签名）。

在执行期间，EVM 会维护一个瞬态*内存*（作为字可寻址的字节数组），该内存不会在交易之间持久存在。

然而，合约确实包含一个 Merkle Patricia *存储* trie（作为可字寻址的字数组），该 trie 与帐户和部分全局状态关联。

已编译的智能合约字节码作为许多 EVM opcodes 执行，它们执行标准的堆栈操作，例如 `XOR`、`AND`、`ADD`、`SUB` 等。 EVM 还实现了一些区块链特定的堆栈操作，如 `ADDRESS`、`BALANCE`、`BLOCKHASH` 等。

表明 EVM 操作需要 Gas 的图表 *图表改编自以太坊 EVM 说明*

EVM 实现 {#evm-implementations}

EVM 的所有实现都必须遵守以太坊黄皮书中描述的规范。

在以太坊的 5 年历史中，EVM 经历了几次修订，并且以各种编程语言完成了 EVM 的多种实现。

所有以太坊客户端都包含一个 EVM 实现。 此外，还有多个独立的实现方法，包括：

- Py-EVM - *Python*
- evmone - *C++*
- ethereumjs-vm - *JavaScript*
- eEVM - *C++*
- Hyperledger Burrow - *Go*

延伸阅读 {#further-reading}

- 以太坊黄皮书
- Jellopaper aka KEVM：K 中的 EVM 语法
- The Beigepaper
- 以太坊虚拟机操作码
- Solidity 文档的简短介绍

相关主题 {#related-topics}

- Gas

---

title: 去中心化应用程序开发框架
description: 探索框架的优势并比较现有的选项。
lang: zh

sidebar: true

框架介绍 {#introduction-to-frameworks}

构建一个完整的去中心化应用程序需要不同的技术。 软件框架包括许多需要的功能，或提供简单的插件系统来选择您需要的工具。

这些框架带有很多非常规的功能，比如：

- 编一个本地区块链的程序功能。
- 编辑和测试您智能合约的实用工具。
- 客户开发附加功能，在同一个项目、仓库中来搭建您面向客户的应用程序。
- 无论是在本地运行的实例，还是在以太坊的公共网络之一，可以连接到以太网并且部署合约的配置。
- 去中心化的应用程序分布——与类似星际文件系统（一种用于存储和访问文件、网站、应用程序和数据的分布式系统）的存储选项一体化。

前置要求 {#prerequisites}

在更深入介绍这个框架之前，我们推荐您先阅读下面对于去中心化应用程序的简介以及以太坊堆栈。

可用的框架 {#available-frameworks}

Truffle - *开发环境、测试框架、部署通道及其他工具。*

- trufflesuite.com
- GitHub

安全帽 - *专业人员的以太坊开发环境。*

- hardhat.org
- GitHub

Ape - *适用于 Pythonista、数据科学家和安全专家的智能合约开发工具.*

- 相关文档
- GitHub

Brownie - *基于 Python 的开发环境和测试框架。*

- 相关文档
- GitHub

Embark - *开发环境、测试框架以及与以太坊、星际文件系统和 Whisper 集成的其他工具。*

- 相关文档
- GitHub

Web3j - *在 JVM 上开发区块链应用程序的平台。*

- 主页

- 相关文档
- GitHub

OpenZeppelin SDK - ***终极智能合约工具包：一套帮助您开发、编译、升级、部署智能合约并与之交互的工具。***

- OpenZepelin SDK
- GitHub
- 社区论坛

创建 Eth App - ***使用一个命令创建以太空驱动的应用程序。 使用范围广泛的 UI 框架和 Defi 模板来选择。***

- GitHub
- 模板

Scaffold-Eth - ***Ethers.js + 安全帽 + React 组件和 web3 钩子函数：这是您开始构建由智能合约驱动的去中心化应用程序所需要的一切。***

- GitHub

The Graph - ***高效查询区块链数据的 Graph。***

- 网站
- 使用教程

Alchemy - ***以太坊开发平台。***

- alchemy.com
- GitHub
- Discord

Dapptools - ***一套专注于以太坊的 CLI 工具，遵循 Unix 设计理念，倾向于可组合、可配置和可扩展性。***

- 主页
- GitHub

Foundry - ***一种快速便携式模块化工具包，用于开发以 Rust 语言编写的以太坊应用。***

- 相关文档
- GitHub
- Foundry 工具

延伸阅读 {#further-reading}

还有哪些社区资源对您有所帮助？ 编辑并添加本页面！

相关主题 {#related-topics}

- 构建您的本地开发环境

---

title: Gas 和费用
description:
lang: zh

sidebar: true

Gas 对以太坊网络至关重要。 正是这种燃料使它能够运行，正如车辆需要汽油一样。

前置要求 {#prerequisites}

为了更好地理解此页面，推荐先阅读交易和 EVM。

什么是 Gas？ {#what-is-gas}

Gas 是指在以太坊网络上执行特定操作所需的计算工作量。

由于每笔以太坊交易都需要计算资源才能执行，每笔交易都需要付费。 在这个方面上，Gas 是指在以太坊成功进行交易所需的费用。

显示 EVM 操作所需 Gas 的图表 *图表来自 Ethereum EVM illustrated*

本质上，Gas 费用是以太坊的货币以太 (ETH) 支付的。 Gas 价格以 Gwei 标明，Gwei 本身就是 ETH 的一个单位——每个 Gwei 等于 0.000000001 ETH (10^{-9} ETH)。 例如，您可以说您的 Gas 成本为 1 Gwei，而不是说您的 Gas 成本为 0.000000001 以太。 “gwei”一词本身表示“giga-wei”，等于 100,000,000 wei。 Wei 本身（以 b-money 的发明者 Wei Dai 命名）是 ETH 中最小的单位。

伦敦升级之前 {#pre-london}

以太坊网络交易费的计算方式在 2021 年 8 月的伦敦升级后发生了改变。 以下回顾了过去的工作方式：

假设 Alice 需要支付 1 ETH 给 Bob。 在交易中，gas 限额为 21,000 单位，gas 的价格是 200 gwei。

总费用为: `Gas 单位（限额） * Gas 单价` 例如 `21,000 * 200 = 4,200,000 gwei` 或者 0.0042 ETH

当 Alice 汇款时，将从 Alice 账户中扣除 1.0042 ETH。 Bob 将获得 1.0000 ETH。 矿工将得到 0.0042 ETH。

这个视频简要概述了 Gas 及其存在原因：

伦敦升级之后 {#post-london}

伦敦升级 于 2021 年 8 月 5 日 实施，目的是通过彻底改造以太坊的交易费用机制，使用户更容易预测以太坊的交易费用。 这一升级带来的更高一层的收益包括更好地估算交易费，通常会加快纳入交易，并通过燃烧一定比例的交易费来抵消 ETH 的发行。

从伦敦网络升级开始，每个区块都有基本费用，由网络根据区块空间需求来计算每单位 gas 最低价格。 由于交易费的基本费用会被燃烧掉，因此用户还要在其交易中设定一个小费（优先费）。 小费用于满足矿工执行和广播包含用户交易的区块，预计将由大多数钱包自动设置。

交易总费用的计算用如下所示：`Gas 单位（限额）*（基本费用 + 小费）`

假设 Jordan 需要向 Taylor 支付 1 ETH。 在交易中，矿工报酬限额为 21,000 单位，基本费用的价格是 100 gwei。 Jordan 支付了 10 gwei 作为小费。

使用上面的公式，我们可以计算 `21,000 * (100 + 10) = 2,310,000 gwei` 或 0.00231 ETH。

当 Jordan 发送钱时，将从 Jordan 账户中扣除 1.00231 ETH。 Taylor 将获得 1.0000 ETH。 矿工得到 0.00021 ETH。 0.0021 ETH 的基本费用被燃烧。

此外，Jordan 还可以为交易设定最高费用 (`maxFeePergas`)。 最高费用与实际收费之间的差额将归还给 Jordan。如：`退款 = 最高费用 -（基本费用 + 优先费）`。 Jordan 可以为执行交易费用设定一个最高金额，而不必担心在执行交易时“超额”支付基本费用。

区块大小 {#block-size}

在伦敦升级之前，以太坊有固定大小的区块。 在网络需求很高的时期，这些区块满负荷运行。 因此，用户常常不得不等待大量需求减少后才能被列入一个区块，这导致用户体验很差。

伦敦升级在以太坊中引入了大小可变区块。 每个区块的目标大小为 1500 万 gas，但区块的大小将根据网络需求增减。因此区块限制最多不超过 3000 万 gas（目标区块大小的 2 倍）。 该协议通过 *tâtonnement* 的过程使区块大小平均达到 1,500 万。 这意味着如果区块大小超出目标区块大小，协议将增加以下区块的基本费用。 同样，如果区块大小低于目标区块大小，协议将减少基本费用。 基本费用的调整金额与当前区块大小和目标区块大小的差距成比例。 关于区块的更多信息。

基本费用 {#base-fee}

每个区块都有作为储备价格的基本费用。 要想有资格被列入区块，gas 费用报价必须至少等于基本费用。 基本费用独立于当前区块计算，是由区块之前的区块决定的，这使得用户更容易预测交易费用。 当区块被开采时，此基本费用将被“燃烧掉”，从循环中移除。

基本费用的计算公式是将上一个区块的大小（所有交易中使用的 gas 数量）与目标大小进行比较。 如果超过目标区块大小，区块的基本费用将最多增加 12.5%。 这种指数级增长使得区块大小无限期保持高位在经济上不可行。

区块编号	已包含 Gas	费用增加	当前基本费用
1	15M	0%	100 gwei
2	30M	0%	100 gwei
3	30M	12.5%	112.5 gwei
4	30M	12.5%	126.6 gwei
5	30M	12.5%	142.4 gwei
6	30M	12.5%	160.2 gwei
7	30M	12.5%	180.2 gwei
8	30M	12.5%	202.7 gwei

相对于伦敦升级之前的 gas 拍卖市场，这种交易费机制的变化使费用更容易预测。 根据以上表格，在 9 号区块创建交易，钱包会让用户确切了解，要添加到下一个区块的**最大基本费用**等于 `当前基本费用 * 112.5%`，即 `202.8 gwei * 112.5% = 228.1 gwei`。

还请注意，由于处理完整块的基本费用增加的速度，我们不太可能看到完整块的延长峰值。

区块编号	已包含 Gas	费用增加	当前基本费用
30	30M	12.5%	2705.6 gwei
...	...	12.5%	...
50	30M	12.5%	28531.3 gwei
...	...	12.5%	...
100	30M	12.5%	10302608.6 gwei

优先费（小费） {#priority-fee}

在伦敦升级之前，矿工将获得区块中所含交易的总 gas 费用。

随着新的基本费被燃烧，伦敦升级引入了优先费（小费），以激励矿工将交易纳入区块。 如果没有小费，矿工会发现开采空区块在经济上可行，因为他们会获得相同的区块奖励。 在常规情况下，一笔金额不大的小费给矿工提供了包含该交易的最低激励。 对于需要在同一区块中优先执行的交易，需要更高的小费来试图出价高于竞争交易。

最高费用 {#maxfee}

要在网络上执行交易，用户可以为他们愿意支付的交易执行费指定最高限额。 此可选参数称为 `maxFeePergas`。 为了执行交易，最高费用必须超过基本费用和小费的总和。 会为交易发送人退还最高费用与基本费用和小费总和之间的差额。

计算费用 {#calculating-fees}

伦敦升级带来的主要好处之一是提高用户在确定交易费用时的体验。 对于支持升级的钱包，钱包提供商将自动设置推荐的交易费（基本费用 + 推荐优先费），而不是明确说明您愿意支付多少费用来完成交易，以便降低用户的复杂程度。

EIP-1559 {#eip-1559}

在伦敦升级中执行 EIP-1559 使得交易费机制比以前的 gas 竞价招标更加复杂，但优点是提高 gas 费用的可预测性，从而使交易费市场更加有效。 用户可以在提交交易时设定 `maxFeePergas`，表示他们愿意为执行交易支付多少费用，同时清楚该数额不会超过 gas 的市场价格（`BaseFeePergas`），并且获得减去小费后的剩余退款。

这个视频解释了 EIP-1559 及其带来的好处：

如果感兴趣，您可以阅读 EIP-1559 规范。

更深层次的见解请参见 EIP-1559 资源。

为什么存在 Gas 费用？ {#why-do-gas-fees-exist}

简而言之，Gas 费用有助于确保以太坊网络的安全。 在网络上执行的每次计算都需要收费，这样可以防止参与者对网络造成垃圾信息。 为了防止代码中无意、恶意的无限循环或其他计算浪费，要求每个事务对代码可以执行的计算步骤设置一个限制。 基本计算单位是“gas”。

尽管交易中包含费用限制，但交易中未使用的 gas 将退还给用户（如：返还 `最高费用 -（基本费用 + 小费）`）。

未使用 Gas 退款情况图 *图表来自 Ethereum EVM 插图*

什么是 gas 限额？ {#what-is-gas-limit}

Gas 限额是指您愿意在交易中消耗的最大 gas 数量。 涉及智能合约的更复杂交易需要更多的计算工作，因此相比较简单的支付，它们需要更高的 gas 限额。 标准的 ETH 转账一般要求 gas 限额为 21,000 单位。

例如，如果您对简单的 ETH 转账设置 50,000 gas 限额。EVM 将消耗 21,000，您将收到剩余的 29,000。 然而，如果您设置的 gas 太少，比如说，对于简单的 ETH 转账，gas 限额为 20,000。EVM 将消耗您 20,000 gas 试图实现交易，但不会完成。 然后，EVM 会恢复所有变化，但由于矿工已经完成了价值 20k gas 的工作，所以 gas 被消耗掉了。

为什么 gas 费会变得如此高？ {#why-can-gas-fees-get-so-high}

Gas 费高是由于以太坊广受欢迎。 在以太坊进行任何操作都需要消耗 gas，每个区块的 gas 空间有限。 这包括计算、存储或操作数据，或转移代币，每种消耗不同数量的“gas”单位。 随着 dapp 功能变得更加复杂，智能合约执行的操作数量也会增加。即每个交易在有限大小区块内占用更多空间。 如果需求太大，用户必须提供更高的小费来尝试超越其他用户的出价。 小费更高即更有可能让你的交易进入下一区块。

光靠 gas 价格并不能实际决定我们必须对特定交易支付的金额。 为了计算交易费，我们必须将交易费所使用的 gas 乘以 gwei。

减少 gas 成本的举措 {#initiatives-to-reduce-gas-costs}

以太坊性能扩展的升级将会最终解决一些燃料成本问题，也会为以太坊平台处理每秒上千次的交易和全面升级做好准备。

第二层扩容是一项主要举措，可大大优化 gas 成本、用户体验和可扩展性。 关于第二层扩容的更多信息。

在信标链上引入的新验证模式应能降低高耗电量，减少对专门硬件的依赖。 这个链将使去中心化的以太坊网络能够达成共识并保持网络安全。同时通过代替要求财务承诺来限制能源消耗。

任何拥有至少 32 个 ETH 的人都可以抵押这些 ETH，成为负责处理交易、验证区块并提出要添加到链中的新区块的验证者。 少于 32 个 ETH 的用户可以加入权益质押池。

降低 gas 成本的策略 {#strategies-for-you-to-reduce-gas-costs}

如果您尝试降低 ETH 的 gas 成本，则可以设置小费来表明交易的优先级别。 矿工会“继续工作”并执行每 gas 提供小费更高的交易。因为他们可以保留您支付的小费，而不会倾向于执行小费较低的交易。

如果想要监控 gas 价格，用较少的费用发送 ETH，您可以使用多种不同的工具，例如：

- Etherscan *交易 gas 价格估算器*
- Blocknative ETH Gas Estimator *支持类型 0 传统交易和类型 2 EIP-1559 交易的 gas 估算 Chrome 插件。*
- ETH Gas Station *以太坊 gas 市场面向消费者的指标*

相关工具 {#related-tools}

- Bloxy Gas Analytics *以太坊网络 gas 统计数据*
- Blocknative’s Gas Platform *由 Blocknative 的全局内存池数据平台提供支持的 Gas 估算 API*

延伸阅读 {#further-reading}

- 以太坊 Gas 详解
- 以太坊价格上涨会导致使用费更贵吗？
- 减少智能合约的 gas 消耗
- 权益证明与工作量证明

相关主题 {#related-topics}

- 挖矿

title: 集成开发环境
description:
lang: zh

sidebar: true

当建立一个集成开发环境时，以太坊上的应用编程类似于任何其他软件项目编程。 这里有许多选项可供选择，最后，请选择一个最适合您偏好的集成开发环境或代码编辑器。 对您的以太坊开发来说，最好的集成开发环境很可能就是您在传统软件开发中使用过的集成开发环境。

基于网络的集成开发环境 {#web-based-ides}

如果您想在设置本地开发环境之前摆弄一下代码，这些网络应用是为以太坊智能合约开发定制的。

Remix - ***基于网页的集成开发环境，内置静态分析和区块链测试虚拟机。***

- remix.ethereum.org

EthFiddle - ***基于网页的集成开发环境，让您能编写、编译和调试智能合约。***

- ethfiddle.com
- Gitter

ChainIDE - ***一个基于云端的多链集成开发环境***

- chainide.com

Replit - ***一个可定制的以太坊开发环境，具有热重载、错误检查和一流测试网支持等功能。***

- Solidity 新手教程

桌面 IDE {#desktop-ides}

大多数成熟的集成开发环境都建立了插件来增强以太坊的开发体验。 至少，它们为智能合约语言提供语法高亮功能。

Visual Studio Code - ***以太坊官方支持的专业跨平台集成开发环境。***

- Visual Studio Code
- 以太坊区块链开发工具包
- Azure 区块链工作台
- 代码示例
- GitHub

Atom - ***21 世纪可破解的文本编辑器***

- Atom 编辑器
- GitHub
- 以太坊包

JetBrains 集成开发环境（IntelliJ IDEA 等） - ***软件开发者和团队的基本工具***

- JetBrains
- GitHub
- IntelliJ Solidity

Remix Desktop - ***在本地计算机上体验 Remix 集成开发环境***

- 下载
- GitHub

插件与扩展 {#plugins-extensions}

- solidity - 以太坊 Visual Studio Code 中的 Solidity 语言
- Prettier Solidity - 使用 prettier 的代码格式化工具

延伸阅读 {#further-reading}

还有哪些社区资源对您有所帮助？ 请编辑本页面并添加！

title: 以太坊开发文档
description: ethereum.org 开发者文档简介。
lang: zh

sidebar: true

本文档旨在帮助您构建以太坊。 它从概念层面，解释了以太坊的技术堆栈，并记录了以太坊更复杂的应用和使用案例。

基于开源社区的努力，您可以随时提出新的主题，添加新内容，并在您认为可能有用的地方提供示例。所有文档都可通过 GitHub 编辑 — 如果您不了解如何使用，请遵循这些说明。

开发单元 {#development-modules}

如果这是您首次尝试以太坊开发，我们建议从头开始，像读书一样奋斗学习。

基本主题 {#foundational-topics}

以太坊堆栈 {#ethereum-stack}

高级 {#advanced}

---

title: ETH 简介
description: 关于 ETH 加密货币的开发者指南
lang: zh

sidebar: true

前提条件 {#prerequisites}

为了更好地理解此页面，推荐先阅读：以太坊简介。

什么是加密货币？ {#what-is-a-cryptocurrency}

加密货币是一种基于分布式账本（区块链）的交换媒介。

交换媒介是指被广泛接受、可支付任何商品和服务的物品，而分布式账本是记录交易的数据存储系统。区块链技术允许用户在分布式账本上进行交易，而无需依赖受信任的第三方来维护账本。

第一个加密货币是由 Satoshi Nakamoto 创建的比特币。 自 2009 年比特币发行以来，人们已经在许多不同的区块链上制作了数以千计的加密货币。

什么是以太币 (ETH)？ {#what-is-ether}

以太币 (ETH) 是用于以太坊网络上许多事物的加密货币。 从根本上讲，它是唯一可接受的交易费用支付方式，而且合并后，还需用于在主网得到验证并提出区块。 以太币还被用作 DeFi 借贷市场的主要抵押形式，NFT 市场的主要记账单位以及提供服务、销售实体商品赚取的付款等。

以太坊允许开发者创建 **去中心化应用 (dapp)**，它们共享算力池。 这个共享池是有限的，因此以太坊需要一种机制来确定谁可以使用它。 否则，某个 dapp 可能会意外或恶意地消耗所有网络资源，从而导致其他应用程序无法访问算力池。

ETH 加密货币支持以太坊算力的定价机制。 当用户想要完成一笔交易时，他们必须支付以太币，使他们的交易被区块链识别。 这些使用成本被称为 gas 费用，gas 费用的多少取决于执行交易所需的算力和全网当时的算力需求。

因此，即使某恶意 dapp 无限循环提交，交易最终也会耗尽 ETH 并终止，从而使网络恢复正常。

人们经常 混 淆以太坊和以太币 (ETH) - 当他们谈及“以太坊价格”时，其实是指 ETH 价格。

铸造 ETH {#minting-ether}

铸造是指在以太坊分类账上创造新以太币的过程。 底层以太坊协议创造出新以太币，单一用户不可能创造。

当矿工在以太坊区块链上创建出一个区块时，就会铸造出新以太币。 作为对矿工的激励，以太坊协议在每个区块中都设置了奖励，增加区块矿工设置的地址余额。 区块奖励随时间而变化，目前奖励为每区块 2 个以太币。

燃烧 ETH {#burning-ether}

除了通过区块奖励创造 ETH，也可以通过被称为“燃烧”的过程销毁 ETH。 当 ETH 被燃烧掉，它也就永久退出流通。

以太坊上的每一笔交易都会发生 ETH 燃烧。 当用户支付交易费用时，基本 gas 费将根据协议被销毁。 根据网络需求，某些区块燃烧的 ETH 超出铸造的 ETH。

ETH 面额 {#denominations}

由于以太坊上许多交易规模较小，ETH 有一些面额单位表示较小金额。 在这些单位中，Wei 与 Gwei 特别重要。

Wei 是最小的 ETH 单位，因此在以太坊黄皮书等众多技术实施中，都以 Wei 为基础进行运算。

Gwei（giga-wei 的缩写），常用于描述以太坊的 gas 费用。

面额	ETH 值	常见用法
Wei	10^{-18}	技术实施
Gwei	10^{-9}	可读 gas 费用

传输 ETH {#transferring-ether}

以太坊上的每一笔交易都包含一个 `值` 域，指定了要从发送者地址发送到接收者地址的 ETH 转移数量（以 Wei 为单位）。

当接收者地址是智能合约时，当智能合约执行其代码时，这些转移的以太币可用于支付 gas 费用。

关于交易的更多信息

查询 ETH {#querying-ether}

用户可以通过检查帐户 `余额` 字段来查询任何帐户的 ETH 余额，显示以 wei 为单位的 ETH 持有面额。

Etherscan 是常用于检查地址余额的基于 web 的应用程序。 例如，此 Etherscan 页面显示以太坊基金会的余额。

延伸阅读 {#further-reading}

- 定义 ETH 和以太坊 – *CME Group*
- 以太坊白皮书：以太坊原始提案 这份资料包括了对 ETH 及其创建动机的整体描述。

还有哪些社区资源对您有所帮助？ 请编辑本页面并添加！

title: 以太坊基金会与社区资助方案
description: 以太坊生态系统的资助方案清单。
sidebar: true

lang: zh

以太坊资助 {#ethereum-grants}

下面列出的方案为向致力于促进以太坊生态系统成功和发展的项目提供各种资金支持。 根据本指南来寻求和申请资金，助力您的以太坊项目成功。

这个列表由我们的社区管理。 如果有内容缺失或错误，请编辑此页面！

庞大的以太坊生态系统 {#broad-ethereum-ecosystem}

这些计划通过向大量项目提供资助来支持庞大的以太坊生态系统。 其中包括可扩容性、社区建设、安全、隐私等解决方案。 这些资助并非专门针对任何一个以太坊平台，如果不确定，可以从这里开始。

- 以太坊基金会生态系统支持计划 - *资助有利于以太坊的开源项目，重点关注通用工具、基础设施、研究和公共物品*
    - 学术资助行动 - *以太坊基金会正在赞助一波与以太坊相关的学术工作*
- 以太坊意见征求 - *以太坊基金会对以太坊生态系统中的工作和项目提出的建议要求*
- MetaCartel - *去中心化应用程序开发，去中心化自治组织创建*
  资助从 1 千美元到 8 千美元不等。
- Moloch DAO - *隐私、第二层扩容、客户端安全性等*
- 开放资助
- DAO 资助 - *提供赠款的组织的 Google 电子表单*

特定项目 {#project-specific}

这些项目为旨在开发和实验自己技术的项目建立了自己的资助。

- Aave 资助计划 – *Aave赠款 DAO*
- Balancer – *Balancer 生态系统基金*
- Compound 补助计划 – *Compound 金融生态系统*
- Consensys 资助计划 – *Consensys 区块链融资& 以太坊赠款*

- dYdX 资助计划 – *dYdX 去中心化交易所*
- Lido Ecosystem 生态系统资助组织 (LEGO) – *Lido 金融生态系统*
- mStable 资助计划 - *mStable 社区*
- The Graph – *The Graph 生态*
- Uniswap 资助计划 – *Uniswap 社区*

二次方融资 {#quadratic-funding}

以太坊的开源属性催生了一种有趣的新型募资模式：二次方融资。 这有可能改善我们在未来为各种公共产品募资的方式。 二次方募资确保获得最多资金的项目是那些具有最独特需求的项目。 换句话说，就是那些能够改善大多数人生活的项目。 关于二次方融资的更多信息。

- Gitcoin
- clr.fund

在以太坊工作 {#work-in-ethereum}

没有准备好启动您自己的项目？ 有数百家公司正在积极寻求热情人士的加入，为以太坊生态系统奉献力量。 正在寻找更多信息？ 点击查看更多以太坊相关工作

title: 以太坊简介
description: 面向 dapp 开发者介绍以太坊的核心概念。
lang: zh

sidebar: true

什么是区块链？ {#what-is-a-blockchain}

对区块链最好的描述是将其描述为一个公共数据库，它由网络中的许多计算机更新和共享。

"区块"指的是数据和状态是按顺序批量或"区块"存储的。 如果你向别人发送 ETH，需要将交易数据添加到一个区块中才算成功。

"链"指的是每个区块加密引用其父块。 换句话说，区块被链接在一起。 在不改变所有后续区块的情况下，区块内数据是无法改变，但改变后续区块需要整个网络的共识。

网络中的每台计算机都必须就每个新区块和链达成一致。 这些计算机被称为“节点”。 节点保证所有与区块链交互的人都有相同的数据。 要完成此分布式协议，区块链需要一个共识机制。

与比特币一样，以太坊目前使用工作量证明 (PoW) 作为共识机制。 这意味着，任何人想要在链上添加新的区块，都必须解决一个困难的难题，该难题需要大量的算力来解决。 解决了这个难题，就 "证明"你已经花费了算力资源。 这种操作被称为挖矿。 挖矿通常充满了粗暴的尝试和错误，但是成功添加区块会得到 ETH 奖励。

新的区块会被广播到网络中的节点，经过检查和验证，同步更新状态。

综上所述，当你向别人发送 ETH 时，这笔交易必须被打包并包含在一个新的区块中。 然后将更新后的状态与整个网络共享。

观看 Austin 带你了解区块链：

如果想查看区块链如何压缩数据，以及以前的区块如何引用所有过去的区块，请务必查看 Anders Brownworth 的演示并观看以下随附视频。

观看 Anders 介绍区块链中的哈希：

以太坊简介 {#what-is-ethereum}

在以太坊网络中，有一台规范化计算机（称为以太坊虚拟机，或 EVM），其状态得到以太坊网络中所有人的一致同意。 每个参与以太坊网络的人（每个以太坊节点）都会保存一份这台计算机的状态。 此外，任何参与者都可以广播请求这台计算机进行任意计算。 每当广播这样的请求网络时，网络上的其他参与者就会验证、确认并进行（“执行”）计算。 这个命令会导致 EVM 的状态变化，并且在整个网络中传播。

计算请求被称为交易请求；所有交易的记录以及 EVM 的当前状态都存储在区块链中，而区块链又由所有节点存储并达成一致。

加密机制确保了一旦交易被验证为有效并添加到区块链中后，之后就不会被篡改。 同样的机制还确保所有交易都以适当的“权限”签名和执行（除了 Alice 本人，任何人都不能从 Alice 的账户发送数字资产）。

什么是以太币（ether）？ {#what-is-ether}

以太币 (ETH) 是以太坊上的的原生加密货币。 为什么要有以太币的存在？目的是为了允许算力市场的存在。 这种市场为参与者提供了一种经济激励，以验证并执行交易请求，为网络提供计算资源。

任何广播交易请求的参与者也必须向网络提供一定数量的以太币作为奖励。 这笔奖金将颁发给最终验证交易、执行交易并将其提交到区块链，并在网络中广播的任何人。

支付的 ETH 数量对应于进行计算所需的时间。 这类奖励也可以防止恶意参与者通过请求执行无限循环或资源密集型脚本来故意堵塞网络，因为这些参与者将为自己的计算时间付费。

什么是智能合约？ {#what-are-smart-contracts}

实际上，参与者不会每当在 EVM 上请求计算时都编写新代码。 而是应用程序开发者将程序（可重复使用的代码片段）上传到 EVM 存储中，然后用户通过不同的参数请求执行这些代码片段。 我们将这些上传至网络并由网络执行的程序称为智能合约。

简单来说，你可以把智能合约想象成一种自动售货机：通过特定参数调用脚本后，如果满足某些特定条件，就会执行一些操作或计算。 例如，如果调用者将以太币发送给特定的接收者，通过简单的卖方智能合约就可以创建和分配数字资产所有权。

任何开发者都可以创建智能合约并在网络上公开，并使用区块链作为其数据层，向网络支付费用。 然后，任何用户都可以调用智能合约来执行其代码，并再次向网络支付费用。

因此，通过智能合约，开发者可以任意构建和部署面向用户的复杂应用程序和服务，例如市场、金融工具、游戏等。

术语 {#terminology}

Blockchain 区块链 {#blockchain}

所有已在网络历史上提交给以太坊网络的区块的序列。 如此命名，是因为每个区块都包含对前一个区块的引用，这有助于我们在所有区块间（同时在精确的历史记录）上保持顺序。

ETH 以太币 {#eth}

以太坊的原生加密货币。 用户向其他用户支付以太币，使自己的代码执行请求得以满足。

ETH 的更多信息

EVM 以太坊虚拟机 {#evm}

以太坊虚拟机是一个全局虚拟计算机，以太坊网络每个参与者都会存储并同意其状态。 任何参与者都可以请求执行 EVM 上的任意代码；代码执行会改变 EVM 的状态。

关于 EVM 的更多信息

Nodes 节点 {#nodes}

存储 EVM 状态的实体机器。 节点间相互通讯传播关于 EVM 状态变化和新状态更改的信息。 任何用户还可以通过广播来自节点的代码执行请求来请求代码执行。 以太坊网络本身就是所有以太坊节点及其通信的集合。

关于节点的更多信息

帐户 {#accounts}

存储以太币之处。 用户可以初始化帐户，将以太币存入帐户，并将以太币转给其他用户。 帐户和帐户余额储存在 EVM 中的一个大表格中，是 EVM 总体状态的一部分。

关于账户的更多信息

交易 {#transactions}

“交易请求”是在 EVM 上执行代码请求的正式术语。“交易”是指已完成的交易请求和相关的 EVM 状态变化。 任何用户都可以从节点向网络广播交易请求。 为了使交易请求影响 EVM 的一致状态，就必须由其他节点对其进行验证、执行和“提交到网络”。 执行任何代码都会导致 EVM 的状态更改；一旦提交后，该状态更改将广播到网络中的所有节点。 以下为一些交易示例：

- 从我的帐户发送 X 个 ETH 到 Alice 的帐户。
- 将一些智能合约代码发布到 EVM 内存中。
- 使用 Y 参数执行 EVM 中 X 地址的智能合约代码。

关于交易的更多信息

区块 {#blocks}

交易量巨大，因此交易分批或分区块“提交”。 区块通常包含数十至数百笔交易。

关于区块的更多信息

智能合约 {#smart-contracts}

是开发者发布在 EVM 内存中的可重用代码片段（程序）。 任何人都可以通过提出“交易请求”来请求执行智能合约代码。 因为开发人员可以通过发布智能合约将任意可执行应用程序写入 EVM（游戏，市场，金融工具等），所以这些通常也称为 dapp 或去中心化应用。

了解关于智能合约的更多信息

延伸阅读 {#further-reading}

- 以太坊白皮书
- 所以以太坊到底如何工作？ - *Preethi Kasireddy*

还有哪些社区资源对您有所帮助？ 请编辑本页面并添加！

相关教程 {#related-tutorials}

- 一个开发者的以太坊指南，第一部分 -*一个初学者使用 Python 和 web3.py 方式非常友好地探索以太坊的方式。*

---

title: 最大可提取价值 (MEV)
description: 最大可提取价值 (MEV) 简介
lang: zh

sidebar: true

最大可提取价值 (MEV) 是指通过包含、排除和更改一个区块中的交易顺序，可以从超过标准区块奖励和 gas 费用的区块生产中提取的最大值。

矿工的可抽取的价值

这个概念最初是在 工作量证明 的背景下应用的，最初称为“矿工的可抽取价值”。 这是因为在工作量证明中，矿工掌握了交易的包含、排除和顺序。 然而，在通过The Merge 转换为权益证明之后，验证者将负责这些角色，并且挖矿将不再适用。 此处的值提取方法在此转换后仍将保留，因此需要更改名称。 为了在保持相同基本含义的同时保持相同的连续性首字母缩写词，现在使用“最大可提取价值”作为更具包容性的替代品。

前提条件 {#prerequisites}

请确保您熟悉 交易, 区块, gas, 和 挖矿 熟悉 dapps 和 DeFi 也很有帮助。

MEV 提取 {#mev-extraction}

理论上讲，矿工可提取价值的累积完全取决于矿工，因为矿工是唯一能保证可获利的挖矿行为得到执行的一方（至少是在当前的工作量证明链中是这样 — 这将在合并后发生改变）。 但实际上，大部分 MEV 是由称为“搜索人”的独立网络参与者提取的。 搜索人在区块链数据上运行复杂的算法来检测盈利的 MEV 机会，并且有机器人自动将这些盈利交易提交到网络。

矿工确实能够获得全额 MEV 的一部分，因为搜索人愿意支付高昂的 gas 费用（向矿工支付），以换取将他们的盈利交易纳入区块的更高机会。 假定搜索人在经济上是合理的。 搜索人愿意支付的 gas 费将是 MEV 的 100% 的金额(因为如果 gas 费更高， 搜索人将亏钱)。

在这种情况下，对于一些高度竞争的 MEV 机会，例如 DEX arbitrage, 搜索者可能必须向矿工支付 90% 甚至更多的 MEV 总收入，因为很多人同时想运行同样的套利交易。 这是因为，确保套利交易运行的唯一方法是提交最高 gas 费用的交易。

Gas 高尔夫 {#mev-extraction-gas-golfing}

这种动态使得 “gas 高尔夫”——编程交易——能够使用最少数量的 gas——这成为一种竞争优势。 因为它允许搜索人设置较高的 gas 价格，同时保持总 gas 费不变(因为使用 gas 费 = gas 价格* gas 用量)。

一些著名的 gas 高尔夫技术包括：使用用长串零开头的地址(如：0x000000000000C521824EaFf97Eac7B73B084ef9306)，因为他们的需要的存储空间较少 (因而 gas 也减少)； 并留下很小 ERC-20 令牌余额在合约中， 因为相比于更新储存插槽，初始化存储槽需要更多的 gas (余额为 0 时)。 寻找如何更多的减少 gas 使用是搜索人在积极研究的一个领域。

通用的领跑者 {#mev-extraction-generalized-frontrunners}

一些搜索人并没有编写复杂的算法来检测盈利的 MEV 机会，而是运行通用的领跑者。 通用的领跑者是监控内存池以检测盈利交易的机器人。 领跑者将复制潜在的盈利交易代码，用领跑者的地址替换其地址。然后在本地执行交易，重复检查修改后的交易是否给领跑者地址带来利润。 如果交易确实有利可图，领跑者将以更替地址和更高的 gas 价格提交修改后的交易。 “领跑”原始交易并获取原始搜索人的 MEV。

Flashbots {#mev-extraction-flashbots}

Flashbots 是一个独立的项目，它将 go-ethereum 客户端扩展，提供的服务使搜索人能够向矿工提交 MEV 交易，而不向公共内存池披露。 这就防止了交易被通用领跑者领跑。

截至本文撰写时，MEV 交易的很大一部分是通过 Flashbots 来进行的，这意味着通用的领跑人不如过去那样有效。

MEV 相关案例 {#mev-examples}

MEV 以几种方式出现在区块链上。

DEX 套利 {#mev-examples-dex-arbitrage}

去中心化交易所 (DEX) 套利是最简单和最著名的 MEV 机会。 因此，它也是最具竞争性的。

它的作用就像这样：如果两种 DEXes 以两种不同的价格提供一个通证。 有人可以在较低价格的 DEX 上购买通证，然后在交易中以较高价格的 DEX 出售它。 多亏了区块链的机制，这是真实的，无风险的套利。

这是一个有利可图的套利交易的示例, 在这个交易中，一个搜索人将 1, 000 ETH 变成了 1,045 ETH，它利用 ETH/DAI 交易对在 Uniswap 和 Sushiswap 的不同价格。

清算 {#mev-examples-liquidations}

贷款协议的清算提供了另一个众所周知的 MEV 机会。

通过要求用户存放某种抵押品 (如 ETH) 的方式出借协议，如 Maker 和 Aave 的功能。 然后，用户可以根据需要从其他人处借入不同的资产和通证（例如，如果他们想对 MakerDAO 治理方案进行投票，他们可以借入 MKR，如果他们想在 Sushis Swap 上赚取一部分交易费用，他们可以借入 SUSHI），最高可借入其存放抵押品的一定金额-例如，30%（准确的借用功率百分比由协议确定）。 在这种情况下，他们从其他通证借出的用户充当出借人。

随着借款人抵押品价值的波动，他们的借款能力也会波动。 如果由于市场波动，借入资产的价值超过其抵押品价值的 30%（同样，准确的百分比由协议确定，协议通常允许任何人清算抵押品，立即偿还贷款人（这类似于传统金融中的 追加保证金通知）。 如果清算，借款人通常必须支付大笔清算费， 其中有些是流向变现人的——这是多 MEV 机会出现的地方。

搜索人竞相以最快的速度解析区块链数据，以确定哪些借款人可以被清算，并成为第一个提交清算交易并自行收取清算费的人。

夹心交易 {#mev-examples-sandwich-trading}

夹心交易是另外一种 MEV 提取的常用方法。

为了实现夹心交易，搜索人会监视内存池内 DEX 的大额交易。 例如，有人想要在 Uniswap 上使用 DAI 购买 10,000 UNI。 这种规模的交易将对 UNI/DAI 对产生有意义的影响，可能会显着提高 UNI 相对于 DAI 的价格。

搜索者可以计算该大额交易对 UNI/DAI 对的大致价格影响，并在大额交易之前**立即执行最优买单，低价买入 UNI，然后执行 在大额交易之后**立即卖出一个卖单，以大额订单导致的更高价格卖出。

然后，夹心交易风险更高，因为它不是原子交易（不像上文所述的 DEX 套利），而且容易出现 salmonella 攻击

NFT MEV {#mev-examples-nfts}

NFT 领域的 MEV 是一种新兴现象，而且不一定能赚钱。

然而，由于 NFT 交易发生在所有其他以太坊交易共享的同一个区块链上，搜寻者也可以在 NFT 市场上使用与传统 MEV 机会类似的技术。

例如，如果有一个流行的 NFT 下降，并且搜索者想要某个 NFT 或一组 NFT，他们可以写一个交易，使他们成为第一个排队购买 NFT 的人，或者他们可以在一个交易中购买整个 NFT 组合。 或者，如果一个 NFT 被错误地以低价挂出，搜寻者就可以抢在其他购买者前面，低价抢购。

NFT MEV 的一个显著例子发生在一个搜寻者花费 700 万美元来购买价格底线的每一个 Cryptopunk。 一位区块链研究员在 Twitter 上解释了买家是如何与 MEV 供应商合作以保持其购买的秘密。

长尾 {#mev-examples-long-tail}

DEX 套利、清算和三明治交易都是非常知名的 MEV 机会，对于新的搜寻者来说不太可能获利。 然而，还有一长串鲜为人知的 MEV 机会（NFT MEV 可以说是这样一个机会）。

刚刚起步的搜索者可能会通过在这个长尾搜索 MEV 而找到更多的成功。 Flashbot 的MEV 招聘板列出了一些新兴的机会。

MEV 的影响 {#effects-of-mev}

MEV 并不都是坏事 - 以太坊的 MEV 既有积极的作用，也有消极的影响。

优点 {#effects-of-mev-the-good}

许多 DeFi 项目依靠经济上的理性行为者，来确保其协议的有用性和稳定性。 例如，DEX 套利确保用户为他们的代币获得最好、最正确的价格，而借贷协议在借款人低于抵押率时依靠快速清算来确保贷款人得到回报。

如果没有理性的搜索者寻求和修复经济上的低效率，并利用协议的经济激励，DeFi 协议和一般的 dapps 可能不会像今天这样强大。

缺点 {#effects-of-mev-the-bad}

在应用层，某些形式的 MEV，如夹心交易，会导致用户的体验明显变差。 被夹在中间的用户面临更高的滑点和更差的交易执行。

在网络层，一般的抢跑者和他们经常参与的矿工费拍卖（当两个或更多的先行者通过逐步提高自己交易的矿工费，从而使他们的交易被打包到下一个区块），导致网络拥堵和试图运行正常交易的其他人的高矿工费。

除了区块*内*发生的，MEV 也可能会在区块*间*产生有害的影响。 如果一个区块中可用的 MEV 大大超过了标准区块的奖励，矿工可能会被激励重新开采区块并为自己捕获 MEV，导致区块链的重新组织和共识的不稳定。

这种区块链重新组织的可能性已经在以前的比特币区块链上探索过。 随着比特币的区块奖励减半，交易费用占区块奖励的比例越来越大，于是出现了这样的情况：矿工放弃下一个区块的奖励，而用更高的费用重新开采过去的区块，这在经济上变得很合理。 随着 MEV 的发展，以太坊也可能出现同样的情况，威胁到区块链的完整性。

MEV 的状况 {#state-of-mev}

2021 年初，MEV 开采量剧增，导致今年前几个月的矿工费价格极高。 Flashbots 的 MEV 中继的出现，降低了普通抢跑者的效力，并将矿工费价格拍卖带出链外，降低了普通用户的矿工费。

虽然许多搜寻者仍在从 MEV 中赚取丰厚的利润，但随着机会变得更加知名，越来越多的搜寻者竞争相同的机会，矿工将捕获越来越多的 MEV 总收入（因为上文最初描述的那种矿工费拍卖也发生在 Flashbots 中，尽管是私下的，矿工将捕获由此产生的矿工费）。 MEV 也不是以太坊独有的，随着以太坊上的机会变得越来越有竞争力，搜索者正在转移到其他区块链，如 Binance Smart Chain，那里存在与以太坊上类似的 MEV 机会，但竞争更少。

随着 DeFi 的发展和普及，MEV 可能很快就会大大超过以太坊区块的基础奖励。 随之而来的是，自私的区块链和共识不稳定的可能性越来越大。 一些人认为这是对以太坊的生存威胁，而抑制自私挖矿是以太坊协议理论的一个积极研究领域。 目前正在探索的一个解决方案是 MEV 奖励平滑。

相关资源 {#related-resources}

- Flashbots GitHub
- MEV-Explore *MEV 交易的仪表板和实时交易浏览器*

延伸阅读 {#further-reading}

- 什么是 MEV?
- MEV 和我
- 以太坊黑暗森林
- 逃离黑暗森林
- Flashbos：在 MEV 危机中抢跑
- @bertcmiller's MEV 评论

---

title: 网络
description: 以太坊网络概述以及在何处获得测试网络以太坊（ETH）来测试您的应用程序。
lang: zh

sidebar: true

由于以太坊是一种协议，因此这意味着可以有多个独立的“网络”与该协议兼容，并且彼此之间不会相互作用。

网络不同于以太坊环境，是用于开发，测试或生产方案。 你的以太坊帐户将在不同的网络上运行，但是你的帐户余额和交易历史记录不会从以太坊主网中继承下来。 出于测试目的，了解哪些网络可用以及如何获取测试网路的 ETH 是很有用的，以便你可以试用它。

前置要求 {#prerequisites}

你需要在浏览不同的网络之前了解以太坊的基础知识，因为测试网络将会提供一个便捷且安全的以太坊版本供你使用。 尝试访问以太坊简介。

公共网络 {#public-networks}

每个人都能通过互联网连接到公共网络。 任何人都可以在公共区块链上读取或创造交易，并且可以验证已经执行的交易。 交易协议以及网络状态由网络上的同行共同决定。

主网 {#mainnet}

主网是指主要的以太坊生态区块链，所有具有实际价值的交易都发生在该链的分散账本中。

大众和交易所涉及的 ETH 价格是主网的 ETH。

测试网 {#testnets}

除了主网外，还有公开的测试网。 这是一种模拟生态环境的网络，协议开发者或智能合约开发者可以使用它们测试尚未部署在主网上的协议升级和智能合约。 你可以把它看作生产与装配服务器的模拟。

在部署到主网上之前，测试你在测试网上编写的任何合约代码通常很重要。 如果你正在构建与现有智能合约集成的 dapp，大多数项目都会将副本部署到你可以与之交互的测试箱中。

大多数测试网使用权威证明的共识机制。 这意味着会选择少量节点来验证交易并创建新区块——在此过程中将他们的身份进行抵押。 很难在工作量证明测试网上激励挖矿，这会使它容易受到攻击。

Görli {#goerli}

一个能够在客户端之间起作用的验证测试网。

Kovan {#kovan}

运行 OpenEthereum 客户端的测试网的权威证明。

Rinkeby {#rinkeby}

运行 Geth 客户端的测试网的权威证明。

Ropsten {#ropsten}

工作量证明测试网。 这意味着它是以太坊最好的相似表示。

测试网水龙头 {#testnet-faucets}

测试网上的 ETH 没有实际价值；因此，测试网 ETH 没有市场。 因为你需要 ETH 与以太坊进行实际交互，所以大多数人都从水龙头获得测试网 ETH。 大多数水龙头是网络应用程序，你可以输入一个要求将 ETH 发送到的地址。

- Görli 水龙头
- Kovan 水龙头
- Rinkeby 水龙头
- Ropsten 水龙头

私有网络 {#private-networks}

如果以太坊网络的节点未连接到公共网络（即 主网或测试网），则以太坊网络是专用网络。 在这种情况下，专用仅指保留或隔离，而不是保护或安全。

开发网络 {#development-networks}

要开发以太坊应用程序，你需要在专用网络上运行它，以便在部署它之前了解它的工作方式。 与在计算机上创建用于 web 开发的本地服务器类似，你可以创建本地区块链实例来测试你的 dapp。 这允许比公共测试网更快的迭代。

有一些项目和工具专门协助这方面的工作。 进一步了解开发网络。

联盟网络 {#consortium-networks}

共识过程由一组预定义的受信任节点控制。 例如，一个由已知学术机构组成的私有网络，每个学术机构管理一个节点，并且块由网络中的签名者的阈值进行验证。

如果公共以太坊网络类似于公共互联网，那么你可以将联合体网络视为专用内部网。

相关工具 {#related-tools}

- Chainlist *EVM 网络列表，可以帮钱包和供应商连接到需要的链 ID 和网络 ID*
- 基于 EVM 的链 *给 Chainlist 提供链元数据的 GitHub 库*

延伸阅读 {#further-reading}

还有哪些社区资源对您有所帮助？ 编辑并添加本页面！

title: 节点和客户端
description: 以太坊节点和客户端软件的概述，以及如何设置节点和为什么您应该这样做。
lang: zh
sidebar: true

sidebarDepth: 2

以太坊是一个由计算机组成的分布式网络，运行可验证区块和交易数据的软件（称为节点）。 您需要一个被称为客户端的应用程序，才能在您的设备上“运行”一个节点。

前置要求 {#prerequisites}

在您更深层次介入和运行自己的以太坊客户端前，您应该理解点对点网络和 EVM 基础概念。 查看我们的以太坊简介。

什么是节点和客户端？ {#what-are-nodes-and-clients}

“节点”是指一种称为客户端的软件。 客户端是一种以太坊的实现，它可以验证每个区块中的所有交易，从而确保网络安全和数据准确。

通过查看此节点图，您可以查看以太坊网络的实时视图。

以太坊客户端存在多种编程语言（如 Go、Rust、JavaScript、Python、C# .NET 和 Java）的版本。 这些实现的共同之处是它们都遵循了正式规范（原为以太坊黄皮书）。 此规范决定了以太坊网络和区块链的功能。

执行客户端 以太坊客户端功能的简化图表。

节点类型 {#node-types}

如果您想 运行自己的节点，就应该明白不同类型的节点会消耗不同数据量。 事实上，客户端可以运行三种不同类型的节点——轻量、完整和归档。 也有不同同步策略的选项，可加快同步时间。 同步是指它能以多快的速度获取有关以太坊状态的最新信息。

完整节点 {#full-node}

- 存储完整的区块链数据。
- 参与区块验证，验证所有区块和状态。
- 所有状态都可以从一个完整的节点推出。
- 提供网络服务，并应要求提供数据。

轻节点 {#light-node}

- 存储头链并请求其他所有内容。
- 可以对照区块头中的状态根来验证数据的有效性。
- 对于低容量的设备，如嵌入式设备或移动电话来说是有用的，这些设备无法储存数千兆字节的区块链数据。

归档节点 {#archive-node}

- 存储保留在完整节点中的所有内容，并建立历史状态档案。 如果您想查询区块 #4,000,000 的帐户余额等内容，或者简单可靠地测试自己的交易集，而不使用 OpenEthereum 挖矿，则需要这样做。
- 这些数据以太字节为单位，这使存档节点对普通用户的吸引力降低，但对于诸如区块浏览器、钱包供应商和链分析之类的服务而言却很方便。

以存档以外的任何方式同步客户端将导致修剪的区块链数据。 这意味着，没有所有历史状态的档案，但是整个节点都可以按需构建它们。

为什么我要运行一个以太坊节点？ {#why-should-i-run-an-ethereum-node}

运行节点可以让您在支持生态系统的同时，不受信任地私下使用以太坊。

对您的好处 {#benefits-to-you}

运行自己的节点使您能够以真正私有、自给自足和不信任的方式使用以太坊。 您无需信任网络，因为您可以自己与客户端验证数据。 “不信任，验证”是流行的区块链口头禅。

- 您的节点根据一致性规则自己验证所有交易和区块。 这意味着您不必依赖网络中的任何其他节点或完全信任它们。
- 您不必将地址和余额泄露到随机节点。 一切都可以由您自己的客户检查。
- 如果您使用自己的节点，则去中心化应用可以更安全和私有。 MetaMask、MyEtherWallet 和其他钱包可以轻松指向您自己的本地节点。
- 您可以编程自己的自定义 RPC 端点。
- 您可以使用**进程间通信 (IPC)**连接到节点，或者重写节点，将您的程序作为插件加载。 这会降低延迟，您需要尽快替换您的交易（例如抢跑交易）。

如何通过您的应用程序和节点访问以太坊

网络优势 {#network-benefits}

多种节点对以太坊的健康、安全和运行弹性非常重要。

- 他们为依赖它的轻量级客户端提供对区块链数据的访问。 在使用高峰期，需要有足够的完整节点来帮助轻量节点同步。 轻量节点不会存储整个区块链，而是通过区块头中的状态根来验证数据。 如果有需要，他们可以要求区块提供更多信息。
- 完整的节点会强制执行工作量证明的共识规则，因此它们不会被欺骗来接受不遵循规则的代码块。 这为网络中提供了额外的安全性，因为如果所有节点都是轻型节点（不进行完全验证），则矿工可能会攻击网络，例如，创建具有更高奖励的区块。

如果您运行一个完整的节点，整个以太坊网络将从中受益。

运行您自己的节点 {#running-your-own-node}

是否有兴趣运行自己的以太坊客户端？ 学习如何 升级您自己的节点！

项目 {#projects}

选择客户端并按说明操作

ethnode - *运行一个以太坊节点（Geth 或 Parity）用于本地开发。*

- GitHub

** DAppNode - ****用于在专用机器上运行 Web3 节点（包括以太坊和信标链）的操作系统 GUI。**

- dappnode.io

资源 {#resources}

- 运行以太坊完整节点：完整指南 *Nov 7, 2019 - Justin Leroux*
- 节点配置备忘单 *Jan 5, 2019 - Afri Schoeden*
- 如何安装和运行一个 Geth 节点 *October 4, 2020 - Sahil Sen*
- 如何安装和运行一个 OpenEthereum(fka. Parity) 节点 *2020 - Sahil Sen*

替代方法 {#alternatives}

运行您自己的节点可能很难，您不必总是运行您自己的实例。 在这种情况下，您可以使用第三方 API 提供商，如 Infura、Alchemy 或 QuikNode。 另外，ArchiveNode 是一个社区资助的归档节点，希望将以太坊上的数据带给无法运行自己节点的独立开发者。 如果想了解使用这些服务的概况，请查看节点即服务。

如果有人在您的社区里运行一个包含公共 API 的以太坊节点，您可以通过自定义 RPC 将您的轻量级钱包（如 MetaMask）指向社区节点，并比随机受托的第三方隐私性更强。

另一方面，如果您运行一个客户端，则可以与可能需要它的朋友分享。

执行客户端（原“Eth1 客户端”） {#execution-clients}

以太坊社区维持多个开放源代码执行客户端（以前称为“Eth1 客户端”，或只称为“以太坊客户端”），由不同的团队使用不同的编程语言开发。 这使得该网络更强大、更多样化。 理想的目标是在没有任何客户端支配的情况下实现多样性，以减少任何单点故障。

本表概括了不同的客户端。 它们均已通过客户端测试，并积极维护以保持与网络升级同步。

客户端	语言	操作系统：	网络	同步策略	状态缓冲
Geth	Go	Linux, Windows, macOS	Mainnet、Görli、Rinkeby、Ropsten	快速，完整	Archive，Pruned
OpenEthereum	Rust	Linux，Windows，macOS	Mainnet、Kovan、Ropsten 及更多	Warp，完整	Archive, Pruned
Nethermind	C#, .NET	Linux，Windows，macOS	Mainnet、Gounderster、Ropsten、Rinkeby 及更多	快速，完整	Archive, Pruned
Besu	Java	Linux，Windows，macOS	Mainnet、Rinkeby、Ropsten 和 Görli	快速，完整	Archive, Pruned
Erigon	Go	Linux、Windows、macOS	Mainnet、Görli、Rinkeby、Ropsten	快速、完整	Archive、Pruned

更多关于支持网络的信息，请在以太坊网络中查阅。

不同实现方式的优势 {#advantages-of-different-implementations}

每个客户端都有独特的用例和优势，所以您应该根据自己的偏好选择。 多样性使得实现工作能够侧重于不同的特性和用户群。 您可能想要根据特性、支持、编程语言或许可证选择客户端。

Go Ethereum {#geth}

Go Ethereum（简称 Geth）是实现以太坊原始协议之一。 目前，它是使用最为广泛的客户端，拥有最大的用户群，为用户和开发者提供各种工具。 它用 Go 语言编写，完全开源，并由 GNU LGPL v3 授权许可。

OpenEthereum {#openethereum}

OpenEthereum 是一个快速、功能丰富、基于 CLI 的高级以太坊客户端。 它的诞生是为了给那些需要快速同步和最大运行时间的快速和可靠的服务提供必要的基础设施。 OpenEthereum 的目标是成为最快、最轻、最安全的以太坊客户端。 它提供了以下简洁的模块化代码：

- 易于自定义。
- 轻度融入服务或产品。
- 最小的内存和存储痕迹。

OpenEthereum 是使用最先进的 Rust 编程语言开发的，并由 GPLv3 授权许可。

注意，OpenEthereum 已被废弃并已停止维护。请谨慎使用，最好切换至其他客户端实现。

Nethermind {#nethermind}

Nethermind 是一个用 C# .NET 技术栈创建的以太坊协议，在包括 ARM 在内的所有主要平台上运行。它表现出了优越性能：

- 优化虚拟机
- 状态访问
- 网络和丰富的功能，如 Promethe/Graphana 仪表板、seq 企业日志支持、JSON RPC 跟踪和分析插件。

Nethermind 同样拥有详细文档，强大的开发支持，在线社区，并为高级用户提供全天候服务。

Besu {#besu}

Hyperledger Besu 是一个企业级以太坊客户端，用于公共和许可网络。 运行所有以太坊主网功能，从追踪到 GraphQL，有广泛的监控并得到 ConsenSys 支持，无论是在开放社区渠道中还是通过企业的商业 SLA。 使用 Java 编写，并由 Apache 2.0 授权许可。

Erigon {#erigon}

Erigon（前称 TurbohyGeth）是 Go Ethereum 的一个分叉，注重速度和磁盘空间效率。 Erigon 是一个完全重新设计的以太坊实现项目，目前用 Go 语言编写，但以其他语言实现。 Erigon 的目标是为以太坊提供更快、更模块化和更优化的实现。 它可以在 3 天内使用不到 2 TB 磁盘空间，执行完整归档节点同步。

同步模式 {#sync-modes}

为了关注和验证网络中的当前数据，以太坊客户端需要与最新的网络状态同步。 通过从其他人那里下载数据，用密码学方法验证其完整性，并建立一个本地区块链数据库来实现。

同步模式代表了这个过程的不同方法，并有不同的权衡。 客户端在实现同步算法方面也有所不同。 有关部署的具体细节，请参考您所选择的客户端的官方文件。

策略概览 {#overview-of-strategies}

主网客户端使用的同步方法概览：

完全同步 {#full-sync}

完整下载所有区块（包括区块头、交易和接收的数据），并通过执行每个区块来逐步生成区块链的状态。

- 通过验证每笔交易，最大限度地减少信任并实现最高安全性。
- 随着交易数量的增加，处理所有交易可能需要几天到几周时间。

快速同步

快速同步会下载所有区块（包括区块头、交易和接收的数据），验证所有区块头，下载区块状态并对区块头进行验证。

- 依赖共识机制的安全性。
- 同步只需要几个小时。

轻量同步

轻量客户端下载所有区块头、区块数据并对其进行随机验证。 仅从信任的检查点同步区块链信息。

- 仅获取最新状态，同时依赖于对开发者的信任和共识机制。
- 客户端已准备好在几分钟内使用当前网络状态。

关于轻量客户端的更多信息

快照同步

由 Geth 实现。 使用动态快照，用户可以检索所有帐户和存储数据，而不下载中间三角形节点，然后在本地重建 Merkle trie。

- Geth 开发的最快同步策略，为当前默认值
- 节省大量磁盘使用和网络带宽，同时不影响安全。

关于快照的更多信息

压缩同步

由 OpenEthereum 实现。 节点定期生成一个对共识至关重要的状态快照，任何节点都可以通过网络获取这些快照，从而能够从这个节点快速同步。

- OpenEthereum 的最快和默认同步模式依赖于节点提供的静态快照。
- 它和快照同步的策略类似，但没有确定的安全效益。

关于压缩的更多信息

Beam 同步

由 Nethermind 和 Trinity 实现。 可以快速同步，但也可以下载执行最新模块所需的数据，这允许您在启动后的最初几分钟内查询链上信息。

- 首先同步状态，并允许您在几分钟内查询 RPC。
- 后台同步速度放慢，RPC 响应可能失败，仍处于开发阶段，但并非完全可靠。

关于 Beam 的更多信息

设置客户端 {#client-setup}

客户端提供丰富的配置选项，可满足您的需要。 根据安全程度、可用数据和成本选择最适合的方法。 除了同步算法之外，您还可以设置对不同类型的旧数据的精简。 精简可以删除过时的数据，例如删除无法从最近区块上访问的节点。

关注客户端的文档或帮助页面来找出默认的同步模式。 您在设置时定义所希望的同步类型，如：

在 GETH 或 ERIGON 设置轻量同步

```
geth --syncmode "light"
```

想要了解更多详情，请查看运行 Geth 轻量节点的教程。

在 Besu 中设置与归档的完整同步

```
besu --sync-mode=FULL
```

像任何其他配置一样，它可以通过启动标志或配置文件来定义。 另一个例子是 Nethermind ，它促使您在初始化过程中选择一个配置并创建一个配置文件。

共识客户端（原“Eth2”客户端） {#consensus-clients}

有多个共识客户端（以前称为“Eth2”客户端）支持 共识升级。 这些客户端正在运行信标链，并将在 合并 后为执行客户端提供权益证明共识机制。

查看共识客户端。

客户端	语言	操作系统:	网络
Teku	Java	Linux, Windows, macOS	信标链，Prater
Nimbus	Nim	Linux，Windows，macOS	信标链，Prater
Lighthouse	Rust	Linux，Windows，macOS	信标链，Prater，Pyrmont
Lodestar	TypeScript	Linux，Windows，macOS	信标链，Prater
Prysm	Go	Linux，Windows，macOS	信标链，Gnosis，Prater，Pyrmont

硬件 {#hardware}

硬件要求因客户端不同而已，但通常要求不会特别高，因为节点只需要保持同步。 不要把它与需要更多算力的挖矿混为一谈。 然而，更强大的硬件的确可以提升同步时间和性能。 根据您的需求和目的，以太坊可以在您的计算机、家庭服务器、单板计算机或云端虚拟私人服务器上运行。

有一个简单的方法可以运行您自己的节点，即使用即插即用的盒子，如 DAppNode。 一个简单的用户接口，为运行客户端和依赖客户端的应用程序提供硬件。

要求 {#requirements}

在安装任何客户端之前，请确保您的计算机有足够的资源运行它。 最低要求和建议请见下文，然而核心部分在于磁盘空间。 同步以太坊区块链是高强度的输入/输出。 最好有一个固态硬盘 (SSD)。 要在机械硬盘上运行以太坊客户端，您将需要至少 8GB 的内存作为缓存。

最低要求 {#recommended-specifications}

- 2 核以上 CPU
- 如果是固态硬盘，则内存需 4GB 以上。如果是机械硬盘，则内存需 8GB 以上。
- 8 MBit/s 带宽

推荐的规格要求 {#recommended-specifications}

- 具有 4 核以上快速 CPU
- 16GB 以上 RAM
- 快速固态硬盘，拥有至少 500 GB 可用空间
- 25 MBit/s 以上带宽

您选择的同步模式将影响空间要求，但我们估计了下面每个客户端需要的磁盘空间。

客户端	磁盘大小（快速同步）	磁盘大小（完整存档）
Geth	400GB 以上	6TB 以上
OpenEthereum	280GB 以上	6TB 以上
Nethermind	200GB 以上	5TB 以上
Besu	750GB 以上	5TB 以上
Erigon	不适用	1TB 以上

- 注意：Erigon 无法快速同步，但可以完全修剪 (~500 GB)。

图表表明，完全同步所需的 GB 数正呈上升趋势

图表表明，存档同步所需的 GB 数正呈上升趋势

这些图表显示了存储要求的变化趋势。 关于 Geth 和 OpenEthereum 的最新数据，请参阅完整同步数据和归档同步数据。

单板计算机上的以太坊 {#ethereum-on-a-single-board-computer}

运行以太坊节点的最方便和最便宜的方法是使用 Raspberry Pi 这样的 ARM 架构的单板计算机。 Ethereum on ARM提供 Geth、OpenEthereum、Nethermind 和 Besu 客户端的图像。 这是一个关于如何构建和设置 ARM 客户端的简单教程。

像这样小型、实惠和高效的设备是在家里运行节点的理想选择。

延伸阅读 {#further-reading}

互联网上有许多关于以太坊客户端的信息。 以下是一些可能有所帮助的资源。

- 以太坊 101 - 第二部分 - 了解节点 - *Wil Barnes，2019 年 2 月 13 日*
- 运气以太坊全节点：勉励者指南 - *Justin Leroux，2019 年 11 月 7 日*
- 运行一个以太坊节点 - *ETHHub，经常更新*
- 分析成为以太坊完全验证节点的硬件要求 - *Albert Palau，2018 年 9 月 24 日*
- 在以太坊主网上运行 Hyperledger Besu 节点：优点、要求和设置 - *Felipe Faraggi，2020 年 5 月 7 日*

相关主题 {#related-topics}

- 块
- 网络

相关教程 {#related-tutorials}

- 用 Geth 运行一个节点 - *如何下载、安装和运行 Geth。 涵盖了同步模式、JavaScript 控制台等内容。*
- 将您的 Raspberry Pi 4 通过写入 MicroSD 卡变成验证节点 - 安装指南 - *写入你的 Raspberry Pi 4，插入网线，连接固态磁盘并给设备供电，将 Raspberry Pi 4 变成一个运行执行层（主网）和/或共识层（信标链/验证者）的完整以太坊节点。*

title: 节点服务
description: 节点服务及其利弊和主流供应商的入门级概述。
lang: zh
sidebar: true

sidebarDepth: 2

介绍 {#Introduction}

运行自己的以太坊节点可能会比较困难，特别是在刚开始时或在快速扩展时。 许多服务都可以为你运行优化的节点基础设施，藉此，你可以专注于开发你的应用程序或产品。 我们将解释节点服务运行的原理以及它们的优缺点，如果有兴趣着手开始，我们将为你列出供应商。

前置要求 {#prerequisites}

如果你还不了解什么是节点和客户端，请查看节点和客户端。

节点服务运作原理 {#how-do-node-services-work}

节点服务提供商在幕后为你运行分布式节点客户端，因此你就不必做这些了。

这些服务通常提供一个您可以用来写入和读取区块链 API 密钥。 除了主网之外，通常还包括对以太坊测试网的访问。

有些服务提供由他们替你管理的你自己的专用节点，而另外一些服务则使用负载均衡器在各节点之间分配活动。

几乎所有的节点服务都非常容易集成，只需要在你的代码中修改一行，就可以换掉你自己的托管节点，甚至可以在服务本身之间切换。

通常节点服务会运行各种节点客户端和各种节点，使得用户可以仅通过一个 API 获得全部和存档的节点。

值得注意的是，节点服务不会也不应该存储你的私钥或信息。

使用节点服务的好处是什么？ {#benefits-of-using-a-node-service}

使用节点服务的主要好处是不需要自己花费用来维护和管理节点的工程时间。 这使你能够专注于构建你的产品，而不必担心基础设施的维护。

从存储到带宽再到宝贵的工程时间，运行自己的节点都是非常昂贵的， 像在扩展时运转更多的节点、将节点升级到最新的版本以及确保状态的一致性，这些都会影响到你在所需的 web3 产品上的建设和花费的资源。

使用节点服务的缺点是什么？ {#cons-of-using-a-node-service}

使用了节点服务，意味着你产品的基础设施部分走向了集中化。 因此，那些把去中心化奉为圭臬的项目可能更喜欢自我托管的节点，而不是外包给第三方。

阅读更多关于运行自己节点的好处。

热门节点服务 {#popular-node-services}

这里列出了一些最受欢迎的以太坊节点服务提供商，如有缺失，随时添加！ 每个节点服务除了免费或付费等级外，还提供不同的福利和功能。 在做出决定之前，你应该调查哪些最符合你的需求。

- **Alchemy**
  - 相关文档
  - 特性
    - 免费等级选项
    - 随时扩展
    - 免费归档数据
    - 分析工具
    - 仪表板
    - 唯一 API 端点
    - 网钩
    - 点对点支持
- **Ankr**
  - 相关文档
  - 特性
    - Ankr 协议 - 为超过 8 条链提供公共 RPC API 端点
    - 负载平衡和节点健康监测，为最近的可用节点提供快速可靠的网关。
    - 支持 WSS 端点和无上限速率限制的高级层
    - 一键式全节点和验证者节点部署，可用于 40 多条链
    - 随时扩展
    - 分析工具
    - 仪表板
    - RPC、HTTP 和 WSS 端点
    - 点对点支持
- **BlockDaemon**
  - 相关文档
  - 优势
    - 仪表板
    - 基于每个节点
    - 数据分析
- **ChainStack**
  - 相关文档
  - 特性
    - 免费共享节点
    - 共享归档节点
    - GraphQL 支持
    - RPC 和 WSS 端点
    - 专用完整归档节点
    - 专用部署快速同步
    - 用自己的云服务
    - 按小时计费定价
    - 全天候点对点支持
- **GetBlock**
  - 相关文档

    - 特性
        - 接入 40 多个区块链节点
        - 40K 每日免费请求
        - 无限量 API 秘钥
        - 1GB/秒的高连接速度
        - 追踪 + 归档
        - 高级分析
        - 自动更新
        - 技术支持
- **InfStones**
    - 特性
        - 免费层级选择
        - 随时扩展
        - 分析
        - 仪表板
        - 唯一 API 端点
        - 专用完整节点
        - 专用部署快速同步
        - 全天候点对点支持
        - 接入 50 多个区块链节点
- **Infura**
    - 相关文档
    - 特性
        - 免费层级选择
        - 随时扩展
        - 付费归档数据
        - 点对点支持
        - 仪表板
- **Moralis**
    - 相关文档
    - 特性
        - 免费共享节点
        - 免费共享归档节点
        - 注重隐私（无日志政策）
        - 跨链支持
        - 随时扩展
        - 仪表板
        - 唯一以太坊 SDK
        - 唯一 API 端点
        - 直接技术支持
- **Pocket Network**
    - 相关文档
    - 功能
        - 去中心化 RPC 协议和市场
        - 每天一百万个免费层级请求（每个端点，最多 2 个）
        - 公共端点

        - Pre-Stake+ 项目（如果需要每天超一百万次的请求）
        - 支持多于 15 个区块链
        - 6400 多个节点为应用程服务赚取 POKT
        - 归档节点，具有追踪及测试网节点支持的归档节点
        - 以太坊主网节点客户端多样性
        - 无单点故障
        - 零停机时间
        - 具有成本效益的近零通证经济（质押 POKT 获得网络带宽）
        - 没有每月沉没成本，将您的基础设施变成资产
        - 协议中规定的负载平衡
        - 每日请求数量和每小时节点数量无限变化
        - 最私密的抗审查选择
        - 实践型开发者支持
        - Pocket Portal 仪表板和分析
- **QuikNode**
    - 功能
        - 7 天免费试用
        - 多样化支持
        - Webhooks
        - 仪表板
        - 数据分析
- **Rivet**
    - 相关文档
    - 特性
        - 免费层级选择
        - 随时扩展
- **SettleMint**
    - 相关文档
    - 特性
        - 免费试用
        - 随时扩展
        - GraphQL 支持
        - RPC 和 WSS 端点
        - 专用完整节点
        - 用自己的云服务
        - 分析工具
        - 仪表板
        - 按小时计费定价
        - 点对点支持

延伸阅读 {#further-reading}

- 以太坊节点服务列表

相关主题 {#related-topics}

- 节点和客户端

相关教程 {#related-tutorials}

- 使用 Alchemy 开始以太坊开发
- 使用 web3 和 Alchemy 发送交易指南

---

title: 运行您自己的以太坊节点
description: 如何运行自己的以太坊客户端简介
lang: zh
sidebar: true

sidebarDepth: 2

运行您自己的节点为您提供各种好处，打开新的可能性，并为支持生态系统提供帮助。 这个页面将引导您启动您自己的节点，并参与验证以太坊交易。

前提条件 {#prerequisites}

你应该明白什么是以太坊节点，以及你可能想要运行客户端的原因。 节点和客户端涵盖了这个问题。

选择一种方式 {#choosing-approach}

要运行您自己的以太坊节点，第一步是决定您的运行方式。 您需要选择客户端（软件），运行环境以及您的启动参数 查看所有可用的主网客户端。

客户端设置 {#client-settings}

客户端启用时有不同的同步模式和各种其他选项 同步模式是指下载和验证区块链数据的不同方法。 在启动节点前，您应该决定使用哪个网络和同步模式 做决定时最重要的考虑因素是，磁盘容量和客户端需要的同步时间。

所有功能和选项都可以在客户端的文档中找到。 您可以通过执行相应参数来为客户端进行不同的配置 关于参数的更多信息，可以从客户端文档或 EthHub 获取。 为了测试的目的，您可能喜欢在测试网上运行一个客户端。 查看支持的网络。

运行环境和硬件设施 {#environment-and-hardware}

本地或云端 {#local-vs-cloud}

以太坊客户端可以在消费级电脑上运行，不需要特殊的硬件，比如挖矿。 因此，您可以根据您的需要来进行部署。 简单一点，我们考虑在本地物理机器和云服务器上运行一个节点：

- 云端
  - 云服务商提供了高可用的服务器以及静态公共 IP 地址
  - 获得专用或虚拟服务器比自己搭建更加方便
  - 取舍是：是否需要信赖云服务商三方
  - 由于全节点需要的内存很大，租用一个服务器的价格可能很贵
- 自有硬件
  - 更可信并且更有主动权
  - 只需一次性投入
  - 可以购买预先配置好的机器
  - 您必须亲自准备、维护和解决潜在的机器故障

上面两种方式各有优点。 如果您正在寻找云端的解决方案，除了许多传统云计算服务商外，还有一些专注于部署以太坊节点的服务商， 例如

- QuikNode,
- Blockdaemon,
- LunaNode.

硬件 {#hardware}

然而，一个抗审查的去中心化网络不应该依赖于云服务商。 如果您在自己的硬件设备上运行节点，这对整个生态系统来说更健康。 最简单的做法就是购买预先配置好的机器，比如

- DappNode
- Avado.

查看每种客户端和同步模式所需要的最小和推荐磁盘容量。 一般来说，普通的算力就足够了。 问题通常是驱动速度。 在最初的数据同步期，以太坊客户端会执行许多读/写操作， 所以强烈推荐 SSD。 使用 HDD 的客户端可能不能同步到最新的以太坊状态，会卡在主网后的几个区块。 您可以使用带有 ARM 芯片的单板机来运行大多数客户端。 您也可以使用带有 Ethbian 操作系统的 Raspberry Pi 4。 这让你可以通过使用 SD 卡来运行客户端。 基于您的软件和硬件选择，初始同步时间和存储容量要求可能不一样，请务必 查看同步时间和存储容量要求， 也请确保您的网络连接没有宽带流量限制。 推荐您使用不计量的网络连接，因为初始同步和广播到网络的数据可能超过您的限额。

操作系统 {#operating-system}

所有的客户端都支持主流的操作系统 — Linux、MacOS、Windows。 这意味着您可以使用带有最适合您的操作系统（OS）的桌面电脑或者服务器来运行节点。 为了避免潜在的故障和安全漏洞，请确保您的操作系统是最新的。

运行节点 {#spinning-up-node}

获取客户端软件 {#getting-the-client}

首先，下载您的首选客户端软件

您可以简单地下载适合您操作系统和架构的可执行应用程序或安装包。 始终验证下载的软件包的签名和校验量。 有些客户端为了简化安装和更新将软件发布在软件库中， 如果您喜欢的话，你可以从源代码开始构建客户端。 当然，所有的客户端都是开源的，您可以用合适的编译器从源码开始构建客户端。

稳定的主网客户端二进制可执行文件可以从其发布页面下载：

- Geth,
- OpenEthereum,,
- Nethermind,
- Besu,
- Erigon.

注意，OpenEthereum 已被废弃，并已停止维护。请谨慎使用，最好切换至其他客户端。

启动客户端 {#starting-the-client}

在启动以太坊客户端软件之前，请最后检查您的环境是否已就绪。 例如，请保证：

- 选定的网络和同步模式下有足够的磁盘空间
- 内存和 CPU 未被其他程序停止。
- 操作系统更新到最新版本。

- 系统有正确的时间和日期。
- 您的路由器和防火墙接受监听端口上的连接。 默认情况下，以太坊客户端使用监听 (TCP) 端口和探测端口 (UDP)，两者均默认为 30303。

首先在测试网上运行您的客户端，帮助确保一切正常工作。 运行一个 Geth 轻节点应该会有所帮助。 您需要设置在启动时不是默认好的客户端配置。 您可以使用参与或配置文件设置您的首选配置。 详细信息可以查看您的客户端文档。客户端启动时将初始化其核心功能，选择端点，并开始寻找其他节点。 在成功探测到其他节点后，客户端开始同步数据。 当客户端成功同步到最新状态时，最新的区块链数据将可使用。

访问客户端 {#using-the-client}

客户端提供 RPC API 端点，您可以用来控制客户端，并以各种方式与以太坊网络交互：

- 使用合适的协议（例如，使用 `curl`）来手动调用
- 附加一个控制台（例如，`geth attache`）
- 在应用程序中执行

不同的客户端对 RPC 端点有不同的实现方式。 但是有一个标准的 JSON RPC，每个客户端都可以使用。 想要了解，请阅读 JSON-RPC 文档。 那些需要从以太坊网络中获取信息的应用，可以使用此 RPC。 例如，受欢迎的钱包 MetaMask，您可以运行一个本地区块链实例并连接到该钱包。

访问 RPC {#reaching-rpc}

JSON-RPC 的默认端口是 `8545`，不过您可以在本地端点的配置文件中修改这个端口。 默认情况下，这个 RPC 接口只能在您电脑的本地主机上访问。 为了能够让该端口可以被远程访问，您可以通过将地址改为 `0.0.0.0` 来进行公开。 这将使该端口可以通过本地或公共 IP 地址访问。 在大多数情况下，您也需要在您的路由器上设置端口转发。

当您设置远程访问时应该保持谨慎，因为这会让互联网上的任何人有可能控制您的节点。 如果您使用您的客户端充当钱包的话，不怀好意者可以让您的钱包系统下线或者窃取您的资金。

与此相应的一个解决办法是防止可能有危害的 RPC 访问方式被修改。 例如，使用 `Geth` 时，您可以将可被修改的访问方式用参数声明：`–http.api web3,eth,txpool`。

您也可以使用指向您客户端本地地址和端口的网络服务器，比如 Nginx 来公开您的 RPC 接口。

最保护隐私和简单的方法还是设置一个托管在自己的洋葱服务器上的可公开访问端点。 这将让您在本地网络外部无需开放的静态公共 IP 地址和端口也能连接 RPC。 操作如下：

- 安装 `tor`
- 编辑 `torrc` 配置文件以启用隐藏您客户端 RPC 地址和端口的服务
- 重启 `tor` 服务

一旦您重启 Tor，您将在您设置的目录中获得隐藏的密钥和主机名。 之后，您的 RPC 接口就可以通过一个以 `.onion` 结尾的主机名进行访问。

操作该节点 {#operating-the-node}

您应该定期监控您的节点，以确保它正常运行。 偶尔您也可能需要维护。

保持节点在线 {#keeping-node-online}

您的节点不需要一直在线，但是为了保持和网络同步，您应该尽可能多的让它在线。 当您需要关闭或重启节点时，以下需要注意：

- 如果磁盘正在写入，关机需要花费几分钟。
- 强制关机会导致数据库损坏。
- 您的客户端将无法与网络同步，当您重启后，您需要重新同步数据。

但是，对于用作*进行共识的验证节点*，就需要一直在线。验证节点离线会影响所有使用它的服务。 如果您是为了*权益质押*而运行节点，您应该尽可能地减少离线时间。

创建客户端证书 {#creating-client-service}

在启动节点时，您应该考虑创建一个客户端自动运行的服务。 例如，在 Linux 服务器上，一个不错的做法是创建一个服务，以适当的配置，在有限权限的用户下执行客户端并自动重新启动。

更新客户端 {#updating-client}

您应该让您的客户端软件具有最新的安全补丁、功能和 EIP。 尤其是在硬分叉之前，请确保您的客户端版本是正确的。

运行额外服务 {#running-additional-services}

运行自己的节点，可以让您使用需要直接访问以太坊客户端 RPC 的服务。 有些服务构建在以太坊架构之上，如第二层解决方案、共识客户端和其它以太坊基础设施。

监测节点 {#monitoring-the-node}

“为了正确地监测您的节点，请考虑收集指标。 客户端提供了节点指标，这样您可以获得关于节点的全面数据。 使用像 InfluxDB 或者 Prometheus 这样的数据库，再加上 Grafana 这样的可视化软件，您可以将节点可视化。 在可视化节点和网络时，Grafana 有许多设置和仪表版样式可供您使用。 作为监测的一部分，也需注意您的机器性能。 在您的节点初始同步期间，客户端软件会占用大量 CPU 和 RAM。 除了 Grafana，您也可以使用系统自带的 `htop` 或 `uptime` 等工具。

延伸阅读 {#further-reading}

- Analyzing the hardware requirements to be an Ethereum full validated node – *Albert Palau，2018 年 9 月 24 日*
- Running Ethereum Full Nodes: A Guide for the Barely Motivated – *Justin Leroux，2019 年 11 月 7 日*
- Running an Ethereum Node – *ETHHub，经常更新*
- Running a Hyperledger Besu Node on the Ethereum Mainnet: Benefits, Requirements, and Setup – *Felipe Faraggi，2020 年 5 月 7 日*
- Deploying Nethermind Ethereum Client with Monitoring Stack – *Nethermind.eth，2020 年 7 月 8 日*

相关主题 {#related-topics}

- 节点和客户端
- 块
- 网络

title: 预言机
description: 预言机有助于将真实世界数据输入您的以太坊应用程序，因为智能合约本身无法查询真实世界的数据。
lang: zh
sidebar: true

incomplete: true

预言机是将以太坊连接到链外、真实世界信息的数据源，以便您可以在智能合约中查询数据。 例如，预测市场的去中心化应用程序使用预言机来根据事件进行结算。 预测市场可能要求您使用以太币来打赌下一任美国总统是谁。 他们将使用一个预言机来确认结果并向获胜者付款。

前置要求 {#prerequisites}

请确保您熟悉节点、共识机制和智能合约解析，尤其是事件。

什么是预言机 {#what-is-an-oracle}

预言机是区块链与真实世界之间的桥梁。 他们充当在链上的应用程序接口，你可以查询以获取信息到你的智能合约。 可以是任何信息，从价格信息到天气预报。 预言机可以是双向的，用于向真实世界“发送”数据。

观看 Patrick 解释预言机：

为什么需要预言机？ {#why-are-they-needed}

使用以太坊这样的区块链，您需要确保：网络中每个节点都能够重放每个交易，并最终产生同样的结果。 应用程序接口可能会引入可变数据。 如果使用价格应用程序接口根据商定的 $USD 值发送以太币，查询将在不同日期返回不同的结果。 更不用说，应用程序接口可能会被破解或弃用。 一旦发生这种情况，网络中的节点将不会接受以太坊的当前状态，从而打破共识。

预言机可通过发布数据到区块链来解决这个问题。 以便任何重放交易的节点使用所有人都能看到的相同且不变的数据。 要做到这一点，预言机通常由一个智能合约和某些可以查询应用程序接口的链下组件组成，然后定期发送交易以更新智能合约的数据。

Oracle 问题 {#oracle-problem}

如前所述，以太坊交易无法直接获取链下数据。 同时，依靠单一的事实来源提供数据并不安全，会使智能合约的去中心化失效。 这便是预言机的问题。

我们可以使用去中心化预言机拉取多个数据来源，以避免预言机的问题；如果某个数据源被破解或失效，智能合约仍将按预期运行。

安全性 {#security}

预言机的安全性等同于其数据源。 如果一个去中心化应用程序使用 Uniswap 作为其以太币/DAI 价格的预言机，攻击者就可以在 Uniswap 上篡改价格，以操纵该去中心化应用程序对当前价格的理解。 如何对付这个隐患的示例包括一种推送系统，如 MakerDAO 所使用的推送系统。它会将来自若干外部数据源的价格数据进行比对，而不是仅仅依靠单一来源。

架构 {#architecture}

这是一个简单预言机架构的示例，但还有更多方法可用来触发链下计算。

1. 通过您的智能合约事件发出一份日志
2. 一个链下服务已经订阅了（通常使用类似 JSON-RPC `eth_subscribe` 的命令）这些特定的日志。
3. 该链下服务会继续做一些日志所定义的任务。
4. 链下服务使用智能合约二级交易中要求的数据作出回应。

如此可以 1 对 1 的方式获取数据，但是为了提高安全性，您可能希望对链下数据的收集方式进行去中心化。

下一步可能是建立这些节点的网络，将这些节点调用到不同的应用程序接口和源，并在链上汇总数据。

Chainlink 链下报告 (Chainlink OCR) 让链下预言机网络相互通信、为它们的回应附加加密签名、汇总它们的链下回应，并且仅随结果发送一个交易，以改进这种方法。 这样可以减少消耗的燃料，但您仍然可以得到去中心化数据的保证，因为每个节点都对其负责的交易进行了签名，使得发送交易的节点无法更改该交易。 如果节点不进行交易处理，而由下一个节点发送交易，上报规则将会介入。

使用方法 {#usage}

使用诸如 Chainlink 等服务，您可以引用链上已经从现实世界中提取并汇总的去中心化数据。 这有点像公共空间，但仅用于去中心化数据。 您也可以构建自己的模块化预言机网络来获取您想要的任意自定义数据。 此外，您可以执行链下运算并向真实世界发送信息。 Chainlink 已具备相关基础设施，以：

- 在你的合约中获取加密货币价格
- 生成可验证的随机数字（对游戏有用）
- 调用外部应用程序接口 – 一种新颖的用法是 检查 wBTC 储备

以下为如何从智能合约中获取最新以太币价格的示例：

Chainlink 数据源 {#chainlink-data-feeds}

```
pragma solidity ^0.6.7;

import "@chainlink/contracts/src/v0.6/interfaces/AggregatorV3Interface.sol";

contract PriceConsumerV3 {

    AggregatorV3Interface internal priceFeed;

    /**
     * Network: Kovan
     * Aggregator: ETH/USD
     * Address: 0x9326BFA02ADD2366b30bacB125260Af641031331
     */
    constructor() public {
        priceFeed =
AggregatorV3Interface(0x9326BFA02ADD2366b30bacB125260Af641031331);
    }

    /**
     * Returns the latest price
     */
    function getLatestPrice() public view returns (int) {
```

```
        (
            uint80 roundID,
            int price,
            uint startedAt,
            uint timeStamp,
            uint80 answeredInRound
        ) = priceFeed.latestRoundData();
        return price;
    }
}
```

[您可以结合此链接进行该测试]

[查看文档]

Chainlink VRF {#chainlink-vrf}

Chainlink VRF（可验证随机函数）是为智能合约设计的可证明公平性和可验证的随机性来源。 智能合约开发者可以使用 Chainlink VRF 作为防篡改随机数生成 (RNG)，为依赖不可预测结果的任何应用程序建立可靠的智能合约：

- 区块链游戏和非同质化代币
- 职责和资源的随机分配（例如随机分配审判者审理案件）
- 为共识机制选择一个有代表性的示例

随机数很难，因为区块链是决定性的。

在数据源之外使用 Chainlink 预言机需要遵循使用 Chainlink 的请求和接收周期。 它们使用 LINK 代币向预言机供应商发送预言机燃料，以获得响应。 LINK 代币专门设计用于与预言机协同作用，基于升级后的 ERC-677 代币，此代币向后兼容 ERC-20。 以下代码如果部署到 Kovan 测试网，将获得一个经加密验证的随机数。 要提出请求，请使用您可以从 Kovan LINK Faucet 获得的测试网 LINK 代币，为合约提供资金。

```
pragma solidity 0.6.6;

import "@chainlink/contracts/src/v0.6/VRFConsumerBase.sol";

contract RandomNumberConsumer is VRFConsumerBase {

    bytes32 internal keyHash;
    uint256 internal fee;

    uint256 public randomResult;

    /**
     * Constructor inherits VRFConsumerBase
     *
     * Network: Kovan
     * Chainlink VRF Coordinator address:
0xdD3782915140c8f3b190B5D67eAc6dc5760C46E9
     * LINK token address:
0xa36085F69e2889c224210F603D836748e7dC0088
     * Key Hash:
0x6c3699283bda56ad74f6b855546325b68d482e983852a7a82979cc4807b641f4
```

```
     */
    constructor()
        VRFConsumerBase(
            0xdD3782915140c8f3b190B5D67eAc6dc5760C46E9, // VRF Coordinator
            0xa36085F69e2889c224210F603D836748e7dC0088  // LINK Token
        ) public
    {
        keyHash =
0x6c3699283bda56ad74f6b855546325b68d482e983852a7a82979cc4807b641f4;
        fee = 0.1 * 10 ** 18; // 0.1 LINK (varies by network)
    }

    /**
     * Requests randomness from a user-provided seed
     */
    function getRandomNumber(uint256 userProvidedSeed) public returns (bytes32
requestId) {
        require(LINK.balanceOf(address(this)) >= fee, "Not enough LINK - fill
contract with faucet");
        return requestRandomness(keyHash, fee, userProvidedSeed);
    }

    /**
     * Callback function used by VRF Coordinator
     */
    function fulfillRandomness(bytes32 requestId, uint256 randomness) internal
override {
        randomResult = randomness;
    }
}
```

Chainlink Keeper {#chainlink-keepers}

智能合约无法在任意时间或任意条件下自动触发或执行。 只有当另一个帐户（如用户、预言机或合约）发起交易时，才会发生状态变化。 Chainlink Keeper 网络为智能合约提供了多种选择，以信任最小化和去中心化的方式将常规维护任务外包出去。

要使用 Chainlink Keeper，智能合约必须执行 KeeperCompatibleInterface，它由两个函数组成：

- `checkUpkeep` - 检查合约是否需要完成工作。
- `performUpkeep` - 如果 checkUpkeep 有指示，随即执行合约内容。

以下示例是一个简单的 Counter 合约。 `counter` 变量通过每次调用 `performUpkeep` 来递增。 您可以使用 Remix 查看以下代码

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.7.0;

// KeeperCompatible.sol imports the functions from both ./KeeperBase.sol and
// ./interfaces/KeeperCompatibleInterface.sol
import "@chainlink/contracts/src/v0.7/KeeperCompatible.sol";

contract Counter is KeeperCompatibleInterface {
    /**
    * Public counter variable
```

```
    */
    uint public counter;

    /**
    * Use an interval in seconds and a timestamp to slow execution of Upkeep
    */
    uint public immutable interval;
    uint public lastTimeStamp;

    constructor(uint updateInterval) {
      interval = updateInterval;
      lastTimeStamp = block.timestamp;

      counter = 0;
    }

    function checkUpkeep(bytes calldata /* checkData */) external override
returns (bool upkeepNeeded, bytes memory /* performData */) {
        upkeepNeeded = (block.timestamp - lastTimeStamp) > interval;
        // We don't use the checkData in this example. Upkeep 注册时会定义
checkData。
    }

    function performUpkeep(bytes calldata /* performData */) external override {
        lastTimeStamp = block.timestamp;
        counter = counter + 1;
        // We don't use the performData in this example. Keeper 调用 checkUpkeep
函数
   }
} 时，会生成 performData
```

部署了兼容 Keeper 的合约后，您必须注册 Upkeep 的合约，并用 LINK 为其添加资金。这样才能通知 Keeper 网络有关您的合约事宜，工作才可以持续进行。

Keeper 项目 {#keepers}

- Chainlink Keeper
- Keep3r 网络

Chainlink API 调用 {#chainlink-api-call}

Chainlink 应用程序接口调用是以网络调用的传统方式（应用程序接口调用）从链下获取数据的最简单方式。 做一个这样的实例，并且只有一个预言机，使得它在本质上是集中的。 要保持它真正去中心化，智能合约平台需要使用在外部数据市场中找到的许多节点。

在 kovan 网络上部署以下代码以进行测试

这也遵循了预言机的请求和接收周期，并且需要从 Kovan LINK（预言机燃料）获得资金才能工作。

```
pragma solidity ^0.6.0;

import "@chainlink/contracts/src/v0.6/ChainlinkClient.sol";

contract APIConsumer is ChainlinkClient {
```

```
    uint256 public volume;

    address private oracle;
    bytes32 private jobId;
    uint256 private fee;

    /**
     * Network: Kovan
     * Oracle: 0x2f90A6D021db21e1B2A077c5a37B3C7E75D15b7e
     * Job ID: 29fa9aa13bf1468788b7cc4a500a45b8
     * Fee: 0.1 LINK
     */
    constructor() public {
        setPublicChainlinkToken();
        oracle = 0x2f90A6D021db21e1B2A077c5a37B3C7E75D15b7e;
        jobId = "29fa9aa13bf1468788b7cc4a500a45b8";
        fee = 0.1 * 10 ** 18; // 0.1 LINK
    }

    /**
     * Create a Chainlink request to retrieve API response, find the target
     * data, then multiply by 1000000000000000000 (to remove decimal places from
data).
     */
    function requestVolumeData() public returns (bytes32 requestId)
    {
        Chainlink.Request memory request = buildChainlinkRequest(jobId,
address(this), this.fulfill.selector);

        // Set the URL to perform the GET request on
        request.add("get", "https://min-
api.cryptocompare.com/data/pricemultifull?fsyms=ETH&tsyms=USD");

        // Set the path to find the desired data in the API response, where the
response format is:
        // {"RAW":
        //   {"ETH":
        //    {"USD":
        //     {
        //      "VOLUME24HOUR": xxx.xxx,
        //     }
        //    }
        //   }
        //  }
        request.add("path", "RAW.ETH.USD.VOLUME24HOUR");

        // Multiply the result by 1000000000000000000 to remove decimals
        int timesAmount = 10**18;
        request.addInt("times", timesAmount);

        // Sends the request
        return sendChainlinkRequestTo(oracle, request, fee);
    }

    /**
```

```
     * Receive the response in the form of uint256
     */
    function fulfill(bytes32 _requestId, uint256 _volume) public
recordChainlinkFulfillment(_requestId)
    {
        volume = _volume;
    }
}
```

您可以阅读 Chainlink 开发者博客，进一步了解 Chainlink 的应用程序。

预言机服务 {#other-services}

- Chainlink
- Witnet
- Provable
- Paralink
- Dos.Network

构建一个预言机智能合约 {#build-an-oracle-smart-contract}

此为 Pedro Costa 编写的预言机合约示例。 您可以在他的以下文章中找到更多注解：在以太网部署区块链预言机。

```
pragma solidity >=0.4.21 <0.6.0;

contract Oracle {
  Request[] requests; //list of requests made to the contract
  uint currentId = 0; //increasing request id
  uint minQuorum = 2; //minimum number of responses to receive before declaring
final result
  uint totalOracleCount = 3; // Hardcoded oracle count

  // defines a general api request
  struct Request {
    uint id;                            //request id
    string urlToQuery;                  //API url
    string attributeToFetch;            //json attribute (key) to retrieve in
the response
    string agreedValue;                 //value from key
    mapping(uint => string) anwers;     //answers provided by the oracles
    mapping(address => uint) quorum;    //oracles which will query the answer
(1=oracle hasn't voted, 2=oracle has voted)
  }

  //event that triggers oracle outside of the blockchain
  event NewRequest (
    uint id,
    string urlToQuery,
    string attributeToFetch
  );

  //triggered when there's a consensus on the final result
  event UpdatedRequest (
```

```
    uint id,
    string urlToQuery,
    string attributeToFetch,
    string agreedValue
  );

  function createRequest (
    string memory _urlToQuery,
    string memory _attributeToFetch
  )
  public
  {
    uint lenght = requests.push(Request(currentId, _urlToQuery,
_attributeToFetch, ""));
    Request storage r = requests[lenght-1];

    // Hardcoded oracles address
    r.quorum[address(0x6c2339b46F41a06f09CA0051ddAD54D1e582bA77)] = 1;
    r.quorum[address(0xb5346CF224c02186606e5f89EACC21eC25398077)] = 1;
    r.quorum[address(0xa2997F1CA363D11a0a35bB1Ac0Ff7849bc13e914)] = 1;

    // launch an event to be detected by oracle outside of blockchain
    emit NewRequest (
      currentId,
      _urlToQuery,
      _attributeToFetch
    );

    // increase request id
    currentId++;
  }

  //called by the oracle to record its answer
  function updateRequest (
    uint _id,
    string memory _valueRetrieved
  ) public {

    Request storage currRequest = requests[_id];

    //check if oracle is in the list of trusted oracles
    //and if the oracle hasn't voted yet
    if(currRequest.quorum[address(msg.sender)] == 1){

      //marking that this address has voted
      currRequest.quorum[msg.sender] = 2;

      //iterate through "array" of answers until a position if free and save the
retrieved value
      uint tmpI = 0;
      bool found = false;
      while(!found) {
        //find first empty slot
        if(bytes(currRequest.anwers[tmpI]).length == 0){
          found = true;
```

```
            currRequest.anwers[tmpI] = _valueRetrieved;
          }
          tmpI++;
        }

        uint currentQuorum = 0;

        //iterate through oracle list and check if enough oracles(minimum quorum)
        //have voted the same answer has the current one
        for(uint i = 0; i < totalOracleCount; i++){
          bytes memory a = bytes(currRequest.anwers[i]);
          bytes memory b = bytes(_valueRetrieved);

          if(keccak256(a) == keccak256(b)){
            currentQuorum++;
            if(currentQuorum >= minQuorum){
              currRequest.agreedValue = _valueRetrieved;
              emit UpdatedRequest (
                currRequest.id,
                currRequest.urlToQuery,
                currRequest.attributeToFetch,
                currRequest.agreedValue
              );
            }
          }
        }
      }
    }
  }
```

我们希望有更多关于构建预言机智能合约的文档。 如果您能帮助我们，就创建一个拉取请求吧！

延伸阅读 {#further-reading}

文章

- 什么是区块链 Oracle？ - *Chainlink*
- 预言机 – *EthHub*
- 什么是区块链预言机？ - *Patrick Collins*
- 去中心化预言机：综述– *Julien Thevenard*
- 在以太坊部署区块链预言机– *Pedro Costa*
- 为什么智能合约无法调用应用程序接口？ - *StackExchange*
- 我们为什么需要去中心化预言机- *Bankless*
- 因此您想要使用价格预言机- *samczsun*

相关视频

- 预言机和区块链实用程序拓展- *Real Vision Finance*

教程

- 如何在 Solidity 中获取以太坊的当前价格- _ Chainlink _

title: 面向 Dart 开发者的以太坊资源
description: 学习如何使用 Dart 语言进行以太坊开发
lang: zh

sidebar: true

智能合约和 Solidity 语言入门 {#getting-started-with-smart-contracts-and-solidity}

教程 {#tutorials}

- Flutter 和区块链 – Hello World 去中心化应用程序带您了解所有步骤，内容包括：
    1. 安装 Truffle 开发套件
    2. 使用 Solidity 编写一个智能合约
    3. 使用 Dart 编写一个用户界面
- 使用 Flutter 创建移动端去中心化应用这篇文章要简短得多，如果您了解基础知识，这篇文章可能更为合适。
- 如果您喜欢通过观看视频来学习，可以观看 创建您的第一个区块链 Flutter 应用，时长大概一个小时
- 如果您不想花这么多时间，您可能喜欢在以太坊上使用 Flutter 和 Dart 创建去中心化的区块链应用，时长仅约 20 分钟
- 将 MetaMask 集成到 Flutter 应用中 — 这个短片会带您了解如何一步步将 MetaMask 集成到您的 Flutter 应用中。

使用以太坊客户端 {#working-with-ethereum-clients}

您可以使用以太坊来创建去中心化应用程序（或称“dapp”），这种应用可以利用加密货币和区块链技术。 Dart 当前至少维护了两个调用以太坊 JSON RPC 应用程序接口的库。

1. 来自 simonbutler.eu 的 Web3dart
2. 来自 darticulate.com 的以太坊 5.0.0

还有其他的一些库，可用于操作特定的以太坊地址或者让您检索各种加密货币的价格。 您可以在这里查看完整的列表。

title: 面向 Delphi 开发者的以太坊
description: 学习如何使用 Delphi 编程语言为以太坊开发
lang: zh
sidebar: true

incomplete: true

学习如何使用 Delphi 编程语言为以太坊开发

使用以太坊来创建去中心化应用程序，发挥加密货币和区块链技术的优势。 这些去中心化应用程序可以是值得信赖的，也即一旦被部署到以太坊上，它们将总是按程序运行。 这些应用程序可以控制数字资产，以便创造新的金融应用； 它们可以是去中心化的，也即没有任何单一实体或个人能够控制它们，而且它们几乎是不可能被审查的。

在以太坊顶部构建去中心化应用，并使用 Delphi 编程语言与智能合约交互！

智能合约和 Solidity 语言入门 {#getting-started-with-smart-contracts-and-the-solidity-language}

迈出第一步，将 Delphi 与以太坊集成

需要更基础的入门知识？ 请查看 ethereum.org/learn 或者 ethereum.org/developers。

- 区块链详解
- 理解智能合约
- 编写您的第一个智能合约
- 学习如何编写和部署 Solidity

初学者参考文献和链接 {#beginner-references-and-links}

介绍 Delphereum 库

- 什么是 Delphereum？
- 连接 Delphi 到本地（内存）区块链中
- 连接 Delphi 到以太坊主网络
- 连接 Delphi 到智能合约

现在想要跳过设置，直接跳转到样本？

- 3 分钟的智能合约和 Delphi - 第 1 部分
- 3 分钟的智能合约和 Delphi - 第 2 部分

面向中等程度用户的文章 {#intermediate-articles}

- 在 Delphi 中生成一个以太坊认证的消息签名
- 使用 Delphi 交易以太币
- 使用 Delphi 交易基于 ERC-20 的代币

面向高等程度用户的使用模式 {#advanced-use-patterns}

- Delphi 和以太坊域名服务
- QuikNode、以太坊和 Delphi
- Delphi 和以太坊黑暗森林

想要获取更多的资源？ 请查看 ethereum.org/developers。

title: 面向 .NET 开发者的以太坊
description: 学习如何使用并通过基于 .NET 的项目及工具参与以太坊的开发
lang: zh
sidebar: true

incomplete: true

学习如何使用并通过基于 .NET 的项目及工具参与以太坊的开发

使用以太坊来创建去中心化应用程序（或称“dapp”），发挥加密货币和区块链技术的优势。 这些去中心化应用程序可以是值得信赖的，也即一旦被部署到以太坊上，它们将总是按程序运行。 这些应用程序可以控制数字资产，以便创造新的金融应用； 它们可以是去中心化的，也即没有任何单一实体或个人能够控制它们，而且它们几乎是不可能被审查的。

使用 Microsoft 技术堆栈中的工具和语言在以太坊上搭建去中心化应用程序和与智能合约交互 —— .NET Framework/.NET Core/.NET Standard 支持 VSCode 和 Visual Studio 等工具上的 C#、# Visual Basic .NET、F#。 使用 Microsoft Azure 区块链在 Azure 上部署一条以太坊区块链，几分钟即可完成。 将人们对 .NET 的喜爱带到以太坊上！

智能合约和 Solidity 语言入门 {#getting-started-with-smart-contracts-and-the-solidity-language}

迈出集成 .NET 与以太坊的第一步

需要更基础的入门知识？ 请查看 ethereum.org/learn 或者 ethereum.org/developers。

- 区块链详解
- 理解智能合约
- 编写您的第一个智能合约
- 学习如何编写和部署 Solidity

初学者参考文献和链接 {#beginner-references-and-links}

介绍 Nethereum 库和 VS Code Solidity

- 开始了解 Nethereum
- 安装 VS Code Solidity
- .NET 开发者创建和调用以太坊智能合约的工作流
- 智能合约与 Nethereum 的集成
- 使用 Nethereum 连接 .NET 和以太坊区块链智能合约，也可参考此中文版
- Nethereum——开源的区块链 .NET 集成库
- 使用 Nethereum 将以太坊交易写入 SQL 数据库
- 查看如何使用 C# 和 VisualStudio 轻松地部署以太坊智能合约

现在想要跳过设置，直接跳转到样本？

- 训练场 - 与以太坊进行交互，学习如何通过浏览器使用 Nethereum。
    - 查询帐户余额C#VB.NET
    - 查询 ERC20 智能合约余额C#VB.NET
    - 转账以太币至帐户 C# VB.NET
    - ... 以及更多！

面向中等程度用户的文章 {#intermediate-articles}

- Nethereum 练习册/样本列表
- 部署您自己的开发测试链
- Solidity 的 VSCode Codegen 插件
- Unity 和以太坊：为何以及如何？
- 为以太坊 dapp 创建 ASP.NET Core Web API
- 使用 Nethereum Web3 来实现一个供应链追踪系统
- Nethereum 区块处理，包含了C# 操作样本
- Nethereum Websocket Streaming

- Kaleido 和 Nethereum
- Quorum 和 Nethereum

面向高等程度用户的使用模式 {#advanced-use-patterns}

- Azure Key Vault 和 Nethereum
- Nethereum.DappHybrid
- Ujo Nethereum 后端参考架构

.NET 项目、工具和其他有趣内容 {#dot-net-projects-tools-and-other-fun-stuff}

- Nethereum 训练场 - *在浏览器中编译、创建和运行 Nethereum 代码片段*
- Nethereum Codegen Blazor - *在 Blazor 中具有 UI 的 Nethereum 代码生成*
- Nethereum Blazor - *.NET Wasm SPA 轻区块链浏览器和简单钱包*
- Wonka Business Rules Engine - *业务规则引擎（同时适用于 .NET 平台和以太坊平台），本质上是由元数据驱动的*
- Nethermind - *.NET 核心以太坊客户端，适用于 Linux、Windows 和 MacOs*
- eth-utils - *使用以太坊相关代码库的实用函数工具*
- TestChains - *预先配置的 .NET devchains，用于快速回应 (PoA)*

想要获取更多的资源？ 请查看 ethereum.org/developers。

.NET 社区贡献者 {#dot-net-community-contributors}

在 Nethereum，我们主要活跃于 Gitter 上，任何人都可以前来提问/回答问题，获得帮助或者前来逛逛。 您可以自由地在 Nethereum GitHub 库上提交拉取请求或者开启问题，或者只是前来浏览我们已有的许多小项目/样本项目。 您可以在 Discord 上找到我们！

在 Nethermind，让我们通过 Gitter 取得联系。 关于拉取请求或者问题，请查看 Nethermind GitHub 库。

其他汇总列表 {#other-aggregated-lists}

Nethereum 官方网站

Nethermind 官方网站

title: 语言资源
description: 了解关于以太坊的非英文资源
sidebar: true

lang: zh

语言资源 {#language-resources}

以太坊社区是全球性的，包括数以百万计的非英语人士。

我们的目标是提供所有语言的教育内容，帮助世界各地的人们克服语言障碍，成功加入以太坊社区。

如果您更喜欢用您的母语阅读或者知道某人不会说英语，您可以在下面找到有用的非英语资源列表。 数十万的以太坊爱好者聚集在这些在线论坛上，分享消息、谈论最近的发展、讨论技术问题并畅想未来。

知道一个您语言的教育资源？ 创建问题，以将其添加到列表！

Ethereum.org 资源 {#ethereum-org}

Ethereum.org 被翻译为 40 多种语言，您可以在我们的语言页面上找到这些语言。

如果您会使用两种语言，而且想帮助我们宣传到更多的人，您也可以参与 ethereum.org 翻译计划，帮助我们翻译该网站。

社区资源 {#community}

巴西葡萄牙语 {#br-pt}

新闻

- BeinCrypto - 提供有关加密货币的新闻和文章，包括一个巴西交易所列表
- Cointegraph - 巴西版 Cointelegraph，一个主要的加密货币新闻机构
- Livecoins - 提供有关加密货币的新闻和工具
- Seudinheiro - 提供有关加密货币的新闻和报告

教育

- web3dev - Web3 开发者们的内容中心和 Discord 社区。
- Web3Brasil - 提供有关 Web3 和去中心化金融的学习资源
- CriptoFacil - 提供有关加密货币的新闻和教育，包括“以太坊入门”和“去中心化金融入门”
- CriptoAtivos - 提供来自加密货币空间、教育和博客的洞察
- Cointimes - 提供有关加密货币的新闻和教育
- Web3 starter pack - 提供有关加密货币最常见和最基础问题解答的指南

中文 {#zh}

通用资源

- Ethereum.cn - 由社区维护网站内容，内容包括共识层升级、所有核心开发者会议记录、以太坊第二层等
- EthFans - 学习从基础到高级以太坊主题的所有知识
- Untimes - 由社区维护内容，内容涵盖以太坊、去中心化金融、非同质化代币和 Web3 相关知识
- 123ETH - 以太坊生态系统的一个门户网站
- Zhen Xiao（肖臻） - 关于加密货币及其应用的免费在线课程
- 以太坊白皮书 - 以太坊白皮书中文版

以太坊生态系统

- ETHPlanet - 可在线或现场参加黑客马拉松，为大学生提供培训
- PrimitivesLane - 一个以区块链技术为重点的非营利研究小组
- Ethereum Translation Community CN - 一个致力于翻译以太坊教育类内容的社区

适用于开发人员

- DappLearning - 每周学习主流去中心化应用程序项目并分享想法和评论的学习小组
- LearnBlockchain - 一个分享区块链技术相关信息的开发者社区

试用于加密研究人员

- 安比实验室 - 一个解释加密和安全等内容的微信公众号
- 星想法 - 一个解释零知识证明技术的微信公众号

法语 {#fr}

- Ethereum France - Ethereum France 组织各种活动、创建内容并鼓励围绕以太坊展开讨论
- Ethereum.fr - 提供有关以太坊的新闻和教育
- BanklessFR - Bankless 法语版新闻通讯
- CryptoFR - 以太坊子页面上的加密货币论坛

德语 {#de}

- Microsoft Learn (Solidity) - 使用 Solidity
- Microsoft Learn（智能合约） - 用 Solidity 撰写以太坊智能合约
- Microsoft Learn（以太坊网络） - 连接并部署以太坊网络
- Microsoft Learn（区块链） - 可进入区块链开发

意大利语 {#it}

- Ethereum Italia - 提供有关以太坊的教育、活动和新闻，专注于智能合约和区块链技术
- Ethereum Italia Podcast - 以太坊意大利语播客
- Microsoft Learn (Solidity) - 学习使用 Solidity
- Microsoft Learning（智能合约） - 学习用 Solidity 撰写智能合约
- Microsoft Learn（去中心化应用程序） - 使用去中心化的应用程序创建用户界面

西班牙语 {#es}

- Ethereum Madrid - 区块链、去中心化金融、治理课程、活动和博客
- Cointegraph - 西班牙语版以太坊初学者指南
- 在线教程 - 在以太坊上学习 Solidity 和编程
- Curso Introducción a Ethereum Development - Solidity 基础知识以及测试和部署您的首个智能合约
- Curso Introducción a Seguridad y Hacking en Ethereum - 了解真实智能合约中常见的漏洞和安全问题
- 去中心化金融开发课程介绍 - 了解去中心化金融智能合约如何在 Solidity 中工作，并创建自己的自动化做市商 (Automated Market Maker) 应用

土耳其语 {#tr}

- BTK Akademi - 专注于区块链和加密货币的课程
- 伟大的重命名：以太坊 2 发生了什么？ -《伟大的重命名》博文的土耳其语译作，解释了“以太坊 2”术语的变化

越南语 {#vi}

- Tino Group - 以太坊、去中心化应用程序、钱包和常见问题的概述
- Tap Chi Bitcoin - 具有以太坊新闻和教育子页面的网页平台
- Coin68 - 提供以太坊新闻和教育内容的加密货币门户网站

日语 {#ja}

- 日本虚拟和加密资产交易协会
- 日本加密资产商业协会
- 开始使用区块链开发 - 学习 | Microsoft Docs - 此学习路径向您介绍区块链和以太坊平台上的开发
- 掌握以太坊 - 用日语掌握以太坊
- 使用 Solidity 和 Ethereum 进行智能合约开发实践 - 日语中使用 Solidity 和 Ethereum 进行智能合约开发实践

title: 面向 Go 开发者的以太坊
description: 学习如何使用基于 Go 的项目和工具参与以太坊的开发
lang: zh
sidebar: true

incomplete: true

学习如何使用基于 Go 的项目和工具参与以太坊的开发

使用以太坊创建去中心化应用（即"dapps"）。 这些去中心化应用程序可被信任，意味着一旦被部署到以太坊上，它们将总是按既定程序运行。 它们是去中心化的，意味着它们运行在一个点对点网络中并且不存在单点故障。 不存在单一实体或者个人可以控制它们，它们也几乎不可能被审查。 它们可以通过控制数字资产来创建新的应用。

智能合约和 Solidity 语言入门 {#getting-started-with-smart-contracts-and-solidity}

迈出集成 Go 与以太坊的第一步

需要更基础的入门知识？ 请查看 ethereum.org/learn 或者 ethereum.org/developers。

- 区块链详解
- 理解智能合约
- 编写您的第一个智能合约
- 学习如何编写和部署 Solidity
- 智能合约教程

初学者文章和书籍 {#beginner-articles-and-books}

- 选择以太坊客户端
- 开始使用 Geth
- 使用 Golang 连接到以太坊
- 使用 Golang 部署以太坊智能合约
- 一步步教您测试和部署以太坊 Go 语言智能合约
- 电子书：使用 Go 开发以太坊 - *使用 Go 开发以太坊应用程序*

面向中等程度用户的文章和文档 {#intermediate-articles-and-docs}

- Go 以太坊相关文档 - *官方以太坊 Golang 相关文档*
- Erigon 编程者指南 - *图文指南，内容涵盖状态树、多重认证和交易处理*
- Erigon 和无状态以太坊 - *2020 Ethereum Community Conference (EthCC 3)*
- Erigon：优化以太坊客户端 - *2018 Devcon 4*
- Go 以太坊 GoDoc
- 在 Go 上使用 Geth 创建一个去中心化应用程序
- 用 Golang 和 Geth 使用以太坊专用网络
- 使用 Go 对以太坊上的 Solidity 合约进行单元测试
- 使用 Geth 作为库的快速参考

面向高等程度用户的使用模式 {#advanced-use-patterns}

- 使用 GETH 模拟后端搭建
- 基于以太坊和 Quorum 的区块链即服务应用程序

- 以太坊区块链应用程序中的分布式存储星际文件系统和 Swarm
- 移动客户端：各种库和 Inproc 以太坊节点
- 本地去中心化应用程序：绑定以太坊合约

Go 项目和工具 {#go-projects-and-tools}

- Geth / Go Ethereum - *以太坊协议的官方 Go 实现*
- Go Ethereum Code Analysis - *审查和分析 Go 以太坊源代码*
- Erigon - *基于 Go Ethereum 的更快的实现*
- Golem - *Golem 正在创建一个算力全球市场*
- Quorum - *支持数据隐私的许可型以太坊实现*
- Prysm - *以太坊 'Serenity'2.0 Go 实现*
- Eth Tweet - *去中心化 Twitter：运行在以太坊区块链上的微博客服务*
- Plasma MVP Golang — *Golang 实现以及最小可执行 Plasma 规范拓展*
- Open Ethereum Mining Pool - *以太坊开源矿池*
- Ethereum HD Wallet - *使用 Go 的以太坊硬件钱包衍生品*
- Multi Geth - *支持多种以太坊网络*
- Geth Light Client - *轻量级以太坊子协议的 Geth 实现*

想要获取更多的资源？ 请浏览 ethereum.org/developers。

Go 社区贡献者 {#go-community-contributors}

- Geth Discord
- Geth Gist
- Gophers Slack - #ethereum 频道
- StackExchange - 以太坊
- Multi Geth Gitter
- Ethereum Gitter
- Geth light Client Gitter

其他汇总列表 {#other-aggregated-lists}

- 强大的以太坊
- Consensys：以太坊开发工具的权威清单 | GitHub 源

---

title: 编程语言
description:
lang: zh

sidebar: true

一个常见的误解是开发人员必须编写 智能合约 才能在以太坊上构建。 这是错误的。 以太坊网络及其社区的一个美好之处是您可以使用任意编程语言参与其中。

以太坊及其社区拥抱开源。 您可以找到各种语言的社区项目 - 客户端实现、API、开发框架、测试工具。

选择您的语言 {#data}

选择您的编程语言来查找项目、资源和虚拟社区：

- 面向 Dart 开发者的以太坊资源

- 面向 Delphi 开发者的以太坊资源
- 面向 .NET 开发者的以太坊资源
- 面向 Go 开发者的以太坊资源
- 面向 Java 开发者的以太坊资源
- 以太坊的 JavaScript 开发者
- 以太坊的 Python 开发者
- 以太坊的 Ruby 开发者
- 面向 Rust 开发者的以太坊资源

如果不支持我的语言怎么办 {#other-lang}

如果您想链接到资源或指向额外编程语言的虚拟社区，您可以通过提出问题 来申请新页面。

如果只想使用当前不支持的语言 编写与区块链对接的代码，您可以使用 JSON-RPC 接口连接以太坊网络。 任何可以使用 TCP/IP 的编程 语言都可以使用此接口。

title: 面向 Java 开发者的以太坊资源
description: 学习如何使用并通过基于 Java 的项目及工具参与以太坊的开发
lang: zh
sidebar: true

incomplete: true

学习如何使用并通过基于 Java 的项目及工具参与以太坊的开发

使用以太坊来创建去中心化应用程序 (或称“dapp”)，发挥加密货币和区块链技术的优势。 这些去中心化应用程序可以是值得信赖的，也即一旦被部署到以太坊上，它们将总是按程序运行。 这些应用程序可以控制数字资产，以便创造新的金融应用； 它们可以是去中心化的，也即没有任何单一实体或个人能够控制它们，而且它们几乎是不可能被审查的。

智能合约和 Solidity 语言入门 {#getting-started-with-smart-contracts-and-solidity}

迈出第一步，将 Java 与以太坊进行集成

需要更基础的入门知识？ 请查看 ethereum.org/learn 或者 ethereum.org/developers。

- 区块链详解
- 理解智能合约
- 编写您的第一个智能合约
- 学习如何编写和部署 Solidity

使用以太坊客户端 {#working-with-ethereum-clients}

了解如何使用 Web3J 和 Hyperledger Besu 这两个领先的 Java 以太坊客户端。

- 使用 Java 、 Eclipse 和 Web3J 连接以太坊客户端
- 使用 Java 和 Web3j 管理以太坊帐户
- 从智能合约中生成 Java 包装器
- 与以太坊智能合约互动
- 监听以太坊智能合约事件
- 使用 Besu (Pantheon)， Linux 下的 Java 以太坊客户端

- 在 Java 集成测试中运行一个 Hyperledger Besu (Pantheon) 节点
- Web3j 备忘单

面向中等程度用户的文章 {#intermediate-articles}

- 使用星际文件系统在 Java 应用程序中管理存储
- 使用 Web3j 在 Java 中管理 ERC20 代币
- Web3j 交易管理器

面向高等程度用户的使用模式 {#advanced-use-patterns}

- 使用 Eventeum 构建 Java 智能合约数据缓存

Java 项目和工具 {#java-projects-and-tools}

- Hyperledger Besu (Pantheon)（以太坊客户端）
- Web3J（与以太坊客户端交互的库）
- Eventeum（事件侦听器）
- Mahuta（IPFS 开发者工具）

想要获取更多的资源？ 请浏览 ethereum.org/developers。

Java 社区贡献者 {#java-community-contributors}

- IO Builders
- Kauri
- Besu HL 聊天室

---

title: 面向 JavaScript 开发者的以太坊资源
description: 学习如何使用并通过基于 JavaScript 的项目及工具参与以太坊的开发。
lang: zh

sidebar: true

JavaScript 是以太坊生态中最受欢迎的语言之一。 事实上，有一个 团队 致力于尽可能多地在以太坊引入 JavaScript。

有机会在堆栈的所有级别使用 JavaScript（或接近）。

Javascript 和以太坊的交互 {#interact-with-ethereum}

JavaScript API 库 {#javascript-api-libraries}

如果您想要写入 JavaScript 来查询区块链、发送交易等，则最方便的方法是使用 JavaScript API 库。 这些 API 允许开发者轻松与以太坊网络节点交互。

您可以使用这些库与以太坊上的智能合约交互，因此只需要使用 JavaScript 与既有合约交互就可以构建一个 dapp。

参阅：

- Web3.js
- Ethers.js – *包含 JavaScript 和 TypeScript 的完整以太坊钱包的实现和工具。*

智能合约 {#smart-contracts}

作为 JavaScript 开发者，如果希望开发自己的智能合约，请了解：Solidity。 这是最受欢迎的智能合约语言，它在语法上类似于 JavaScript，可能会更容易学习。

关于智能合约的更多信息。

理解协议 {#understand-the-protocol}

以太坊虚拟机 {#the-ethereum-virtual-machine}

用 JavaScript 实现以太坊的虚拟机。 它支持最新的分叉（fork）规则。 分叉规则是指因计划的升级而对 EVM 进行的改造。

它被拆分成各种 JavaScript 软件包，以便您可以更好地了解：

- 帐户
- 区块
- 区块链本身
- 交易
- 更多...

这将有助于您理解像“一个帐户的数据结构是什么？”这样的问题。

如果您倾向于阅读代码，下面的 JavaScript 代码可以很好地成为通读文档的替代方案。

参阅 monorepo
`ethereumjs`

节点和客户端 {#nodes-and-clients}

Ethereumjs 客户端正在开发中。 这将让您深入了解以太坊客户端如何使用您理解的语言运行。

参阅客户端
`ethereumjs-client`

其它项目 {#other-projects}

以太坊的 JavaScript 领域内还有许多其它东西：

- 钱包工具库。
- 生成、导入和导出以太坊密钥的工具。
- 一个 `merkle-patricia-tree` 的实现 - 一种在以太坊黄皮书中描述的数据结构。

发掘最让您感兴趣的： EthereumJS repo

延伸阅读 {#further-reading}

还有哪些社区资源对您有所帮助？ 编辑并添加本页面！

title: 面向 Python 开发者的以太坊资源
description: 学习如何使用并通过基于 Python 的项目及工具参与以太坊的开发
lang: zh
sidebar: true

incomplete: true

学习如何使用并通过基于 Python 的项目及工具参与以太坊的开发

使用以太坊来创建去中心化应用程序 (或称“dapp”)，发挥加密货币和区块链技术的优势。 这些 dapp 可以是值得信赖的，也即一旦被部署到以太坊上，它们将总是按程序运行。 这些应用程序可以控制数字资产，以便创造新的金融应用； 它们可以是去中心化的，也即没有任何单一实体或个人能够控制它们，而且它们几乎是不可能被审查的。

智能合约和 Solidity 语言入门 {#getting-started-with-smart-contracts-and-solidity}

迈出第一步，将 Python 与以太坊集成

需要更基础的入门知识？ 请查看 ethereum.org/learn 或者 ethereum.org/developers。

- 区块链详解
- 理解智能合约
- 编写您的第一个智能合约
- 学习如何编写和部署 Solidity

初学者文章 {#beginner-articles}

- 以太坊开发者指南 (Python)
- 基于 Vyper 的智能合约简介
- 使用 Python 和 Brownie 部署您自己的 ERC20 令牌
- 如何使用 Python Flask 开发以太坊合约？
- Web3.py 简介 ·面向 Python 开发者的以太坊
- 如何通过 Python 和 web3.py 调用智能合约函数？

面向中等程度用户的文章 {#intermediate-articles}

- 面向 Python 程序员的去中心化应用程序开发
- 创建 Python 以太坊接口：第 1 部分
- 基于 Python 的以太坊智能合约开发：完整（入门）教程
- 使用 Brownie 和 Python 部署智能合约
- 使用 Brownie 在 OpenSea 上创建 NFT

面向高等程度用户的使用模式 {#advanced-use-patterns}

- 使用 Python 编译、部署和调用以太坊智能合约
- 使用 Slither 分析 Solidity 智能合约
- Blockchain Fintech 教程：使用 Python 借贷和借贷

Python 项目和工具 {#python-projects-and-tools}

活跃： {#active}

- Web3.py - *用于与以太坊交互的 Python 库*
- Ape - *适用于 Pythonistas、数据科学家和安全专业人员的智能合约开发工具*
- Brownie - *一个用于部署、测试和与以太坊智能合约交互的 Python 框架*
- Vyper - *一个具有 Python 风格的以太坊智能合约编程语言*
- py-evm - *以太坊虚拟机的实现*
- eth-tester - *用于测试基于以太坊应用程序的工具*
- eth-utils - *使用以太坊相关代码库的实用函数工具*

- py-solc-x - *Python 封装的 solidity 0.5.x 编译器*
- py-wasm - *Web 汇编解释器的 Python 实现*
- pydevp2p - *P2P 协议栈的 Python 实现*
- pymaker - _Maker 合约的 Python API _
- siwe - *注册以太坊 (siwe) 以应用 Python*
- 用于测试的智能合约 - *ERC-20、Uniswap 和其他常见合约随附的一个 Python 包，用于使用 Web3.py 进行测试*

已归档/不再维护： {#archived--no-longer-maintained}

- Trinity - *以太坊的 Python 客户端*
- Mamba - *一个用 Vyper 语言编写、编译和部署智能合约的框架*

想要获取更多的资源？ 请查看 ethereum.org/developers。

使用 Python 工具的项目 {#projects-using-python-tooling}

以下基于以太坊的项目使用本页提到的工具。 相关的开源代码库可作为一个很好的参考，例如代码和最佳做法。

- Yearn Finance 和 Yearn Vault Contracts 库
- Curve 和 Curve 智能合约库
- BadgerDAO 和 使用 Brownie 工具链的智能合约
- Sushiswap 使用 Python 来管理和部署他们的归属合约
- 因 Alpha Homora 而知名的 Alpha Finance 使用 Brownie 来测试和部署他们的智能合约

Python 社区贡献者 {#python-community-contributors}

以太坊 Python Discord 社区是一个迅速发展壮大的社区，专门用于讨论上述任何项目和相关主题的资源。

其他汇总列表 {#other-aggregated-lists}

维基百科中关于 Vyper 的可靠资源列表

title: 以太坊的 Ruby 开发者
description: 了解如何使用基于 Ruby 的项目和工具为以太坊进行开发。
lang: zh

sidebar: true

了解如何使用基于 Ruby 的项目和工具为以太坊进行开发。

使用以太坊来创建去中心化应用程序，发挥加密货币和区块链技术的优势。 这些 dapp 可以是无需信任的，这意味着一旦将它们部署到以太坊，它们将始终按程序运行。 他们可以控制数字资产来创建新的金融应用程序。 它们可以是去中心化的，也即没有任何单一实体或个人能够控制它们，而且它们几乎是不可能被审查的。

智能合约和 Solidity 语言入门 {#getting-started-with-smart-contracts-and-solidity}

迈出将 Ruby 与以太坊集成的第一步

需要更基础的入门知识？ 请查看 ethereum.org/learn 或者 ethereum.org/developers。

- 区块链详解
- 理解智能合约
- 编写您的第一个智能合约

- 学习如何编写和部署 Solidity

初学者文章 {#beginner-articles}

- 终于明白以太坊账户了
- 最后使用 MetaMask 对 Rails 用户进行身份验证
- 使用 Ethereum 登录 - Ruby 库和 Rails 示例发布
- 如何使用 Ruby 连接到以太坊网络
- 如何在 Ruby 中生成新的以太坊地址

面向中等程度用户的文章 {#intermediate-articles}

- 用 Ruby 写的区块链 App
- 使用连接到以太坊的 Ruby 来执行智能合约

Ruby 项目和工具 {#ruby-projects-and-tools}

活跃的 {#active}

- eth.rb - *Ruby 库和 RPC 客户端，用于处理以太坊账户、消息, 和交易*
- keccak.rb - *以太坊使用的 Keccak (SHA3) 哈希*
- siwe-ruby - *使用 Ruby 来实现登录以太坊*
- siwe_rails - *添加 SIWE 本地登录路由的 Rails gem*
- siwe-rails-examples - *SIWE 示例使用 Ruby on Rails 和自定义 控制器*
- omniauth-siwe - _OmniAuth strategy 用于登录以太坊 (SIWE) _
- omniauth-nft - *OmniAuth strategy 用于通过 NFT 所有权进行身份验证*
- ethereum-on-rails - *Ethereum on Rails 模板，允许连接 MetaMask 到 Ruby on Rails*

存档/不再维护 {#archived--no-longer-maintained}

- web3-eth - *用 Ruby 调用以太坊节点的 RPC 方法*
- ethereum_tree - _Ruby 库，用于根据 BIP32 标准从分层确定性钱包生成 ETH 地址 _
- etherlite - *Ruby on Rails 的以太坊集成*
- ethereum.rb - *Ruby 以太坊客户端使用 JSON-RPC 接口发送交易 ， 创建合约并与之交互以及与以太坊节点一起使用的有用工具包*
- omniauth-ethereum.rb - *以太坊平台为实施 OmniAuth 提供了策略*

想要获取更多的资源？ 查看我们的开发者之家。

Ruby 社区贡献者 {#ruby-community-contributors}

Ethereum Ruby Telegram group 是一个快速发展的社区，是讨论上述任何项目和相关主题的专用资源。

title: 面向 Rust 开发者的以太坊
description: 学习如何使用并通过基于 rust 的项目及工具参与以太坊的开发
lang: zh
sidebar: true

incomplete: true

学习如何使用并通过基于 rust 的项目及工具参与以太坊的开发

使用以太坊来创建去中心化应用程序（或称“dapp”），发挥加密货币和区块链技术的优势。 这些 dapp 可以是值得信赖的，也即一旦被部署到以太坊上，它们将总是按程序运行。 这些应用程序可以控制数字资产，以便创造新的金融应用； 它们可以是去中心化的，也即没有任何单一实体或个人能够控制它们，而且它们几乎是不可能被审查的。

智能合约和 Solidity 语言入门 {#getting-started-with-smart-contracts-and-solidity}

迈出第一步，将 Rust 与以太坊进行集成

需要更基础的入门知识？ 请查看 ethereum.org/learn 或者 ethereum.org/developers。

- 区块链详解
- 理解智能合约
- 编写您的第一个智能合约
- 学习如何编写和部署 Solidity

初学者文章 {#beginner-articles}

- 选择以太坊客户端
- Rust 以太坊客户端 * **注意 OpenEthereum 已被废弃并已停止维护。** 请谨慎使用，最好切换至其他客户端实现。
- 使用 Rust 向以太坊发送交易
- Parity 以太坊客户端智能合约简介
- 设置 Oasis SDK 开发环境
- 一步步教您如何用 Rust 为 Kovan 编写 Wasm 合约

面向中等程度用户的文章 {#intermediate-articles}

- Rust-Web3 相关文档
- Rust-Web3 工作示例

面向高等程度用户的使用模式 {#advanced-use-patterns}

- pwasm_ethereum 外部库与类以太坊网络交互
- 使用 JavaScript 和 Rust 搭建去中心化聊天室
- 使用 Vue.js 和 Rust 构建一个去中心化待办事项应用程序
- 从 Enigma 开始 - 用 Rust 编程语言
- Secret Contracts 介绍
- 在 Oasis (Compound) 上部署 Solidity 合约

Rust 项目和工具 {#rust-projects-and-tools}

- pwasm-ethereum - *与类以太坊网络交互的外部库集合*
- Lighthouse - *以太坊快速共识层客户端*
- Ethereum WebAssembly
- oasis_std - *OASIS 应用程序界面参考*
- Solaris
- SputnikVM - *采用 Rust 实现以太坊虚拟机*
- rust-web3 - _ Web3.js 库 Rust 语言实现_
- Wavelet - *Rust 语言的 Wavelet 智能合约*
- Foundry - *以太坊应用开发工具包*

- Ethers_rs- *可实现以太坊库和钱包功能*
- evm_rs- *采用 Rust 实现以太坊虚拟机*
- SewUp - *一个帮助您以 Rust 语言构建以太坊 Webassembly 合约的库，正如在一个共同后端开发一样*

想要获取更多的资源？ 请浏览 ethereum.org/developers。

Rust 社区贡献者 {#rust-community-contributors}

- Ethereum WebAssembly
- Oasis Gitter
- Parity Gitter
- Enigma

---

title: 扩容
description: 介绍以太坊社区目前正在开发的不同扩容选择。
lang: zh
sidebar: true

sidebarDepth: 3

扩容概述 {#scaling-overview}

随着以太坊使用人数增加，区块链已经达到了一定的容量限制。 这提高了网络使用成本，从而导致需要“扩容解决方案”。 目前正在研究、测试和执行多种解决方案，这些方案采取不同的办法来实现类似的目标。

扩容的主要目标是提升交易速度（更快确定交易）和交易吞吐量（提高每秒交易量），而不影响去中心化或安全性（详情请见以太坊愿景）。 在第一层以太坊区块链上，高需求导致交易速度减慢和燃料价格难以持续。 提高网络速度和吞吐量是有意义地大规模采用以太坊的基础。

虽然速度和吞吐量很重要，但实现这些目标的扩容解决方案必须保持去中心化和安全性。 降低节点运营商的进入门槛，对于防止向不安全的中心化计算能力发展至关重要。

从概念上说，我们首先将扩容分为链上扩容和链外扩容两类。

前提条件 {#prerequisites}

你应对所有基础性课题有很好的了解。 实施扩容解决方案是一项先进的任务，因为该技术没有经过多少实践检验，还在进一步研发中。

链上扩容 {#on-chain-scaling}

这种扩容方法需要变更以太坊协议（第一层主网）。 区块分片目前是这种扩容方法的重点。

区块分片 {#sharding}

区块分片是一个横向分割数据库以扩展网络承载能力的过程。 在以太坊中，区块分片将通过创建新链（称为“分片”）来减少网络拥塞和增加每秒交易量。 这还可以减轻每位验证者的负担，因为他们不再需要处理整个网络的所有交易。

详细了解区块分片。

链下扩容 {#off-chain-scaling}

链下解决方案与第一层主网分开实施，无需更改现有以太坊协议。 部分解决方案称为“第二层”解决方案，直接从第一层以太坊共识中获得安全性，例如乐观卷叠、零知识卷叠或状态通道。 其他解决方案包括建立不同形式的新链，这些新链单独从主网获得安全性，例如侧链或 plasma 链。 这些解决方案与主网进行通信，但为了实现各种不同的目标，它们获得安全性的方式也有所不同。

二层扩容 {#layer-2-scaling}

此类链下解决方案的安全性来自以太坊主网。

第二层是解决方案的一个集体术语，用于通过处理以太坊主网（第一层）的交易，同时利用主网的强大分散安全模式，帮助扩展您的应用程序。 当网络繁忙时，交易速度会受到影响，这可能使某些类型的去中心化应用程序的用户体验变差。 而且，随着网络越来越繁忙，由于交易发送者的目标是超越对方的出价，燃料价格也随之上升。 这可能会让以太坊的使用成本非常高。

大多数第二层解决方案均围绕着一个服务器或服务器群集，可以将它们称之为节点、验证者、操作者、排序器、区块生产者或类似术语。 根据实施情况，这些第二层节点可由使用它们的个人、企业或实体运行，或者由第三方运营商或一大群个人（与主网相似）运行。 一般而言，交易会提交给第二层节点，而非直接提交给第一层（主网）。 对于部分解决方案， 第二层实例会将它们分组，，然后锚定到第一层，之后受第一层保护且不能更改。 对于不同的第二层技术和实施而言，如何做到这一点的细节差异很大。

某个特定的第二层实例可能是开放的，由许多应用程序共享，也可能由一个项目部署，专供支持该项目的应用程序。

为什么需要第二层？ {#why-is-layer-2-needed}

- 每秒增加交易量会大大提高用户体验，并减少以太坊主网上的网络拥塞情况。
- 卷叠就是将多个交易打包到一个交易中，然后发到以太坊主网上，这为用户减少了燃料费用。以太坊将更具包容性，任何人都可以用得起以太坊。
- 关于可扩容性的任何更新都不应以分散安全性为代价 - 第二层建立在以太坊的基础上。
- 有一些特定应用的第二层网络，在大规模处理资产时有它们自己的效率。

卷叠 {#rollups}

卷叠在第一层之外执行交易执行任务，并在达成共识时，在第一层公开数据。 由于交易数据包含在第一层区块中，因此可以通过原生的以太坊安全性来保证卷叠的安全性。

有两种不同安全模式的卷叠：

- **乐观卷叠**：假设交易在默认情况下有效，并且在遇到挑战的情况下只通过**欺诈证明**运行计算。 关于乐观卷叠的更多信息。
- **零知识卷叠**：在链下运行计算并向链上提交**有效性证明**。 关于零知识卷叠的更多信息。

状态通道 {#channels}

状态通道采用多签合约，使参与者能够在链下快速自由地进行交易，然后再与主网结算。 这将最大限度地减少网络拥塞、费用和延迟。 现在有两种通道：状态通道和支付通道。

了解更多关于状态通道的信息。

侧链 {#sidechains}

侧链是与主网并行运行且兼容以太坊虚拟机的独立区块链。 它们通过双向桥接与以太坊兼容，按照自行选择的共识规则和区块参数运行。

了解更多有关侧链的信息。

以太坊 Plasma 扩容解决方案 {#plasma}

Plasma 是一条独立的区块链，锚定至以太坊主链，并使用欺诈证明（如乐观卷叠）来仲裁争议。

了解更多关于以太坊 Plasma 扩容解决方案的信息。

Validium {#validium}

Validium 链使用诸如零知识汇总之类的有效性证明，但数据不存储在一层以太坊主链上。 这可以导致每条 Validium 链每秒处理 10k 事务，并且可以并行运行多个链。

详细了解 Validium。

为何需要如此多扩容解决方案？ {#why-do-we-need-these}

- 多重解决方案有助于减少网络任意部分的总体阻塞情况，也可防止单点故障。
- 整体大于各部分的总和。 不同的解决方案可以同时存在，并且可以协同工作，对未来的交易速度和吞吐量产生指数效应。
- 并非所有解决方案都需要直接利用以太坊共识算法，替代办法或许能带来难以获得的好处。
- 没有一个单独的扩容方案可以完全满足以太坊愿景。

更愿意通过视频学习？ {#visual-learner}

请注意，视频中使用"第二层"这一术语指代所有链下扩容解决方案。 而我们通常所说的"第二层"是指通过第一层主网共识获得安全性的链下解决方案。

延伸阅读 {#further-reading}

- 以卷叠为中心的以太坊路线图 *Vitalik Buterin*
- 有关以太坊第二层扩容解决方案的最新分析
- 评估以太坊第二层扩容解决方案：一个比较框架
- 卷叠不完全指南
- 以太坊赋能的零知识卷叠：强者
- "乐观卷叠"对比"零知识卷叠"
- 零知识区块链的可扩展性
- 为什么卷叠 + 数据分片是高可扩展性的唯一可持续的解决办法

还有哪些社区资源对您有所帮助？ 请编辑本页面并添加！

title: 乐观卷叠
description: 乐观卷叠介绍
lang: zh

sidebar: true

前提条件 {#prerequisites}

您应该很好地了解所有基本主题，并且对以太坊扩容具有高层次的理解。 实现诸如卷叠之类的扩容解决方案是一个高阶主题，因为该技术没有经过多少实战测试，还在继续研发中。

乐观卷叠 {#optimistic-rollups}

乐观卷叠与第二层的以太坊主链同步进行。 它们可以提供可扩容性方面的改进，因为它们默认不做任何计算。 相反，它们会在一笔交易之后向主网提议新的状态或公证交易。

通过乐观卷叠，交易以 `calldata` 的形式写入以太坊主网，通过减少燃料成本进一步优化。

由于计算是使用以太坊缓慢而昂贵的部分，乐观卷叠可以改进高达 10-100 倍的可扩容性，具体取决于交易。 在引入分片链后，这个数字还会大大增加，因为如果交易有争议，将会产生更多的数据。

对交易提出异议 {#disputing-transactions}

乐观卷叠并不实际计算交易，所以需要有一个机制来确保交易是合法而不是欺诈性的。 这正是欺诈证明的用武之地。 如果有人注意到有欺诈性交易，卷叠将执行防欺诈，并使用可用的状态数据运行交易的计算。 这意味着您的交易确认等待时间可能比零知识卷叠更长，因为交易可能受到质疑。

显示如何处理在以太坊的乐观卷叠中发生的欺诈性交易的图表

您运行欺诈证明计算需要消耗的燃料甚至可以报销。 来自 Optimism 的 Ben Jones 介绍了现有的联系制度。

"任何人都有可能能够采取某种行动，您为了保证您的资金安全，必须证明该行动是否具有欺诈性，因此这个人就必须缴纳保证金。 这基本上等同于您拿出一些以太币，将其锁定，然后说 "嘿，我保证说实话"... 如果我不说实话，欺诈被证实，这笔钱就会被扣减。 这笔钱不仅有一部分会被扣减，而且还有一部分将支付人们做欺诈证明时所花的燃料。

您可以看到奖惩措施：参与者因欺诈而受到惩罚，也会因提供欺诈证明而获得补偿。

优点和缺点 {#optimistic-pros-and-cons}

优点	缺点
您能在以太坊第一层做的任何事情，都能用乐观卷叠做，因为它兼容 EVM 和 Solidity。	由于可能存在对欺诈的质疑，链上交易的等待时间会很长。
所有交易数据都存储在第一层链上，这意味着它是安全和分散的。	运营者可以影响交易顺序。

乐观卷叠的直观解释 {#optimistic-video}

观看 Finematics 解说乐观卷叠的视频：

使用乐观卷叠 {#use-optimistic-rollups}

存在多种乐观卷叠的实现方式，您可以整合到您的去中心化应用程序中。

乐观卷叠相关阅读

- 关于 Optimistic Rollup，你所需要知道的一切
- Optimistic rollup 上的 EthHub
- Arbitrum 指南
- Optimistic Rollup 究竟是如何起作用的？
- 深入研究 OVM

---

title: Plasma 扩容解决方案
description: 这是一篇来源于以太坊社区的关于使用 plasma 来实现扩容解决方案的介绍文章
lang: zh
sidebar: true

incomplete: true

sidebarDepth: 3

Plasma 是一条独立的区块链，锚定至以太坊主链，并使用欺诈证明（如乐观卷叠）来仲裁争议。 这些链有时被称为 "子"链，因为它们本质上是父链以太坊区块链的较小复制。 Merkle 树能够创建这些链的无限堆栈，可以从父链（包括主网）上分流带宽。 它们通过欺诈证明来确保安全性，每条子链都有自己的区块验证机制。

前提条件 {#prerequisites}

您应该对所有以太坊基础有很好的理解，并对“以太坊扩容” 有着一定的理解。 实现诸如 Plasma 之类的扩展解决方案是一个高阶主题，因为该技术没有经过多少实战测试，而且还在继续研究和开发。

优点和缺点 {#pros-and-cons}

优点	缺点
高吞吐量，每笔交易成本较低。	不支持通用计算。 只支持特定逻辑的基本通证转账、兑换和几种其他交易类型。
适合任意用户之间的交易（如果双方都建立在 Plasma 链上，每个用户都几乎没有成本）	需要定期监视（有效性要求）或者委托其他人监视网络，从而确保你的资金安全。
	依靠一个或多个运营者来存储数据，并根据其需求提供数据。
	为了等待挑战期，提款会延迟几天。 对于可替换的资产，这可以由流动性提供者来缓解，但需要支付相关的资本成本。

使用 Plasma {#use-plasma}

多个项目使用了 Plasma 技术，你可以将其整合到你的去中心化应用程序中。

- OMG 网络
- Polygon（原 Matic 网络）
- Gluon
- LeapDAO

延伸阅读 {#further-reading}

- Plasma 上的 EthHub
- 学习 Plasma

您知道有什么社区资源帮助过您吗？ 编辑并添加本页面！

title: 以太坊支持
description: 在以太坊生态系统中获得支持。
sidebar: true

lang: zh

以太坊支持 {#support}

以太坊官方支持 {#official-support}

您正在寻求以太坊官方支持吗？ 您应该知道的第一件事是以太坊是去中心化的。 这意味着以太坊不属于任何中心化组织、实体或个人，因此没有官方支持渠道。

了解以太坊的去中心化性质至关重要，因为任何自称是以太坊官方支持人员的人都可能在试图诈骗您！ 预防诈骗的最好办法就是自学并认真对待安全问题。

以太坊安全和预防诈骗措施

学习以太坊基础知识

尽管缺少官方支持，但以太坊生态系统中的许多团体、社区和项目都乐于提供帮助。

钱包支持 {#wallet-support}

您的钱包有问题吗？ 大多数钱包都有专门的支持团队可以提供帮助：

- MetaMask
- Argent
- MyEtherWallet

这并不是完整的列表。 需要帮助寻找特定钱包的支持？ 加入 ethereum.org Discord，我们会尝试帮助您。

正在寻找以太坊钱包？ 深入了解我们完整的以太坊钱包列表。

构建去中心化应用程序 {#building-support}

构建这类应用可能很困难。 以下提供了一些专注于开发的平台，其中有经验丰富的以太坊开发人员，他们会乐意提供帮助。

- CryptoDevs discord
- 以太坊 StackExchange
- StackOverflow
- Web3 大学

您还可以在我们的以太坊开发者资源部分找到文档和开发指南。

工具 {#dapp-tooling}

您的问题是否与某个特定的工具、项目或程序库有关？ 大多数项目都有专用于用户支持的聊天服务器或论坛。

以下是一些比较常见的例子：

- Solidity
- ethers.js
- web3.js
- 安全帽
- Truffle
- Alchemy

运行节点 {#node-support}

如果您正在运行节点或验证者节点，下面是一些专门帮助您入门的社区。

- EthStaker discord
- EthStaker reddit

大多数构建以太坊客户端的团队也有专门的、面向公众的平台，您可以在这些平台获取支持和提问。

执行客户端 {#execution-clients}

- Geth
- Nethermind
- Besu
- Erigon

共识客户端 {#consensus-clients}

- Prysm
- Nimbus
- Lighthouse
- Teku
- Lodestar

您还可以在此处了解如何运行节点。

常见问题 {#faq}

我把以太币发到了错误的钱包中 {#wrong-wallet}

以太坊上发送的交易是不可逆的。 如果您将以太币发送到错误的钱包，很遗憾，没有办法追回这些资金。 以太坊不属于任何中心化组织、实体或个人，这意味着没有人可以逆转交易。 因此，在发送交易之前，一定要仔细检查您的交易，这一点至关重要。

如何获取以太坊的赠品？ {#giveaway-scam}

以太坊赠品是用来窃取您以太币的骗局。 不要被看起来好得令人难以置信的优惠所诱惑 — 如果您将以太币发送到赠品地址，您不但收不到赠品，还将无法追回您的资金。

关于预防诈骗的更多信息

我的交易被卡住了 {#stuck-transaction}

如果你提交的交易费低于网络需求，以太坊上的交易有时会被卡住。 许多钱包提供了一个选项，即以更高的交易费重新提交同一项交易，使该交易得以处理。 或者，您可以发送交易到您自己的地址，并使用与待定交易相同的随机数，以取消待定交易。

如何在 MetaMask 上加速或取消待定交易

如何取消待定以太坊交易

我如何开采以太坊？ {#mining-ethereum}

如果您尚未开采以太坊，我们不建议购买挖矿设备。 2022 年第 2 季度，将发生合并，将以太坊从工作量证明转换为权益证明。 这就意味着以太坊将不再可能被开采。 如果您仍然对开采以太坊感兴趣，可以从采矿社区寻求帮助，例如 /r/EtherMining。

title: 侧链
description: 这是一篇来源于以太坊社区的关于使用侧链来实现扩容解决方案的介绍文章
lang: zh
sidebar: true
incomplete: true

sidebarDepth: 3

侧链是一个独立的区块链，与以太坊主网并行且独立地运行。 它有自己的共识算法（例如权威证明、委托权益证明、拜占庭容错）。 它通过双向主网来实现链接。

让人们对侧链感兴趣的是，它的工作环境和主链相同，因为同样基于 EVM（以太坊虚拟机）。 它不使用以太坊，它自己就是以太坊。 这意味着如果想要在侧链上使用 dapp ，只需将它部署在侧链上。 它的外观、给人的感受和行为与主链相似，仍然通过 Solidity 写合约，并提供 Web3 API 完成互动。

前提条件 {#prerequisites}

您应该对所有基础有很好的理解，并对“以太坊扩容. 有着一定的理解。

优点和缺点 {#pros-and-cons}

优点	缺点
既有技术。	不够去中心化。
支持通用计算，EVM 兼容。	采用单独的共识机制。 未受一层保护（因此，从技术上而言，它不是第二层）。
	测链验证者达到一定数量可以进行欺诈。

使用侧链 {#use-sidechains}

多个项目提供侧链的实施，以便您集成至 dApp：

- Polygon PoS
- Skale
- Gnosis Chain（原 xDai）

延伸阅读 {#further-reading}

- 侧链上的 EthHub
- Scaling Ethereum Dapps through Sidechains（通过侧链扩容以太坊 Dapp） *Feb 8, 2018 - Georgios Konstantopoulos*

你知道有什么社区资源帮助过你吗？ 编辑并添加本页面！

title: 状态通道
description: 介绍了状态通道和付款渠道，作为以太坊社区目前使用的扩容解决方案。
lang: zh
sidebar: true
incomplete: true

sidebarDepth: 3

通道允许参与者在链下进行`x`次交易，而在链上只能向网络提交两次交易。 这允许极高的交易吞吐量.

前提条件 {#prerequisites}

您应该很好地了解所有基本主题，并且对以太坊扩容有高级别的理解。 实现诸如渠道之类的扩展解决方案是一个高级主题，因为该技术没有经过多少实战测试，而且还在继续研究和开发。

通道 {#channels}

参与者必须将以太坊的一部分状态，如 ETH 存款，锁定在一个多签合约中。 多签合约是一种需要多个私钥的签名（从而达成一致）才能执行的合约。

以这种方式锁定状态是第一个交易，并打开了通道。 然后参与者可以在链下快速自由地进行交易。 当互动结束后，提交最后一笔链上交易，解锁状态。

适用于:

- 需要大量状态更新
- 当参与者数量为公开时
- 当参与者始终在线时

现在有两种通道：状态通道和支付通道。

状态通道 {#state-channels}

状态通道或许最好通过示例进行解释，比如井字棋：

1. 在以太坊主链上创建一个井字游戏智能合约 "法官"，它了解井字游戏的规则，并能识别爱丽丝和鲍勃是我们游戏中的两个玩家。 该合约持有 1 个 ETH 的奖励。
2. 然后，Alice 和 Bob 开始玩游戏，打开状态通道。 每一个动作都会创建一个包含“nonce”的链下交易，nonce 代表这些动作发生的顺序。
3. 当有赢家时，他们通过向审判合约提交最终状态（如交易清单）来关闭通道，这时候只需要支付一笔交易费用。 审判者确保双方签署这一“最后状态”，并等待一段时间以确保没有人合法地对结果提出质疑，然后向 Alice 支付 1 ETH 的奖励。

支付通道 {#payment-channels}

仅处理支付的简化状态通道（例如 ETH 转账）。 允许两个参与者之间的链下转账，只要他们的转账总额不超过存入的代币数量。

优点和缺点 {#channels-pros-and-cons}

优点	缺点
在主网上即时提款/结算（如果通道的双方合作）。	对于偶尔转账给对方的用户来说，创建和结算通道的时间和经济成本都比较高，不太友好。
可能出现极高的吞吐量	需要定期监视（有效性要求）或者委托其他人监视网络，从而确保你的资金安全。
每笔交易成本最低 - 适合主流的小额支付	必须在开启的支付通道中锁定资金
	不支持开放式参与

使用状态通道 {#use-state-channels}

多个项目提供了您可以整合到您的 dapp 的状态频道的实现方式：

- Connext
- Kchannels
- Perun
- Raiden
- Statechannels.org

延伸阅读 {#further-reading}

状态通道

- 状态通道上的 EthHub
- Making Sense of Ethereum's Layer 2 Scaling Solutions: State Channels, Plasma, and Truebit（理解以太坊的第二层扩容解决方案：状态通道、Plasma 和 Truebit） *– Josh Stark, Feb 12 2018*
- State Channels - an explanation（状态通道）- 解释 *Nov 6, 2015 - Jeff Coleman*
- 状态通道的基础知识 *District0x*

支付通道

- 支付通道的 EthHub

你知道有什么社区资源帮助过你吗？ 编辑并添加本页面！

title: Validium
description: 介绍 Validium - 以太坊社区目前使用的一种扩容解决方案。
lang: zh
sidebar: true
incomplete: true

sidebarDepth: 3

使用 zk-rollups 等有效性证明，但数据不存储在第一层以太坊主链上。 这可能导致每个 validium 链每秒有 1 万次交易，并且多个链可并行运行。

前提条件 {#prerequisites}

您应对所有基础性课题有很好的了解，并且对以太坊扩容有深入的了解。 实施 Validium 等扩容解决方案是一个先进的课题，因为这种技术没有经过充分的实践检验，仍将继续进行研发。

优点和缺点 {#pros-and-cons}

优点	缺点
没有提款延迟（链上/跨链转账没有延迟）；因此，资本效益更高。	对通用计算/智能合约的支持有限；需要专用语言。
在高价值应用程序中，不容易受到可防欺诈系统所面临的某些经济攻击。	生成零知识证明需要很高的计算能力；对于低吞吐量的应用程序来说不具成本效益。
	主观终结时间较慢（生成零知识证明的时间为 10-30 分钟）（但由于没有争议时间延迟，所以到完全终结的时间较快）。
	生成证明需要链下数据在任何时候都可用。

使用 Validium {#use-validium}

多个项目提供 Validium 的部署，您可以整合到您的去中心化应用中：

- Starkware
- Matter Labs zkPorter

延伸阅读 {#further-reading}

- Validium 和二层 2 x 2 矩阵，第 99 期

你知道有什么社区资源帮助过你吗？ 编辑并添加本页面！

title: 零知识卷叠
description: 零知识卷叠介绍
lang: zh

sidebar: true

前提条件 {#prerequisites}

您应该很好地了解所有基本主题，并且对以太坊扩容具有高层次的理解。 实现诸如卷叠之类的扩容解决方案是一个高阶主题，因为该技术没有经过多少实战测试，还在继续研发中。

零知识卷叠 {#zk-rollups}

零知识卷叠将数百个链下交易捆绑（或“卷叠”），并生成一个加密证明。 这些证明可以采用 SNARK（简洁的非交互式知识论证）或 STARK（可扩展的透明知识论证）的形式。 “简洁的非交互式知识论证”和“可扩展的透明知识论证”均称为有效性证明，并发布到第一层。

零知识卷叠智能合约在第二层维护所有转账的状态，这个状态只能用有效性证明来更新。 这意味着零知识卷叠只需要有效性证明，而不是所有交易数据。 使用零知识卷叠验证一个区块会更快、更便宜，因为包含的数据更少。

通过零知识卷叠，当资金从第二层转移到第一层时不会有任何延迟，因为零知识卷叠智能合约接受的有效性证明已经验证了资金。

零知识卷叠处于第二层，可以优化以进一步减小交易的大小。 例如，一个帐户由一个索引而不是一个地址来表示，这就把一个交易从 32 个字节减少到只有 4 个字节。 交易也作为 `calldata` 写入以太坊，可减少燃料消耗。

优点和缺点 {#zk-pros-and-cons}

优点	缺点
完成时间更快，因为一旦有效性证明被送到主链上，状态将即时验证。	有一些不支持以太坊虚拟机。
不太容易受到乐观卷叠可能容易受到的经济攻击。	有效性证明的计算量很大 -- 对于链上活动很少的应用来说，不值得这样做。
安全且去中心化，因为恢复状态所需的数据储存在第一层链上。	运营者可以影响交易顺序

零知识卷叠的直观解释 {#zk-video}

观看 Finematics 解说零知识卷叠的视频：

使用零知识卷叠 {#use-zk-rollups}

存在多种零知识卷叠的实现方式，您可以整合到您的去中心化应用程序中。

零知识卷叠相关阅读

- 什么是零知识 Rollup？
- Zk-rollup 上的 EthHub
- “简洁的非交互式知识论证”对比“可扩展的透明知识论证”

---

title: 安全性
description: 以太坊开发者的安全考虑
lang: zh

sidebar: true

以太坊智能合约是极为灵活的。它能够存储超过非常大量的虚拟货币（超过十亿美元），并且根据先前部署的智能合同运行不可修改的代码。 虽然这创造了一个充满活力和创造性的生态系统，但其中包含的无信任、相互关联的智能合约，也吸引了攻击者利用智能合约中的漏洞和以太坊中的未知错误来赚取利润。 智能合约代码*通常*无法修改来修复安全漏洞，因此从智能合约中被盗窃的资产是无法收回的，且被盗资产极难追踪。 由于智能合约问题而被盗或丢失的价值总额已经达到了 10 亿美元。 一些因为智能合约代码编写错误导致较大经济损失的例子：

- 钱包问题 #1 - 3000 万美金损失
- 钱包问题 #2 - 3 亿美金锁定
- TheDAO 被黑，360 万 ETH 被盗！ 目前价值超过 10 亿美元

前置要求 {#prerequisites}

这个章节将介绍智能合约安全问题，以确保你在处理安全问题之前熟悉智能合约。

如何编写更安全的智能合约代码 {#how-to-write-more-secure-smart-contract-code}

在启动任何代码到主网之前，采取足够的预防措施来保护你的智能合约所委托的任何有价值的东西是很重要的。 在本文中，我们将讨论一些特定的攻击，提供学习了解更多攻击类型的资源，并为你提供一些基本工具和最佳实践，以确保你的合约正确、安全地运行。

审核不是完美的解决方案 {#audits-are-not-a-silver-bullet}

几年前，用于编写、编译、测试和部署智能合约的工具还非常不成熟，许多项目被随意地编写 Solidity 代码，并将其交给审查员，审查员将审查代码以确保其安全并按预期运行。 在 2020 年，编写 Solidity 代码的开发过程和工具得到了显著改善，不仅可以确保项目更易于管理，而且能够组成项目安全性的一部分。 仅仅在项目结束时对你的智能合约进行审计已经不足以成为项目的唯一安全考虑。 **安全性来源于适当的设计和开发过程**，所以在你编写第一行智能合约代码之前，安全性就应该被考虑。

智能合约开发过程 {#smart-contract-development-process}

最低安全限度：

- 所有代码应该被存在于一个版本控制系统当中，例如 git
- 所有的代码修改都应该通过拉取请求来进行
- 所有的拉取请求都应该有至少一个审核员。 *如果这是一个个人项目，请考虑寻找另一位个人作者和一个交易代码审核员。*
- 使用开发以太坊环境（请参阅：Truffle），只需一个命令就可以编译、部署和运行一套针对你的代码的测试
- 已经通过 Mythril 和 Sliter 等基本代码分析工具运行了代码，最好是在合并每个拉取请求之前，比较输出中的差异。
- Solidity 代码编辑器不会发出任何警告
- 你的代码有据可查

上面的这些条目是编写智能合约的一个良好的开始，但是在编写代码过程中还有很多要值得注意。 关于更多条目和条目的解释，请参阅DeFiSafety 提供的过程质量检查表。 DefiSafety 是一个对各种大型的公开以太坊 dApps 进行评论的非官方公共服务提供者。 DeFiSafete 对项目的安全评级等级的一部分包括该项目是否遵守了质量检查表。 遵循这些审核过程：

- 通过可复现的自动化测试，产生更安全的代码
- 审查员将能够更有效地审查你的项目
- 对新开发者友好
- 允许开发者快速迭代、测试并在修改时获得反馈
- 回滚代码的可能性较低

攻击和脆弱性 {#attacks-and-vulnerabilities}

既然你正在使用高效的开发过程编写 Solidity 代码，那么让我们看看一些常见的 Solidity 漏洞问题。

重入攻击 {#re-entrancy}

重入攻击时在编写智能合约代码时应该考虑的最大且最重要的安全问题。 虽然以太坊虚拟机不能同时运行多个合约，一个合约可以调用另一个合约来暂停一个合约的执行和内存状态，直到被重新调用。这时，代码将会继续被正常执行。 暂停和重新启动的过程可能会造成一种被称为“重入攻击”的漏洞。

这是一个容易受到重入攻击的合约：

```
// THIS CONTRACT HAS INTENTIONAL VULNERABILITY, DO NOT COPY
contract Victim {
    mapping (address => uint256) public balances;

    function deposit() external payable {
```

```
        balances[msg.sender] += msg.value;
    }

    function withdraw() external {
        uint256 amount = balances[msg.sender];
        (bool success, ) = msg.sender.call.value(amount)("");
        require(success);
        balances[msg.sender] = 0;
    }
}
```

为了让用户可以退回先前存储在智能合约里的 ETH，这个功能会

1. 读取用户的余额
2. 发送用户的余额
3. 将余额重置为 0，因此它们不能再次提取余额。

如果收到来自一个普通帐户的调用（例如你自己的 MetaMask 帐户），这个函数 msg.sender.call.value() 能够发送你的帐户里的 ETH。 但是，智能合约也能调用其他合约。 如果一个自制的恶意合约调用 `withdraw()`，msg.sender.call.value() 会不仅发送 `amount` 个 ETH，还会暗中调用合约来开始执行代码。 想象这个恶意合约：

```
contract Attacker {
    function beginAttack() external payable {
        Victim(VICTIM_ADDRESS).deposit.value(1 ether)();
        Victim(VICTIM_ADDRESS).withdraw();
    }

    function() external payable {
        if (gasleft() > 40000) {
            Victim(VICTIM_ADDRESS).withdraw();
        }
    }
}
```

调用 Attacker.beginAttack() 会开始循环寻找一些像：

```
0.) 攻击者帐户首先使用 1 个 ETH 并借助 EOA 协议调用 Attacker.beginAttack()
0.) Attacker.beginAttack() 会将 1 个 ETH 存入到受害者帐户中
 1.) 攻击者帐户 ->（调用）Victim.withdraw()
 1.) 受害者帐户调用函数 balanceOf[msg.sender]
 1.) 受害者帐户将自己帐户中的 ETH 发送到攻击者帐户中（该操作将使用默认函数进行）
    2.) 攻击者帐户 ->（调用）Victim.withdraw()
    2.) 受害者帐户调用函数 balanceOf[msg.sender]
    2.) 受害者帐户将自己帐户中的 ETH 发送到攻击者帐户中（该操作将使用默认函数进行）
      3.) 攻击者帐户 ->（调用）Victim.withdraw()
      3.) 受害者帐户调用函数 balanceOf[msg.sender]
      3.) 受害者帐户将自己帐户中的 ETH 发送到攻击者帐户中（该操作将使用默认函数进行）
        4.) 攻击者帐户最终会因受害者帐户中无足够的 ETH 而停止调用函数
      3.) balances[msg.sender] = 0;
    2.) balances[msg.sender] = 0; (受害者帐户中 ETH 已经为 0)
  1.) balances[msg.sender] = 0; (受害者帐户中 ETH 已经为 0)
```

攻击者帐户使用 1 个 ETH 调用 Attacker.beginAttack 函数将会重复攻击受害者帐户，并将赚取远超其提供 ETH 的数量（这些额外的 ETH 会从其他用户帐户的余额中赚取，这样会造成受害者账户余额减少）

如何解决重入攻击（一种错误的方式） {#how-to-deal-with-re-entrancy-the-wrong-way}

防止重入攻击的一种简单方法便是让您的任何代码不与任何智能合约进行交互。 但当你搜索 stackoverflow 时，会发现这段被许多人使用的代码片段：

```
function isContract(address addr) internal returns (bool) {
  uint size;
  assembly { size := extcodesize(addr) }
  return size > 0;
}
```

这似乎有道理：合约本身拥有一些代码，如果调用者还拥有其他代码，则不能允许合约代码去存储数据。 让我们加上它吧：

```
// 该合约存在已知漏洞，请勿复制
contract ContractCheckVictim {
    mapping (address => uint256) public balances;

    function isContract(address addr) internal returns (bool) {
        uint size;
        assembly { size := extcodesize(addr) }
        return size > 0;
    }

    function deposit() external payable {
        require(!isContract(msg.sender)); // <- 新增行
        balances[msg.sender] += msg.value;
    }

    function withdraw() external {
        uint256 amount = balances[msg.sender];
        (bool success, ) = msg.sender.call.value(amount)("");
        require(success);
        balances[msg.sender] = 0;
    }
}
```

现在为了要存入 ETH，你的地址里不能有智能合约的代码。 然而，通过如下的攻击者合约可以很轻松地击败它：

```
contract ContractCheckAttacker {
    constructor() public payable {
        ContractCheckVictim(VICTIM_ADDRESS).deposit(1 ether); // <- 新增行
    }

    function beginAttack() external payable {
        ContractCheckVictim(VICTIM_ADDRESS).withdraw();
    }

    function() external payable {
        if (gasleft() > 40000) {
            Victim(VICTIM_ADDRESS).withdraw();
        }
```

```
    }
}
```

先前的攻击是对合约逻辑的攻击，而这一次则是对以太坊合约部署行为的攻击。 在部署过程中，合约尚未返回要在其地址部署完毕的代码，但在此过程中保留了完整的 EVM 控制阶段。

从技术上讲，可以使用以下代码来防止智能合约调用你的代码：

```
require(tx.origin == msg.sender)
```

然而，这依旧不是一个很好的解决办法。 因为以太坊最令人兴奋的方面之一是它的可组合性，智能合约相互集成并融合、发展。 通过使用上面的代码，你会限制你项目的实用性。

如何解决重入攻击（一种正确的方式） {#how-to-deal-with-re-entrancy-the-right-way}

只需切换存储更新和外部调用的顺序，我们就可以防止启用攻击的重新进入条件。 虽然有可能，回调退却不会给攻击者带来好处，因为 `balances` 存储已经设置为 0。

```
contract NoLongerAVictim {
    function withdraw() external {
        uint256 amount = balances[msg.sender];
        balances[msg.sender] = 0;
        (bool success, ) = msg.sender.call.value(amount)("");
        require(success);
    }
}
```

上面的代码遵循“检查-生效-相互作用”设计模式，它有助于防止冲入。 你可以在这里 [阅读更多关于检查-生效-交互作用的信息]

如何解决重入攻击（核心选择） {#how-to-deal-with-re-entrancy-the-nuclear-option}

任何时候你都会将 ETH 发送到一个不信任的地址或与一个未知合约进行交互（例如调用 `transfer()` 到用户提供的代币地址），你可以自行开启重入。 **通过设计既不发送 ETH 也不调用不信任合约的智能合约，你将防止重入攻击的可能性！**

更多攻击类型 {#more-attack-types}

上述攻击类型包括智能合约编码问题（重入）和以太坊奇数（在合约构造器内运行代码，合约地址才有编码）。 有许多更多的攻击类型需要了解，如：

- 抢跑
- ETH 发送拒绝
- 整数上溢/下溢

延伸阅读:

- [共识智能合约已知攻击] - 对最重要弱点的可读解释，有很多样本代码。
- [SWC 注册] - 适用于以太坊和智能合约的 CWE 的管理列表

安全工具 {#security-tools}

虽然了解以太坊安全基础知识和聘请专业审计公司审查你的代码是无可替代的，但有许多工具可以帮助突出你的代码中的潜在问题。

智能合约安全 {#smart-contract-security}

Slither - ***用 Python 3 编写的 Solidity 静态分析框架。***

- GitHub

MythX - ***以太坊智能合约的安全分析 API。***

- mythx.io
- 相关文档

Mythril - ***EVM 字节码安全分析工具。***

- mythril
- 相关文档

Manticore - ***在智能合约和二进制文件上使用符号执行工具的命令行界面。***

- GitHub
- 相关文档

Securify - ***以太坊智能合约安全分析工具。***

- securify.chainsecurity.com
- Discord

ERC20 Verifier - ***用于检查合约是否符合 ERC20 标准的验证工具。***

- erc20-verifier.openzeppelin.com
- 论坛

形式化验证 {#formal-verification}

有关形式化验证的信息

- How formal verification of smart-contacts works *July 20, 2018 - Brian Marick*
- How Formal Verification Can Ensure Flawless Smart Contracts *Jan 29, 2018 - Bernard Mueller*

使用工具 {#using-tools}

智能合约安全分析最受欢迎的两个工具是：

- Slither by Trail of Bits（托管版本：Crytic）
- Mythril by ConsenSys（托管版本：MythX）

两者都是分析你的代码和报告问题的有用工具。 每个人都有一个 [commercial] 托管版本，但也可以免费在本地运行。 下面是如何运行 Slither 的一个快速示例，这个示例是在方便的 Docker 映像 `trailofbits/eth-security-toolbox` 中提供的。 如果你还没有安装 ，你将需要 安装 Docker。

```
$ mkdir test-slither
$ curl
https://gist.githubusercontent.com/epheph/460e6ff4f02c4ac582794a41e1f103bf/raw/9e761af793
d4414c39370f063a46a3f71686b579/gistfile1.txt > bad-contract.sol
$ docker run -v `pwd`:/share  -it --rm trailofbits/eth-security-toolbox
docker$ cd /share
docker$ solc-select 0.5.11
docker$ slither bad-contract.sol
```

将生成此输出：

```
ethsec@1435b241ca60:/share$ slither bad-contract.sol
INFO:Detectors:
Reentrancy in Victim.withdraw() (bad-contract.sol#11-16):
    External calls:
    - (success) = msg.sender.call.value(amount)() (bad-contract.sol#13)
    State variables written after the call(s):
    - balances[msg.sender] = 0 (bad-contract.sol#15)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-
vulnerabilities
INFO:Detectors:
Low level call in Victim.withdraw() (bad-contract.sol#11-16):
    - (success) = msg.sender.call.value(amount)() (bad-contract.sol#13)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls
INFO:Slither:bad-contract.sol analyzed (1 contracts with 46 detectors), 2 result(s) found
INFO:Slither:Use https://crytic.io/ to get access to additional detectors and GitHub
integration
```

Slither 已经在这里确定了重新进入的可能性。 确定问题可能出现的关键行号，并为我们提供一个链接以获得有关问题的更多细节：

> 参考：https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities

允许你快速了解你代码的潜在问题。 像所有自动化测试工具一样，Slither 并不完美，它在报告一侧出现了太多错误。 即使在不存在可开发的脆弱性的情况下，它也可以就潜在的重返提出警告。 在代码更改之间在 Slither 输出中查看 DIFERENER 常常是非常有启发性的，可以帮助发现比等待你的项目代码完成早得多的漏洞。

延伸阅读 {#further-reading}

智能合约安全性最佳实践指南

- consensys.github.io/smart-contract-best-practices/
- GitHub
- 安全性建议和最佳实践合集

智能合约安全验证标准 (SCSVS)

- securing.github.io/SCSVS/

你知道有什么社区资源帮助过你吗？ 编辑并添加本页面！

相关教程 {#related-tutorials}

- 安全的开发工作流程
- 如何使用 Slither 查询智能合约漏洞

- 如何使用 Manticore 查询智能合约漏洞
- 安全准则
- 令牌安全

---

title: 详解智能合约
description: 深入解读智能合约：函数、数据和变量。
lang: zh

sidebar: true

智能合约是一种在以太坊某个地址上运行的程序。 它们是由数据和函数组成的，可以在收到交易时执行。 以下概述一个智能合约的组成。

前置要求 {#prerequisites}

确保您已经先阅读了智能合约。 本文档假设您已经熟悉某种编程语言，例如 JavaScript 或 Python。

数据 {#data}

任何合约数据必须分配到一个位置：要么是`存储`，要么是`内存`。 在智能合约中修改存储消耗很大，因此您需要考虑数据在哪里存取。

存储 {#storage}

持久性数据被称之为存储，由状态变量表示。 这些值被永久地存储在区块链上。 您需要声明一个类型，以便于合约在编译时可以跟踪它在区块链上需要多少存储。

```
// Solidity example
contract SimpleStorage {
    uint storedData; // State variable
    // ...
}
```

```
# Vyper example
storedData: int128
```

如果用过面向对象编程语言，应该会熟悉大多数类型。 但如果是刚接触以太坊开发，则会发现 `address` 是一个新类型。

一个 `address` 类型可以容纳一个以太坊地址，相当于 20 个字节或 160 位。 它以十六进制的形式返回，前导是 0x。

其它类型包括：

- 布尔
- 整数（integer）
- 定点数（fixed point numbers）
- 固定大小的字节数组（fixed-size byte arrays）
- 动态大小的字节数组（dynamically-sized byte arrays）
- 有理数和整数常量（Rational and integer literals）
- 字符常量（String literals）
- 十六进制常量（Hexadecimal literals）

- 枚举（Enums）

了解更多信息，请参阅文档：

- 查看 Vyper 类型
- 查看 Solidity 类型

内存 {#memory}

仅在合约函数执行期间存储的值被称为内存变量。 由于这些变量不是永久地存储在区块链上，所以它们的使用成本要低得多。

在 Solidity 文档中了解更多关于以太坊虚拟机如何存储数据（存储、内存和栈）。

环境变量 {#environment-variables}

除了在自己合约上定义的变量之外，还有一些特殊的全局变量。 它们主要用于提供有关区块链或当前交易的信息。

示例：

属性	状态变量	描述
`block.timestamp`	uint256	当前区块的时间戳
`msg.sender`	地址	消息的发送者（当前调用）

函数 {#functions}

用最简单的术语来说，函数可以获得信息或设置信息，以响应传入的交易。

有两种函数调用方式：

- `internal` – 不会创建以太坊虚拟机调用
  - Internal 函数和状态变量只能在内部访问（只能在合约内部或者从其继承的合约内部访问）。
- `external` – 会创建以太坊虚拟机调用
  - External 函数是合约接口的一部分，这意味着他可以被其它合约和交易调用。 一个 external 函数 `f` 不可以被内部调用（即 `f()` 不行，但 `this.f()` 可以）。

它们可以是 `public` 或 `private`

- `public` 函数可以在合约内部调用或者通过消息在合约外部调用
- `private` 函数仅在其被定义的合约内部可见，并且在该合约的派生合约中不可见。

函数和状态变量都可以被定义为 public 或 private

下面是更新合约上一个状态变量的函数：

```
// Solidity example
function update_name(string value) public {
    dapp_name = value;
}
```

- `string` 类型的参数 `value` 传入函数 `update_name`
- 函数声明为 `public`，意味着任何人都能访问它

- 函数没有被声明为 `view`，因此它可以修改合约状态

View 函数 {#view-functions}

这些函数保证不会修改合约数据的状态。 常见的示例是 "getter" 函数 - 例如，它可以用于接收用户的余额。

```
// Solidity 示例
function balanceOf(address _owner) public view returns (uint256 _balance) {
    return ownerPizzaCount[_owner];
}
```

```
dappName: public(string)

@view
@public
def readName() -> string:
  return dappName
```

这些操作被视为修改状态：

1. 写入状态变量。
2. 正在导出事件。
3. 创建其它合约。
4. 使用 `selfdestruct`。
5. 通过调用发送 ether。
6. 调用任何未标记为 `view` 或 `pure` 的函数。
7. 使用底层调用。
8. 使用包含某些操作码的内联程序组。

构造函数 {#constructor-functions}

`constructor` 函数只在首次部署合约时执行一次。 与许多基于类的编程语言中的 `constructor` 函数类似，这些函数常将状态变量初始化到指定的值。

```
// Solidity 示例
// 初始化合约数据，设置 `owner`为合约的创建者。
constructor() public {
    // 所有智能合约依赖外部交易来触发其函数。
    // `msg` 是一个全局变量，包含了给定交易的相关数据，
    // 例如发送者的地址和交易中包含的 ETH 数量。
    // 了解更多：https://solidity.readthedocs.io/en/v0.5.10/units-and-global-variables.html#block-and-transaction-properties
    owner = msg.sender;
}
```

```
# Vyper 示例

@external
def __init__(_beneficiary: address, _bidding_time: uint256):
    self.beneficiary = _beneficiary
    self.auctionStart = block.timestamp
    self.auctionEnd = self.auctionStart + _bidding_time
```

内置函数 {#built-in-functions}

除了自己在合约中定义的变量和函数外，还有一些特殊的内置函数。 最明显的例子是：

- `address.send()` – Solidity
- `send(address)` – Vyper

这使合约可以发送以太币给其它帐户。

编写函数 {#writing-functions}

您的函数需要：

- 参数变量及其类型（如果它接受参数）
- 声明为 internal/external
- 声明为 pure/view/payable
- 返回类型（如果它返回值）

```
pragma solidity >=0.4.0 <=0.6.0;

contract ExampleDapp {
    string dapp_name; // state variable

    // Called when the contract is deployed and initializes the value
    constructor() public {
        dapp_name = "My Example dapp";
    }

    // Get Function
    function read_name() public view returns(string) {
        return dapp_name;
    }

    // Set Function
    function update_name(string value) public {
        dapp_name = value;
    }
}
```

一个完整的合约可能就是这样。 在这里，`constructor` 函数为 `dapp_name` 变量提供了初始化值。

事件和日志 {#events-and-logs}

事件可以让您通过前端或其它订阅应用与您的智能合约通信。 当交易被挖矿执行时，智能合约可以触发事件并且将日志写入区块链，然后前端可以进行处理。

附带说明的例子 {#annotated-examples}

这是一些用 Solidity 写的例子。 如果希望运行这些代码，您可以在 Remix 中调试。

Hello world {#hello-world}

```
// Specifies the version of Solidity, using semantic versioning.
// 了解更多：https://solidity.readthedocs.io/en/v0.5.10/layout-of-source-files.html#pragma
pragma solidity ^0.5.10;

// 定义合约名称 `HelloWorld`。
// 一个合约是函数和数据（其状态）的集合。
// 一旦部署，合约就会留在以太坊区块链的一个特定地址上。
// 了解更多： https://solidity.readthedocs.io/en/v0.5.10/structure-of-a-contract.html
contract HelloWorld {

    // 定义`string`类型变量 `message`
    // 状态变量是其值永久存储在合约存储中的变量。
    // 关键字 `public` 使得可以从合约外部访问。
    // 并创建了一个其它合约或客户端可以调用访问该值的函数。
    string public message;

    // 类似于很多基于类的面向对象语言，
    // 构造函数是仅在合约创建时执行的特殊函数。
    // 构造器用于初始化合约的数据。
    // 了解更多：https://solidity.readthedocs.io/en/v0.5.10/contracts.html#constructors
    constructor(string memory initMessage) public {
        // 接受一个字符变量 `initMessage`
        // 并为合约的存储变量`message` 赋值
        message = initMessage;
    }

    // 一个 public 函数接受字符参数并更新存储变量 `message`
    function update(string memory newMessage) public {
        message = newMessage;
    }
}
```

代币（Token） {#token}

```
pragma solidity ^0.5.10;

contract Token {
    // 一个 `address` 类比于邮件地址 - 它用来识别以太坊的一个帐户。
    // 地址可以代表一个智能合约或一个外部（用户）帐户。
    // 了解更多：https://solidity.readthedocs.io/en/v0.5.10/types.html#address
    address public owner;

    //  `mapping` 是一个哈希表数据结构。
    // 此 `mapping` 将一个无符号整数（代币余额）分配给地址（代币持有者）。
    // 了解更多： https://solidity.readthedocs.io/en/v0.5.10/types.html#mapping-types
    mapping (address => uint) public balances;

    // 事件允许在区块链上记录活动。
    // 以太坊客户端可以监听事件，以便对合约状态更改作出反应。
    // 了解更多： https://solidity.readthedocs.io/en/v0.5.10/contracts.html#events
    event Transfer(address from, address to, uint amount);
```

```
    // 初始化合约数据，设置 `owner`为合约创建者的地址。
    constructor() public {
        // 所有智能合约依赖外部交易来触发其函数。
        // `msg` 是一个全局变量，包含了给定交易的相关数据，
        // 例如发送者的地址和包含在交易中的 ETH 数量。
        // 了解更多：https://solidity.readthedocs.io/en/v0.5.10/units-and-global-variables.html#block-and-transaction-properties
        owner = msg.sender;
    }

    // 创建一些新代币并发送给一个地址。
    function mint(address receiver, uint amount) public {
        // `require` 是一个用于强制执行某些条件的控制结构。
        // 如果 `require` 的条件为 `false`，则异常被触发，
        // 所有在当前调用中对状态的更改将被还原。
        // 学习更多: https://solidity.readthedocs.io/en/v0.5.10/control-structures.html#error-handling-assert-require-revert-and-exceptions

        // 只有合约创建人可以调用这个函数
        require(msg.sender == owner, "You are not the owner.");

        // 强制执行代币的最大数量
        require(amount < 1e60, "Maximum issuance exceeded");

        // 将 "收款人"的余额增加"金额"
        balances[receiver] += amount;
    }

    // 从任何调用者那里发送一定数量的代币到一个地址。
    function transfer(address receiver, uint amount) public {
        // 发送者必须有足够数量的代币用于发送
        require(amount <= balances[msg.sender], "Insufficient balance.");

        // 调整两个帐户的余额
        balances[msg.sender] -= amount;
        balances[receiver] += amount;

        // 触发之前定义的事件。
        emit Transfer(msg.sender, receiver, amount);
    }
}
```

唯一的数字资产 {#unique-digital-asset}

```
pragma solidity ^0.5.10;

// 从其它文件向当前合约中导入符号。
// 本例使用一系列来自 OpenZeppelin 的辅助合约。
// 了解更多：https://solidity.readthedocs.io/en/v0.5.10/layout-of-source-files.html#importing-other-source-files

import "../node_modules/@openzeppelin/contracts/token/ERC721/IERC721.sol";
import "../node_modules/@openzeppelin/contracts/token/ERC721/IERC721Receiver.sol";
import "../node_modules/@openzeppelin/contracts/introspection/ERC165.sol";
import "../node_modules/@openzeppelin/contracts/math/SafeMath.sol";
```

```
// `is` 关键字用于从其它外部合约继承函数和关键字。
// 本例中，`CryptoPizza` 继承 `IERC721` 和 `ERC165` 合约。
// 了解更多：https://solidity.readthedocs.io/en/v0.5.10/contracts.html#inheritance
contract CryptoPizza is IERC721, ERC165 {
    // 使用 OpenZeppelin's SafeMath 库来安全执行算数操作。
    // 了解更多：https://docs.openzeppelin.com/contracts/2.x/api/math#SafeMath
    using SafeMath for uint256;

    // Solidity 语言中的常量状态变量与其他语言类似。
    // 但是必须用一个表达式为常量赋值，而这个表达式本身必须在编译时是一个常量。
    // Learn more: https://solidity.readthedocs.io/en/v0.5.10/contracts.html#constant-
state-variables
    uint256 constant dnaDigits = 10;
    uint256 constant dnaModulus = 10 ** dnaDigits;
    bytes4 private constant _ERC721_RECEIVED = 0x150b7a02;

    // Struct types let you define your own type
    // Learn more: https://solidity.readthedocs.io/en/v0.5.10/types.html#structs
    struct Pizza {
        string name;
        uint256 dna;
    }

    // Creates an empty array of Pizza structs
    Pizza[] public pizzas;

    // Mapping from pizza ID to its owner's address
    mapping(uint256 => address) public pizzaToOwner;

    // Mapping from owner's address to number of owned token
    mapping(address => uint256) public ownerPizzaCount;

    // Mapping from token ID to approved address
    mapping(uint256 => address) pizzaApprovals;

    // You can nest mappings, this example maps owner to operator approvals
    mapping(address => mapping(address => bool)) private operatorApprovals;

    // Internal function to create a random Pizza from string (name) and DNA
    function _createPizza(string memory _name, uint256 _dna)
        // The `internal` keyword means this function is only visible
        // within this contract and contracts that derive this contract
        // Learn more:
https://solidity.readthedocs.io/en/v0.5.10/contracts.html#visibility-and-getters
        internal
        // `isUnique` is a function modifier that checks if the pizza already exists
        // Learn more: https://solidity.readthedocs.io/en/v0.5.10/structure-of-a-
contract.html#function-modifiers
        isUnique(_name, _dna)
    {
        // Adds Pizza to array of Pizzas and get id
        uint256 id = SafeMath.sub(pizzas.push(Pizza(_name, _dna)), 1);

        // Checks that Pizza owner is the same as current user
```

```
        // Learn more: https://solidity.readthedocs.io/en/v0.5.10/control-structures.html#error-handling-assert-require-revert-and-exceptions

        // note that address(0) is the zero address,
        // indicating that pizza[id] is not yet allocated to a particular user.

        assert(pizzaToOwner[id] == address(0));

        // Maps the Pizza to the owner
        pizzaToOwner[id] = msg.sender;
        ownerPizzaCount[msg.sender] = SafeMath.add(
            ownerPizzaCount[msg.sender],
            1
        );
    }

    // Creates a random Pizza from string (name)
    function createRandomPizza(string memory _name) public {
        uint256 randDna = generateRandomDna(_name, msg.sender);
        _createPizza(_name, randDna);
    }

    // Generates random DNA from string (name) and address of the owner (creator)
    function generateRandomDna(string memory _str, address _owner)
        public
        // Functions marked as `pure` promise not to read from or modify the state
        // Learn more: https://solidity.readthedocs.io/en/v0.5.10/contracts.html#pure-functions
        pure
        returns (uint256)
    {
        // Generates random uint from string (name) + address (owner)
        uint256 rand = uint256(keccak256(abi.encodePacked(_str))) +
            uint256(_owner);
        rand = rand % dnaModulus;
        return rand;
    }

    // Returns array of Pizzas found by owner
    function getPizzasByOwner(address _owner)
        public
        // Functions marked as `view` promise not to modify state
        // Learn more: https://solidity.readthedocs.io/en/v0.5.10/contracts.html#view-functions
        view
        returns (uint256[] memory)
    {
        // Uses the `memory` storage location to store values only for the
        // lifecycle of this function call.
        // 了解更多: https://solidity.readthedocs.io/en/v0.5.10/introduction-to-smart-contracts.html#storage-memory-and-the-stack
        uint256[] memory result = new uint256[](ownerPizzaCount[_owner]);
        uint256 counter = 0;
        for (uint256 i = 0; i < pizzas.length; i++) {
            if (pizzaToOwner[i] == _owner) {
```

```
                result[counter] = i;
                counter++;
            }
        }
        return result;
    }

    // 转移 Pizza 和归属关系到其它地址
    function transferFrom(address _from, address _to, uint256 _pizzaId) public {
        require(_from != address(0) && _to != address(0), "Invalid address.");
        require(_exists(_pizzaId), "Pizza does not exist.");
        require(_from != _to, "Cannot transfer to the same address.");
        require(_isApprovedOrOwner(msg.sender, _pizzaId), "Address is not approved.");

        ownerPizzaCount[_to] = SafeMath.add(ownerPizzaCount[_to], 1);
        ownerPizzaCount[_from] = SafeMath.sub(ownerPizzaCount[_from], 1);
        pizzaToOwner[_pizzaId] = _to;

        // 触发继承自 IERC721 合约中定义的事件。
        emit Transfer(_from, _to, _pizzaId);
        _clearApproval(_to, _pizzaId);
    }

    /**
     * 安全转账给定代币 ID 的所有权到其它地址
     * 如果目标地址是一个合约，则该合约必须实现 `onERC721Received`函数，
     * 该函数调用了安全转账并且返回一个 magic value。
     * `bytes4(keccak256("onERC721Received(address,address,uint256,bytes)"))`;
     * 否则，转账被回退。
    */
    function safeTransferFrom(address from, address to, uint256 pizzaId)
        public
    {
        // solium-disable-next-line arg-overflow
        this.safeTransferFrom(from, to, pizzaId, "");
    }

    /**
     * 安全转账给定代币 ID 所有权到其它地址
     * 如果目标地址是一个合约，则该合约必须实现 `onERC721Received` 函数，
     * 该函数调用安全转账并返回一个 magic value
     * `bytes4(keccak256("onERC721Received(address,address,uint256,bytes)"))`;
     * 否则，转账被回退。
     */
    function safeTransferFrom(
        address from,
        address to,
        uint256 pizzaId,
        bytes memory _data
    ) public {
        this.transferFrom(from, to, pizzaId);
        require(_checkOnERC721Received(from, to, pizzaId, _data), "Must implement
onERC721Received.");
    }
```

```
    /**
     * Internal function to invoke `onERC721Received` on a target address
     * The call is not executed if the target address is not a contract
     */
    function _checkOnERC721Received(
        address from,
        address to,
        uint256 pizzaId,
        bytes memory _data
    ) internal returns (bool) {
        if (!isContract(to)) {
            return true;
        }

        bytes4 retval = IERC721Receiver(to).onERC721Received(
            msg.sender,
            from,
            pizzaId,
            _data
        );
        return (retval == _ERC721_RECEIVED);
    }

    // Burns a Pizza - destroys Token completely
    // The `external` function modifier means this function is
    // part of the contract interface and other contracts can call it
    function burn(uint256 _pizzaId) external {
        require(msg.sender != address(0), "Invalid address.");
        require(_exists(_pizzaId), "Pizza does not exist.");
        require(_isApprovedOrOwner(msg.sender, _pizzaId), "Address is not approved.");

        ownerPizzaCount[msg.sender] = SafeMath.sub(
            ownerPizzaCount[msg.sender],
            1
        );
        pizzaToOwner[_pizzaId] = address(0);
    }

    // Returns count of Pizzas by address
    function balanceOf(address _owner) public view returns (uint256 _balance) {
        return ownerPizzaCount[_owner];
    }

    // Returns owner of the Pizza found by id
    function ownerOf(uint256 _pizzaId) public view returns (address _owner) {
        address owner = pizzaToOwner[_pizzaId];
        require(owner != address(0), "Invalid Pizza ID.");
        return owner;
    }

    // Approves other address to transfer ownership of Pizza
    function approve(address _to, uint256 _pizzaId) public {
        require(msg.sender == pizzaToOwner[_pizzaId], "Must be the Pizza owner.");
        pizzaApprovals[_pizzaId] = _to;
        emit Approval(msg.sender, _to, _pizzaId);
```

```
    }

    // Returns approved address for specific Pizza
    function getApproved(uint256 _pizzaId)
        public
        view
        returns (address operator)
    {
        require(_exists(_pizzaId), "Pizza does not exist.");
        return pizzaApprovals[_pizzaId];
    }

    /**
     * Private function to clear current approval of a given token ID
     * Reverts if the given address is not indeed the owner of the token
     */
    function _clearApproval(address owner, uint256 _pizzaId) private {
        require(pizzaToOwner[_pizzaId] == owner, "Must be pizza owner.");
        require(_exists(_pizzaId), "Pizza does not exist.");
        if (pizzaApprovals[_pizzaId] != address(0)) {
            pizzaApprovals[_pizzaId] = address(0);
        }
    }

    /*
     * Sets or unsets the approval of a given operator
     * An operator is allowed to transfer all tokens of the sender on their behalf
     */
    function setApprovalForAll(address to, bool approved) public {
        require(to != msg.sender, "Cannot approve own address");
        operatorApprovals[msg.sender][to] = approved;
        emit ApprovalForAll(msg.sender, to, approved);
    }

    // Tells whether an operator is approved by a given owner
    function isApprovedForAll(address owner, address operator)
        public
        view
        returns (bool)
    {
        return operatorApprovals[owner][operator];
    }

    // Takes ownership of Pizza - only for approved users
    function takeOwnership(uint256 _pizzaId) public {
        require(_isApprovedOrOwner(msg.sender, _pizzaId), "Address is not approved.");
        address owner = this.ownerOf(_pizzaId);
        this.transferFrom(owner, msg.sender, _pizzaId);
    }

    // Checks if Pizza exists
    function _exists(uint256 pizzaId) internal view returns (bool) {
        address owner = pizzaToOwner[pizzaId];
        return owner != address(0);
    }
```

```
    // Checks if address is owner or is approved to transfer Pizza
    function _isApprovedOrOwner(address spender, uint256 pizzaId)
        internal
        view
        returns (bool)
    {
        address owner = pizzaToOwner[pizzaId];
        // Disable solium check because of
        // https://github.com/duaraghav8/Solium/issues/175
        // solium-disable-next-line operator-whitespace
        return (spender == owner ||
            this.getApproved(pizzaId) == spender ||
            this.isApprovedForAll(owner, spender));
    }

    // Check if Pizza is unique and doesn't exist yet
    modifier isUnique(string memory _name, uint256 _dna) {
        bool result = true;
        for (uint256 i = 0; i < pizzas.length; i++) {
            if (
                keccak256(abi.encodePacked(pizzas[i].name)) ==
                keccak256(abi.encodePacked(_name)) &&
                pizzas[i].dna == _dna
            ) {
                result = false;
            }
        }
        require(result, "Pizza with such name already exists.");
        _;
    }

    // Returns whether the target address is a contract
    function isContract(address account) internal view returns (bool) {
        uint256 size;
        // Currently there is no better way to check if there is a contract in an address
        // than to check the size of the code at that address.
        // 参阅 https://ethereum.stackexchange.com/a/14016/36603
        // 了解更多信息。
        // TODO: 在 Serenity 发布前再次检查这里。
        // 否则到时所有地址都将判断为合约。
        // solium-disable-next-line security/no-inline-assembly
        assembly {
            size := extcodesize(account)
        }
        return size > 0;
    }
}
```

延伸阅读 {#further-reading}

查阅 Solidity 和 Vyper 文档，以获得关于智能合约的更完整概述：

- Solidity
- Vyper

相关主题 {#related-topics}

- 智能合约
- 以太坊虚拟机

相关教程 {#related-tutorials}

- 减少合约大小以应对合约大小的限制 – *一些减少智能合约大小的实用提示。*
- 用事件记录智能合约的数据 *——对智能合约事件的介绍以及如何使用它们来记录数据。*
- 在 Solidity 中与其它合约交互 *——如何从现有合约中部署智能合约并与之交互。*

---

title: 编译智能合约
description: 解释您为什么需要编译智能合约以及编译实际上做了什么。
lang: zh
sidebar: true

incomplete: true

您需要编译您的合约，以便您的 web 应用程序和以太坊虚拟机能够理解它。

前提条件 {#prerequisites}

在阅读关于编译的文档之前，请阅读我们的智能合约和以太坊虚拟机可能会有帮助。

以太坊虚拟机 {#the-evm}

要使以太坊虚拟机能够运行您的合约，您的合约必须被编译为**字节码**。 编译过程把如下代码：

```
pragma solidity 0.4.24;

contract Greeter {

    function greet() public constant returns (string) {
        return "Hello";
    }

}
```

转化为

```
PUSH1 0x80 PUSH1 0x40 MSTORE PUSH1 0x4 CALLDATASIZE LT PUSH2 0x41 JUMPI PUSH1 0x0
CALLDATALOAD PUSH29 0x10000000000000000000000000000000000000000000000000000000 SWAP1 DIV
PUSH4 0xFFFFFFFF AND DUP1 PUSH4 0xCFAE3217 EQ PUSH2 0x46 JUMPI JUMPDEST PUSH1 0x0 DUP1
REVERT JUMPDEST CALLVALUE DUP1 ISZERO PUSH2 0x52 JUMPI PUSH1 0x0 DUP1 REVERT JUMPDEST POP
PUSH2 0x5B PUSH2 0xD6 JUMP JUMPDEST PUSH1 0x40 MLOAD DUP1 DUP1 PUSH1 0x20 ADD DUP3 DUP2
SUB DUP3 MSTORE DUP4 DUP2 DUP2 MLOAD DUP2 MSTORE PUSH1 0x20 ADD SWAP2 POP DUP1 MLOAD
SWAP1 PUSH1 0x20 ADD SWAP1 DUP1 DUP4 DUP4 PUSH1 0x0 JUMPDEST DUP4 DUP2 LT ISZERO PUSH2
0x9B JUMPI DUP1 DUP3 ADD MLOAD DUP2 DUP5 ADD MSTORE PUSH1 0x20 DUP2 ADD SWAP1 POP PUSH2
0x80 JUMP JUMPDEST POP POP POP POP SWAP1 POP SWAP1 DUP2 ADD SWAP1 PUSH1 0x1F AND DUP1
ISZERO PUSH2 0xC8 JUMPI DUP1 DUP3 SUB DUP1 MLOAD PUSH1 0x1 DUP4 PUSH1 0x20 SUB PUSH2
0x100 EXP SUB NOT AND DUP2 MSTORE PUSH1 0x20 ADD SWAP2 POP JUMPDEST POP SWAP3 POP POP POP
PUSH1 0x40 MLOAD DUP1 SWAP2 SUB SWAP1 RETURN JUMPDEST PUSH1 0x60 PUSH1 0x40 DUP1 MLOAD
SWAP1 DUP2 ADD PUSH1 0x40 MSTORE DUP1 PUSH1 0x5 DUP2 MSTORE PUSH1 0x20 ADD PUSH32
0x48656C6C6F000000000000000000000000000000000000000000000000000000 DUP2 MSTORE POP SWAP1
POP SWAP1 JUMP STOP LOG1 PUSH6 0x627A7A723058 KECCAK256 SLT 0xec 0xe 0xf5 0xf8 SLT 0xc7
0x2d STATICCALL ADDRESS SHR 0xdb COINBASE 0xb1 BALANCE 0xe8 0xf8 DUP14 0xda 0xad DUP13
LOG1 0x4c 0xb4 0x26 0xc2 DELEGATECALL PUSH7 0x8994D3E002900
```

Web 应用程序 {#web-applications}

编译器还将生成您需要的**应用程序二进制接口 (ABI)**，以便您的应用程序能够理解合约并调用合约的功能。

ABI 是一份 JSON 文件，描述了部署的合约及这个智能合约的函数。 这在 web2 和 web3 之间的鸿沟上架起交流的桥梁

Anchor Link Xpath: /ul[19]/li/a 文件： Decentralized Storage

以下是 ERC-20 代币合约的应用程序二进制接口。 ERC-20 是您可以在以太坊交易的代币。

```
[
  {
    "constant": true,
    "inputs": [],
    "name": "name",
    "outputs": [
      {
        "name": "",
        "type": "string"
      }
    ],
    "payable": false,
    "stateMutability": "view",
    "type": "function"
  },
  {
    "constant": false,
    "inputs": [
      {
        "name": "_spender",
        "type": "address"
      },
      {
        "name": "_value",
        "type": "uint256"
      }
```

```
    ],
    "name": "approve",
    "outputs": [
      {
        "name": "",
        "type": "bool"
      }
    ],
    "payable": false,
    "stateMutability": "nonpayable",
    "type": "function"
  },
  {
    "constant": true,
    "inputs": [],
    "name": "totalSupply",
    "outputs": [
      {
        "name": "",
        "type": "uint256"
      }
    ],
    "payable": false,
    "stateMutability": "view",
    "type": "function"
  },
  {
    "constant": false,
    "inputs": [
      {
        "name": "_from",
        "type": "address"
      },
      {
        "name": "_to",
        "type": "address"
      },
      {
        "name": "_value",
        "type": "uint256"
      }
    ],
    "name": "transferFrom",
    "outputs": [
      {
        "name": "",
        "type": "bool"
      }
    ],
    "payable": false,
    "stateMutability": "nonpayable",
    "type": "function"
  },
  {
    "constant": true,
```

```
    "inputs": [],
    "name": "decimals",
    "outputs": [
      {
        "name": "",
        "type": "uint8"
      }
    ],
    "payable": false,
    "stateMutability": "view",
    "type": "function"
  },
  {
    "constant": true,
    "inputs": [
      {
        "name": "_owner",
        "type": "address"
      }
    ],
    "name": "balanceOf",
    "outputs": [
      {
        "name": "balance",
        "type": "uint256"
      }
    ],
    "payable": false,
    "stateMutability": "view",
    "type": "function"
  },
  {
    "constant": true,
    "inputs": [],
    "name": "symbol",
    "outputs": [
      {
        "name": "",
        "type": "string"
      }
    ],
    "payable": false,
    "stateMutability": "view",
    "type": "function"
  },
  {
    "constant": false,
    "inputs": [
      {
        "name": "_to",
        "type": "address"
      },
      {
        "name": "_value",
        "type": "uint256"
```

```
      }
    ],
    "name": "transfer",
    "outputs": [
      {
        "name": "",
        "type": "bool"
      }
    ],
    "payable": false,
    "stateMutability": "nonpayable",
    "type": "function"
  },
  {
    "constant": true,
    "inputs": [
      {
        "name": "_owner",
        "type": "address"
      },
      {
        "name": "_spender",
        "type": "address"
      }
    ],
    "name": "allowance",
    "outputs": [
      {
        "name": "",
        "type": "uint256"
      }
    ],
    "payable": false,
    "stateMutability": "view",
    "type": "function"
  },
  {
    "payable": true,
    "stateMutability": "payable",
    "type": "fallback"
  },
  {
    "anonymous": false,
    "inputs": [
      {
        "indexed": true,
        "name": "owner",
        "type": "address"
      },
      {
        "indexed": true,
        "name": "spender",
        "type": "address"
      },
      {
```

```
        "indexed": false,
        "name": "value",
        "type": "uint256"
      }
    ],
    "name": "Approval",
    "type": "event"
  },
  {
    "anonymous": false,
    "inputs": [
      {
        "indexed": true,
        "name": "from",
        "type": "address"
      },
      {
        "indexed": true,
        "name": "to",
        "type": "address"
      },
      {
        "indexed": false,
        "name": "value",
        "type": "uint256"
      }
    ],
    "name": "Transfer",
    "type": "event"
  }
]
```

延伸阅读 {#further-reading}

- ABI spec – *Solidity*

相关主题 {#related-topics}

- JavaScript 客户端库
- 以太坊虚拟机

---

title: 智能合约的可组合性
description:
lang: zh
sidebar: true

incomplete: true

简介 {#a-brief-introduction}

智能合约在以太坊上是公开的，可视为开放式应用程序接口。 您不需要写自己的智能合约才能成为一个去中心化应用程序开发者，您只需要知道如何与它们交互。 例如，您可以使用现有的智能合约 Uniswap，一个去中心化的交易所，在您的应用中处理代币交易逻辑 – 您并不需要一切从零开始。 看看他们的一些 v2 和 v3 合约。

相关教程 {#related-tutorials}

- 合约的可组合性：以太坊智能合约开发的基石
- 使用“创建以太坊应用程序”启动去中心化应用程序前端开发 – *概述如何使用“创建以太坊应用程序”创建带有热门智能合约的应用程序。*

延伸阅读 {#further-reading}

还有哪些社区资源对您有所帮助？ 编辑并添加本页面！

- 可组合性是创新

---

title: 部署智能合约
description:
lang: zh

sidebar: true

需要部署智能合约才能提供给以太坊网络的用户使用。

要部署一个智能合约，只需发送一个包含编译后的智能合约代码的以太坊交易，而不需要指定任何收件人。

前置要求 {#prerequisites}

在部署智能合约之前，您需要理解以太坊网络, 交易和详解智能合约。

部署一个合约也需要耗费以太币，所以您应该熟悉以太坊的燃料和费用。

最后，您需要在部署之前编译您的合约，所以请确保您已经阅读了编译智能合约。

如何部署智能合约 {#how-to-deploy-a-smart-contract}

您所需要的 {#what-youll-need}

- 您的合约字节码 – 这是通过编译获得的。
- 用作燃料的以太币 – 像其他交易一样，您需要设定燃料限制，这样就知道部署合约比简单的以太币交易需要更多的燃料。
- 一个部署脚本或插件。
- 访问以太坊节点，通过运行自己的节点连接到公共节点，或通过应用程序接口秘钥使用节点服务，如 Infura 和 Alchemy。

部署智能合约的步骤 {#steps-to-deploy}

所涉及的具体步骤将取决于您使用的工具。 例如，查看关于部署合约的安全帽文档或关于网络和应用程序部署的 Truffle 文档。 这是两个最受欢迎的智能合约部署工具，它们涉及到编写脚本来处理部署步骤。

一旦部署，您的合约将有一个以太坊地址，就像其它帐户一样。

相关工具 {#related-tools}

Remix - ***Remix 集成开发环境可以开发、部署和管理类似区块链的以太坊智能合约。***

- Remix

Tenderly - ***用实时数据模拟、调试和监视以太坊虚拟机兼容链上的任何内容***

- tenderly.co
- 相关文档
- GitHub
- Discord

安全帽 - *用于编译、部署、测试和调试您的以太坊软件的开发环境*

- hardhat.org
- 关于部署合约的文档
- GitHub
- Discord

Truffle - *开发环境、测试框架、部署通道及其他工具。*

- trufflesuite.com
- 关于网络和应用部署的文档
- GitHub

相关教程 {#related-tutorials}

- 部署您的第一个智能合约 _ - 介绍如何在以太坊测试网络上部署您的第一个智能合约。_
- 在 Solidity 中与其它合约交互 _ - 如何从现有合约中部署智能合约并与之交互。_
- 如何减少合约的大小 _ - 如何减少合约的大小，使其保持在限制之下并节省燃料_

延伸阅读 {#further-reading}

- https://docs.openzeppelin.com/learn/deploying-and-interacting - *OpenZeppelin*
- 使用安全帽部署合约 - *Nomic Labs*

还有哪些社区资源对您有所帮助？ 请编辑本页面并添加！

相关主题 {#related-topics}

- 开发框架
- 运行以太坊节点

---

title: 智能合约简介
description: 智能合约概述，重点介绍其独特的特征和局限性。
lang: zh

sidebar: true

什么是智能合约？ {#what-is-a-smart-contract}

智能合约只是一个运行在以太坊链上的一个程序。 它是位于以太坊区块链上一个特定地址的一系列代码（函数）和数据（状态）。

智能合约也是一个以太坊帐户，我们称之为合约帐户。 这意味着他们有余额，他们可以通过网络进行交易。 但是，他们无法被人操控，他们是被部署在网络上作为程序运行着。 个人用户可以通过提交交易执行智能合约的某一个函数来与智能合约进行交互。 智能合约能像常规合约一样定义规则，并通过代码自动强制执行。 默认情况下，您无法删除智能合约，与它们的交互是不可逆的。

前置要求 {#prerequisites}

确保在您已了解有关帐户、交易和以太坊虚拟机后再进入智能合约。

数字自动售货机 {#a-digital-vending-machine}

正如 Nick Szabo 所述，对智能合约的最佳比喻也许就是一台售货机。 有了正确的投入，就能保证一定的产出。

从售货机中获取快餐：

```
money + snack selection = snack dispensed
```

这种逻辑已在自动售货机中设定好。

一个智能合约，就像自动售货机一样，逻辑已设定好。 以下示例简单展示了智能合约会如何看起来像自动售货机：

```
pragma solidity 0.8.7;

contract VendingMachine {

    // Declare state variables of the contract
    address public owner;
    mapping (address => uint) public cupcakeBalances;

    // When 'VendingMachine' contract is deployed:
    // 1. set the deploying address as the owner of the contract
    // 2. set the deployed smart contract's cupcake balance to 100
    constructor() {
        owner = msg.sender;
        cupcakeBalances[address(this)] = 100;
    }

    // Allow the owner to increase the smart contract's cupcake balance
    function refill(uint amount) public {
        require(msg.sender == owner, "Only the owner can refill.");
        cupcakeBalances[address(this)] += amount;
    }

    // Allow anyone to purchase cupcakes
    function purchase(uint amount) public payable {
        require(msg.value >= amount * 1 ether, "You must pay at least 1 ETH per
cupcake");
        require(cupcakeBalances[address(this)] >= amount, "Not enough cupcakes in stock
to complete this purchase");
        cupcakeBalances[address(this)] -= amount;
        cupcakeBalances[msg.sender] += amount;
    }
}
```

就像自动售货机消除了对供应商雇员的需求一样，智能合约可以在许多行业中取代中介机构。

无需准入性 {#permissionless}

任何人都可以编写智能合约，并将其部署到网络上。 您只需要学习如何用智能合约语言编码，并有足够的以太币来部署您的合约。 在技术上，部署智能合约是一项交易，所以您需要支付气体，就像您需要为简单的以太币转账支付气体一样。 然而，部署合约需要支付的气体成本要高得多。

以太坊为编写智能合约提供的语言对开发者比较友好：

- Solidity
- Vyper

关于语言的更多信息

然而，它们必须要先编译才能部署，以便以太坊虚拟机可以解释并储存合约。 关于编译的更多信息

可组合性 {#composability}

智能合约在以太坊上是公开的，可视为开放式应用程序接口。 这意味着您可以在自己的智能合约中调用其他智能合约，极大地扩展了可以执行的操作范围。 合约甚至可以部署其他合约。

了解关于智能合约可组合性的更多信息。

局限性 {#limitations}

智能合约本身无法获取关于"真实世界"的事件信息，因为它们无法发送超文本传输协议请求。 这是由设计决定的。 依赖外部信息可能会危及共识，这一点对安全性和去中心化而言十分重要。

这可以使用预言机通过多种方法来规避。

智能合约的另一个限制是最大合约大小。 智能合约最大可为 24KB，否则会消气体耗完气体。 可以使用钻石模式来规避这一问题。

智能合约资源 {#smart-contract-resources}

OpenZeppelin 合约——**安全智能合约开发的库。****

- openzeppelin.com/contracts/
- GitHub
- 社区论坛

DappSys - ****安全、简单、灵活的智能合约构建块。**

- DappSys
- GitHub

延伸阅读 {#further-reading}

- 智能合约：将取代 Lawyer 的区块链技术 - *Blockgeek*
- 智能合约开发的最佳做法 – *2019 年 11 月 10 日 - Yos Riady*
- 清洁合约 - 智能合约模式与实践指南 – *wslyvh，2020 年 7 月 30 日*

---

title: Ethereum.org 翻译指南
lang: zh
description: 如何帮助翻译 ethereum.org

sidebar: true

ethereum.org 翻译手册 {#translating-ethereum-guide}

如果您是翻译项目的新手，而且不知道从何下手，这里的一些常见问题会有所帮助。 使用本手册来寻找常见问题的答案。

如何翻译带有 `<HTML tags>` 的语句？ {#tags}

并非每个语句都以纯文本形式写入。 部分语句包含 HTML 标签 (`<0>` `</0>`) 等混合脚本。通常是句子中的超链接或不同格式。

- 翻译标签内的文本，但不翻译标签本身。 不能翻译或删除 `<` and `>` 中的任何内容。
- 为了保持语句完整，建议您点击左下方的"复制源代码"按钮。 这将复制原文语句并粘贴到文本框内。 这让您明确标签的位置，有助于避免错误。

高亮显示复制源按钮的 Crowdin 界面

您可以在语句中移动标签的位置，使其在句子中更加自然，但请确保移动整个标签。

上下文在哪里？ {#strings}

通常情况下，仅靠原文语句可能不足以让您提供准确翻译。

- 查看“屏幕截图”和“上下文”以获取更多信息。 在原文语句部分，您将看到附加的屏幕截图图像，从而了解使用该语句的上下文。
- 如果仍然不确定，请在“评论部分”中添加标记。 不知道如何评论？

如何通过屏幕截图为语句提供上下文信息

为上下文添加一个示例截图

如何留下评论或问题？ 我想标记一个问题或错别字... {#comment}

如果您想标记某个需要注意的语句，请随时提交评论。

- 点击右上角菜单栏的第二个按钮。 隐藏标签将出现在您的右侧。 留下新评论并点击底部的“问题”复选框。 您可以从下拉菜单选择一个选项，表明问题类型。
- 问题一旦提交，就将报告给我们的团队。 我们将解决这个问题，并通过回复评论和关闭问题来让您知道。

如何发表评论和问题

什么是翻译记忆库 (TM)？ {#translation-memory}

翻译记忆库 (TM) 是一项 Crowdin 功能，可在 ethereum.org 中存储所有先前已翻译的语句。 翻译过的语句会自动保存到 TM 中。 这款实用工具可以帮助您节省时间!

- 请看“TM 和 MT 建议”部分，您会看到其他翻译者如何翻译相同或类似语句。 如果发现匹配率很高的建议，可以点击以引用该翻译内容。
- 如果列表中没有任何内容，您可以搜索 TM 中以前做过的翻译并重新使用以保持一致性。

翻译记忆库屏幕截图

如何使用 Crowdin 词汇表？ {#glossary}

以太坊术语是我们翻译工作的另一个关键部分，因为通常新的术语在许多语言中尚未本地化。 另外，有些术语在不同背景下的含义不同。 关于以太坊术语翻译的更多信息

Crowdin 词汇表是阐明术语和定义的最佳位置。 有两种方法来参照词汇表。

- 首先，当您在原文语句上发现带下划线的术语时，将鼠标移到上面，即可看到简要定义。

词汇表定义示例

- 第二，如果看到一个不熟悉但没有下划线的术语，您可以在词汇表选项卡（右侧列第三个按钮）中搜索。 您会找到特定术语和项目中常用术语的解释。

Crowdin 中“词汇表”选项卡位置的屏幕截图

- 如果仍然找不到，即可借此机会添加新术语！ 我们建议您在搜索引擎上进行查找，并将描述添加到词汇表中。 这将非常有助于其他翻译者更好地理解该术语。

如何向 Crowdin 添加词汇表术语的屏幕截图

术语翻译政策 {#terminology}

适用于名称（品牌、公司、人物）和新的技术术语（Eth2、信标链等）

以太坊提出了很多最近出现的新术语。 由于没有各自语言的官方译本，因此翻译者对有些术语的翻译不同。 这种不一致会造成误解，降低可读性。

由于语言多样性和每种语言的标准化程度不同，几乎不可能提出可适合所有支持语言的统一术语翻译规则。

经过慎重考虑，我们决定将最常用的术语留给你们翻译者。

如果发现您不熟悉的术语，我们建议您：

- 参考术语表，您可能会发现其他翻译者以前的译法。 如果您认为之前翻译的术语不恰当，请随时通过向 Crowdin 词汇表添加新术语来恢复您的译文。
- 如果词汇表中之前没有翻译，我们建议您在搜索引擎或媒体文章中查找，显示该术语在社区中的实际使用情况。
- 如果根本找不到任何参考资料，请按您的直觉和理解翻译成自己的语言！
- 如果不太确定，可以不翻译这一术语。 有时，英语术语足以提供准确定义。

我们建议不翻译品牌、公司和人员名称，因为翻译可能会造成不必要的混乱和 SEO 困难。

联系我们 {#contact}

感谢阅读所有内容。 我们希望这对您加入我们的项目有所帮助。 欢迎加入我们的Discord 翻译频道，提出问题并与其他翻译者合作！

title: 智能合约语言
description: 两种主要的智能合约语言（Solidity 和 Vyper）的概述和比较。
lang: zh

sidebar: true

关于以太坊的一个重要方面是，智能合约可以使用相对友好的开发者语言编程。 如果您熟悉 Python 或任何大括号语言，可以找到一种语法熟悉的语言。

最受欢迎和维护得最好的两种语言是：

- Solidity
- Vyper

更有经验的开发者也可能想要使用 Yul：一种用于以太坊虚拟机的中间语言，或者是 Yul+ 语言，这是一种 Yul 扩展。

如果您很好奇，喜欢帮助测试仍在大力发展的新语言，则可以尝试使用 Fe，这是一种新兴的智能合约语言，目前仍处于起步阶段。

前置要求 {#prerequisites}

如果已经有编程语言（特别是关于 JavaScript 或 Python）知识，可以帮助您体验到智能合约语言的差异。 同时，我们建议您在深入理解语言差异之前，先理解作为概念的智能合约。 智能合约简介。

Solidity {#solidity}

- 执行智能合约的目标导向高级语言。
- 受 C++ 影响最深的大括号编程语言。
- 静态类型（编译时已知变量类型）。
- 支持：
  - 继承（您可以拓展其它合约）。
  - 库（您可以创建从不同的合约调用的可重用代码 - 就像静态函数在其它面向对象编程语言的静态类中一样）。
  - 复杂的用户自定义类型。

重要链接 {#important-links}

- 相关文档
- Solidity 语言网站
- Solidity 示例
- GitHub
- Solidity Gitter Chatroom 桥接到 Solidity Matrix Chatroom
- 备忘单
- Solidity 博客
- Solidity Twitter

合约示例 {#example-contract}

```
/ SPDX-License-Identifier: GPL-3.0
pragma solidity >= 0.7.0;

contract Coin {
    // The keyword "public" makes variables
    // accessible from other contracts
    address public minter;
    mapping (address => uint) public balances;

    // Events allow clients to react to specific
    // contract changes you declare
    event Sent(address from, address to, uint amount);

    // Constructor code is only run when the contract
```

```
    // is created
    constructor() {
        minter = msg.sender;
    }

    // Sends an amount of newly created coins to an address
    // Can only be called by the contract creator
    function mint(address receiver, uint amount) public {
        require(msg.sender == minter);
        require(amount < 1e60);
        balances[receiver] += amount;
    }

    // Sends an amount of existing coins
    // from any caller to an address
    function send(address receiver, uint amount) public {
        require(amount <= balances[msg.sender], "Insufficient balance.");
        balances[msg.sender] -= amount;
        balances[receiver] += amount;
        emit Sent(msg.sender, receiver, amount);
    }
}
```

这个示例应该能让您感觉到 Solidity 合约语法是什么样子的。 关于函数和变量的详细描述，请查看文档。

Vyper {#vyper}

- 类 Python 编程语言
- 强类型
- 小而且易懂的编译器代码
- 为了更安全和易于审阅，特意提供比 Solidity 少的功能。 Vyper 不支持：
  - 修饰符
  - 继承
  - 内联装配
  - 函数重载
  - 操作重载
  - 递归调用
  - 无限长度循环
  - 二进制定长浮点

更多信息，请查阅 Vyper 原理。

重要的链接 {#important-links-1}

- 相关文档
- Vyper 示例
- GitHub
- Vyper Gitter 聊天室
- 备忘单
- 更新日期：2020 年 1 月 8 日

示例 {#example}

```
# Open Auction

# Auction params
# Beneficiary receives money from the highest bidder
beneficiary: public(address)
auctionStart: public(uint256)
auctionEnd: public(uint256)

# Current state of auction
highestBidder: public(address)
highestBid: public(uint256)

# Set to true at the end, disallows any change
ended: public(bool)

# Keep track of refunded bids so we can follow the withdraw pattern
pendingReturns: public(HashMap[address, uint256])
@external
def __init__(_beneficiary: address, _bidding_time: uint256):
    self.beneficiary = _beneficiary
    self.auctionStart = block.timestamp
    self.auctionEnd = self.auctionStart + _bidding_time

# Bid on the auction with the value sent
# together with this transaction.
# The value will only be refunded if the
# auction is not won.
@external
@payable
def bid():
    # Check if bidding period is over.
    assert block.timestamp < self.auctionEnd
    # Check if bid is high enough
    assert msg.value > self.highestBid
    # Track the refund for the previous high bidder
    self.pendingReturns[self.highestBidder] += self.highestBid
    # Track new high bid
    self.highestBidder = msg.sender
    self.highestBid = msg.value The withdraw pattern is
# used here to avoid a security issue. If refunds were directly
# sent as part of bid(), a malicious bidding contract could block
# those refunds and thus block new higher bids from coming in.
@external
def withdraw():
    pending_amount: uint256 = self.pendingReturns[msg.sender]
    self.pendingReturns[msg.sender] = 0
    send(msg.sender, pending_amount)

# End the auction and send the highest bid
# to the beneficiary.
@external
def endAuction():
    # It is a good guideline to structure functions that interact
```

```
    # with other contracts (i.e. they call functions or send ether)
    # into three phases:
    # 1. checking conditions
    # 2. performing actions (potentially changing conditions)
    # 3. interacting with other contracts
    # If these phases are mixed up, the other contract could call
    # back into the current contract and modify the state or cause
    # effects (ether payout) to be performed multiple times.
    # If functions called internally include interaction with external
    # contracts, they also have to be considered interaction with
    # external contracts.

    # 1. Conditions
    # Check if auction endtime has been reached
    assert block.timestamp >= self.auctionEnd
    # Check if this function has already been called
    assert not self.ended

    # 2. Effects
    self.ended = True

    # 3. Interaction
    send(self.beneficiary, self.highestBid)
```

这个例子应该让您了解 Vyper 合约语法是什么样的。 有关函数和变量的详细说明，请参阅文档。

Yul 和 Yul+ {#yul}

如果您是以太坊的新手并且尚未使用智能合约语言进行任何编码，我们建议您开始使用 Solidity 或 Vyper。 只有在您熟知智能合约安全最佳做法和使用 EVM 的具体细节后，才可以查看 Yul 或 Yul+。

Yul

- 以太坊的中继语言。
- 支持 EVM 和 Ewasm，一种以太坊风格的 WebAssembly，以及旨在成为两个平台均可用的公分母。
- 高级优化阶段的良好目标，既使 EVM 和 eWASM 平台均等受益。

Yul+

- Yul 的低级、高效扩展。
- 最初设计用于乐观卷叠合约。
- Yul+ 可以被视为对 Yul 的实验性升级建议，为其添加新功能。

重要的链接 {#important-links-2}

- Yul 相关文档
- Yul+ 相关文档
- Yul+ 实战场
- Yul+ 介绍帖子

合约示例 {#example-contract-2}

以下简单示例实现了幂函数。 它可以使用 `solc --strict-assembly --bin input.yul` 编译。 这个例子应该 存储在 input.yul 文件中。

```
{
    function power(base, exponent) -> result
    {
        switch exponent
        case 0 { result := 1 }
        case 1 { result := base }
        default
        {
            result := power(mul(base, base), div(exponent, 2))
            if mod(exponent, 2) { result := mul(base, result) }
        }
    }
    let res := power(calldataload(0), calldataload(32))
    mstore(0, res)
    return(0, 32)
}
```

如果您已经熟悉智能合约，可以在 此处找到 Yul 中的完整 ERC20 实例 。

Fe {#fe}

- 以太坊虚拟机 (EVM) 静态类型语言。
- 受到 Python 和 Rust 的启发。
- 目标是容易学习 - 甚至对以太坊生态系统为新的开发者来说也是如此。
- Fe 开发仍处于早期阶段，该语言于 2021 年 1 月发行。

重要链接 {#important-links-3}

- GitHub
- Fe 公告
- Fe 2021 路线图
- Fe Discord 聊天室
- Fe Twitter

合约示例 {#example-contract-3}

以下是在 Fe 中执行的简单的智能合约。

```
type BookMsg = bytes[100]

contract GuestBook:
    pub guest_book: map<address, BookMsg>

    event Signed:
        book_msg: BookMsg

    pub def sign(book_msg: BookMsg):
        self.guest_book[msg.sender] = book_msg
```

```
        emit Signed(book_msg=book_msg)

    pub def get_msg(addr: address) -> BookMsg:
        return self.guest_book[addr].to_mem()
```

如何选择 {#how-to-choose}

与任何其他编程语言一样，它主要是关于为合适的工作以及个人喜好选择合适的工具。

如果您还没有尝试过任何一种语言，请考虑以下几点：

Solidity 的优点是什么？ {#solidity-advantages}

- 如果您是初学者，这里有很多教程和学习工具。 在通过编码学习部分了解更多相关信息。
- 提供出色的开发者工具。
- Solidity 拥有庞大的开发人员社区，这意味着您很可能会很快找到问题的答案。

Vyper 的优点是什么？ {#vyper-advatages}

- 想要编写智能合约的 Python 开发人员入门的好方法。
- Vyper 的功能较少，因此非常适合快速制作创意原型。
- Vyper 旨在易于审计并最大限度地提高人类可读性。

Yul 和 Yul+ 的优点是什么？ {#yul-advantages}

- 简单而实用的低级语言。
- 允许更接近原始 EVM，这有助于优化合约的 gas 使用量。

语言比较 {#language-comparisons}

关于基本语法的比较、合同生命周期、接口、操作员、数据结构、功能、控制流程以及更多，请查看由 Auditless 编写的备忘清单

延伸阅读 {#further-reading}

- OpenZeppelin 的 Solidity 合约库
- Solidity 示例

---

title: 智能合约库
description:
lang: zh

sidebar: true

您无需从头开始编写项目中的每一个智能合约 我们有许多开源代码的智能合约库可为您的项目提供可重复利用的构建块，从而使您不必重新开始。

前置要求 {#prerequisites}

在我们跳转到智能合约库之前，清楚地了解一个智能合约的构成是一个不错的主意。 如果尚未进行智能合约的了解，请直接点击智能合约。

资料库中的内容 {#whats-in-a-library}

您通常可以在智能合约库中找到两种构建模块：可以添加到合约中的可复用代码，与各种标准的实现。

行为 {#behaviors}

当编写智能合约时，您很可能会发现自己在写重复的代码。 比如说在智能合约中指派一个*管理员*地址执行受保护的操作，或添加一个紧急*暂停*按钮以应对预料不到的问题。

智能合约库通常提供这些行为的可复用实现方式为标准库或在 solidity 中通过继承的方式实现。

例如，以下是`不可拥有的`合约的简化版本来自 OpenZepelin 合约库，它设计了一个作为合约所有者的地址，并且提供了一个修饰者来限制该所有者获得一种方法。

```
contract Ownable {
    address public owner;

    constructor() internal {
        owner = msg.sender;
    }

    modifier onlyOwner() {
        require(owner == msg.sender, "Ownable: caller is not the owner");
        _;
    }
}
```

在您的合约中使用这个构建模块，您需要先导入它，然后在您自己的合约中扩展它。 这个将会允许您使用`Ownable`合约提供的修饰符来保护您的函数。

```
import ".../Ownable.sol"; // Path to the imported library

contract MyContract is Ownable {
    // The following function can only be called by the owner
    function secured() onlyOwner public {
        msg.sender.transfer(1 ether);
    }
}
```

另一个比较受欢迎的例子是 SafeMath 或DsMath。 这些库（与基础合约不同）提供了语言本身不具有的带有溢出检查的算术函数。 使用这些库而不是本地的算术操作可以来防止您的合约出现溢出错误，这些错误可能会导致灾难性的后果！

标准 {#standards}

为了促进可组合性和互操作性，以太坊社区已经以**以太坊意见征求**的形式定义了几个标准。 您可以在标准部分阅读更多关于他们的信息。

当将以太坊意见征求作为您的合约的一部分时，更好的做法是寻找已有的标准去实现而不是试图推出您自己的方式。 许多智能合约库包含了最流行的以太坊意见征求标准的实现。 例如，普遍存在的 ERC20 同质化通证标准可在 HQ20 DappSys 和 OpenZepelin 中找到。 此外，一些以太坊意见征求还提供规范实现作为以太坊意见征求本身的一部分。

值得一提的是，一些以太坊意见征求不是独立的，而是对其他以太坊意见征求的补充。 例如， ERC2612 为 ERC20 添加了一个扩展，以提高其可用性。

如何添加库 {#how-to}

始终参考您所包含的库的文档，以获得关于如何将其包含在您的项目中的具体说明 一些 Solidity 合约库使用 `npm` 来打包，所以您可以直接 `npm` 安装它们。 大多数编译合约的工具会在您的 node_modules 中查找智能合约库，所以您可以做以下工作。

```
// This will load the @openzeppelin/contracts library from your node_modules
import "@openzeppelin/contracts/token/ERC721/ERC721.sol";

contract MyNFT is ERC721 {
    constructor() ERC721("MyNFT", "MNFT") public { }
}
```

无论您使用哪种方法，当包括一个库时，总是要注意语言的版本。 例如，如果您用 Solidity 0.5 编写您的合约，您就不能使用 Solidity 0.6 的库。

何时使用 {#when-to-use}

为您的项目使用智能合约库有几个好处。 首先，它为您提供了现成的构建模块，您可以将其纳入您的系统，而不必自己编码，从而节省了您的时间。

安全性也是一个重要的优点。 开源智能合约库也经常受到严格审查。 鉴于许多项目都依赖于它们，社区有强烈的动机来对它们持续审计。 在应用程序代码中发现错误比在可重用的合约库中发现错误要常见得多。 一些库还接受了外部审计，以提高安全性。

然而，使用智能合约库有可能将您不熟悉的代码纳入您的项目。 导入一个合约并将其直接包含在您的项目中是很诱人的，但如果没有很好地理解该合约的作用，您可能会由于一个意外的行为而无意中在您的系统中引入一个问题。 一定要确保阅读您要导入的代码的文档，然后在使其成为您的项目的一部分之前审查代码本身。

最后，在决定是否包括一个库时，要考虑其总体使用情况。 一个被广泛采用的方案的好处是有一个更大的社区和更多的眼睛来关注它的问题。 在使用智能合约进行建设时，安全应该是您的首要关注点！

相关工具 {#related-tools}

OpenZeppelin 合约- *安全的智能合约开发库。*

- 相关文档
- GitHub
- 社区论坛

DappSys - *安全、简单、灵活的智能合约构建模块。*

- 相关文档
- GitHub

HQ20 - *一个带有合约、库和案例的 Solidity 项目，帮助您为现实世界建立功能齐全的分布式应用。*

- GitHub

相关教程 {#related-tutorials}

- 以太坊开发者的安全考虑 - *构建智能合约时的安全考虑教程，包括库的使用。*
- 了解 ERC-20 代币智能合约 _-关于 ERC20 标准的教程，由多个库提供。

延伸阅读 {#further-reading}

还有哪些社区资源对您有所帮助？ 编辑并添加本页面！

title: 智能合约安全性
description: 以太坊开发者的安全考虑
lang: zh

sidebar: true

以太坊智能合约是极为灵活的。它能够存储超过非常大量的虚拟货币（超过十亿美元），并且根据先前部署的智能合约运行不可修改的代码。 虽然这创造了一个充满活力和创造性的生态系统，但其中包含的无信任、相互关联的智能合约，也吸引了攻击者利用智能合约中的漏洞和以太坊中的未知错误来赚取利润。 智能合约代码*通常*无法修改来修复安全漏洞，因此从智能合约中被盗窃的资产是无法收回的，且被盗资产极难追踪。 由于智能合约问题而被盗或丢失的价值总额很容易超过 $1B。 一些因为智能合约代码编写错误导致较大经济损失的例子：

- 钱包问题 #1 - 3000 万美金损失
- 钱包问题 #2 - 3 亿美金锁定
- 去中心化自治组织被黑，360 万以太币被盗！ 目前价值超过 10 亿美元

前置要求 {#prerequisites}

这个章节将介绍智能合约安全问题，以确保您在处理安全问题之前熟悉智能合约。

如何编写更安全的智能合约代码 {#how-to-write-more-secure-smart-contract-code}

在启动任何代码到主网之前，采取足够的预防措施来保护您的智能合约所委托的任何有价值的东西是很重要的。 在本文中，我们将讨论一些特定的攻击，提供学习了解更多攻击类型的资源，并为您提供一些基本工具和最佳做法，以确保您的合约正确、安全地运行。

审核不是完美的解决方案 {#audits-are-not-a-silver-bullet}

几年前，用于编写、编译、测试和部署智能合约的工具还非常不成熟，许多项目被随意地编写 Solidity 代码，并将其交给审查员，审查员将审查代码以确保其安全并按预期运行。 在 2020 年，编写 Solidity 代码的开发过程和工具得到了显著改善，不仅可以确保项目更易于管理，而且能够组成项目安全性的一部分。 仅仅在项目结束时对您的智能合约进行审计已经不足以成为项目的唯一安全考虑。 **安全性来源于适当的设计和开发过程**，所以在您编写第一行智能合约代码之前，安全性就应该被考虑。

智能合约开发过程 {#smart-contract-development-process}

最低安全限度：

- 所有代码应该被存在于一个版本控制系统当中，例如 git
- 所有的代码修改都应该通过拉取请求来进行
- 所有的拉取请求都应该有至少一个审核员。 *如果这是一个个人项目，请考虑寻找另一位个人作者和一个交易代码审核员。*
- 使用开发以太坊环境（请参阅：Truffle），只需一个命令就可以编译、部署和运行一套针对您的代码的测试
- 已经通过 Mythril 和 Sliter 等基本代码分析工具运行了代码，最好是在合并每个拉取请求之前，比较输出中的差异。
- Solidity 代码编辑器不会发出任何警告
- 您的代码有据可查

上面的这些条目是编写智能合约的一个良好的开始，但是在编写代码过程中还有很多要值得注意。 关于更多条目及其详细解释，请参阅DeFiSafety 提供的过程质量检查清单。 DefiSafety 是一个对各种大型的公开以太坊去中心化应用程序进行评论的非官方公共服务提供者。 DeFiSafete 对项目的安全评级等级的一部分包括该项目是否遵守了质量检查表。 遵循这些审核过程：

- 通过可复现的自动化测试，产生更安全的代码
- 审查员将能够更有效地审查您的项目
- 对新开发者友好
- 允许开发者快速迭代、测试并在修改时获得反馈
- 回滚代码的可能性较低

攻击和脆弱性 {#attacks-and-vulnerabilities}

既然您正在使用高效的开发过程编写 Solidity 代码，那么让我们看看一些常见的 Solidity 漏洞问题。

重入攻击 {#re-entrancy}

重入攻击时在编写智能合约代码时应该考虑的最大且最重要的安全问题。 虽然以太坊虚拟机不能同时运行多个合约，一个合约可以调用另一个合约来暂停一个合约的执行和内存状态，直到被重新调用。这时，代码将会继续被正常执行。 暂停和重新启动的过程可能会造成一种被称为“重入攻击”的漏洞。

这是一个容易受到重入攻击的合约：

```
// THIS CONTRACT HAS INTENTIONAL VULNERABILITY, DO NOT COPY
contract Victim {
    mapping (address => uint256) public balances;

    function deposit() external payable {
        balances[msg.sender] += msg.value;
    }

    function withdraw() external {
        uint256 amount = balances[msg.sender];
        (bool success, ) = msg.sender.call.value(amount)("");
        require(success);
        balances[msg.sender] = 0;
    }
}
```

为了让用户可以退回先前存储在智能合约里的以太币，这个功能会

1. 读取用户的余额
2. 发送用户的余额
3. 将余额重置为 0，因此它们不能再次提取余额。

如果从普通账户（例如您自己的 MetaMask 账户）调用，这将按预期运行：msg.sender.call.value() 只是向您的账户发送 ETH。 但是，智能合约也能调用其他合约。 如果一个自制的恶意合约调用 `withdraw()`，msg.sender.call.value() 会不仅发送 `amount` 个以太币，还会暗中调用合约来开始执行代码。 想象这个恶意合约：

```
contract Attacker {
    function beginAttack() external payable {
        Victim(VICTIM_ADDRESS).deposit.value(1 ether)();
        Victim(VICTIM_ADDRESS).withdraw();
    }

    function() external payable {
        if (gasleft() > 40000) {
            Victim(VICTIM_ADDRESS).withdraw();
        }
    }
}
```

调用 Attacker.beginAttack() 会开始循环寻找一些像：

```
0.) Attacker's EOA calls Attacker.beginAttack() with 1 ETH
0.) Attacker.beginAttack() deposits 1 ETH into Victim

  1.) Attacker -> Victim.withdraw()
  1.) Victim reads balances[msg.sender]
  1.) Victim sends ETH to Attacker (which executes default function)
    2.) Attacker -> Victim.withdraw()
    2.) Victim reads balances[msg.sender]
    2.) Victim sends ETH to Attacker (which executes default function)
      3.) Attacker -> Victim.withdraw()
      3.) Victim reads balances[msg.sender]
      3.) Victim sends ETH to Attacker (which executes default function)
        4.) Attacker no longer has enough gas, returns without calling again
      3.) balances[msg.sender] = 0;
    2.) balances[msg.sender] = 0; (it was already 0)
  1.) balances[msg.sender] = 0; (it was already 0)
```

攻击者帐户使用 1 个以太币调用 Attacker.beginAttack 函数将会重复攻击受害者帐户，并将赚取远超其提供以太币的数量（这些额外的以太币会从其他用户帐户的余额中赚取，这样会造成受害者帐户余额减少）

如何解决重入攻击（一种错误的方式） {#how-to-deal-with-re-entrancy-the-wrong-way}

防止重入攻击的一种简单方法便是让您的任何代码不与任何智能合约进行交互。 但当您搜索 stackoverflow 时，会发现这段被许多人使用的代码片段：

```
function isContract(address addr) internal returns (bool) {
  uint size;
  assembly { size := extcodesize(addr) }
  return size > 0;
}
```

这似乎有道理：合约本身拥有一些代码，如果调用者还拥有其他代码，则不能允许合约代码去存储数据。 让我们加上它吧：

```
// 该合约存在已知漏洞，请勿复制
contract ContractCheckVictim {
```

```
    mapping (address => uint256) public balances;

    function isContract(address addr) internal returns (bool) {
        uint size;
        assembly { size := extcodesize(addr) }
        return size > 0;
    }

    function deposit() external payable {
        require(!isContract(msg.sender)); // <- 新增行
        balances[msg.sender] += msg.value;
    }

    function withdraw() external {
        uint256 amount = balances[msg.sender];
        (bool success, ) = msg.sender.call.value(amount)("");
        require(success);
        balances[msg.sender] = 0;
    }
}
```

现在为了要存入以太币，您的地址里不能有智能合约的代码。 然而，通过如下的攻击者合约可以很轻松地击败它：

```
contract ContractCheckAttacker {
    constructor() public payable {
        ContractCheckVictim(VICTIM_ADDRESS).deposit(1 ether); // <- New line
    }

    function beginAttack() external payable {
        ContractCheckVictim(VICTIM_ADDRESS).withdraw();
    }

    function() external payable {
        if (gasleft() > 40000) {
            Victim(VICTIM_ADDRESS).withdraw();
        }
   }
}
```

先前的攻击是对合约逻辑的攻击，而这一次则是对以太坊合约部署行为的攻击。 在部署过程中，合约尚未返回要在其地址部署完毕的代码，但在此过程中保留了完整的以太坊虚拟机控制阶段。

从技术上讲，可以使用以下代码来防止智能合约调用您的代码：

```
require(tx.origin == msg.sender)
```

然而，这依旧不是一个很好的解决办法。 因为以太坊最令人兴奋的方面之一是它的可组合性，智能合约相互集成并融合、发展。 通过使用上面的代码，您会限制您项目的实用性。

如何解决重入攻击（一种正确的方式） {#how-to-deal-with-re-entrancy-the-right-way}

只需切换存储更新和外部调用的顺序，我们就可以防止启用攻击的重新进入条件。 虽然有可能，回调退却不会给攻击者带来好处，因为 `balances` 存储已经设置为 0。

```
contract NoLongerAVictim {
    function withdraw() external {
        uint256 amount = balances[msg.sender];
        balances[msg.sender] = 0;
        (bool success, ) = msg.sender.call.value(amount)("");
        require(success);
    }
}
```

上面的代码遵循“检查-效果-相互作用”设计模式，它有助于防止重入攻击。 您可以在这里 [阅读更多关于检查-生效-交互作用的信息]

如何解决重入攻击（核心选择） {#how-to-deal-with-re-entrancy-the-nuclear-option}

任何时候您都会将 ETH 发送到一个不信任的地址或与一个未知合约进行交互（例如调用 `transfer()` 到用户提供的代币地址），您可以自行开启重入。 **通过设计既不发送以太币也不调用不信任合约的智能合约，您将防止重入攻击的可能性！**

更多攻击类型 {#more-attack-types}

上述攻击类型包括智能合约编码问题（重入）和以太坊奇数（在合约构造器内运行代码，合约地址才有编码）。 还有更多攻击类型需要注意，例如：

- 前台运行
- 以太币发送拒绝
- 整数上溢/下溢

延伸阅读:

- [共识智能合约已知攻击] - 对最重要弱点的可读解释，有很多样本代码。
- [SWC 注册] - 适用于以太坊和智能合约的 CWE 的管理列表

安全工具 {#security-tools}

虽然了解以太坊安全基础知识和聘请专业审计公司审查您的代码是无可替代的，但有许多工具可以帮助突出您的代码中的潜在问题。

智能合约安全性 {#smart-contract-security}

Slither - ***用 Python 3 编写的 Solidity 静态分析框架。***

- [GitHub]

MythX - ***以太坊智能合约的安全分析应用程序接口。***

- [mythx.io]
- [相关文档]

Mythril - *以太坊虚拟机字节码安全分析工具。*

- mythril
- 相关文档

Manticore - *在智能合约和二进制文件上使用符号执行工具的命令行界面。*

- GitHub
- 相关文档

Securify - *以太坊智能合约安全分析工具。*

- securify.chainsecurity.com
- Discord

ERC20 Verifier - *用于检查合约是否符合 ERC20 标准的验证工具。*

- erc20-verifier.openzeppelin.com
- 论坛

形式化验证 {#formal-verification}

有关形式验证的信息

- 智能合约的形式验证如何工作 *2018 年 7 月 20 日 - Brian Marick*
- 形式验证如何确保智能合约无懈可击 *2018 年 1 月 29 日 - Bernard Mueller*

使用工具 {#using-tools}

智能合约安全分析最受欢迎的两个工具是：

- Slither 来自 Trail of Bits（托管版本：Crytic）
- Mythril 来自 ConsenSys（托管版本：MythX）

两者都是分析代码和报告问题的实用工具。 每个人都有一个 [commercial] 托管版本，但也可以免费在本地运行。 下面是如何运行 Slither 的一个快速示例，这个示例是在方便的 Docker 映像 `trailofbits/eth-security-toolbox` 中提供的。 如果您还没有安装 ， 您将需要 安装 Docker。

```
$ mkdir test-slither
$ curl
https://gist.githubusercontent.com/epheph/460e6ff4f02c4ac582794a41e1f103bf/raw/9e761af793
d4414c39370f063a46a3f71686b579/gistfile1.txt > bad-contract.sol
$ docker run -v `pwd`:/share  -it --rm trailofbits/eth-security-toolbox
docker$ cd /share
docker$ solc-select 0.5.11
docker$ slither bad-contract.sol
```

将生成此输出：

```
ethsec@1435b241ca60:/share$ slither bad-contract.sol
INFO:Detectors:
Reentrancy in Victim.withdraw() (bad-contract.sol#11-16):
    External calls:
    - (success) = msg.sender.call.value(amount)() (bad-contract.sol#13)
    State variables written after the call(s):
    - balances[msg.sender] = 0 (bad-contract.sol#15)
```

```
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities
INFO:Detectors:
Low level call in Victim.withdraw() (bad-contract.sol#11-16):
    - (success) = msg.sender.call.value(amount)() (bad-contract.sol#13)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls
INFO:Slither:bad-contract.sol analyzed (1 contracts with 46 detectors), 2 result(s) found
INFO:Slither:Use https://crytic.io/ to get access to additional detectors and GitHub integration
```

Slither 已经在这里确定了重新进入的可能性。确定问题可能出现的关键行号，并为我们提供一个链接以获得有关问题的更多细节：

> 参考：https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities

允许您快速了解代码的潜在问题。 像所有自动化测试工具一样，Slither 并不完美，它在报告一侧出现了太多错误。 即使在不存在可开发的脆弱性的情况下，它也可以就潜在的重返提出警告。 在代码更改之间在 Slither 输出中查看 DIFERENER 常常非常有启发性，可以帮助发现比等待您的项目代码完成早得多的漏洞。

延伸阅读 {#further-reading}

智能合约安全性最佳做法指南

- consensys.github.io/smart-contract-best-practices/
- GitHub
- 安全性建议和最佳做法合集

智能合约安全验证标准 (SCSVS)

- securing.github.io/SCSVS/

还有哪些社区资源对您有所帮助？ 请编辑本页面并添加！

相关教程 {#related-tutorials}

- 安全的开发工作流程
- 如何使用 Slither 来查找智能合约漏洞
- 如何使用 Manticore 查找智能合约漏洞
- 安全准则
- 令牌安全

---

title: 测试智能合约
description:
lang: zh
sidebar: true

incomplete: true

测试工具和库 {#testing-tools-and-libraries}

Waffle - *高级智能合约开发和测试的框架（基于 ethers.js）。*

- getwaffle.io
- GitHub

Solidity-Coverage - ***替代性 Solidity 代码覆盖工具。***

- GitHub

hevm - ***以太坊虚拟机实现，专门用于单元测试以及调试智能合约。***

- GitHub
- DappHub 聊天室

Whiteblock Genesis - ***端到端开发沙盒和区块链测试平台。***

- Whiteblock.io
- 相关文档
- GitHub

OpenZepelin 测试环境 - ***闪电快速智能合约测试。 带来优秀测试体验的单行设置。***

- GitHub
- 相关文档

OpenZeppelin 测试助手—— ***用于以太坊智能合约测试的断言库。 请确保您的合约行为如同预期的一样！***

- GitHub
- 相关文档

相关教程 {#related-tutorials}

- Solidity 和 Truffle 持续集成设置*——如何为 Truffle 测试搭配有用的插件，并设置 Travis 或 Circle CI。*
- 测试产品概述*——不同测试产品的概述和比较。*
- 如何使用 Echidna 测试智能合约
- 如何使用 Manticore 查找智能合约漏洞
- 如何使用 Slither 来查找智能合约漏洞
- 如何模拟测试 Solidity 合约
- 如何将 Truffle 测试迁移至 OpenZeppelin 测试环境
- 如何在部署到网络后测试智能合约

延伸阅读 {#further-reading}

还有哪些社区资源对您有所帮助？ 编辑并添加本页面！

title: 以太坊开发标准
description:
lang: zh
sidebar: true

incomplete: true

标准概述 {#standards-overview}

以太坊社区已经采用了许多标准，这有助于在项目的不同实现中保持统一的兼容性（例如以太坊客户和钱包），并确保智能合约和 dapps 仍保持兼容。

通常，标准作为以太坊改进提议 (EIP) 提出，由社区成员通过标准流程讨论确定。

- EIPS 简介
- EIP 列表

- EIP GitHub 存储库
- EIP 讨论板
- 以太坊治理简介
- 以太坊治理概述 *2019 年 3 月 31 日 - Boris Mann*
- 以太坊协议开发治理和网络升级协调 *2020 年 3 月 23 日 - Hudson Jameson*
- 以太坊核心开发者会议播放列表 *（YouTube 播放列表）*

标准的类型 {#types-of-standards}

某些 EIP 与应用层规范相关（例如智能合约标准格式），这类规范被称为以太坊意见征求 (ERC)。 许多 ERC 是以太坊生态系统中被广泛使用的关键标准。

- ERC 列表

代币标准 {#token-standards}

- ERC-20 - 同质化（可互换）代币的标准接口，比如投票代币、质押代币或虚拟货币。
- ERC-721 - 非同质化代币的标准接口，比如艺术作品或歌曲的契约。
- ERC-777 - 关于 ERC-20 的代币标准改进。
- ERC-1155 - 一个能包括同质化和非同质化资产的代币标准。

了解更多关于 代币标准的信息。

延伸阅读 {#further-reading}

你知道有什么社区资源帮助过你吗？ 编辑并添加本页面！

title: ERC-1155 多代币标准
description:
lang: zh

sidebar: true

介绍 {#introduction}

用于多种代币管理的合约标准接口。 单个部署的合约可以包括同质化代币、非同质化代币或其他配置（如半同质化代币）的任何组合。

多代币标准是什么?

它的目的很单纯，就是创建一个智能合约接口，可以代表和控制任何数量的同质化和非同质化代币类型。 这样一来，ERC-1155 代币就具有与 ERC-20 和 ERC-721 代币相同的功能，甚至可以同时使用这两者的功能。 而最重要的是，它能改善这两种标准的功能，使其更有效率，并纠正 ERC-20 和 ERC-721 标准上明显的实施错误。

EIP-1155 中对 ERC-1155 代币进行了全面的描述。

前提条件 {#prerequisites}

为了更好地理解这一页面的内容，我们建议您先阅读代币标准、ERC-20 和 ERC-721。

ERC-1155 的功能和特点： {#body}

- 批量传输：通过一次合约调用传输多种资产。
- 批量余额：在一次调用中获取多个资产的余额。

- 批量审批：审批同一地址的所有代币。
- Hook：接收代币的钩子函数。
- 支持非同质化代币：如果供应量仅为 1，将其作为非同质化代币处理。
- 安全转账规则：安全转账规则集。

批量传输 {#batch-transfers}

批量传输与常规 ERC-20 传输非常相似。 让我们看看常规的 ERC-20 传输功能：

```
// ERC-20
function transferFrom(address from, address to, uint256 value) external returns (bool);

// ERC-1155
function safeBatchTransferFrom(
    address _from,
    address _to,
    uint256[] calldata _ids,
    uint256[] calldata _values,
    bytes calldata _data
) external;
```

ERC-1155 中唯一的区别是我们将数值作为数组传递，我们也传递了数组 id。 例如，给出 `ids=[3, 6, 13]` 和 `values=[100, 200, 5]`，传输结果将是

1. 将 id 3 的 100 个代币从 `_from` 传输到 `_to`。
2. 将 id 6 的 200 个代币从 `_from` 传输到 `_to`。
3. 将 id 13 的 5 个代币从 `_from` 转移到 `_to`。

在 ERC-1155 中，我们只有 `transferFrom`，没有 `transfer`。 要像常规的 `transfer`一样使用它，只需将 "from" 地址设为调用该函数的地址。

批量余额 {#batch-balance}

相应的 ERC-20 `balanceOf` 调用同样具有支持批处理的相应函数。 作为对比，这是 ERC-20 版本：

```
// ERC-20
function balanceOf(address owner) external view returns (uint256);

// ERC-1155
function balanceOfBatch(
    address[] calldata _owners,
    uint256[] calldata _ids
) external view returns (uint256[] memory);
```

调用余额查询更简单的是，我们可以在单次调用中获取多个余额。 参数中传递所有者账户数组和代币的 id 数组。

例如，对于给出的 `_ids=[3, 6, 13]` 和 `_owners=[0xbeef..., 0x1337..., 0x1111...]`，返回值将为：

```
[
    balanceOf(0xbeef...),
    balanceOf(0x1337...),
    balanceOf(0x1111...)
]
```

批量审批 {#batch-approval}

```
// ERC-1155
function setApprovalForAll(
    address _operator,
    bool _approved
) external;

function isApprovedForAll(
    address _owner,
    address _operator
) external view returns (bool);
```

审批过程与 ERC-20 略有不同。 这里不是批准特定金额，而是通过 `setApprovalForall` 函数设置操作帐户为已批准或未批准。

查看当前的审批状态可以通过 `isApprovedForall` 完成。 如您所见，要么全部批准，要么不批准。 不能定义要批准代币的数量，甚至代币类型。

这是考虑到简洁性而故意设计的。 您只能批准一个地址的所有代币。

接收钩子 {#receive-hook}

```
function onERC1155BatchReceived(
    address _operator,
    address _from,
    uint256[] calldata _ids,
    uint256[] calldata _values,
    bytes calldata _data
) external returns(bytes4);
```

基于 EIP-165 的协议支持，ERC-1155 只支持智能合约的接收钩子函数。 钩子函数必须返回一个事先预定义的 4 字节值，这个值被指定为：

```
bytes4(keccak256("onERC1155BatchReceived(address,address,uint256[],uint256[],bytes)"))
```

当接收合约返回这一值时，意味着合约知道如何处理 ERC-1155 代币并接受转账。 太好了，代币不会再卡在合约中了！

支持非同质化代币 {#nft-support}

当供应量仅为 1 时，代币本质上就是一个非同质化的代币 (NFT)。 按照 ERC-721 的标准，您可以定义一个元数据网址。 客户端可以读取并修改网址，请参阅这里。

安全转账规则 {#safe-transfer-rule}

在前面的解释中，我们已经提到过一些安全转账规则。 现在我们来看一下最重要的规则：

1. 调用者必须获得批准才能从 `_from` 的账户地址消费代币，或者调用者账户地址必须与 `_from` 的账户地址相同。
2. 在以下情况下，转账调用将回退
    1. `_to` 地址为 0；

2. `_ids` 的长度与 `_values` 的长度不同；
3. `_ids` 中代币持有者的任何余额低于发送给接收者的相应 `_value` 金额。
4. 出现任何其他错误。

注意：包括钩子在内的所有批处理函数也均作为非批处理的版本存在。 这样做是为了提高燃料效率，考虑到只转移一种资产可能仍然是最常用的方式。 简洁起见，我们没有在这里介绍这些非批处理的版本，包括安全转账规则。 名称是相同的，只需移除 'Batch'。

延伸阅读 {#further-reading}

- EIP-1155：多代币标准
- ERC-1155：Openzepelin 文档
- ERC-1155：Github 代码库
- Alchemy NFT API

---

title: ERC-20 代币标准
description:
lang: zh

sidebar: true

介绍 {#introduction}

什么叫做代币?

代币可以在以太坊中表示任何东西：

- 在线平台中的信誉积分
- 游戏中一个角色的技能
- 彩票卷
- 金融资产类似于公司股份的资产
- 像美元一样的法定货币
- 一盎司黄金
- 及更多...

以太坊的这种强大特点必须以强有力的标准来处理，对吗？ 这正是 ERC-20 发挥其作用的地方！ 此标准允许开发者构建可与其他产品和服务互相操作的代币应用程序。

什么是 ERC-20?

ERC-20 提供了一个同质化代币的标准，换句话说，每个代币与另一个代币（在类型和价值上）完全相同。 例如，一个 ERC-20 代币就像以太币一样，意味着一个代币会并永远会与其他代币一样。

前置要求 {#prerequisites}

- 帐户
- 智能合约
- 代币标准

正文内容 {#body}

ERC-20（以太坊意见征求 20）由 Fabian Vogelsteller 提出于 2015 年 11 月。这是一个能实现智能合约中代币的应用程序接口标准。

ERC-20 的功能示例包括：

- 将代币从一个帐户转到另一个帐户
- 获取帐户的当前代币余额
- 获取网络上可用代币的总供应量
- 批准一个帐户中一定的代币金额由第三方帐户使用

如果智能合约实施了下列方法和事件，它可以被称为 ERC-20 代币合约， 一旦部署，将负责跟踪以太坊上创建的代币。

来自 EIP-20：

方法 {#methods}

```
function name() public view returns (string)
function symbol() public view returns (string)
function decimals() public view returns (uint8)
function totalSupply() public view returns (uint256)
function balanceOf(address _owner) public view returns (uint256 balance)
function transfer(address _to, uint256 _value) public returns (bool success)
function transferFrom(address _from, address _to, uint256 _value) public returns (bool
success)
function approve(address _spender, uint256 _value) public returns (bool success)
function allowance(address _owner, address _spender) public view returns (uint256
remaining)
```

事件 {#events}

```
event Transfer(address indexed _from, address indexed _to, uint256 _value)
event Approval(address indexed _owner, address indexed _spender, uint256 _value)
```

示例 {#web3py-example}

让我们看看如此重要的一个标准是如何使我们能够简单地检查以太坊上的任何 ERC-20 代币合约。 我们只需要合约的应用程序二进制接口 (ABI) 来创造一个 ERC-20 代币界面。 下面我们将使用一个简化的应用程序二进制接口，使其成为一个简单易用的例子。

Web3.py 示例 {#web3py-example}

首先，确认已经安装了 Web3.pyPython 库。

```
$ pip install web3
```

```
from web3 import Web3


w3 = Web3(Web3.HTTPProvider("https://cloudflare-eth.com"))

dai_token_addr = "0x6B175474E89094C44Da98b954EedeAC495271d0F"     # DAI
weth_token_addr = "0xC02aaA39b223FE8D0A0e5C4F27eAD9083C756Cc2"    # Wrapped ether (WETH)

acc_address = "0xA478c2975Ab1Ea89e8196811F51A7B7Ade33eB11"        # Uniswap V2: DAI 2
```

```
# This is a simplified Contract Application Binary Interface (ABI) of an ERC-20 Token
Contract.
# It will expose only the methods: balanceOf(address), decimals(), symbol() and
totalSupply()
simplified_abi = [
    {
        'inputs': [{'internalType': 'address', 'name': 'account', 'type': 'address'}],
        'name': 'balanceOf',
        'outputs': [{'internalType': 'uint256', 'name': '', 'type': 'uint256'}],
        'stateMutability': 'view', 'type': 'function', 'constant': True
    },
    {
        'inputs': [],
        'name': 'decimals',
        'outputs': [{'internalType': 'uint8', 'name': '', 'type': 'uint8'}],
        'stateMutability': 'view', 'type': 'function', 'constant': True
    },
    {
        'inputs': [],
        'name': 'symbol',
        'outputs': [{'internalType': 'string', 'name': '', 'type': 'string'}],
        'stateMutability': 'view', 'type': 'function', 'constant': True
    },
    {
        'inputs': [],
        'name': 'totalSupply',
        'outputs': [{'internalType': 'uint256', 'name': '', 'type': 'uint256'}],
        'stateMutability': 'view', 'type': 'function', 'constant': True
    }
]

dai_contract = w3.eth.contract(address=w3.toChecksumAddress(dai_token_addr),
abi=simplified_abi)
symbol = dai_contract.functions.symbol().call()
decimals = dai_contract.functions.decimals().call()
totalSupply = dai_contract.functions.totalSupply().call() / 10**decimals
addr_balance = dai_contract.functions.balanceOf(acc_address).call() / 10**decimals

#  DAI
print("===== %s =====" % symbol)
print("Total Supply:", totalSupply)
print("Addr Balance:", addr_balance)

weth_contract = w3.eth.contract(address=w3.toChecksumAddress(weth_token_addr),
abi=simplified_abi)
symbol = weth_contract.functions.symbol().call()
decimals = weth_contract.functions.decimals().call()
totalSupply = weth_contract.functions.totalSupply().call() / 10**decimals
addr_balance = weth_contract.functions.balanceOf(acc_address).call() / 10**decimals

#  WETH
print("===== %s =====" % symbol)
print("Total Supply:", totalSupply)
print("Addr Balance:", addr_balance)
```

延伸阅读 {#further-reading}

- EIP-20：ERC-20 代币标准
- OpenZepelin - 代币
- OpenZepelin - ERC-20 实施

---

title: ERC-721 非同质化代币标准
description:
lang: zh

sidebar: true

介绍 {#introduction}

什么是非同质化代币?

非同质化代币（NFT）用于以唯一的方式标识某人或者某物。 此类型的代币可以被完美地用于出售下列物品的平台：收藏品、密钥、彩票、音乐会座位编号、体育比赛等。 这种类型的代币有着惊人的潜力，因此它需要一个适当的标准。ERC-721 就是为解决这个问题而来！

ERC-721 是什么?

ERC-721 为 NFT 引入了一个标准，换言之， 这种类型的代币是独一无二的，并且可能与来自同一智能合约的另一代币有不同的价值，也许是因为它的年份、稀有性、甚至是它的观感。 稍等，看起来怎么样呢?

是的。 所有 NFTs 都有一个 `uint256` 变量，名为 `tokenId`，所以对于任何 ERC-721 合约，这对值 `contract address, tokenId` 必须是全局唯一的。 也就是说，dApp 可以有一个“转换器”，该转换器使用 `tokenId` 输入和输出一些非常有趣的事物的图像， 例如僵尸、武器、技能或非常可爱的猫咪！

前置要求 {#prerequisites}

- 帐户
- 智能合约
- 代币标准

正文 {#body}

ERC-721（Ethereum Request for Comments 721），由 William Entriken、Dieter Shirley、Jacob Evans、Nastassia Sachs 在 2018 年 1 月提出，是一个在智能合约中实现代币 API 的非同质化代币标准。

它提供了一些功能，例如将代币从一个帐户转移到另一个帐户，获取帐户的当前代币余额，获取代币的所有者，以及整个网络的可用代币总供应量。 除此之外，它还具有其他功能，例如批准帐户中一定数量的代币可以被第三方帐户转移。

如果一个智能合约实现了下列方法和事件，它就可以被称为 ERC-721 非同质化代币合约。 一旦被部署，它将负责跟踪在以太坊上创建的代币。

来自 EIP-721 :

方法 {#methods}

```
    function balanceOf(address _owner) external view returns (uint256);
    function ownerOf(uint256 _tokenId) external view returns (address);
    function safeTransferFrom(address _from, address _to, uint256 _tokenId, bytes data)
external payable;
    function safeTransferFrom(address _from, address _to, uint256 _tokenId) external
payable;
    function transferFrom(address _from, address _to, uint256 _tokenId) external payable;
    function approve(address _approved, uint256 _tokenId) external payable;
    function setApprovalForAll(address _operator, bool _approved) external;
    function getApproved(uint256 _tokenId) external view returns (address);
    function isApprovedForAll(address _owner, address _operator) external view returns
(bool);
```

事件 {#events}

```
    event Transfer(address indexed _from, address indexed _to, uint256 indexed _tokenId);
    event Approval(address indexed _owner, address indexed _approved, uint256 indexed
_tokenId);
    event ApprovalForAll(address indexed _owner, address indexed _operator, bool
_approved);
```

示例 {#web3py-example}

让我们看看一个标准是多么重要，它使我们能够简单地在以太坊上检查任何 ERC-721 代币合约。 我们只需要合约的应用程序二进制接口（ABI）就可以创造任何 ERC-721 代币的接口。 你可以在下面看到，我们将使用简化的 ABI，来做一个低难度的例子。

Web3.py 示例 {#web3py-example}

首先，请确保您已安装 Web3.py Python 库：

```
$ pip install web3
```

```
from web3 import Web3
from web3._utils.events import get_event_data


w3 = Web3(Web3.HTTPProvider("https://cloudflare-eth.com"))

ck_token_addr = "0x06012c8cf97BEaD5deAe237070F9587f8E7A266d"    # CryptoKitties Contract

acc_address = "0xb1690C08E213a35Ed9bAb7B318DE14420FB57d8C"      # CryptoKitties Sales
Auction

# This is a simplified Contract Application Binary Interface (ABI) of an ERC-721 NFT
Contract.
# It will expose only the methods: balanceOf(address), name(), ownerOf(tokenId),
symbol(), totalSupply()
simplified_abi = [
    {
        'inputs': [{'internalType': 'address', 'name': 'owner', 'type': 'address'}],
```

```
        'name': 'balanceOf',
        'outputs': [{'internalType': 'uint256', 'name': '', 'type': 'uint256'}],
        'payable': False, 'stateMutability': 'view', 'type': 'function', 'constant': True
    },
    {
        'inputs': [],
        'name': 'name',
        'outputs': [{'internalType': 'string', 'name': '', 'type': 'string'}],
        'stateMutability': 'view', 'type': 'function', 'constant': True
    },
    {
        'inputs': [{'internalType': 'uint256', 'name': 'tokenId', 'type': 'uint256'}],
        'name': 'ownerOf',
        'outputs': [{'internalType': 'address', 'name': '', 'type': 'address'}],
        'payable': False, 'stateMutability': 'view', 'type': 'function', 'constant': True
    },
    {
        'inputs': [],
        'name': 'symbol',
        'outputs': [{'internalType': 'string', 'name': '', 'type': 'string'}],
        'stateMutability': 'view', 'type': 'function', 'constant': True
    },
    {
        'inputs': [],
        'name': 'totalSupply',
        'outputs': [{'internalType': 'uint256', 'name': '', 'type': 'uint256'}],
        'stateMutability': 'view', 'type': 'function', 'constant': True
    },
]

ck_extra_abi = [
    {
        'inputs': [],
        'name': 'pregnantKitties',
        'outputs': [{'name': '', 'type': 'uint256'}],
        'payable': False, 'stateMutability': 'view', 'type': 'function', 'constant': True
    },
    {
        'inputs': [{'name': '_kittyId', 'type': 'uint256'}],
        'name': 'isPregnant',
        'outputs': [{'name': '', 'type': 'bool'}],
        'payable': False, 'stateMutability': 'view', 'type': 'function', 'constant': True
    }
]

ck_contract = w3.eth.contract(address=w3.toChecksumAddress(ck_token_addr),
abi=simplified_abi+ck_extra_abi)
name = ck_contract.functions.name().call()
symbol = ck_contract.functions.symbol().call()
kitties_auctions = ck_contract.functions.balanceOf(acc_address).call()
print(f"{name} [{symbol}] NFTs in Auctions: {kitties_auctions}")

pregnant_kitties = ck_contract.functions.pregnantKitties().call()
print(f"{name} [{symbol}] NFTs Pregnants: {pregnant_kitties}")
```

```
# Using the Transfer Event ABI to get info about transferred Kitties.
tx_event_abi = {
    'anonymous': False,
    'inputs': [
        {'indexed': False, 'name': 'from', 'type': 'address'},
        {'indexed': False, 'name': 'to', 'type': 'address'},
        {'indexed': False, 'name': 'tokenId', 'type': 'uint256'}],
    'name': 'Transfer',
    'type': 'event'
}

# We need the event's signature to filter the logs
event_signature = w3.sha3(text="Transfer(address,address,uint256)").hex()

logs = w3.eth.getLogs({
    "fromBlock": w3.eth.blockNumber - 120,
    "address": w3.toChecksumAddress(ck_token_addr),
    "topics": [event_signature]
})

# Notes:
#   - 120 blocks is the max range for CloudFlare Provider
#   - If you didn't find any Transfer event you can also try to get a tokenId at:
#       https://etherscan.io/address/0x06012c8cf97BEaD5deAe237070F9587f8E7A266d#events
#       Click to expand the event's logs and copy its "tokenId" argument

recent_tx = [get_event_data(w3.codec, tx_event_abi, log)["args"] for log in logs]

kitty_id = recent_tx[0]['tokenId'] # Paste the "tokenId" here from the link above
is_pregnant = ck_contract.functions.isPregnant(kitty_id).call()
print(f"{name} [{symbol}] NFTs {kitty_id} is pregnant: {is_pregnant}")
```

除了标准事件之外，CryptoKitties 合约还有其它一些有趣的事件。

让我们看看其中的两个，`Pregnant` 和 `Birth`。

```
# Using the Pregnant and Birth Events ABI to get info about new Kitties.
ck_extra_events_abi = [
    {
        'anonymous': False,
        'inputs': [
            {'indexed': False, 'name': 'owner', 'type': 'address'},
            {'indexed': False, 'name': 'matronId', 'type': 'uint256'},
            {'indexed': False, 'name': 'sireId', 'type': 'uint256'},
            {'indexed': False, 'name': 'cooldownEndBlock', 'type': 'uint256'}],
        'name': 'Pregnant',
        'type': 'event'
    },
    {
        'anonymous': False,
        'inputs': [
            {'indexed': False, 'name': 'owner', 'type': 'address'},
            {'indexed': False, 'name': 'kittyId', 'type': 'uint256'},
            {'indexed': False, 'name': 'matronId', 'type': 'uint256'},
            {'indexed': False, 'name': 'sireId', 'type': 'uint256'},
```

```
            {'indexed': False, 'name': 'genes', 'type': 'uint256'}],
        'name': 'Birth',
        'type': 'event'
    }]

# We need the event's signature to filter the logs
ck_event_signatures = [
    w3.sha3(text="Pregnant(address,uint256,uint256,uint256)").hex(),
    w3.sha3(text="Birth(address,uint256,uint256,uint256,uint256)").hex(),
]

# Here is a Pregnant Event:
# -
https://etherscan.io/tx/0xc97eb514a41004acc447ac9d0d6a27ea6da305ac8b877dff37e49db42e1f8ce
f#eventlog
pregnant_logs = w3.eth.getLogs({
    "fromBlock": w3.eth.blockNumber - 120,
    "address": w3.toChecksumAddress(ck_token_addr),
    "topics": [ck_event_signatures[0]]
})

recent_pregnants = [get_event_data(w3.codec, ck_extra_events_abi[0], log)["args"] for log
in pregnant_logs]

# Here is a Birth Event:
# -
https://etherscan.io/tx/0x3978028e08a25bb4c44f7877eb3573b9644309c044bf087e335397f16356340
a
birth_logs = w3.eth.getLogs({
    "fromBlock": w3.eth.blockNumber - 120,
    "address": w3.toChecksumAddress(ck_token_addr),
    "topics": [ck_event_signatures[1]]
})

recent_births = [get_event_data(w3.codec, ck_extra_events_abi[1], log)["args"] for log in
birth_logs]
```

热门的 NFT {#popular-nfts}

- Etherscan NFT Tracker 列出了以太坊上交易量最大的 NFT。
- CryptoKitties 是一个围绕着我们称之为加密猫的可繁殖、可收藏和可爱的生物游戏。
- Sorare 是一场全球迷幻足球赛，您可以在这里收集有限版本的收藏品，管理您的球队，参加比赛以获得奖品。
- 以太坊域名服务 (ENS) 提供了一种安全和去中心化的方式，用人类可读的名字来处理链上和链下的资源。
- Unstoppable Domains 是一个坐落在旧金山，在区块链上创建域名的公司。 区块链域名用人类可读的名字替换加密货币地址，并且可用于支撑抗审查的网站。
- Gods Unchained Cards 是以太坊区块链上的 TCG（集换式卡牌游戏），它使用 NFT 来给游戏中的资产带来实际所有权。
- 无聊猿游艇俱乐部是一个由 10,000 个独特 NFT 组成的系列收藏品，这些收藏品不仅仅是非常罕见的艺术品，还是俱乐部入会的标志，可为成员提供多种特权和福利。在社区的努力下，这些特权和福利还会随着时间的推移不断增加。

延伸阅读 {#further-reading}

- EIP-721： ERC-721 非同质化代币标准
- OpenZeppelin - ERC-721 文档
- OpenZeppelin - ERC-721 实施
- Alchemy NFT API

---

title: ERC-777 代币标准
description:
lang: zh

sidebar: true

介绍? {#introduction}

ERC-777 是一个易于交易的通证标准，可改进现有的 ERC-20 标准。

前体条件 {#prerequisites}

为了更好地理解本文，我们建议你首先阅读以下内容 ERC-20。

ERC-777 对 ERC-20 提出了哪些改进？ {#-erc-777-vs-erc-20}

与 ERC-20 相比，ERC-777 提供了以下改进。

钩子 {#hooks}

钩子是智能合约代码中描述的一个函数。 钩子将会在代币通过合约发送或者接收时调用。 这将允许智能合约对进出的通证做出互动。

钩子是使用 ERC-1820 标准注册及发现利用的。

为什么钩子很棒？ {#why-are-hooks-great}

1. 钩子允许向合同发送代币并在单笔交易中通知合约，不像 ERC-20， 需要双重呼叫（`approve`/`transferFrom`）才能实现这一点。
2. 未登记钩子的合约与 ERC-777 不相容。 接收合约没有注册钩子时，发送合约会中止交易。 这可以防止意外向非 ERC-777 智能合约转账。
3. 钩子可以拒绝交易。

小数位数 {#decimals}

该标准还解决了在 ERC-20 中造成的 `decimals` 的混乱。 这种清晰度提升了开发者体验。

后向兼容 ERC-20 {#backwards-compatibility-with-erc-20}

EC-777 合同可以与类似于 ERC-20 合同的合同进行互动。

延伸阅读 {#further-reading}

ERC-777 代币标准

title: 代币标准
description:
lang: zh
sidebar: true

incomplete: true

介绍 {#introduction}

许多以太坊开发标准之一侧重于代币接口。 这些标准有助于确保智能合约仍然可以被撰写，例如，当新项目签发代币时，它与现有分散化的交易是兼容的。

前置要求 {#prerequisites}

- 以太坊开发标准
- 智能合约

代币标准 {#token-standards}

以下是以太坊上最受欢迎的一些代币标准：

- ERC-20 - 同质化（可互换）代币的标准接口，比如投票代币、质押代币或虚拟货币。
- ERC-721 - 非同质化代币的标准接口，比如艺术作品或歌曲的契约。
- ERC-777 - ERC-777 使人们能够通过代币上创建额外的功能，例如用于改善交易私密性的混合合约，或是在您不慎丢失私钥时的紧急恢复功能。
- ERC-1155 - ERC-1155 允许更有效的交易和打包交易，从而节省燃料成本。 此代币标准允许创建实用代币（例如 $BNB 或 $BAT）和加密朋克之类的非同质化代币。

延伸阅读 {#further-reading}

你知道有什么社区资源帮助过你吗？ 编辑并添加本页面！

相关教程 {#related-tutorials}

- 代币集成清单 - *与代币互动时要考虑的事项清单。*
- 了解 ERC20 代币智能合约 — — *介绍如何在以太坊测试网络上部署您的第一个智能合约。*
- 通过 Solidity 智能合约转账和批准 ERC20 代币_ - 如何使用智能合约与使用 Solidity 语言的代币进行交互。_
- 实现 ERC721 市场[指导手册] - *如何将代币化的物品放在分散的分类板上。*

---

title: 翻译计划
lang: zh
description: 如何为 ethereum.org 的翻译计划贡献力量

sidebar: true

翻译计划 {#translation-program}

翻译计划是一项将 ethereum.org 翻译成不同语言的协同工作计划，方便全球数十亿非英语人士更容易访问该网站。

我们迄今取得的进展：

- 超过 **2,000** 名翻译者参与
- 网站支持 **37** 种语言
- 2021 年翻译了 **190 万**单词

如果您想参与其中并帮助我们通过将网站翻译成您的语言来发展全球以太坊社区，请遵循下面的步骤！

领取您的 POAP 代币！ 如果您在 2021 年参与了 ethereum.org 翻译计划，则会为您准备独特 POAP。
关于 POAP 的更多信息

使命和愿景 {#mission-and-vision}

以太坊社区旨在成为全球化和包容性的社区，但其大部分内容仅面向英语使用者，而忽略了全球 60 亿非英语使用者。 为了使 ethereum.org 成为全球社区进入以太坊的门户，我们认为很有必要为非英语国家人士按母语提供以太坊内容。

Ethereum.org 翻译计划旨在通过将 Ethereum.org 和其他以太坊内容翻译成尽可能多的语言，让所有人都能参与以太坊。

我们的使命

- 提供网站的翻译版本，支持世界各地的访问者通过母语了解以太坊
- 促进更多成员加入全球以太坊社区
- 允许以更易获得和更具包容性的方式分享以太坊的信息和知识
- 支持社区成员为以太坊贡献翻译，并在生态系统中留下自己的痕迹
- 寻找、联系希望参与生态系统的热情贡献者，并为其提供指导

我们的愿景

- 为来自世界上尽可能多的国家/地区的以太坊社区成员翻译基本内容
- 支持跨语言的知识共享，创建一个信息灵通、受过更好教育的以太坊社区
- 通过消除非英语人士加入生态系统的语言障碍，提高以太坊的包容性和可访问性

如何翻译 {#how-to-translate}

1. **加入我们的 Crowdin 项目**
   您需要登录 Crowdin 账户，如果没有账户，请注册。 只需要提供电子邮件帐户和密码。
2. **打开您要翻译的语言，找到要翻译的文档**
   网站内容被分为若干文档和语言版本。 您可以在右边查看每份文档的进度 - 如果翻译进度低于 100%，请贡献您的力量！

   找不到您的语言列表？ 提出问题

   Crowdin 上已翻译和未翻译的文件

   关于内容版本的说明：我们使用 Crowdin 版本管理来避免已添加内容的翻译延误。 当您查看一种语言时（例如菲律宾语），您会看到每个网站版本（“v2.0”、“v2.1”等）的文件夹。

   我们建议您先翻译最低的版本，然后再逐步升级到更高的版本 (v.2.0 → v.2.1 → v.2.2 → ⋯)，因为网站上会先更新较低版本。

   了解关于 ethereum.org 版本的更多信息
3. **翻译**
   选择您要翻译的文件后，随即在线编辑器中打开。 如果您从未使用过 Crowdin，您可以通过这个快速指南来了解基础知识。

   Crowdin 在线编辑器

1 - 左侧边栏

- 未翻译（红色）- 尚未翻译的文本。 这些是您应该翻译的文本。
- 已翻译（绿色）- 已翻译但尚未审核的文本。 欢迎您提出其他翻译建议，或使用编辑器中的"“+”"和“-”按钮对现有翻译投票。
- 已审核（勾号）- 已审核并且目前已经在网站上展示的文本。

您也可以使用顶部的按钮搜索特定文本，按状态筛选或更改视图。

2 - 编辑区域

主要翻译区域 - 源文本显示在顶部，还有更多上下文和屏幕截图（如果有）。 要添加新翻译，请在“请在此处输入翻译”字段中输入您的翻译，然后点击“保存”。

您还可以在此部分找到现有文本翻译和其他语言的翻译，以及翻译记忆库的匹配和机器翻译建议。

3 - 右侧边栏

您可以在这里找到评论、翻译记忆库和词汇表条目。 默认视图会显示评论，允许翻译者相互交流、提出问题或报告错误翻译。

通过使用上面的按钮，您还可以切换到翻译记忆库，在其中搜索现有翻译，或者切换到词汇表，其中包含关键术语的描述和标准翻译。

想了解更多信息？ 可随时查看文档：关于使用 Crowdin 在线编辑器。

4. **审核过程**
一旦完成翻译（即一个内容版本的所有文件显示为 100%），将由我们的专业翻译服务审核（并有可能编辑）内容。 一旦审核完成（即审核进度为 100%），我们会将其添加到网站上。

还有其他问题？ 或者想与我们的团队和其他翻译者合作？ 请在我们的 ethereum.org Discord 服务器的 #translations channel 中发布

您也可以通过向 translations@ethereum.org 发送邮件联系我们

感谢您参与 ethereum.org 翻译计划！

如果您在帮助我们进行翻译，您可能会在翻译常见问题中找到一些实用信息。
{" "}翻译 ethereum.org 常见问题

更多资源 {#resources}

工具

- Microsoft 语言门户 - *用于查找和检查技术术语的标准翻译*
- Linguee。 - *翻译和字典搜索引擎，可按词或短语搜索。*
- Proz 术语搜索。 - *专业术语的翻译字典和词汇表数据库。*
- Eurotermbank - *42 种语言的欧洲术语集*

社区

- 特定语言翻译组。 - *让 ethereum.org 翻译者加入翻译组的计划。*
- 中文翻译组 - *方便中文翻译者之间协作的 Notion 页面*

版本概述

- 内容版本 - *ethereum.org 的每个内容版本中包含哪些页面*

最近更新

要了解翻译计划的最新进展，您可以关注以太坊基金会博客。

- 2020 年 12 月里程碑更新
- 2020 年 7 月里程碑更新
- 2019 年 8 月翻译计划启动

以太坊 Staking Launchpad 翻译 {#staking-launchpad}

我们的翻译爱好者社区也在进行 Staking Launchpad 的翻译工作。 这使得任何人都可以成为以太坊验证者，并帮助保护以太坊网络。 Launchpad 目前支持 17 种语言。

如果您感兴趣，请加入 Crowdin 的 Ethereum Staking Launchpad 翻译项目。 如果 Crowdin 未提供您的语言，您可以在 GitHub 上创建问题，请求增加新语言。

进行中的翻译 {#in-progress}

---

title: 去中心化存储
description: 概述什么是分布式存储和集成到去中心化应用程序中的相关工具。
lang: zh

sidebar: true

不同于由一个公司或组织控制的中心服务器，分布式存储系统由分别持有全局数据中部分数据的用户操作者组成 P2P 网络，形成了一个具有弹性的文件储存、共享系统。 这些可以应用于基于区块链的应用程序或任何 P2P 网络中。

以太坊本身可以用作分布式存储系统，所有智能合约的编码储存就是一种分布式存储。 然而，当涉及大量的数据存储时，就不太符合以太坊的最初目标。 这一区块链正在稳步增长，在本报告撰写之时，以太坊区块链约为 500GB - 1TB（取决于客户端），而网络上的每个节点都需要存储所有这些数据。 如果链上数据量继续扩大（例如 5TB），那么将导致所有节点都无法继续运行。 而且，由于燃料费用，将这么多数据部署到主网的费用将非常昂贵。

由于这些制约因素，我们需要使用不同的区块链或方法，以分布式的方式储存大量数据。

在查看去中心化存储选项时，用户必须牢记几件事情。

- 持久性机制/激励结构
- 数据保留执行
- 去中心化
- 共识

持久性机制/激励结构 {#persistence-mechanism}

基于区块链 {#blockchain-based}

为了使某个数据永久保存，我们需要使用一种持久性机制。 例如，在以太坊中，持久性机制是当运行一个节点时，需要考虑整条链的情况。 新建的数据持续不断地堆积到链的末端，并且要求每个节点复制所有新加入的数据。

这被称为**基于区块链**的持久性机制。

区块链的持久性存储机制会出现区块链过大，维护和存储所有数据十分困难的问题（比如许多机构预测整个区块链网络需要 40ZB 的存储容量）。

区块链还必须有某种类型的激励结构。 出于基于区块链的持久性原因，需要为矿工支付报酬。 当数据被添加到链中时，会为添加数据的节点付费。

基于区块链持久性的平台：

- 以太坊
- Arweave

基于合约 {#contract-based}

我们能直观地感受到，**基于合约**的持久性使得数据不能被每个节点复制并永久存储，而必须根据合约协议进行维护。 这些是与多个节点达成的协议，这些节点承诺在一段时间内保存一份数据。 每当费用耗尽或数据更新时，就必须向这些节点续费，以保持数据的持续性。

在大多数情况下，不是在链上储存所有数据，而是在链上存储定位数据的哈希值。 这样，整条链不需要扩大规模，就能保存所有数据。

基于合约持久性的平台：

- Filecoin
- Skynet
- Storj
- 0Chain

其他注意事项 {#additional-consideration}

星际文件系统是一个储存和访问文件、网站、应用程序和数据的分布式系统。 虽然它没有内置激励计划，但可以与上述任何基于合同的激励解决方案一起使用，以获得更长期的持久性。 另一个将数据持久存储在星际文件系统上的办法是与某项固定服务（表示将您的数据固定在某处）一起使用。 您甚至可以运行自己的星际文件系统节点来为该网络做出贡献，从而将您和/或他人的数据免费持久地存储在星际文件系统上。

- 星际文件系统
- Pinata *（星际文件系统固定服务）*
- web3.storage *（星际文件系统/菲乐币固定服务）*
- Infura *（星际文件系统固定服务）*

数据留存 {#data-retention}

为了保留数据，系统必须有某种机制，以确保数据得到保留。

质疑机制 {#challenge-mechanism}

确保保留数据的一个最常见方法是使用向节点发出的某种类型的密码质询，以确保它们仍然持有数据。 一种简单的方法是查看 Arweave 的访问证明。 他们向节点发出质疑，以查看它们是否在最近的区块和过去的随机区块中都具有数据。 如果节点无法给出答案，则会受到惩罚。

具有质疑机制的去中心化存储类型：

- 0Chain
- Skynet
- Arweave
- Filecoin

去中心化 {#decentrality}

没有很好的工具来衡量平台的去中心化程度，但一般来说，您可能想使用那些没有某种形式的 KYC 的工具证明它们实际上是去中心化的。

不使用 KYC 的去中心化工具：

- 0Chain（实现非 KYC 版本）
- Skynet
- Arweave
- Filecoin
- 星际文件系统
- 以太坊

共识 {#consensus}

这些工具大多有自己的 共识机制 版本，但一般都是基于**工作量证明（PoW）** 或者**权益证明（PoS）**。

基于 PoW：

- Skynet
- Arweave
- 以太坊

基于 PoS：

- 信标链
- Filecoin
- 0Chain

相关工具 {#related-tools}

IPFS - ***即星际文件系统，是以太坊的去中心化存储和文件引用系统。***

- Ipfs.io
- 相关文档
- GitHub

Storj DCS - ***安全、私有、与 S3 兼容的去中心化云对象存储，供开发者使用***

- Storj.io
- 相关文档

Skynet - ***Skynet 是一条去中心化 PoW 链，专用于去中心化网络。***

- Skynet.net
- 相关文档
- GitHub

Filecoin - ***Filecoin 是由星际文件系统背后的同一团队打造。 它是星际文件系统之上的一个激励层。***

- Filecoin.io
- 相关文档
- GitHub

Arweave - ***Arweave 是去中心化数据存储平台。***

- Arweave.org

- 相关文档
- Arweave

0chain - ***0Chain 是一个具有区块分片和 blobbers 的基于权益证明的去中心化存储平台***

- 0Chain.net
- 相关文档
- GitHub

Swarm - ***以太坊 web3 堆栈的分布式存储平台和内容分发服务。***

- EthSwarm.org
- 相关文档
- GitHub

OrbitDB - ***基于星际文件系统的去中心化点对点数据库。***

- OrbitDB.org
- 相关文档
- GitHub

Aleph.im - ***去中心化云项目（数据库、文件存储、计算和 DID）。 独特的链下和链上点对点技术融合。 星际文件系统以及多链兼容性。***

- Aleph.im
- 相关文档
- GitHub

Ceramic - ***用户控制的星际文件系统数据库存储，用于数据丰富和吸引人的应用程序***

- Ceramic.network
- 相关文档
- GitHub

Filebase - ***第一个兼容 S3 的对象存储平台，允许您跨多个去中心化存储网络存储数据，包括 IPFS、Sia、Skynet 和 Storj。***

- Filebase.com
- 相关文档
- GitHub

延伸阅读 {#further-reading}

- 什么是去中心化存储？ *- CoinMarketCAp*
- 五个关于去中心化存储的常见错误观念 *- Storj*

还有哪些社区资源对您有所帮助？ 请编辑本页面并添加！

相关主题 {#related-topics}

- 开发框架

---

title: 交易
description: 以太坊交易 – 工作原理、数据结构以及如何通过应用发送。
lang: zh

sidebar: true

交易是由帐户发出，带密码学签名的指令。 帐户将发起交易以更新以太坊网络的状态。 最简单的交易是将 ETH 从一个账户转到另一个帐户。

前置要求 {#prerequisites}

为了帮助您更好地理解这个页面，我们建议您先阅读账户和我们的以太坊简介。

什么是交易？ {#whats-a-transaction}

以太坊交易是指由外部持有账户发起的行动，换句话说，是指由人管理而不是智能合约管理的账户。 例如，如果 Bob 发送 Alice 1 ETH，则 Bob 的帐户必须减少 1 ETH，而 Alice 的账户必须增加 1 ETH。 此项操作发生在交易中，会变更状态。

显示交易导致状态更改的图表 *图表来自 Ethereum EVM 插图*

改变 EVM 状态的交易需要广播到整个网络。 任何节点都可以在 EVM 上广播交易请求；此后，矿工将执行交易并将由此产生的状态变化传播到网络的其他部分。

交易需要收费并且必须开采才能有效。 为了使这种概述更加简单，我们将其称为 Gas 费和挖矿。

所提交的交易包括下列信息：

- `recipient` – 接收地址（如果为一个外部持有的帐户，交易将传输值。 如果为合约帐户，交易将执行合约代码）
- `signature` – 发送者的标识符。 当通过发送者的私钥签名交易来确保发送者已授权此交易时，生成此签名。
- `value` – 从发件人向收件人转移 ETH 的金额 （以 WEI 为单位，ETH 的一种面值单位）
- `data` – 可包括任意数据的可选字段
- `gasLimit` – 交易可以消耗的 Gas 的最大数量。 Gas 单位代表了计算步骤
- `maxPriorityFeePerGas` - 作为矿工小费包含的最大 gas 数量
- `maxFeePerGas` - 愿意为交易支付的最大 gas 数量（包括 `baseFeePerGas` 和 `maxPriorityFeePerGas`）

Gas 是指矿工处理交易所需的算力。 用户必须为此计算支付费用。 `gasLimit` 和 `gasPrice` 决定支付给矿工的最高交易费用。 关于 Gas 的更多信息。

交易对象看起来像这样：

```
{
  from: "0xEA674fdDe714fd979de3EdF0F56AA9716B898ec8",
  to: "0xac03bb73b6a9e108530aff4df5077c2b3d481e5a",
  gasLimit: "21000",
  maxFeePerGas: "300"
  maxPriorityFeePerGas: "10"
  nonce: "0",
  value: "10000000000",
}
```

但交易对象需要使用发送者的私钥签名。 这证明交易只可能来自发送者，而不是欺诈。

Geth 这样的以太坊客户端将处理此签名过程。

示例 JSON-RPC 调用：

```
{
```

```
  "id": 2,
  "jsonrpc": "2.0",
  "method": "account_signTransaction",
  "params": [
    {
      "from": "0x1923f626bb8dc025849e00f99c25fe2b2f7fb0db",
      "gas": "0x55555",
      "maxFeePerGas": "0x1234",
      "maxPriorityFeePerGas": "0x1234",
      "input": "0xabcd",
      "nonce": "0x0",
      "to": "0x07a565b7ed7d7a678680a4c162885bedbb695fe0",
      "value": "0x1234"
    }
  ]
}
```

示例响应：

```
{
  "jsonrpc": "2.0",
  "id": 2,
  "result": {
    "raw":
"0xf88380018203339407a565b7ed7d7a678680a4c162885bedbb695fe080a44401a6e4000000000000000000
0000000000000000000000000000000000000000000001226a0223a7c9bcf5531c99be5ea7082183816eb20cfe
0bbc322e97cc5c7f71ab8b20ea02aadee6b34b45bb15bc42d9c09de4a6754e7000908da72d48cc77049714916
63",
    "tx": {
      "nonce": "0x0",
      "maxFeePerGas": "0x1234",
      "maxPriorityFeePerGas": "0x1234",
      "gas": "0x55555",
      "to": "0x07a565b7ed7d7a678680a4c162885bedbb695fe0",
      "value": "0x1234",
      "input": "0xabcd",
      "v": "0x26",
      "r": "0x223a7c9bcf5531c99be5ea7082183816eb20cfe0bbc322e97cc5c7f71ab8b20e",
      "s": "0x2aadee6b34b45bb15bc42d9c09de4a6754e7000908da72d48cc7704971491663",
      "hash": "0xeba2df809e7a612a0a0d444ccfa5c839624bdc00dd29e3340d46df3870f8a30e"
    }
  }
}
```

- `raw` 是已签名交易的 RLP（Recursive Length Prefix）编码形式。
- `tx` 是已签名交易的 JSON 形式。

如有签名哈希，可通过加密技术证明交易来自发送者并提交网络。

交易类型 {#types-of-transactions}

以太坊有几种不同类型的交易：

- 常规交易：从一个钱包到另一个钱包的交易。
- 合约部署交易：没有“to”地址的交易，数据字段用于合约代码。

关于 Gas {#on-gas}

如上所述，执行交易需要花费 Gas。 简单的转让交易需要 21000 个 Gas。

假设 Bob 要向 Alice 发送 1 ETH，需支付 190 gwei 的 `baseFeePergas` 和 10 gwei 的 `maxPriorityFeePerGas`，那么 Bob 需要支付以下费用：

```
(190 + 10) * 21000 = 4,200,000 gwei
--or--
0.0042 ETH
```

Bob 的账户将会减少 **1.0042 ETH**

Alice 的账户将会增加 **+1.0 ETH**

扣掉 **0.00399 ETH** 的基本费用

矿工保留小费 **+0.000210 ETH**

任何智能合约交互也需要 Gas。

未使用 Gas 退款情况图 *图表来自 Ethereum EVM 插图*

任何未用于交易的 gas 都退还至用户账户。

交易生命周期 {#transaction-lifecycle}

一旦提交交易，就会发生以下情况：

1. 一旦您发送交易，加密法生成交易哈希：`0x97d99bc7729211111a21b12c933c949d4f31684f1d6954ff477d0477538ff017`
2. 然后将该交易转播到网络，并且与大量其他交易一起包含在一个集合中。
3. 矿工必须选择您的交易并将它包含在一个区块中，以便验证交易并认为它“成功”。
    - 如果网络繁忙，矿工无法跟上，您可能会在这个阶段等候。
4. 您的交易将收到"确认"。 确认的数量是自包含您交易的区块以来创建的区块数。 这个数字越大，交易被网络处理和承认的确定性就越强。
    - 最近的区块可能会被重组，给人留下交易失败的印象；但交易可能仍然有效，但包含在另一个区块中。
    - 重组的概率随着其后每一次挖掘的区块而降低，即确认次数越多，交易就越不可改变。

直观演示 {#a-visual-demo}

观看 Austin 引导您了解交易、Gas 和挖矿。

Typed Transaction Envelope 交易 {#typed-transaction-envelope}

以太坊最初有一个交易格式。 每笔交易都包含一个随机数、gas 价格、gas 限额、发送地址、价值、数据、v、r 和 s。 这些字段采用 RLP 编码，如下所示：

`RLP([nonce, gasPrice, gasLimit, to, value, data, v, r, s])`

以太坊经过演变，现已支持多种类型的交易，能够在不影响传统交易格式的情况下实现接入列表和 EIP-1559 等新功能。

EIP-2718：Typed Transaction Envelope 定义了交易类型，是未来交易类型的“信封”。

EIP-2718 是用于类型化交易的新通用信封。 在新的标准中，交易被理解为：

`TransactionType || TransactionPayload`

字段定义为：

- `TransactionType` - 一个在 0 到 0x7f 之间的数字，总共为 128 种可能的交易类型。
- `TransactionPayload` - 由交易类型定义的任意字节数组。

延伸阅读 {#further-reading}

- EIP-2718：Typed Transaction Envelope

还有哪些社区资源对您有所帮助？ 请编辑本页面并添加！

相关主题 {#related-topics}

- 帐户
- 以太坊虚拟机 (EVM)
- Gas
- 挖矿

---

title: Web2 与 Web3 的对比
description:
lang: zh

sidebar: true

Web2 指的是我们如今众所周知的互联网版本。 一个在互联网公司掌控下提供服务来交换个人数据的互联网。 就以太坊而言，Web3 指的是在区块链上运行的去中心化应用。 任何用户都可以参与以太坊上的这些应用，个人数据却不会被出卖。

Web3 的优势 {#web3-benefits}

许多 Web3 开发者由于以太坊的内在去中心化而选择构建去中心化应用程序：

- 网络上的任何人都有使用服务的权限 — — 换言之，不需要许可。
- 没有人可以屏蔽您或拒绝您访问该服务。
- 付款是通过本地令牌即 ether (ETH) 构建的。
- 以太坊是图灵完备的，这意味着你可以有很多程序部署在上面。

具体比较 {#practical-comparisons}

Web2	Web3
Twitter 可以检查任何帐户或推文	Web3 推文不可审查，因为控制已经被分散了
支付服务可能决定不允许对某些类型的工作进行支付	Web3 付款应用程序不需要个人数据，也不能阻止付款
零工经济应用的服务器可能会停用并影响劳动者收入	Web3 服务器不会下线——他们使用以太坊，一个由数以千计的计算机组成的去中心化网络作为他们的后端

这并不意味着所有服务都需要变成 dapp。 这些例子说明了 Web2 和 Web3 服务之间的主要差异。

Web3 局限性 {#web3-limitations}

Web3 现在有一些局限：

- 可扩展性——交易在 web3 上进度较慢，因为它们是去中心化的。 更改状态像付款一样，需要由矿工处理并在整个网络中传播。
- UX — 与 web3 应用程序互动可能需要额外的步骤、软件和学习。 这可能是应用的一个障碍。
- 可访问性 - 大多数用户因为缺少现代化的网页浏览器而无法访问 web3。
- 成本 — 因为太过昂贵，大多数成功的去中心化应用程序仅将其代码的很小一部分放入区块链。

中心化与去中心化 {#centralization-vs-decentralization}

在下表中，我们粗略列举了中心化和去中心化数字网络的优缺点。

中心化系统	去中心化系统
网络半径短（所有参与者都与中心化组织连接）； 信息传递快，因为是由一个拥有大量计算资源的中心化组织处理的。	网络上最远的参与者可能彼此相距甚远。 网络中一方的信息广播可能需要很长时间才能传达给另一方。
性能通常较高（吞吐量较高、总计算资源消耗较少）以及构建较为容易。	性能通常较低（吞吐量较低，总计算资源消耗较高），构建起来更为复杂。
在出现相互冲突的数据时，解决办法明确且简单：最终信任的数据来源是中心化组织。	解决争端需要一项协议（通常是复杂的协议）。 如果其他参与者对参与者打算同步的数据状态提出相互冲突的主张。
单点故障：恶意行为者可能能够以中心化组织为目标来破坏网络。	无单点故障：即使有很大比例的参与者受到攻击或下线，网络仍然可以运作。
参与者之间的协调要容易得多，由中心化组织负责。 中心化组织可以迫使网络参与者接受升级、更新协议等，阻力较小。	协调往往很困难，因为没有任何单一角色在网络级的决策、协议升级等方面拥有最终发言权。 在最糟糕的情况下，当对协议更改有分歧时，网络容易破裂。
中心化组织可以审查数据，可能会切断网络的某些部分与网络其他部分的交互。	审查制度要困难得多，因为信息有多种途径在网络上传播。
网络准入由中心化组织控制。	任何人都可以加入网络；没有“守门人”。 理想的情况是，参与费用非常低。

请注意，这些是一般模式，可能不适用于每一个网络。 此外，在现实情况下，网络集中/分散的程度取决于一系列因素；没有一个网络完全中心化或完全去中心化。

延伸阅读 {#further-reading}

- The Architecture of a Web 3.0 application - *Preethi Kasireddy*
- The Meaning of Decentralization *Feb 6, 2017 - Vitalik Buterin*
- Why Decentralization Matters *Feb 18, 2018 - Chris Dixon*
- What Is Web 3.0 & Why It Matters *Dec 31, 2019 - Max Mersch and Richard Muirhead*

---

title: 面向 Python 开发者的以太坊介绍，第一部分
description: 这是一篇介绍以太坊开发的文章，对那些熟悉 Python 编程语言的人来说尤其有用。
author: Marc Garreau
lang: zh

sidebar: true
tags:

- "入门指南"
- "python"
- "区块链"
- "web3.py"
  skill: 初学者
  published: 2020-09-08
  source: Snake charmers
  sourceUrl: https://snakecharmers.ethereum.org/a-developers-guide-to-ethereum-pt-1/

想必您已经听说过以太坊，那么，您准备好投身于这个领域了吗？ 本篇文章将快速介绍一些区块链基础知识，然后让您与模拟的以太坊节点进行互动， 比如读取区块数据、检查账户余额和发送交易。 在这个过程中，我们会着重强调用传统方式构建应用与这种新的去中心化范式之间的差异。

（软）前提条件 {#soft-prerequisites}

本文希望面向所有开发者。 在文章里会涉及 Python 工具，不过它们只是思想的载体，如果您不是 Python 开发者也没有问题。 不过，我将对您已经了解的知识作一些假设，以便我们能够迅速地进入以太坊部分。

本文假定：

- 您熟悉终端操作，
- 您写过一些 Python 代码，
- 您的机器上有安装 Python 3.6 或更高版本 (强烈推荐使用 虚拟环境)，并且
- 您使用过 `pip`，Python 的软件包安装程序。 再次强调，如果您不符合其中任何一条，或者您不打算敲本文中的代码，您照着学仍然可以学得很好。

区块链简述 {#blockchains-briefly}

描述以太坊有很多方法，但其核心还是区块链。 区块链由一系列区块组成，所以让我们从区块链开始。 用最简单的话来说，以太坊区块链上的每个区块只是一些元数据和一个交易的列表。 在 JSON 格式中，它看起来像这样：

```
{
   "number": 1234567,
   "hash": "0xabc123...",
   "parentHash": "0xdef456...",
   "miner": "0xa1b2c3...",
   ...,
   "transactions": [...]
}
```

每个 块 会引用它前面的区块； `parentHash` 是前一个区块的哈希值。

注： 以太坊广泛使用 哈希函数 来生成固定大小的值（“哈希值”）。 哈希值在以太坊中发挥着重要作用，但您现在可以放心地将其视为是唯一的 ID 值。

描述区块链的示意图，其中包括每个区块内的数据

区块链本质上是一个链表；每个区块都有一个对前一个区块的引用。

这种数据结构并不新颖，但治理网络的规则（即点对点协议）却很新颖。 区块链没有中央机构；网络中的对等节点必需协作以维持网络，并且通过竞争决定将哪些交易纳入下一个区块。 因此，当您想给朋友转账时，您需要将这笔交易广播到网络上， 然后等待它被纳入即将产生的区块。

区块链验证资金确实从一个用户发送给另一个用户的唯一方法是使用该区块链原生货币（即，由该区块链创建和管理的货币）。 在以太坊，这种货币被称为 ETH，以太坊区块链是账户余额的唯一正式记录。

一种新范式 {#a-new-paradigm}

这种新的去中心化技术栈催生了新的开发者工具。 许多编程语言都有这样的工具，但我们将通过 Python 的视角来观察。 重申一下：即使 Python 不是您的首选语言，跟上文章也不会有什么太大的问题。

想要与以太坊进行互动的 Python 开发人员可能会接触到 Web3.py。 Web3.py 是一个库，可以帮助我们简化连接以太坊节点，以及发送和接收数据。

注："以太坊节点"和"以太坊客户端"可互换使用。 这两种说法都是指以太坊网络中参与者所运行的软件。 该软件可以读取区块数据，在新区块被添加到链上（"挖矿"）时接收更新、广播新交易等。

以太坊客户端 可以配置为通过进程间通信、超文本传输协议或 Websockets 进行访问，因此 Web3.py 也需要完成这个配置。 Web3.py 将这些连接选项称为**提供者**。 您需要从三个提供者中选择一个来连接 Web3.py 实例和您的节点。

描述 web3.py 如何使用 IPC 将应用程序连接到以太坊节点的示意图

将以太坊节点和 Web3.py 配置为通过相同通信的协议（例如，本图中的 IPC）进行通信。

正确配置了 Web3.py 之后，您就可以开始与区块链交互了。 下面是一些 Web3.py 使用示例，算是抛砖引玉：

```
# read block data:
w3.eth.get_block('latest')

# send a transaction:
w3.eth.send_transaction({'from': ..., 'to': ..., 'value': ...})
```

安装 {#installation}

在这个演练中，我们仅在 Python 解释器中进行。 我们不会创建任何目录、文件、类或函数。

注：在下面的例子中，以"$"开头的命令是要在终端中运行的。 （不要输入 `$`，它只是表示行的开始。）

首先，安装 IPython，以方便用户在其中进行探索。 IPython 提供了 tab 补全等功能，使得我们更容易看到 Web3.py 中有哪些可用方法。

```
$ pip install ipython
```

Web3.py 以 `web3` 的名称发布。 安装方式如下：

```
$ pip install web3
```

另外， 我们后面要模拟一个区块链，这就需要更多依赖项。 可以通过下面的命令安装这些依赖项：

```
$ pip install 'web3[tester]'
```

您已经设置完毕！

开启沙盒环境 {#spin-up-a-sandbox}

在终端中运行 `ipython`，打开一个新的 Python 环境。 这与运行 `python` 类似，但更友好。

```
$ ipython
```

这将打印出一些关于您正在运行的 Python 和 IPython 版本的信息，然后您应该会看到一个等待输入的提示：

```
In [1]:
```

您现在看到的是一个交互式的 Python shell， 实际上，它是一个沙盒。 如果您已经做到了这一点，现在可以导入 Web3.py 了：

```
In [1]: from web3 import web3
```

Web3 模块介绍 {#introducing-the-web3-module}

除了作为以太坊的网关， Web3 模块还提供了一些便利的功能。 让我们来探究探究。

在以太坊应用中，您通常需要转换货币面额。 Web3 模块为此提供几个辅助方法： fromWei 和 toWei。

注：计算机不擅长处理十进制数学。 为了规避这个问题，开发者通常会以美分存储美元。 例如，价格为 5.99 美元的物品在数据库中存储为 599。

在以 **ETH** 处理交易时，也使用了类似的模式。 但是，ETH 不是只有两个小数位，而是有 18 位。 ether 的最小单位是 **wei**，所以发送交易时指定的就是这个值。

1 ETH = 1000000000000000 wei

1 wei = 0.0000000000000001 ETH

试一下将一些数值转换为 wei 或反向转换。 请注意， ETH 和 wei 之间还有其他面额名称。 其中比较有名的是 **gwei**，因为它通常用于表示交易费用。

```
In [2]: web3.toWei(1, 'ether')
Out[2]: 1000000000000000000

In [3]: web3.fromWei(500000000, 'gwei')
Out[3]: Decimal('0.5')
```

Web3 模块上的其他实用方法包括数据格式转换器（例如 `toHex`）， 地址助手， （例如 `is_address`），以及哈希函数（例如 `keccak`）。 其中许多内容将在后面的系列文章中介绍。 要查看所有可用的方法和属性，可以利用 IPython 的自动补全功能，输入 `web3.`， 然后在点号后面按两次 tab 键。

与链交互 {#talk-to-the-chain}

方便的方法很受欢迎，但让我们继续来说区块链。 接下来配置 Web3.py 与以太坊节点通信。 在这里，我们可以选择使用 IPC、HTTP 或 Websocket 提供者。

我们不会完整地进行这个步骤，但一个使用 HTTP 提供者的完整工作流程的例子可能如下所示：

- 下载一个以太坊节点，例如 Geth。
- 在一个终端窗口启动 Geth，等待它同步网络。 默认的 HTTP 端口是 `8545`，但可以配置成其它端口。

- 告诉 Web3.py 通过 HTTP 连接到节点，使用 `localhost:8545`。 `w3 = Web3(Web3.HTTPProvider('http://127.0.0.1:8545'))`
- 使用 `w3` 实例与节点交互。

虽然这是一种“正式”的方式，但同步过程需要几个小时，如果您只是想要一个开发环境，则没有必要同步过程。 Web3.py 为此公开了第四个提供者，即 **EthereumTesterProvider**。 这个测试器提供者连接到一个模拟的以太坊节点，它有更宽松的权限，还有虚拟以太币可供操作。

描述将 web3.py 应用程序连接到模拟以太坊节点的 EthereumTesterProvider 的示意图

EthereumTesterProvider 连接到一个模拟节点，对于快速开发环境来说非常方便。

这个模拟节点叫做 eth-tester，我们把它作为 `pip install web3[tester]` 命令的一部分进行安装。 配置 Web3.py 来使用这个测试器提供者很简单：

```
In [4]: w3 = Web3(Web3.EthereumTesterProvider())
```

现在，您已经准备好在链上冲浪了！ 这不是人们常说的以太链。 我只是虚构了它。 我们来快速了解一下。

快速了解 {#the-quick-tour}

第一件事，先进行连接检查。

```
In [5]: w3.isConnected()
Out[5]: True
```

由于我们使用的是测试器提供者，这不是一个非常有价值的测试，但如果它确实失败了，很可能是在实例化 `w3` 变量时发生了输入错误。 仔细检查您是否包含了内括号，即 `Web3.EthereumTesterProvider()`。

第一站：帐户 {#tour-stop-1-accounts}

为了方便起见，测试器提供者创建了一些帐户，并预先分配了测试以太币。

首先，我们来看看这些帐户的列表：

```
In [6]: w3.eth.accounts
Out[6]: ['0x7E5F4552091A69125d5DfCb7b8C2659029395Bdf',
 '0x2B5AD5c4795c026514f8317c7a215E218DcCD6cF',
 '0x6813Eb9362372EEF6200f3b1dbC3f819671cBA69', ...]
```

如果运行这个命令，您应该会看到一个以 `0x` 开头的十个字符串的列表。 每一个字符串都是一个的**公共地址**，在某些方面，类似于支票帐户上的帐号。 如果有人要给您转 ETH，您可以把这个地址给他。

如前所述，测试提供者已经为这些账户中的每一个账户预分配了一些测试以太币。 我们来看看第一个帐户上有多少 ETH。

```
In [7]: w3.eth.get_balance(w3.eth.accounts[0])
Out[7]: 100000000000000000000000
```

好多零啊！ 在你一路笑醒之前，先回忆一下之前关于货币面额的介绍。 ETH 币值用最小的面额 wei 来表示。 将其转换为 ETH：

```
In [8]: w3.fromWei(1000000000000000000000000, 'ether')
Out[8]: Decimal('1000000')
```

100 万测试 ETH，也不算太寒酸。

第二站：区块数据 {#tour-stop-2-block-data}

我们来看看这个模拟区块链的状态。

```
In [9]: w3.eth.get_block('latest')
Out[9]: AttributeDict({
   'number': 0,
   'hash': HexBytes('0x9469878...'),
   'parentHash': HexBytes('0x0000000...'),
   ...
   'transactions': []
})
```

这里返回了大量关于区块的信息，但这里只介绍以下几点：

- 该区块编号是零 - 无论您在多久以前配置了测试器提供者 。 与真实的以太坊网络不同，真实的以太坊网络大概每隔 15 秒就会挖掘一个新的区块，而模拟链则则需要您给它一些工作才增加区块。
- `transactions` 是一个空列表，原因相同：我们还没有做任何事情。 第一个区块是一个**空区块**，只是为了开个头。
- 注意，`parentHash` 只是一堆空的字节。 这标志着它是链条上的第一个区块，也就是所谓的**创世区块**

第三站：交易 {#tour-stop-3-transactions}

在没有交易之前，我们停留在零区块，所以我们给它一个交易。 从一个账户向另一个账户发送一些测试 ETH

```
In [10]: tx_hash = w3.eth.send_transaction({
   'from': w3.eth.accounts[0],
   'to': w3.eth.accounts[1],
   'value': w3.toWei(3, 'ether'),
   'gas': 21000
})
```

这时通常会等上几秒钟，等待交易挖到一个新的区块。 完整的流程是这样的：

1. 提交交易并持有交易哈希。 在没有被挖出来之前，交易是“等待中”的。 `tx_hash = w3.eth.send_transaction({ … })`
2. 等待交易被打包： `w3.eth.wait_for_transaction_receipt(tx_hash)`
3. 继续应用逻辑。 查看成功的交易： `w3.eth.get_transaction(tx_hash)`

我们的模拟环境会在一个新的区块中即时添加交易，所以我们可以立即查看交易：

```
In [11]: w3.eth.get_transaction(tx_hash)
Out[11]: AttributeDict({
   'hash': HexBytes('0x15e9fb95dc39...'),
   'blockNumber': 1,
   'transactionIndex': 0,
   'from': '0x7E5F4552091A69125d5DfCb7b8C2659029395Bdf',
   'to': '0x2B5AD5c4795c026514f8317c7a215E218DcCD6cF',
   'value': 3000000000000000000,
   ...
})
```

您将在这里看到一些熟悉的细节：`from`、`to`和`value`字段应该与我们的`send_transaction`调用的输入相匹配。 另一个令人欣慰的是，这项交易被列为 1 号区块内的第一笔交易（`'transactionIndex': 0`）。

我们也可以通过检查两个相关帐户的余额，轻松验证此次交易是否成功。 三个 ETH 应从一个帐户转移到另一个帐户。

```
In [12]: w3.eth.get_balance(w3.eth.accounts[0])
Out[12]: 999996999999999999969000

In [13]: w3.eth.get_balance(w3.eth.accounts[1])
Out[13]: 100000300000000000000000
```

后者看起来不错！ 余额从 1000000 增加到 100003 个 ETH。 但第一个账户发生了什么情况？ 它减少的数量看起来略大于三个 ETH？ 是的，没有免费的午餐，使用以太坊公网需要支付矿工手续费， 它从进行交易的账户中扣除了一笔小额的交易费，金额为 31000wei。

注：在公共网络上，交易费用根据网络需求和您希望交易处理的速度而变化。 如果您对费用的计算方式感兴趣，请查看我之前关于[如何将交易包含在一个区块中](#)的文章。

结尾 {#and-breathe}

我们已经学习一段时间，现在可以休息一下。 要学习的内容还有很多，我们将在本系列的第二部分继续进行探索。 探索这些概念：连接真实节点、智能合约和代币， 仍有后续问题？ 请告诉我！ 您的反馈对我们今后的学习至关重要！ 欢迎通过 Twitter 与我联系。

title: 通过JavaScript调用智能合约
description: 以DAI通证为例展示如何通过JavaScript调用智能合约函数
author: jdourlens
tags:

- "交易"
- "前端"
- "JavaScript"
- "web3.js"
  skill: 初学者
  lang: zh
  sidebar: true
  published: 2020-04-19

source: EthereumDev
sourceUrl: https://ethereumdev.io/calling-a-smart-contract-from-javascript/
address: "0x19dE91Af973F404EDF5B4c093983a7c6E3EC8ccE"

在本教程中，我们将会看到如何通过 JavaScript 调用智能合约。 首先读取智能合约的状态（例如：ERC20 持有者的余额），然后通过通证转账修改区块链的状态。 您首先需要了解设置 JS 环境与区块链交互。

在本例中，我们将使用 DAI 通证，基于测试目的，我们将使用 ganache-cli 分叉区块链并解锁一个已经拥有很多 DAI 的地址。

```
ganache-cli -f https://mainnet.infura.io/v3/[YOUR INFURA KEY] -d -i 66 1 --unlock
0x4d10ae710Bd8D1C31bd7465c8CBC3add6F279E81
```

要与智能合约交互，我们需要它的地址和 ABI：

```
const ERC20TransferABI = [
  {
    constant: false,
    inputs: [
      {
        name: "_to",
        type: "address",
      },
      {
        name: "_value",
        type: "uint256",
      },
    ],
    name: "transfer",
    outputs: [
      {
        name: "",
        type: "bool",
      },
    ],
    payable: false,
    stateMutability: "nonpayable",
    type: "function",
  },
  {
    constant: true,
    inputs: [
      {
        name: "_owner",
        type: "address",
      },
    ],
    name: "balanceOf",
    outputs: [
      {
        name: "balance",
        type: "uint256",
      },
```

```
    ],
    payable: false,
    stateMutability: "view",
    type: "function",
  },
]

const DAI_ADDRESS = "0x6b175474e89094c44da98b954eedeac495271d0f"
```

对于此项目，我们剥离完整的 ERC20 ABI ，仅保留 `balanceOf` 和 `transfer` 函数，不过您可以在这里获取完整的 ERC20 ABI。

然后我们需要实例化我们的智能合约：

```
const web3 = new Web3("http://localhost:8545")

const daiToken = new web3.eth.Contract(ERC20TransferABI, DAI_ADDRESS)
```

我们还会设置两个地址：

- 一个接受转账
- 一个已经解锁并将发送

```
const senderAddress = "0x4d10ae710Bd8D1C31bd7465c8CBC3add6F279E81"
const receiverAddress = "0x19dE91Af973F404EDF5B4c093983a7c6E3EC8ccE"
```

在下一部分中，我们会调用 `balance Of` 函数来检索当前的代币数量，此时这两个地址的代币数量都被冻结。

调用：从智能合约读取值 {#call-reading-value-from-a-smart-contract}

第一个例子，将调用“常量（constant）”方法并且在 EVM 中执行这个智能合约方法，并不发送任何交易。 为此我们将读取一个地址的 ECR20 余额。 阅读关于 ECR20 代币的文章.

您可以访问为其提供 ABI 的实例化智能合约方法，如下所示： `yourContract.methods.methodname`。 通过使用 `call` 函数，您可以接收执行函数的结果。

```
daiToken.methods.balanceOf(senderAddress).call(function (err, res) {
  if (err) {
    console.log("An error occured", err)
    return
  }
  console.log("The balance is: ", res)
})
```

请记住，DAI ERC20 有 18 位小数，这意味着您需要移除 18 个零才能获得正确的数额。 uint256 作为字符串返回，因为 JavaScript 不处理大数值。 如果不确定，请了解我们关于 bignumber.js 的教程如何在 JS 中处理大数。

发送：将交易发送给智能合约函数 {#send-sending-a-transaction-to-a-smart-contract-function}

对于第二个示例，我们将调用 DAI 智能合约的 transfer 函数，发送 10 个 DAI 到第二个地址。 transfer 函数接受两个参数：收件人地址和转账代币的数量：

```
daiToken.methods
  .transfer(receiverAddress, "10000000000000000000")
  .send({ from: senderAddress }, function (err, res) {
    if (err) {
      console.log("An error occured", err)
      return
    }
    console.log("Hash of the transaction: " + res)
  })
```

调用函数返回将被在区块链中挖矿(mine) 的交易的哈希。 在以太坊中，交易哈希是可以预测的 - 这里是我们在执行前获得交易哈希值的方法(了解如何计算哈希)。

因为该函数只向区块链提交交易，除非通过挖矿并包含在区块链中，否则我们无法看到结果。 在下一个教程中，我们将学习如何通过了解其哈希值来等待一个交易在区块链上的执行。

title: 创建并部署一个去中心化金融应用程序
description: 将 ERC20 代币存入智能合约，铸造流动性矿池代币
author: "strykerin"
tags:

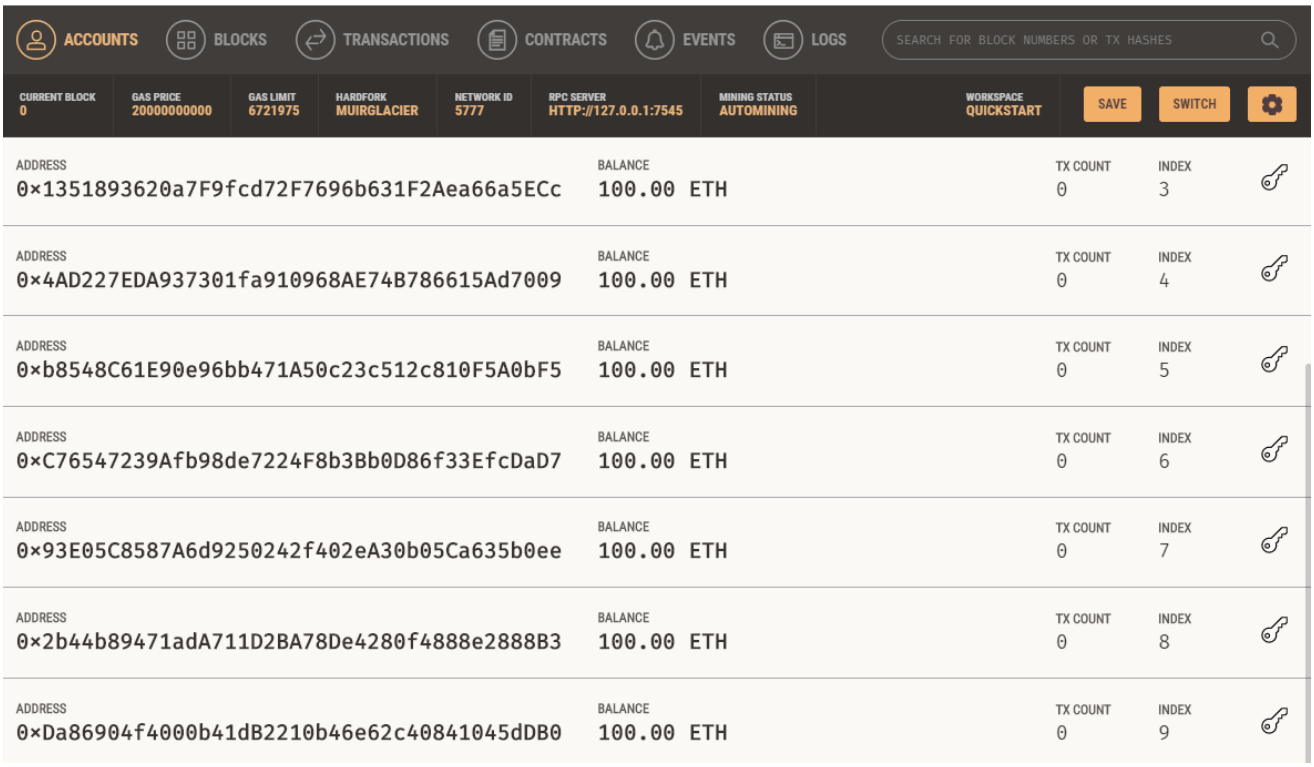
- "solidity"
- "deFi"
- "web3.js"
- "truffle"
- "ganache"
- "智能合约"
  skill: 中级
  lang: zh
  sidebar: true
  published: 2020-12-31
  source: www.github.com
  sourceUrl: https://github.com/strykerin/DeFi-Token-Farm

在本教程中，我们将使用 Solidity 构建一个去中心化金融应用程序。用户可以将 ERC20 代币存入智能合约，然后铸造流动性矿池代币并将这些代币转给他们。 用户之后可以通过消耗智能合约上的流动性矿池代币来赎回他们的 ERC20 代币，然后 ERC20 代币将会转回给他们。

安装 Truffle 和 Ganache {#install-truffle-and-ganache}

如果这是您第一次编写智能合约，您需要搭建您的环境。 我们将使用两个工具：Truffle 和 Ganache。

Truffle 是用于开发以太坊智能合约的开发环境和测试框架。 使用 Truffle 可以很容易地在区块链中创建和部署智能合约。 Ganach 可以帮助我们创建一个本地以太坊区块链，用以测试智能合约。 它模拟真实的网络功能，前 10 个帐户存入了 100 个测试以太币，如此可以随意部署和测试智能合约。 Ganache 可提供桌面应用程序和命令行工具。 在本文中，我们将使用有图形界面的桌面应用程序。



Ganache UI desktop application

要创建项目，请运行以下命令：

```
mkdir your-project-name
cd your-project-name
truffle init
```

这将为智能合约的开发和部署创建一个空白项目， 创建的项目结构如下：

- `contracts`： 存放 solidity 智能合约的文件夹
- `migrations`： 存放部署脚本的文件夹
- `test`： 存放智能合约测试文件的文件夹
- `truffle-config.js`： Truffle 配置文件

创建 ERC20 代币 {#create-the-erc20-token}

首先，我们需要创建 ERC20 代币，用于在智能合约上质押。 要创建同质化代币，我们首先需要安装 OpenZeppelin 库。 该库包含 ERC20 和 ERC721 等标准的实现。 如需安装，请运行以下命令：

```
npm install @openzeppelin/contracts
```

使用 OpenZepelin 库，我们可以通过写入 `contracts/MyToken.sol` 创建我们的 ERC20 代币，需用到以下 solidity 代码：

```
pragma solidity ^0.8.0;

import "@openzeppelin/contracts/token/ERC20/ERC20.sol";

contract MyToken is ERC20 {
    constructor() public ERC20("MyToken", "MTKN"){
        _mint(msg.sender, 100000000000000000000000);
    }
}
```

在上述代码中：

- 第 3 行：我们从包含此代币标准实现的 openzepelin 导入 ERC20.sol 合约。
- 第 5 行：我们继承了 ERC20.sol 合约。
- 第 6 行：我们要调用 ERC20.sol 构造函数，并将名称和符号参数分别设为 `"MyToken"` 和 `"MTKN"`。
- 第 7 行：我们为正在部署智能合约的账户铸造和转移 100 万代币（我们默认为 ERC20 代币使用 18 位小数）。这意味着，如果我们想要铸造 1 个代币，则需将其表示为 100000000000000，即 1 和 18 个零）。

我们可以看到，ERC20.sol 构造函数实现下方，`_decimals` 字段设置为 18：

```
string private _name;
string private _symbol;
uint8 private _decimals;

constructor (string memory name_, string memory symbol_) public {
    _name = name_;
    _symbol = symbol_;
    _decimals = 18;
}
```

编译 ERC20 代币 {#compile-the-erc20-token}

要编译我们的智能合约，我们必须首先检查我们的 solidity 编译器版本。 可以通过运行以下命令来检查：

```
truffle version
```

默认版本是 `Solidity v0.5.16`。 由于我们的代币基于 solidity 版本 `0.6.2` 编写，如果我们运行命令来编译合约，我们会遇到编译器错误。 要使用指定的 solidity 编译器版本，请转到文件 `truffle-config.js`，设为所需的编译器版本，如下所示：

```
// Configure your compilers
compilers: {
  solc: {
    version: "^0.8.0",    // Fetch exact version from solc-bin (default: truffle's
version)
    // docker: true,        // Use "0.5.1" you've installed locally with docker (default:
false)
    // settings: {          // See the solidity docs for advice about optimization and
evmVersion
    //  optimizer: {
    //    enabled: false,
    //    runs: 200
    //  },
```

```
    //  evmVersion: "byzantium"
    // }
  }
}
```

现在，我们可以通过运行以下命令来编译智能合约：

```
truffle compile
```

部署 ERC20 代币 {#deploy-erc20-token}

编译后，我们现在便可以部署代币了。

在 `migrations` 文件夹中，创建一个名为 `2_depu_Tokens.js` 的文件。 我们将在该文件中部署 ERC20 代币和 FarmToken 智能合约。 以下代码用于部署我们的 MyToken.sol 合约：

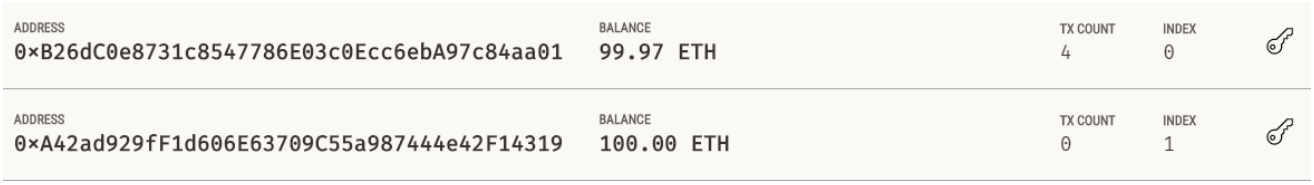
```
const MyToken = artifacts.require("MyToken")

module.exports = async function (deployer, network, accounts) {
  // Deploy MyToken
  await deployer.deploy(MyToken)
  const myToken = await MyToken.deployed()
}
```

打开 Ganache 并选择“快速启动”选项，启动本地以太坊区块链。 要部署合约，请运行：

```
truffle migrate
```

用于部署合同的地址是 Ganache 显示的地址列表中的第一个地址。 为核实这一点，我们可以打开 Ganache 桌面应用程序，然后可以核实第一个帐户的以太币余额已经减少，这是我们部署智能合约的以太币成本：



Ganache desktop application

为了验证 100 万 MyToken 代币是否已发送到部署者地址，我们可以使用 Truffle 控制台与我们部署的智能合约进行交互。

> Truffle 控制台是一个基础型交互式控制台，可连接到任何以太坊客户端。

要与智能合约交互，请运行以下命令：

```
truffle console
```

现在，我们可以在终端写入以下命令：

- 获取智能合约：`myToken = await MyToken.depolyed()`
- 从 Ganache 获取账户组：`accounts = await web3.eth.getAccounts()`
- 获取第一个帐户的余额：`balance = await myToken.balance Of(accounts[0])`
- 格式化具有 18 位小数的余额：`web3.utils.fromWei(Balance.toString())`

通过运行上述命令，我们会看到第一个地址实际上有 100 万 MyToken：

```
truffle(ganache)> web3.utils.fromWei(balance.toString())
'1000000'
```

第一个地址有 1000000 MyToken

创建 FarmToken 智能合约 {#create-farmtoken-smart-contract}

FarmToken 智能合约将包含 3 个函数：

- `balance()`：获取 FarmToken 智能合约上的 MyToken 余额。
- `deposit(uint256 _amount)`：代表用户将 MyToken 转移到 FarmToken 智能合约，然后铸造 FarmToken 并转账给用户。
- `withdraw(uint256 _amount)`：消耗用户的 FarmToken 并将 MyToken 转到用户的地址。

我们来看看 FarmToken 构造函数：

```
pragma solidity ^0.6.2;

import "@openzeppelin/contracts/token/ERC20/IERC20.sol";
import "@openzeppelin/contracts/utils/Address.sol";
import "@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol";
import "@openzeppelin/contracts/token/ERC20/ERC20.sol";

contract FarmToken is ERC20 {
    using Address for address;
    using SafeMath for uint256; // As of Solidity v0.8.0, mathematical operations can be
done safely without the need for SafeMath
    using SafeERC20 for IERC20;

    IERC20 public token;

    constructor(address _token)
        public
        ERC20("FarmToken", "FRM")
    {
        token = IERC20(_token);
    }
```

- 第 3-6 行：我们要从 openzepelin 导入下列合约：IERC20.sol、Address.sol、SafeERC20.sol 和 ERC20.sol。
- 第 8 行：FarmToken 将继承 ERC20 合约。
- 14-19 行：FarmToken 构造函数将接收 MyToken 合约地址作为参数，我们将把此地址分配给我们名为 `token` 的公共变量。

我们来实现 `balance()` 函数。 它将不会收到任何参数，将返回此智能合约上的 MyToken 余额。 可按如下所示实现：

```
function balance() public view returns (uint256) {
    return token.balanceOf(address(this));
}
```

对于 `deposit(uint256 _amount)` 函数，将接收用户想要存入的金额作为参数，铸造 FarmTokens 并转给用户：

```
function deposit(uint256 _amount) public {
    // Amount must be greater than zero
    require(_amount > 0, "amount cannot be 0");

    // Transfer MyToken to smart contract
    token.safeTransferFrom(msg.sender, address(this), _amount);

    // Mint FarmToken to msg sender
    _mint(msg.sender, _amount);
}
```

对于 `withdraw(uint256 _amount)` 函数，将接收用户想要消耗的 FarmTokens 数额作为参数，然后将相同数额的 MyToken 返还给用户：

```
function withdraw(uint256 _amount) public {
    // Burn FarmTokens from msg sender
    _burn(msg.sender, _amount);

    // Transfer MyTokens from this smart contract to msg sender
    token.safeTransfer(msg.sender, _amount);
}
```

现在我们将部署我们的智能合约。 为此，我们将返回文件 `2_depu_Tokens.js` 并添加要部署的新合约：

```
const MyToken = artifacts.require("MyToken")
const FarmToken = artifacts.require("FarmToken")

module.exports = async function (deployer, network, accounts) {
  // Deploy MyToken
  await deployer.deploy(MyToken)
  const myToken = await MyToken.deployed()

  // Deploy Farm Token
  await deployer.deploy(FarmToken, myToken.address)
  const farmToken = await FarmToken.deployed()
}
```

请注意，在部署 FarmToken 时，我们将已部署的 MyToken 合约的地址作为参数传递。

现在，运行 `truffle compile` 和 `truffle migrate` 来部署我们的合约。

我们来测试一下我们的智能合约。 不要使用 `truffle console` 与我们的智能合约交互，我们将创建一个脚本来自动执行此过程。 创建一个名为 `scripts` 的文件夹，并添加以下文件 `getMyTokenBalance.js`。 该文件将核查 FarmToken 智能合约中的 MyToken 余额：

```
const MyToken = artifacts.require("MyToken")
const FarmToken = artifacts.require("FarmToken")

module.exports = async function (callback) {
  myToken = await MyToken.deployed()
  farmToken = await FarmToken.deployed()
  balance = await myToken.balanceOf(farmToken.address)
  console.log(web3.utils.fromWei(balance.toString()))
  callback()
}
```

要执行此脚本，请运行以下 cli 命令：

```
truffle exec .\scripts\getMyTokenBalance.js
```

我们将得到预期结果 0。 如果您收到尚未部署 FarmToken 的错误，则表明 truffle 网络尚未收到最新版本的合约代码。 只需关闭 ganache，然后再快速启动，并确保运行 `truffle migrate`。

现在，让我们把 MyToken 押在智能合约上。 由于 `deposit(uint256 _amount)` 函数调用了来自 ERC20 的 `safeTransferFrom` 函数，用户必须首先批准智能合约，才能以用户的名义转账 MyToken。 因此，在下面的脚本中，我们将首先批准此步骤，然后调用函数：

```
const MyToken = artifacts.require("MyToken")
const FarmToken = artifacts.require("FarmToken")

module.exports = async function (callback) {
  const accounts = await new web3.eth.getAccounts()
  const myToken = await MyToken.deployed()
  const farmToken = await FarmToken.deployed()

  // Returns the remaining number of tokens that spender will be allowed to spend on
behalf of owner through transferFrom.
  // This is zero by default.
  const allowanceBefore = await myToken.allowance(
    accounts[0],
    farmToken.address
  )
  console.log(
    "Amount of MyToken FarmToken is allowed to transfer on our behalf Before: " +
      allowanceBefore.toString()
  )

  // In order to allow the Smart Contract to transfer to MyToken (ERC-20) on the
accounts[0] behalf,
  // we must explicitly allow it.
  // We allow farmToken to transfer x amount of MyToken on our behalf
  await myToken.approve(farmToken.address, web3.utils.toWei("100", "ether"))

  // Validate that the farmToken can now move x amount of MyToken on our behalf
  const allowanceAfter = await myToken.allowance(accounts[0], farmToken.address)
  console.log(
    "Amount of MyToken FarmToken is allowed to transfer on our behalf After: " +
      allowanceAfter.toString()
  )
```

```
  // Verify accounts[0] and farmToken balance of MyToken before and after the transfer
  balanceMyTokenBeforeAccounts0 = await myToken.balanceOf(accounts[0])
  balanceMyTokenBeforeFarmToken = await myToken.balanceOf(farmToken.address)
  console.log("*** My Token ***")
  console.log(
    "Balance MyToken Before accounts[0] " +
      web3.utils.fromWei(balanceMyTokenBeforeAccounts0.toString())
  )
  console.log(
    "Balance MyToken Before TokenFarm " +
      web3.utils.fromWei(balanceMyTokenBeforeFarmToken.toString())
  )

  console.log("*** Farm Token ***")
  balanceFarmTokenBeforeAccounts0 = await farmToken.balanceOf(accounts[0])
  balanceFarmTokenBeforeFarmToken = await farmToken.balanceOf(farmToken.address)
  console.log(
    "Balance FarmToken Before accounts[0] " +
      web3.utils.fromWei(balanceFarmTokenBeforeAccounts0.toString())
  )
  console.log(
    "Balance FarmToken Before TokenFarm " +
      web3.utils.fromWei(balanceFarmTokenBeforeFarmToken.toString())
  )
  // Call Deposit function from FarmToken
  console.log("Call Deposit Function")
  await farmToken.deposit(web3.utils.toWei("100", "ether"))
  console.log("*** My Token ***")
  balanceMyTokenAfterAccounts0 = await myToken.balanceOf(accounts[0])
  balanceMyTokenAfterFarmToken = await myToken.balanceOf(farmToken.address)
  console.log(
    "Balance MyToken After accounts[0] " +
      web3.utils.fromWei(balanceMyTokenAfterAccounts0.toString())
  )
  console.log(
    "Balance MyToken After TokenFarm " +
      web3.utils.fromWei(balanceMyTokenAfterFarmToken.toString())
  )

  console.log("*** Farm Token ***")
  balanceFarmTokenAfterAccounts0 = await farmToken.balanceOf(accounts[0])
  balanceFarmTokenAfterFarmToken = await farmToken.balanceOf(farmToken.address)
  console.log(
    "Balance FarmToken After accounts[0] " +
      web3.utils.fromWei(balanceFarmTokenAfterAccounts0.toString())
  )
  console.log(
    "Balance FarmToken After TokenFarm " +
      web3.utils.fromWei(balanceFarmTokenAfterFarmToken.toString())
  )

  // End function
  callback()
}
```

要运行此脚本：`truffle exec .\script\transferMyTokenToFarmToken.js`。 您将在控制台上看到如下输出：

```
*** My Token ***
Balance MyToken Before accounts[0] 1000000
Balance MyToken Before TokenFarm 0
*** Farm Token ***
Balance FarmToken Before accounts[0] 0
Balance FarmToken Before TokenFarm 0
Call Deposit Function
*** My Token ***
Balance MyToken After accounts[0] 999900
Balance MyToken After TokenFarm 100
*** Farm Token ***
Balance FarmToken After accounts[0] 100
Balance FarmToken After TokenFarm 0
```

transferMyTokenToFarmToken.js 的输出

正如我们所见，我们已成功将 MyToken 存入智能合约，因为第一个帐户现在已经有了 FarmToken。

要取款：

```
const MyToken = artifacts.require("MyToken")
const FarmToken = artifacts.require("FarmToken")

module.exports = async function (callback) {
  const accounts = await new web3.eth.getAccounts()
  const myToken = await MyToken.deployed()
  const farmToken = await FarmToken.deployed()

  // Verify accounts[0] and farmToken balance of MyToken before and after the transfer
  balanceMyTokenBeforeAccounts0 = await myToken.balanceOf(accounts[0])
  balanceMyTokenBeforeFarmToken = await myToken.balanceOf(farmToken.address)
  console.log("*** My Token ***")
  console.log(
    "Balance MyToken Before accounts[0] " +
      web3.utils.fromWei(balanceMyTokenBeforeAccounts0.toString())
  )
  console.log(
    "Balance MyToken Before TokenFarm " +
      web3.utils.fromWei(balanceMyTokenBeforeFarmToken.toString())
  )

  console.log("*** Farm Token ***")
  balanceFarmTokenBeforeAccounts0 = await farmToken.balanceOf(accounts[0])
  balanceFarmTokenBeforeFarmToken = await farmToken.balanceOf(farmToken.address)
  console.log(
    "Balance FarmToken Before accounts[0] " +
      web3.utils.fromWei(balanceFarmTokenBeforeAccounts0.toString())
  )
  console.log(
    "Balance FarmToken Before TokenFarm " +
      web3.utils.fromWei(balanceFarmTokenBeforeFarmToken.toString())
  )
```

```
  // Call Deposit function from FarmToken
  console.log("Call Withdraw Function")
  await farmToken.withdraw(web3.utils.toWei("100", "ether"))

  console.log("*** My Token ***")
  balanceMyTokenAfterAccounts0 = await myToken.balanceOf(accounts[0])
  balanceMyTokenAfterFarmToken = await myToken.balanceOf(farmToken.address)
  console.log(
    "Balance MyToken After accounts[0] " +
      web3.utils.fromWei(balanceMyTokenAfterAccounts0.toString())
  )
  console.log(
    "Balance MyToken After TokenFarm " +
      web3.utils.fromWei(balanceMyTokenAfterFarmToken.toString())
  )

  console.log("*** Farm Token ***")
  balanceFarmTokenAfterAccounts0 = await farmToken.balanceOf(accounts[0])
  balanceFarmTokenAfterFarmToken = await farmToken.balanceOf(farmToken.address)
  console.log(
    "Balance FarmToken After accounts[0] " +
      web3.utils.fromWei(balanceFarmTokenAfterAccounts0.toString())
  )
  console.log(
    "Balance FarmToken After TokenFarm " +
      web3.utils.fromWei(balanceFarmTokenAfterFarmToken.toString())
  )

  // End function
  callback()
}
```

要运行此脚本：`truffle exec .\script\transferMyTokenToFarmToken.js`。 正如我们在下面的输出中所看到的，我们已经成功地取回了 MyToken，并消耗掉了 FarmToken：

```
*** My Token ***
Balance MyToken Before accounts[0] 999900
Balance MyToken Before TokenFarm 100
*** Farm Token ***
Balance FarmToken Before accounts[0] 100
Balance FarmToken Before TokenFarm 0
Call Withdraw Function
*** My Token ***
Balance MyToken After accounts[0] 1000000
Balance MyToken After TokenFarm 0
*** Farm Token ***
Balance FarmToken After accounts[0] 0
Balance FarmToken After TokenFarm 0
```

withdrawMyTokenFromTokenFarm.js 的输出

参考文献 {#references}

[合约 - OpenZepelin 文档]

[智能合约的适用工具 | Truffle 套装]

Ganache | Truffle 套装

什么是去中心化金融？ 初学者指南（2021 年更新）(99bitcoins.com)

去中心化金融 - DeFi Pulse 的去中心化金融排行榜

title: 部署第一个智能合约
description: 介绍如何在以太坊测试网络上部署您第一个智能合约
author: "jdourlens"
tags:

- "智能合约"
- "remix"
- "solidity"
- "入门指南"
- "部署"
  skill: 初学者
  lang: zh
  sidebar: true
  published: 2020-04-03
  source: 以太坊开发团队
  sourceUrl: https://ethereumdev.io/deploying-your-first-smart-contract/
  address: "0x19dE91Af973F404EDF5B4c093983a7c6E3EC8ccE"

我猜您和我们一样会很兴奋在以太坊区块链上部署智能合约并与之交互。

别担心，作为我们的第一个智能合约，我们会将其部署在本地测试网络上，因此您不需要任何开销就可以随意部署和运行它。

编写合约 {#writing-our-contract}

第一步访问 Remix并创建一个新文件。 在 Remix 界面的左上角添加一个新文件，并输入所需的文件名。

在Remix界面中添加一个新文件

在这个新文件中，我们将粘贴如下代码：

```
// SPDX-License-Identifier: MIT
pragma solidity >=0.5.17;

contract Counter {

    // Public variable of type unsigned int to keep the number of counts
    uint256 public count = 0;

    // Function that increments our counter
    function increment() public {
        count += 1;
    }

    // Not necessary getter to get the count value
    function getCount() public view returns (uint256) {
        return count;
    }
```

```
}
```

如果您曾经写过程序，应该可以轻松猜到这个程序是做什么的。 下面按行解释：

- 第 3 行：定义了一个名为`Counter`的合约。
- 第 6 行：我们的合约存储了一个无符号整型`count`，从 0 开始。
- 第 9 行：第一个函数将修改合约的状态并且`increment()`变量 `count`。
- 第 14 行，第二个函数是一个 getter 函数，能够从智能合约外部读取`count`变量的值。 请注意，因为我们将`count`变量定义为公共变量，所以这个函数是不必要的，但它可以作为一个例子展示。

第一个简单的智能合约到此结束。 正如您所知，它看上去像是 Java、C++这样的面向对象编程语言中的一个类。 现在可以运行我们的合约了。

部署合约 {#deploying-our-contract}

当我们写了第一个智能合约后，我们现在可以将它部署在区块链中并运行它。

在区块链上部署智能合约实际上只是发送了一个包含已编译智能合约代码的交易，并且没有指定任何收件人。

我们首先点击左侧的编译图标来编译合约：

Remix工具栏中的编译图标

然后点击编译按钮：

Remix solidity编译器的编译按钮

您可以选择“自动编译”选项，以便在您将内容保存到文本编辑器时始终编译合约。

然后切换到部署和运行交易屏幕：

Remix工具栏的部署图标

在“部署和运行交易”屏幕上，仔细检查显示的合约名称并点击“部署”。 正如您在页面顶部看到的那样，当前环境是“JavaScript VM”，这意味着我们将在本地测试区块链上部署我们的智能合约并与之交互，以便能够更快地进行测试并且无需任何费用。

Remix solidity编译器的部署按钮

点击“部署”按钮后，您可以看到合约在底部显示出来。 点击左侧的箭头展开，可以看到合约的内容。 这里有我们的变量`counter`、函数`increment()`和 getter `getCounter()`。

如果您点击`count`或`getCount`按钮，它将实际检索合约的`count`变量的内容，并显示出来。 因为我们尚未调用`increment`函数，它应该显示 0。

Remix solidity编译器的函数按钮

现在点击按钮来调用`increment`函数。 您可以在窗口底部看到交易产生的日志。 当按下检索数据按钮而非`increment`按钮时，您看到的日志有所不同。 这是因为读取区块链的数据不需要任何交易（写入）或费用。 因为只有修改区块链的状态需要进行交易。

交易日志

在按下 increment 按钮后，将产生一个交易来调用我们的`increment()`函数，如果我们点击 count 或 getCount 按钮，将读取我们的智能合约的最新状态，count 变量大于 0。

智能合约状态的最新更新

在下一个教程中，我们将论述如何在智能合约中添加事件。 记录事件是调试智能合约了解调用函数时所发生情况的方便方式。

title: "如何缩减合约以规避合约大小限制"
description: 您可以做些什么避免智能合约变得太大?
author: Markus Waas
lang: zh
sidebar: true
tags:

- "solidity"
- "智能合约"
- "存储"
- "truffle"
  skill: 中级难度
  published: 2020-06-26
  source: soliditydeveloper.com
  sourceUrl: https://soliditydeveloper.com/max-contract-size

为什么会有限制？ {#why-is-there-a-limit}

在 2016 年 11 月 22 日， Spilious Dragon 硬分叉引入了 EIP-170，增加了 24.576 kb 的智能合约大小限制。 对于作为 Solidity 开发者的您来说，这意味着当您向合约中添加越来越多的功能时，在某个时候您会达到极限，并且在部署时会看到错误：

`警告：合约代码大小超过 24576 字节（Spurious Dragon 分叉中引入的限制）。 这个合约可能无法在主网上部署。 请考虑启用优化器（其“运行”值较低！），关闭 revert 字符串，或使用库。`

引入这一限制是为了防止拒绝服务 (DOS) 攻击。 任何对合约的调用从矿工费上来说都是相对便宜的。 然而，根据被调用合约代码的大小（从磁盘读取代码、预处理代码、将数据添加到 Merkle 证明），合约调用对以太坊节点的影响会不成比例地增加。 每当您出现这样的情况，攻击者只需要很少的资源就能给别人造成大量的工作，您就有可能遭受 DOS 攻击。

最初，这不是什么大问题，因为一个自然合约大小限制是区块 Gas 限制。 很明显，一个合约需要被部署在一个交易中，这个交易持有合约的所有字节码。 如果您只将一个交易纳入一个区块，您可以用完所有的 Gas，但 gas 不是无限的。 但这种情况下的问题是，区块 Gas 限制随时间发生变化，理论上是无限制的。 在引入 EIP-170 的时候，区块 Gas 的限制仅为 470 万。 现在区块 gas 上限在上个月再次提高至 1,190 万。

开始战斗吧！ {#taking-on-the-fight}

不幸的是，没有简单的方法来获得合约的字节码大小。 如果您使用 Truffle ，有一个很好的工具可以帮助您：那就是 truffle-contract-size 插件。

1. `npm install truffle-contract-size`
2. 将插件添加到 *truffle-config.js*： `plugins: ["truffle-contract-size"]`
3. 运行 `truffle run contract-size`

这将帮助您了解您的更改对合约总大小有何影响。

在下文中，我们将根据其潜在的影响顺序来研究一些方法。 从减肥的角度来谈一谈。 对于一个人来说，要达到他们的目标体重（在我们的例子中是 24 kb），最好的策略是首先关注影响较大的方法。 在大多数情况下，只要调整您的饮食就能达到目标，但有时您需要做得更多一些。 然后您可以增加一些锻炼（中等影响）或甚至补充剂（小影响）。

较大影响 {#big-impact}

把合约分开 {#separate-your-contracts}

这应始终是您的第一个方法。 如何将合约分成多个较小的合约？ 一般来说，它会迫使您为您的合约想出一个好的架构。 从代码可读性的角度来看，较小的合约总是首选。 对于拆分合同，请问您自己：

- 哪些函数属于同一类？ 每一组函数最好能在自己的合约中。
- 哪些函数不需要读取合约状态或仅需要读取状态的特定子集？
- 您能把存储和功能分开吗？

使用库 {#libraries}

将功能代码移出存储空间的一个简单方法是使用库。 不要将库函数声明为内部函数，因为这些函数将在编译过程中直接被添加到合约中。 但是，如果您使用公共函数，那么这些函数事实上将在一个单独的库合约中。 可以考虑使用命令 using for，使库的使用更加方便。

使用代理 {#proxies}

一个更先进的策略是代理系统。 库在后台使用 `DELEGATECALL`，它只是用调用合约的状态执行另一个合约的函数。 查看这篇博客文章来了解更多关于代理系统的信息。 它们可以为您提供了更多的功能，例如它们能够升级，但它们也增加了很多复杂性。 我不会仅仅为了减少合约的大小而增加这些东西，除非由于某种原因这是唯一的选择。

中等影响 {#medium-impact}

移除函数 {#remove-functions}

这一点应该是显而易见的， 即函数在一定程度上会增加合约的大小。

- **外部函数**：为了方便起见，我们常常添加大量视图函数。 这完全没有问题，直到您遇到大小限制。 然后，您可能需要真正考虑，移除绝对必要以外的所有内容。
- **内部函数**：您也可以移除内部/ 私有函数，只要函数只被调用一次，就可以简单地内联代码。

避免额外的变量 {#avoid-additional-variables}

像这样一个简单的更改：

```
function get(uint id) returns (address,address) {
    MyStruct memory myStruct = myStructs[id];
    return (myStruct.addr1, myStruct.addr2);
}
```

```
function get(uint id) returns (address,address) {
    return (myStructs[id].addr1, myStructs[id].addr2);
}
```

大小差异为** 0.28kb**。 您有可能在合约中找到许多类似的情况，这些情况加起来可以达到很大的数额。

缩短错误信息 {#shorten-error-message}

长的恢复信息，尤其是许多不同的回滚消息可能会使合约变得臃肿。 相反，使用简短的错误代码，并在合约中对其进行解码。 一条长信息可以变得很短：

```
require(msg.sender == owner, "Only the owner of this contract can call this function");
```

```
require(msg.sender == owner, "OW1");
```

在优化器中考虑一个低运行值 {#consider-a-low-run-value-in-the-optimizer}

您也可以更改优化器设置。 默认值为 200，表示它试图在一个函数被调用 200 次的情况下优化字节码。 如果您将其改为 1，相当于告诉优化器针对每个函数只运行一次的情况进行优化。 一个仅运行一次的优化函数意味着它对部署本身进行了优化。 请注意，**这将会增加运行函数的 gas 成本**，所以，您可能不想这样做。

轻微的影响 {#small-impact}

避免将结构体传递给函数 {#avoid-passing-structs-to-functions}

如果您正在使用 ABIEncoderV2，它可以帮助您不将结构体传递给函数。 不会将参数作为结构体传递...

```
function get(uint id) returns (address,address) {
    return _get(myStruct);
}

function _get(MyStruct memory myStruct) private view returns(address,address) {
    return (myStruct.addr1, myStruct.addr2);
}
```

```
function get(uint id) returns(address,address) {
    return _get(myStructs[id].addr1, myStructs[id].addr2);
}

function _get(address addr1, address addr2) private view returns(address,address) {
    return (addr1, addr2);
}
```

... 而是直接传递所需参数。 在此示例中，我们又节省了 **0.1kb**。

声明函数和变量的正确可见性 {#declare-correct-visibility-for-functions-and-variables}

- 函数或变量仅从外部调用？ 那么，将他们声明为 `external` 而不是 `public`。
- 函数或变量仅从合约内调用？ 那么，将它们声明为 `private` 或 `internal` 而不是 `public`。

移除修改器 {#remove-modifiers}

修改器，尤其是在密集使用的情况下，可能会对合约大小产生重大影响。 可以考虑移除它们，而改用函数。

```
modifier checkStuff() {}

function doSomething() checkStuff {}
```

```
function checkStuff() private {}

function doSomething() { checkStuff(); }
```

这些提示应有助您大幅度减少合约的大小。 我要再次强调的是，如果可能的话，务必将重点放在拆分合约上，以获得最大的效果。

合约大小限制的未来 {#the-future-for-the-contract-size-limits}

有一个公开提议，要取消合约大小限制。 这个设想实际上是为了让大型合约的调用更加昂贵。 这个实现起来不会太困难，具有简单的向后兼容性（将所有以前部署的合约都归入最便宜的类别），但是并不是每个人都认同这个提议。

只有时间才能证明这些限制将来是否会改变，大家的反应（见右图）无疑表明了对我们开发人员的明确要求。 不幸的是，这不是您可以指望很快会发生的事情。

title: "ERC-20 合约概览"
description: OpenZepelin 的 ERC-20 合约内容和解读
author: Ori Pomerantz
lang: zh
sidebar: true
tags:

- "solidity"
- "erc-20"
  skill: 初学者
  published: 2021-03-09

简介 {#introduction}

以太坊最常见的用途之一是由一个团队来打造一种可以交易的代币，在某种意义上是他们自己的货币。 这些代币通常遵循一个标准， ERC-20。 此标准使得人们能够以此来开发可以用于所有 ERC-20 代币的工具，如流动资金池和钱包。 在这篇文章中，我们将带领大家分析 OpenZepelin Solidity ERC20 实现以及 ERC20 接口定义。

这里使用的是附加说明的源代码。 如果想要实现 ERC-20， 请阅读此教程。

接口 {#the-interface}

像 ERC-20 这样的标准，其目的是允许符合标准的多种代币，都可以在应用程序之间进行互操作，例如钱包和分布式交易所。 为实现这个目的，我们要创建一个 接口。 任何需要使用代币合约的代码 可以在接口中使用相同的定义，并且与使用它的所有代币合约兼容。无论是像 Metamask 这样的钱包、 诸如 etherscan.io 之类的去中心化应用程序，或一种不同的合约，例如流动资金池。

ERC-20 接口说明

如果您是一位经验丰富的程序员，您可能记得在 Java 中，甚至在 C 头文件 中看到过类似的构造。

这是来自 OpenZepelin 的 ERC-20 接口 的定义。 这是将人类可读标准转换为 Solidity 代码。 当然， 接口本身并不定义*如何*做事。 这一点在下文合约的源代码中作了解释。

```
// SPDX-License-Identifier: MIT
```

Solidity 文件中一般需要标识软件许可证。 您可以在这里看到许可证列表。 如果需要不同的 许可证，只需在注释中加以说明。

```
pragma solidity >=0.6.0 <0.8.0;
```

Solidity 语言仍在迅速地发展，新版本可能不适配旧的代码 (请点击此处查看)。 因此，最好不仅指定一个最低的 语言版本，也指定一个最高的版本，即测试过代码的最新版本。

```
/**
 * @dev Interface of the ERC20 standard as defined in EIP.
 */
```

注释中的 `@dev` 是 NatSpec 格式的一部分，用于 从源代码生成文档。

```
interface IERC20 {
```

根据惯例，接口名称以 `I` 开头。

```
    /**
     * @dev Returns the amount of tokens in existence.
     */
    function totalSupply() external view returns (uint256);
```

此函数标记为 `external`，表示它只能从合约之外调用。 它返回的是合约中代币的总供应量 这个值按以太坊中最常见的类型返回，即无符号的 256 位（256 位是 以太坊虚拟机的原生字长宽度）。 此函数也是视图 `view` 类型，这意味着它不会改变合约状态，这样它可以在单个节点上执行，而不需要在区块链的每个节点上执行。 这类函数不会生成交易，也不会消耗燃料。

注意：理论上讲，合约创建者可能会通过返回比实际数量少的总供应量来做骗局，让每个代币 比实际看起来更有价值。 然而，这种担忧忽视了区块链的真正内涵。 所有在区块链上发生的事情都要通过每个节点 进行验证。 为了实现这一点，每个合约的机器语言代码和存储都可以在每个节点上找到。 虽然无需发布您的合约代码，但这样其它人都不会认真对待您，除非您发布源代码和用于编译的 Solidity 版本，这样人们可以用它来验证您提供的机器语言代码。 例如，请查看此合约。

```
/**
 * @dev Returns the amount of tokens owned by `account`.
 */
function balanceOf(address account) external view returns (uint256);
```

顾名思义，`balanceOf` 返回一个账户的余额。 以太坊帐户在 Solidity 中通过 `address` 类型识别，该类型有 160 位。 它也是 `external` 和 `view` 类型。

```
/**
 * @dev Moves `amount` tokens from the caller's account to `recipient`.
 *
 * Returns a boolean value indicating whether the operation succeeded.
 *
 * Emits a {Transfer} event.
 */
function transfer(address recipient, uint256 amount) external returns (bool);
```

`transfer` 函数将代币从调用者地址转移到另一个地址。 这涉及到状态的更改，所以它不是 `view` 类型。 当用户调用此函数时，它会创建交易并消耗燃料。 还会触发一个 `Transfer` 事件，以通知区块链上的所有人。

该函数有两种输出，对应两种不同的调用：

- 直接从用户接口调用函数的用户。 此类用户通常会提交一个交易 并且不会等待响应，因为响应可能需要无限期的时间。 用户可以查看交易收据 （通常通过交易哈希值识别）或者查看 `Transfer` 事件，以确定发生了什么。
- 将函数作为整个交易一部分调用的其他合约 这些合约可立即获得结果， 由于它们在相同的交易里运行，因此可以使用函数返回值。

更改合约状态的其他函数创建的同类型输出。

限额允许帐户使用属于另一位所有者的代币。 比如，当合约作为卖方时，这个函数就很实用。 合约无法 监听事件，如果买方要将代币直接转给卖方合约， 该合约无法知道已经获得付款。 因此，买方允许 卖方合约支付一定的额度，而让卖方转账相应金额。 这通过卖方合约调用的函数完成，这样卖方合约 可以知道是否成功。

```
/**
 * @dev Returns the remaining number of tokens that `spender` will be
 * allowed to spend on behalf of `owner` through {transferFrom}. This is
 * zero by default.
 *
 * This value changes when {approve} or {transferFrom} are called.
 */
function allowance(address owner, address spender) external view returns (uint256);
```

`allowance` 函数允许任何人查询一个 地址（`owner`）给另一个地址（`spender`）的许可额度。

```
    /**
     * @dev Sets `amount` as the allowance of `spender` over the caller's tokens.
     *
     * Returns a boolean value indicating whether the operation succeeded.
     *
     * IMPORTANT: Beware that changing an allowance with this method brings the risk
     * that someone may use both the old and the new allowance by unfortunate
     * transaction ordering. One possible solution to mitigate this race
     * condition is to first reduce the spender's allowance to 0 and set the
     * desired value afterwards:
     * https://github.com/ethereum/EIPs/issues/20#issuecomment-263524729
     *
     * Emits an {Approval} event.
     */
    function approve(address spender, uint256 amount) external returns (bool);
```

`approve` 函数创建了一个许可额度。 请务必阅读关于 如何避免函数被滥用的信息。 在以太坊中，您可以控制自己交易的顺序， 但无法控制其他方交易的执行顺序， 除非在看到其他方的交易发生之前 不提交您自己的交易。

```
    /**
     * @dev Moves `amount` tokens from `sender` to `recipient` using the
     * allowance mechanism. `amount` is then deducted from the caller's
     * allowance.
     *
     * Returns a boolean value indicating whether the operation succeeded.
     *
     * Emits a {Transfer} event.
     */
    function transferFrom(address sender, address recipient, uint256 amount) external
returns (bool);
```

最后，消费者使用 `transferFrom` 函数用来使用许可额度。

```
    /**
     * @dev Emitted when `value` tokens are moved from one account (`from`) to
     * another (`to`).
     *
     * Note that `value` may be zero.
     */
    event Transfer(address indexed from, address indexed to, uint256 value);

    /**
     * @dev Emitted when the allowance of a `spender` for an `owner` is set by
     * a call to {approve}. `value` is the new allowance.
     */
    event Approval(address indexed owner, address indexed spender, uint256 value);
```

```
}
```

在 ERC-20 合约状态发生变化时就会激发这些事件。

实际合约 {#the-actual-contract}

这是实现 ERC-20 标准的实际合约， 摘自此处。 不能照原样使用，但可以 通过继承将其扩展，使之可用。

```
// SPDX-License-Identifier: MIT
pragma solidity >=0.6.0 <0.8.0;
```

导入声明 {#import-statements}

除了上述接口定义外，合约定义还要导入两个其他文件：

```
import "../../GSN/Context.sol";
import "./IERC20.sol";
import "../../math/SafeMath.sol";
```

- `GSN/Context.sol` 是使用 OpenGSN 所需的文件，该系统允许用户在没有以太币的情况下 使用区块链。 请注意，这里的文件是旧版本，如果需要集成 OpenGSN， 请使用此教程。
- SafeMath 库，用于 完成没有溢出问题的加法和减法。 这非常必要，否则会出现，用户仅有一个代币，花掉两个代币后，反而有了 2^256-1 个代币。

这里的注释说明了合约的目的。

```
/**
 * @dev Implementation of the {IERC20} interface.
 *
 * This implementation is agnostic to the way tokens are created. This means
 * that a supply mechanism has to be added in a derived contract using {_mint}.
 * For a generic mechanism see {ERC20PresetMinterPauser}.
 *
 * TIP: For a detailed writeup see our guide
 * https://forum.zeppelin.solutions/t/how-to-implement-erc20-supply-mechanisms/226[How
 * to implement supply mechanisms].
 *
 * We have followed general OpenZeppelin guidelines: functions revert instead
 * of returning `false` on failure. This behavior is nonetheless conventional
 * and does not conflict with the expectations of ERC20 applications.
 *
 * Additionally, an {Approval} event is emitted on calls to {transferFrom}.
 * This allows applications to reconstruct the allowance for all accounts just
 * by listening to said events. Other implementations of the EIP may not emit
 * these events, as it isn't required by the specification.
 *
 * Finally, the non-standard {decreaseAllowance} and {increaseAllowance}
 * functions have been added to mitigate the well-known issues around setting
 * allowances. See {IERC20-approve}.
```

```
 */
```

合约定义 {#contract-definition}

```
contract ERC20 is Context, IERC20 {
```

此行指定继承，在本例中来自上面的 `IERC20` 和 `Context`，用于 OpenGSN。

```
    using SafeMath for uint256;
```

此行将 `SafeMath` 库附加到 `uint256` 类型。 您可以在 此处找到此程序库。

变量的定义 {#variable-definitions}

这些定义具体指定了合约的状态变量。 虽然声明这些变量为 `private`，但 这只意味着区块链上的其他合约无法读取它们。 *区块链上 没有秘密*，所有节点上的软件在每个区块上 都有每个合约的状态。 根据惯例，状态变量名称为 `_<something>`。

前两个变量是映射， 表示它们的结果与关联数组相同， 不同之处在于关键词为数值。 存储空间仅分配给数值不同于 默认值（零）的条目。

```
    mapping (address => uint256) private _balances;
```

第一个映射，`_balances`，是代币地址和对应的余额。 要查看 余额，请使用此语法：`_balances[<address>]`。

```
    映射 (address => mapping (address => uint256)) private _allowances;
```

此变量，`_allowances` 存储之前提到过的许可限额。 第一个索引是 代币的所有者，第二个索引是获得许可限额的合约。 要查询地址 A 可以 从地址 B 账户中支出的额度，请使用 `_allowances[B][A]`。

```
    uint256 private _totalSupply;
```

顾名思义，此变量记录代币供应总量。

```
    string private _name;
    string private _symbol;
    uint8 private _decimals;
```

这三个变量用于提高可读性。 前两项的含义不言自明，但 `_decimals` 并非如此。

一方面，以太坊不具有浮点数或分数变量。 另一方面， 人们希望能够拆分代币。 人们选择将黄金做为货币的一个原因是 当有人想要购买一只牛的一小部分时，就很难找零。

解决方案是保持整数值，但是计数时使用一个价值非常小的分数代币， 而不是真正的代币。 就以太币而言，分数代币称为 wei，10^18 个 wei 等于一个 以太币。 在撰写本文时，10,000,000,000,000 wei 约等于一美分或欧分。

应用程序需要知道如何显示代币余额。 如果某位用户有 3,141,000,000,000,000,000 wei，那是否是 3.14 个以太币？ 31.41 个以太币？ 还是 3,141 个以太币？ 对于以太币，10^18 个 wei 等于 1 个以太币，但对于您的 代币，您可以选择一个不同的值。 如果无法合理拆分代币，您可以将 `_decimals` 值设为零。 如果想要使用与以太币相同的标准，请使用 **18**。

构造函数 {#the-constructor}

```
/**
 * @dev Sets the values for {name} and {symbol}, initializes {decimals} with
 * a default value of 18.
 *
 * To select a different value for {decimals}, use {_setupDecimals}.
 *
 * All three of these values are immutable: they can only be set once during
 * construction.
 */
constructor (string memory name_, string memory symbol_) public {
    _name = name_;
    _symbol = symbol_;
    _decimals = 18;
}
```

构造函数在首次创建合约时调用。 根据惯例，函数参数名为 `<something>_`。

用户接口函数 {#user-interface-functions}

```
/**
 * @dev Returns the name of the token.
 */
function name() public view returns (string memory) {
    return _name;
}

/**
 * @dev Returns the symbol of the token, usually a shorter version of the
 * name.
 */
function symbol() public view returns (string memory) {
    return _symbol;
}

/**
 * @dev Returns the number of decimals used to get its user representation.
 * For example, if `decimals` equals `2`, a balance of `505` tokens should
 * be displayed to a user as `5,05` (`505 / 10 ** 2`).
 *
 * Tokens usually opt for a value of 18, imitating the relationship between
 * ether and wei. This is the value {ERC20} uses, unless {_setupDecimals} is
```

```
     * called.
     *
     * NOTE: This information is only used for _display_ purposes: it in
     * no way affects any of the arithmetic of the contract, including
     * {IERC20-balanceOf} and {IERC20-transfer}.
     */
    function decimals() public view returns (uint8) {
        return _decimals;
    }
```

这些函数，`name`、`symbol` 和 `decimals` 帮助用户界面了解合约，从而正常演示合约。

返回类型为 `string memory`，意味着返回在内存中存储的字符串。 变量，如 字符串，可以存储在三个位置：

	有效时间	合约访问	燃料成本
内存	函数调用	读/写	几十到几百不等（距离越远费用越高）
调用数据	函数调用	只读	不可用作返回类型，只可用作函数参数
存储	直到被修改	读/写	高（读取需要 800，写入需要 2 万）

在这种情况下，`memory` 是最好的选择。

读取代币信息 {#read-token-information}

这些是提供代币信息的函数，不管是总量还是 账户余额。

```
    /**
     * @dev See {IERC20-totalSupply}.
     */
    function totalSupply() public view override returns (uint256) {
        return _totalSupply;
    }
```

`totalSupply` 函数返回代币的总量。

```
    /**
     * @dev See {IERC20-balanceOf}.
     */
    function balanceOf(address account) public view override returns (uint256) {
        return _balances[account];
    }
```

读取一个帐户的余额。 请注意，任何人都可以查看他人账户的余额。 试图隐藏此信息没有意义，因为它在每个节点上 都是可见的。 *区块链上没有秘密*

代币转账 {#transfer-tokens}

```
    /**
     * @dev See {IERC20-transfer}.
     *
     * Requirements:
     *
     * - `recipient` cannot be the zero address.
     * - the caller must have a balance of at least `amount`.
     */
    function transfer(address recipient, uint256 amount) public virtual override returns
(bool) {
```

调用 `transfer` 函数以从发送人的帐户转移代币到另一个帐户。 注意 虽然函数返回的是布尔值，但那个值始终为**真实值**。 如果转账失败， 合约会撤销调用。

```
        _transfer(_msgSender(), recipient, amount);
        return true;
    }
```

`_transfer` 函数完成了实际工作。 这是一个私有函数，只能由 其他合约函数调用。 根据常规，私人函数名为 `_<something>`，与状态 变量相同。

在 Solidity 中，我们通常使用 `msg.sender` 代表信息发送人。 然而，这会破坏 OpenGSN 的规则。 如果我们想使用代币进行交易而不用以太币，我们 需要使用 `_msgSender()`。 对于正常交易，它返回 `msg.sender`，但是对于没有以太币的交易， 则返回原始签名而不是传递信息的合约。

许可额度函数 {#allowance-functions}

这些是实现许可额度功能的函数：`allowance`、`approve`、`transferFrom` 和 `_approve`。 此外，除基本标准外，OpenZepelin 实现还包含了一些能够提高 安全性的功能：`increaseallance` 和 `decreaseAllowance`。

许可额度函数 {#allowance}

```
    /**
     * @dev See {IERC20-allowance}.
     */
    function allowance(address owner, address spender) public view virtual override
returns (uint256) {
        return _allowances[owner][spender];
    }
```

`allowance` 函数使每个人都能检查任何许可额度。

审批函数 {#approve}

```
    /**
     * @dev See {IERC20-approve}.
     *
     * Requirements:
     *
     * - `spender` cannot be the zero address.
     */
    function approve(address spender, uint256 amount) public virtual override returns
(bool) {
```

调用此函数以创建许可额度。 它与上述 `transfer` 函数相似：

- 该函数仅调用一个完成真正工作的内部函数（本例中为 `_approval`）。
- 函数要么返回 `true`（如果成功），要么撤销（如果失败）。

```
        _approve(_msgSender(), spender, amount);
        return true;
    }
```

我们使用内部函数尽量减少发生状态变化之处。 *任何*可以改变状态的 函数都是一种潜在的安全风险，需要对其安全性进行审核。 这样我们就能减少出错的机会。

TransferFrom 函数 {#transferFrom}

这个函数被消费者用于使用许可额度。 这里需要两步操作：将消费的金额转账， 并在许可额度中减去这笔金额。

```
    /**
     * @dev See {IERC20-transferFrom}.
     *
     * Emits an {Approval} event indicating the updated allowance. This is not
     * required by the EIP. See the note at the beginning of {ERC20}.
     *
     * Requirements:
     *
     * - `sender` and `recipient` cannot be the zero address.
     * - `sender` must have a balance of at least `amount`.
     * - the caller must have allowance for ``sender``'s tokens of at least
     * `amount`.
     */
    function transferFrom(address sender, address recipient, uint256 amount) public
virtual
                                                    override returns (bool) {
        _transfer(sender, recipient, amount);
```

`a.sub(b, "message")` 函数调用做了两件事。 首先，它计算了 `a-b`，这是新的许可额度。 之后，它检查这一结果是否为负数。 如果结果为负，将撤销调用，并发出相应的信息。 请注意，撤销调用后，之前在调用中完成的任何处理都会被忽略，所以我们不需要 撤消 `_transfer`。

```
        _approve(sender, _msgSender(), _allowances[sender][_msgSender()].sub(amount,
            "ERC20: transfer amount exceeds allowance"));
        return true;
    }
```

OpenZepelin 安全加法 {#openzeppelin-safety-additions}

将许可额度从一个非零值设定为另一个非零值是有危险的， 因为您只能控制自己的交易顺序，而无法控制其他人的交易顺序。 假设现在有两个用户，天真的 Alice 和不诚实的 Bill。 Alice 想要从 Bill 处获取一些服务， 她认为值五个代币，所以她给了 Bill 五个代币的许可额度。

之后有了一些变化，Bill 的价格提高到了十个代币。 Alice 仍然想要购买服务，就发送了一笔交易，将 Bill 的许可额度设置为 10。 当 Bill 在交易池中看到这个新的交易时， 他就会发送一笔交易，以花费 Alice 的五个代币，并且设定高得多的 燃料价格，这样就会更快挖矿。 这样的话，Bill 可以先花五个代币，然后 当 Alice 的新许可额度放款后，他就可以再花费十个代币，这样总共花费了 15 个代币， 超过了 Alice 本欲授权的金额。 这种技术叫做 抢先交易。

Alice 的交易	Alice 的随机数	Bill 的交易	Bill 的随机数	Bill 的许可额度	Bill 从 Alice 处获得的总收入
approve(Bill, 5)	10			5	0
		transferFrom(Alice, Bill, 5)	10,123	0	5
approve(Bill, 10)	11			10	5
		transferFrom(Alice, Bill, 10)	10,124	0	15

为了避免这个问题，有两个函数（`increaseAllowance` 和 `decreaseAllowance`）使您 能够修改指定数额的许可额度。 所以，如果 Bill 已经花费了五个代币， 他就只能再花五个代币。 根据时间的不同，有两种方法可以工作，介绍时 Bill 只得到十个代币：

A：

Alice 的交易	Alice 的随机数	Bill 的交易	Bill 的随机数	Bill 的许可额度	Bill 从 Alice 处获得的总收入
approve(Bill, 5)	10			5	0
		transferFrom(Alice, Bill, 5)	10,123	0	5
increaseAllowance(Bill, 5)	11			0+5 = 5	5
		transferFrom(Alice, Bill, 5)	10,124	0	10

B:

Alice 的交易	Alice 的随机数	Bill 的交易	Bill 的随机数	Bill 的许可额度	Bill 从 Alice 处获得的总收入
approve(Bill, 5)	10			5	0
increaseAllowance(Bill, 5)	11			5+5 = 10	0
		transferFrom(Alice, Bill, 10)	10,124	0	10

```
    /**
     * @dev Atomically increases the allowance granted to `spender` by the caller.
     *
     * This is an alternative to {approve} that can be used as a mitigation for
     * problems described in {IERC20-approve}.
     *
     * Emits an {Approval} event indicating the updated allowance.
     *
     * Requirements:
     *
     * - `spender` cannot be the zero address.
     */
    function increaseAllowance(address spender, uint256 addedValue) public virtual
returns (bool) {
        _approve(_msgSender(), spender, _allowances[_msgSender()]
[spender].add(addedValue));
        return true;
    }
```

`a.add(b)` 函数是一个安全加法。 在罕见的情况下，`a + b >= 2^256`，不会发生 普通加法会出现的溢出错误。

```
    /**
     * @dev Atomically decreases the allowance granted to `spender` by the caller.
     *
     * This is an alternative to {approve} that can be used as a mitigation for
     * problems described in {IERC20-approve}.
     *
     * Emits an {Approval} event indicating the updated allowance.
     *
     * Requirements:
     *
     * - `spender` cannot be the zero address.
     * - `spender` must have allowance for the caller of at least
     * `subtractedValue`.
     */
    function decreaseAllowance(address spender, uint256 subtractedValue) public virtual
returns (bool) {
        _approve(_msgSender(), spender, _allowances[_msgSender()]
[spender].sub(subtractedValue,
```

```
                "ERC20: decreased allowance below zero"));
        return true;
    }
```

修改代币信息的函数 {#functions-that-modify-token-information}

这些是完成实际工作的四个函数：`_transfer`、`_mint`、`_burn` 和 `_approval`。

_transfer 函数 {#_transfer}

```
    /**
     * @dev Moves tokens `amount` from `sender` to `recipient`.
     *
     * This is internal function is equivalent to {transfer}, and can be used to
     * e.g. implement automatic token fees, slashing mechanisms, etc.
     *
     * Emits a {Transfer} event.
     *
     * Requirements:
     *
     * - `sender` cannot be the zero address.
     * - `recipient` cannot be the zero address.
     * - `sender` must have a balance of at least `amount`.
     */
    function _transfer(address sender, address recipient, uint256 amount) internal
virtual {
```

`_transfer` 这个函数将代币从一个账户转到另一个账户。 有两个函数调用它，分别是 `transfer`（从发送人本人账户发送）和 `transferFrom`（使用许可额度，从其他人的账户发送）。

```
        require(sender != address(0), "ERC20: transfer from the zero address");
        require(recipient != address(0), "ERC20: transfer to the zero address");
```

实际上以太坊中没有人拥有零地址（即不存在对应公钥可以转换为零地址的私钥）。 有人使用该地址时，通常是一个软件漏洞，所以 如果将零地址用作发送人或接收人，交易将失败。

```
        _beforeTokenTransfer(sender, recipient, amount);
```

使用该合约有两种方法：

1. 将其作为模板，编写自己的代码
2. 从它继承一个合约，并且重写您需要修改的函数

第二种方法要好得多，因为 OpenZepelin ERC-20 代码已经过审核，其安全性也已得到证实。 当您的合约继承它时， 可以清楚地表明修改了哪些函数，只需要审核这些特定的函数，人们就会信任您的合约。

代币每次易手时，通常都需要调用一个函数。 然而，`_transfer` 是一个非常重要的函数， 重新编写可能会不安全（见下文），所以最好不要重写。 解决方案是重写 `_beforeTokenTransfer` 函数，这是一个挂钩函数。 您可以重写此函数，之后每次转账都会调用它。

```
        _balances[sender] = _balances[sender].sub(amount, "ERC20: transfer amount exceeds balance");
        _balances[recipient] = _balances[recipient].add(amount);
```

这些是实际实现转账的代码。 请注意，将转账金额从发送人帐户上扣除，然后加到接收人帐户之间， 不得有任何**动作**。 这很重要，因为如果 中间调用不同的合约，可能会被用来骗过这个合约。 目前转账为最小操作单元，即中间什么都不会发生。

```
        emit Transfer(sender, recipient, amount);
    }
```

最后，激发一个 `Transfer` 事件。 智能合约无法访问事件，但区块链外运行的代码 可以监听事件并对其作出反应。 例如，钱包可以跟踪所有者获得更多代币事件。

_mint 和 _burn 函数 {#_mint-and-_burn}

这两个函数（`_mint` 和 `_burn`）修改代币的总供应量。 它们都是内部函数，在原有合约中没有任何调用它们的函数。 因此，仅通过继承合约并添加您自己的逻辑， 来决定在什么条件下可以铸造新代币或消耗现有代币时， 它们才是有用的。

注意：每一个 ERC-20 代币都通过自己的业务逻辑来决定代币管理。 例如，一个固定供应总量的合约可能只在构造函数中调用 `_mint`，而从不调用 `_burn`。 一个销售代币的合约 将在支付时调用 `_mint`，并大概在某个时间点调用 `_burn`， 以避免过快的通货膨胀。

```
    /** @dev Creates `amount` tokens and assigns them to `account`, increasing
     * the total supply.
     *
     * Emits a {Transfer} event with `from` set to the zero address.
     *
     * Requirements:
     *
     * - `to` cannot be the zero address.
     */
    function _mint(address account, uint256 amount) internal virtual {
        require(account != address(0), "ERC20: mint to the zero address");
        _beforeTokenTransfer(address(0), account, amount);
        _totalSupply = _totalSupply.add(amount);
        _balances[account] = _balances[account].add(amount);
        emit Transfer(address(0), account, amount);
    }
```

当代币总数发生变化时，请务必更新 `_totalSupply`。

```
    /**
     * @dev Destroys `amount` tokens from `account`, reducing the
     * total supply.
     *
     * Emits a {Transfer} event with `to` set to the zero address.
```

```
     *
     * Requirements:
     *
     * - `account` cannot be the zero address.
     * - `account` must have at least `amount` tokens.
     */
    function _burn(address account, uint256 amount) internal virtual {
        require(account != address(0), "ERC20: burn from the zero address");

        _beforeTokenTransfer(account, address(0), amount);

        _balances[account] = _balances[account].sub(amount, "ERC20: burn amount exceeds
balance");
        _totalSupply = _totalSupply.sub(amount);
        emit Transfer(account, address(0), amount);
    }
```

`_burn` 函数与 `_mint` 函数几乎完全相同，但它们的方向相反。

_approve 函数 {#_approve}

这是实际设定许可额度的函数。 请注意，它允许所有者指定 一个高于所有者当前余额的许可额度。 这是允许的，因为在转账时 会核查余额，届时可能不同于 创建许可额度时的金额。

```
    /**
     * @dev Sets `amount` as the allowance of `spender` over the `owner` s tokens.
     *
     * This internal function is equivalent to `approve`, and can be used to
     * e.g. set automatic allowances for certain subsystems, etc.
     *
     * Emits an {Approval} event.
     *
     * Requirements:
     *
     * - `owner` cannot be the zero address.
     * - `spender` cannot be the zero address.
     */
    function _approve(address owner, address spender, uint256 amount) internal virtual {
        require(owner != address(0), "ERC20: approve from the zero address");
        require(spender != address(0), "ERC20: approve to the zero address");

        _allowances[owner][spender] = amount;
```

激发一个 `Approval` 事件。 根据应用程序的编写， 消费者合约可以从代币所有者或监听事件的服务器获知审批结果。

```
        emit Approval(owner, spender, amount);
    }
```

修改小数点设置变量 {#modify-the-decimals-variable}

```
    /**
     * @dev Sets {decimals} to a value other than the default one of 18.
     *
     * WARNING: This function should only be called from the constructor. Most
     * applications that interact with token contracts will not expect
     * {decimals} to ever change, and may work incorrectly if it does.
     */
    function _setupDecimals(uint8 decimals_) internal {
        _decimals = decimals_;
    }
```

此函数修改了 `>_decimals` 变量，此变量用于设置用户接口如何计算金额。 您应该从构造函数里面调用。 在之后的任何时候调用都是不正当的， 应用程序一般不会处理。

钩子 {#hooks}

```
    /**
     * @dev Hook that is called before any transfer of tokens. This includes
     * minting and burning.
     *
     * Calling conditions:
     *
     * - when `from` and `to` are both non-zero, `amount` of ``from``'s tokens
     * will be to transferred to `to`.
     * - when `from` is zero, `amount` tokens will be minted for `to`.
     * - when `to` is zero, `amount` of ``from``'s tokens will be burned.
     * - `from` and `to` are never both zero.
     *
     * To learn more about hooks, head to xref:ROOT:extending-contracts.adoc#using-
hooks[Using Hooks].
     */
    function _beforeTokenTransfer(address from, address to, uint256 amount) internal
virtual { }
}
```

这是转账过程中要调用的挂钩函数。 这里是空的，但如果你需要 它做一些你只是覆盖它的事情。

结论 {#conclusion}

复习一下，这些是我认为此合约中最重要的概念（你们的看法可能与我不同）

- *区块链上没有秘密* 智能合约可以访问的任何信息 都可以提供给全世界。
- 您可以控制自己交易的订单，但在其他人的交易发生时， 则不能控制。 这就是为什么更改许可额度时会有风险，因为它 允许消费者花掉这两个许可额度的总和。
- `uint256` 类型值的溢出。 换言之，*0-1=2^256-1*。 如果这不是预期的 行为，您必须自行检查（或使用 SafeMath 库执行该服务）。 请注意， Solidity 0.8.0 中对此进行了更改。
- 将特定类型变量的状态改变放在一个特定的地方，这样可以使审核更容易。 这就是我们使用以下等函数的原因，例如 `_approval` 函数，它可以被 `approve`、 `transferFrom`、 `increaseAllowance` 和

`decreaseAllowance` 调用。

- 状态更改应为最小操作单元，其中没有任何其他动作 （如在 `_transfer` 中所见）。 这是因为在状态更改期间，会出现不一致的情况。 例如， 在减少发送人的余额，和增加接收人的余额之间， 代币总量会小于应有总量。 如果在这两个时刻之间有任何操作， 特别是调用不同的合约，则可能出现滥用。

现在您已经了解了 OpenZeppelin ERC-20 合约是怎么编写的，尤其是如何使之更加安全，您即可编写自己的安全合约和应用程序。

title: 以太坊开发入门
description: "这是一份关于以太坊开发的初学者指南。 我们将带领您从启动一个 API 终端节点开始，到提出一个命令行请求，再到编写您的第一个 web3 脚本。 无需区块链开发经验！"
author: "Elan Halpern"
tags:

- "入门指南"
- "javascript"
- "ethers.js"
- "节点"
- "查询"
- "Alchemy"
  skill: 初学者
  lang: zh
  sidebar: true
  published: 2020-10-30
  source: Medium
  sourceUrl: https://medium.com/alchemy-api/getting-started-with-ethereum-development-using-alchemy-c3d6a45c567f

---

以太坊和Alchemy徽标

这是一份关于以太坊开发的初学者指南。 在本教程中，我们将使用 Alchemy，这是一个领先的区块链开发者平台，为 70% 的顶级区块链应用程序（包括 Maker、0x、MyEtherWallet、Dharma 和 Kyber）的数百万用户提供支持。 Alchemy 使我们能够访问以太坊链上的 API 端点，这样我们就可以读写交易。

我们将带您注册 Alchemy 来编写您的第一个 web3 脚本！ 无需区块链的开发经验！

1. 注册免费 Alchemy 帐户 {#sign-up-for-a-free-alchemy-account}

创建一个 Alchemy 帐户很容易。在此免费注册。

2. 创建一个 Alchemy 应用程序 {#create-an-alchemy-app}

为了与以太坊通信，以及为了使用 Alchemy 的产品，您需要一个 API 密钥来验证您的请求。

您可以通过仪表板创建 API 密钥。 要创建一个新密钥，导航到如下所示的“Create App”：

特别感谢ShapeShift*让我们展示他们的仪表板！*

Alchemy仪表板

填写“Create App”下的详细信息以获取您的新密钥。 在此处还可以看到您以前创建的应用以及您的团队创建的应用。 通过点击任何应用的“View Key”来查看现有密钥。

使用Alchemy创建应用程序的截图

您也可以通过将鼠标悬停在““Apps”上并选择一个来获取现有 API 密钥。 您可以在这里“查看密钥”，以及“编辑应用程序”来特定域名加入白名单、查看几个开发者工具，并查看分析。

显示用户如何获取API密钥的GIF图

3. 在命令行中发送请求 {#make-a-request-from-the-command-line}

使用 JSON-RPC 和 curl 通过 Alchemy 与以太坊区块链交互。

对于手动请求，我们建议通过 `JSON RPC` 发送 `POST` 请求来进行交互。 只需传入 `Content-Type: application/json` 标头和查询作为 `POST` 主体，具有以下字段：

- `jsonrpc`：JSON-RPC 版本，目前只支持 `2.0`。
- `method`： ETH API 方法。 [请参阅 API 参考。]
- `params`： 要传递到方法的参数列表。
- `id`： 请求的 ID。 将通过响应返回，这样就可以跟踪一个响应属于哪个请求。

这是一个可通过命令行运行的示例，用于查询当前燃气价格：

```
curl https://eth-mainnet.alchemyapi.io/v2/demo \
-X POST \
-H "Content-Type: application/json" \
-d '{"jsonrpc":"2.0","method":"eth_gasPrice","params":[],"id":73}'
```

***注：** 将 https://eth-mainnet.alchemyapi.io/v2/demo 替换为您自己的 API 密钥 https://eth-mainnet.alchemyapi.io/v2/**your-api-key**。*

结果：

```
{ "id": 73,"jsonrpc": "2.0","result": "0x09184e72a000" // 10000000000000 }
```

4. 设置 Web3 客户端 {#set-up-your-web3-client}

如果您已有客户端， 将您当前的节点提供商的 URL 更改为您的 API 密钥的 Alchemy URL： `“https://eth-mainnet.alchemyapi.io/v2/your-api-key”`

***注意：** 下面的脚本需要在一个**节点环境**中运行或**保存到一个文件运行**，而不是通过命令行运行。 如果您尚未安装节点或 npm ，请查看此适用于 mac 的快速设置指南。*

有大量可与与 Alchemy 集成的Web3 库。但是，我们建议使用Alchemy Web3，它是 web3.js 的替代插件，可以与 Alchemy 无缝配合工作。 这个库有很多优点，例如自动重试和可靠的 WebSocket 支持。

要安装 AlchemyWeb3.js，请**导航到项目目录**并运行：

使用 yarn：

```
yarn add @alch/alchemy-web3
```

使用 NPM：

```
npm install @alch/alchemy-web3
```

要与 Alchemy 的节点基础设施交互，请在 NodeJS 中运行或将其添加到 JavaScript 文件：

```
const { createAlchemyWeb3 } = require("@alch/alchemy-web3")
const web3 = createAlchemyWeb3(
  "https://eth-mainnet.alchemyapi.io/v2/your-api-key"
)
```

5. 编写您的第一个 Web3 脚本！ {#write-your-first-web3-script}

现在用一个小的 web3 编程来练习，我们将编写一个简单的脚本，用于打印出以太坊主网中最新的区块高度。

1. 在终端中创建一个新的项目目录并通过 cd 命令进入该目录（如果尚未这样做）：

```
mkdir web3-example
cd web3-example
```

2. 在项目中安装 Alchemy web3（或任何 web3）依赖项（如果尚未这样做）：

```
npm install @alch/alchemy-web3
```

3. 创建一个名为`index.js`的文件并添加以下内容： **

> 最终应将`demo`替换为您的 Alchemy HTTP API 密钥 。

```
async function main() {
  const { createAlchemyWeb3 } = require("@alch/alchemy-web3")
  const web3 = createAlchemyWeb3("https://eth-mainnet.alchemyapi.io/v2/demo")
  const blockNumber = await web3.eth.getBlockNumber()
  console.log("The latest block number is " + blockNumber)
}
main()
```

不熟悉 async stuff？ 来看看这篇Medium 文章。

4. 使用 node 在终端中运行程序

```
node index.js
```

5. 现在应该会在控制台中看到最新的区块数量输出结果！

```
The latest block number is 11043912
```

哇！ 恭喜！ 您刚刚使用 Alchemy 编写了您的第一个 web3 脚本 🎉

不知道下一步怎么办？ 尝试部署您的第一个智能合约，并在我们的 Hello World 智能合约指南中用 solidity 编程，或者用 Dashboard Demo App 测试您的仪表盘知识！

免费注册 Alchemy，查看我们的文档，关注我们的Twitter获取最新消息。

title: 去中心化自治组织 (DAO)
description: 以太坊上的 DAO 简介
lang: zh
template: use-cases
emoji: "🤝"
sidebar: true
sidebarDepth: 2
image: ../../../../assets/use-cases/dao-2.png
alt: DAO 代表在对提案投票。
summaryPoint1: 没有集中领导的成员制社区。
summaryPoint2: 与网上陌生人合作的安全方式。

summaryPoint3: 为特定事业投入资金的安全场所。

什么是 DAO？ {#what-are-daos}

DAO 是与全球志同道合之士共同工作的安全有效的方式。

可以将 DAO 视作由成员集体所有和共同管理的互联网原生企业。 DAO 拥有内置资产，未经该组织批准，任何人都无权动用。 DAO 通过提案和投票来决策，以确保组织内的每个人都有发言权。

不会有仅凭心血来潮就随意授权支出的 CEO，也不会有徇私枉法操纵账簿的 CFO。 一切行为公开，有关开支的规则都会通过代码编入 DAO 组织。

我们为什么需要 DAO？ {#why-dao}

与他人创办涉及资金和金钱的组织，需要对与合作对象高度信任。 不过，显然很难相信互联网上素不相识的人。 通过 DAO，您不需要相信组织中的其他人，只需要相信 DAO 的代码就够了，它是 100% 公开透明的，任何人都可以验证。

这为全球合作和协调提供了许多新机会。

对比 {#dao-comparison}

去中心化自治组织 (DAO)	传统组织
通常是平等的，并且完全民主。	通常等级鲜明。
需要成员投票才能实施任何更改。	可能部分人就能进行决策，也可能投票表决，具体取决于组织结构。
不需要可信的中间人就可以自动计算投票、执行结果。	如果允许投票，则在内部计票，投票结果必须由人工处理。
以去中心化方式自动提供服务（例如慈善基金的分配）。	需要人工处理或自动集中控制，易受操纵。
所有活动公开透明。	活动通常是私密进行，不向公众开放。

DAO 示例 {#dao-examples}

为了帮助您更好地理解，这里有一些 DAO 的应用示例：

- 捐赠 - 您可以接受来自世界上任何人的成员资格和捐赠，组织可以决定如何使用捐款。
- 自由职业者网络 - 您可以创建承包商网络，为办公室和软件订阅募集资金。
- 风险投资和赠款 - 您可以成立一个风险基金，汇集投资资本并投票进行商业投资。 后续收益可以分配给相应 DAO 成员。

DAO 成员资格 {#dao-membership}

DAO 成员资格分为多种模式。 成员资格可以决定投票方式和其他 DAO 关键事项。

基于代币的成员资格 {#token-based-membership}

通常无需许可，取决于使用的代币。 这些治理型代币大部分可以在去中心化交易所进行无限制交易。 其余部分要通过提供流动性或者进行工作量证明 (POW) 才能赚取。 无论何种方式，只要持有代币就可以参与投票。

通常用来管理广泛去中心化协议和/或代币本身。

知名案例 {#token-example}

MakerDAO - 可在各大去中心化交易所找到 MakerDAO 的代币 MKR。 因此，任何人都可以购买对 Maker 协议未来的投票权。

基于份额的成员资格 {#share-based-membership}

基于份额的 DAO 通常拥有更多权限，但仍然相当公开透明。 任何潜在的成员都可以提交加入 DAO 的建议，通常以代币或工作的形式提供有价贡献。 份额直接代表投票权和所有权。 成员可以随时带着资金库份额退出。

通常用于联系更紧密、以人为中心的组织，例如慈善机构、工人团体和投资俱乐部等。 也可以管理协议和代币。

知名案例 {#share-example}

MolochDAO - MolochDAO 致力于为以太坊项目募集资金。 他们需要成员资格提案，以便小组可以评估您是否具有必要的专业知识和资本来对潜在资助方做出明智判断。 您无法通过直接购买代币来加入组织。

DAO 如何运作？ {#how-daos-work}

智能合约是 DAO 的核心。 合约界定了组织的规则，管理组织资金。 一旦在以太链上启用合约，除非表决通过，否则任何人都不能修改规则， 任何违背代码规则和逻辑的行为都将失败。 由于资金库由智能合约定义，任何人都不能未经组织批准而挪用资金。 这意味着 DAO 不需要集中管理机构。 相反，组织共同作出决定，而付款会在投票通过后自动获批。

之所以能够做到这一点，是因为智能合约一旦在以太坊中生效，就无法被篡改。 一切都是公开的，只要修改代码（DAO 组织规则）就会被发现。

了解更多关于智能合约的信息

以太坊与 DAO {#ethereum-and-daos}

以太坊为 DAO 提供了良好基础，原因如下：

- 以太坊本身已经建立足够的共识，足以让各类组织信任以太坊网络。
- 智能合约一旦生效就无法更改，即便是其所有者也是如此。 这使得 DAO 能够按照编程规则运行。

- 智能合约可以发送/接收资金。 没有这点，您就需要可信的中间人来管理组织资金。
- 比起竞争，以太坊社区更趋向于合作，这使得各类应用程序和服务系统蓬勃发展。

加入/创立 DAO {#join-start-a-dao}

加入 DAO {#join-a-dao}

- 以太坊社区中的 DAO
- DAOHaus 的 DAO 列表

启动 DAO {#start-a-dao}

- 使用 DAOHaus 创立 DAO
- 创立 Aragon 支持的 DAO
- 启动 colony
- 使用 DAOstack 创立 DAO

延伸阅读 {#further-reading}

DAO 相关文章 {#dao-articles}

- What's a DAO?（什么是 DAO？） – Aragon
- House of DAOs（DAO 之家） – Metagame
- What is a DAO and what is it for?（什么是 DAO 及其目的？） – DAOhaus
- How to Start a DAO-Powered Digital Community（如何启动 DAO 提供支持的数字社区） – DAOhaus
- What is a DAO?（什么是 DAO？） – Coinmarketcap

相关视频 {#videos}

- 什么是加密货币中的 DAO？

---

title: 智能合约安全工具指南
description: 三种不同的测试和程序分析技术概述
author: "Trailofbits"
lang: zh
sidebar: true
tags:

- "solidity"
- "智能合约"
- "安全性"
  skill: 中级难度
  published: 2020-09-07
  source: 构建安全合约
  sourceUrl: https://github.com/crytic/building-secure-contracts/tree/master/program-analysis

---

我们将使用三种独特的测试和程序分析技术：

- **使用 Slither 进行静态分析。**通过不同的程序演示（例如控制流程图），同时对程序的所有路径进行模拟和分析。

- **使用 Echidna 进行模糊测试。** 代码是通过伪随机生成的交易来执行的， 模糊器将尝试找到一个违反某个给定的合约特性的交易序列。
- **使用 Manticore 进行符号执行。** 一种正式的验证技术，它将每个路径转换为数学公式，在此基础上可对最重要的约束加以检查。

每种技术都有优缺点，在特定情况下会很有用：

技术	工具	使用方法	速度	错误遗漏	误报
静态分析	Slither	CLI 和脚本	秒	中度	低
模糊测试	Echidna	Solidity 属性	分钟	低	无
符号执行	Manticore	Solidity 属性和脚本	小时	无*	无

* 如果在没有超时情况下遍历所有路径

Slither 可以在几秒钟内分析合同，但是静态分析可能会导致误报，不太适合复杂的检查（例如算术检查）。 通过用于按钮访问内置检测器的 API 或用于用户自定义检查的 API 运行 Slither。

Echidna 需要运行几分钟，并且只会产生真阳性的测试结果。 Echidna 会检查用户提供的用 Solidity 编写的安全属性。 由于它的随机侦测的特性，它也许会错失一些漏洞。

Manticore 进行的是“最大权重”分析。 像 Echidna 一样，Manticore 会验证用户提供的特性。 它需要更多的时间来运行，但它可以证明某个特性的有效性，并且不会报告误报。

推荐工作流程 {#suggested-workflow}

从 Slither 的内置检测器开始，确保现在没有或以后不会引入简单的漏洞。 使用 Slither 检查与继承、变量依赖关系和结构问题相关的属性。 随着代码库的增大，可以使用 Echidna 来测试状态机更复杂的特性。 再次使用 Slither 开发专门用于那些 Solidity 不提供保护的自定义检查，比如防止某个函数被覆盖。 最后，使用 Manticore 对关键安全属性进行有针对性的验证，例如算术运算。

- 使用 Slither 的 CLI 来捕捉常见问题
- 使用 Echidna 测试合约的高级安全属性
- 使用 Slither 编写自定义静态检查
- 如果需要深入保证关键安全属性，请使用 Manticore

关于单元测试的说明。 单元测试是构建高质量软件的必要条件。 然而，要找出安全漏洞，这些技术并不是最合适的。 它们通常用来测试正向代码行为（例如：代码在正常情况下按预期工作）， 但安全漏洞往往存在于开发者未考虑的边缘情况。 在我们进行的数十次智能合约安全测试研究中，单元测试覆盖率对于我们在客户端的代码中发现的安全漏洞的数量或者严重程度没有影响。

确定安全属性 {#determining-security-properties}

为了有效测试和验证代码，您必须确定需要注意的地方。 因为花费在安全上的资源是有限的，只有划分出代码库中的薄弱和高价值部分，才能优化您的工作。 威胁建模可以提供帮助。 供考虑的审核方法有：

- 快速风险评估（时间紧迫时我们的首选方法）
- 数据中心系统威胁建模指南 (aka NIST 800-154)
- Shostack 线程建模
- STRIDE / DREAD
- PASTA
- 使用断言

组件 {#components}

了解要检查的内容也有助于选择正确的工具。

通常与智能合约相关的领域比较广泛，包括：

- **状态机**。大多数合约都可以表示为状态机。 考虑检查以下几点：(1) 不能到达无效状态，(2) 如果状态有效则可以到达，以及 (3) 没有状态会限制合约。
  - Echidna 和 Manticore 是测试状态机规范的首选工具。
- **访问控制**。如果您的系统有特权用户（例如负责人、控制人等），您必须确保：(1) 每个用户只能执行授权操作，并且 (2) 没有用户可以阻止拥有更多特权的用户的操作。
  - Slither、Echidna 和 Manticore 都可以检查访问控制正确与否。 例如，Slither 可以检查是否只有白名单上的函数缺少 onlyOwner 修饰符。 Echidna 和 Manticore 可用于更复杂的访问控制，例如仅当合约达到给定状态时才授予权限。
- **算术运算**。检查算术运算的可靠性至关重要。 在所有地方使用 `SafeMath` 是防止上溢/下溢很好的做法，但您还必需考虑其他算术缺陷，包括四舍五入和限制合约的缺陷。
  - 在这里，Manticore 是最佳选择。 如果算术超出 SMT 求解器的范围，则可以使用 Echidna。
- **继承的正确性**。Solidity 合约在很大程度上依赖于多重继承。 因此，很容易出现一些诸如像缺少 `超级` 调用的遮蔽函数和曲解了 C3 线性化的顺序之类的错误。
  - Slither 是确保检测出这些问题的工具。
- **外部交互**。这里指合约之间的互动，一些外部合约不应该被信任。 例如，如果您的合约依赖于外部预言机，那么如果现有预言机有一半被泄漏，它是否仍然是安全的？
  - Manticore 和 Echidna 是测试您的合约外部互动的最佳选择。 Manticore 有一个内置机制来存留外部合约。
- **标准一致性**。 以太坊标准（例如 ERC20）记录了他们在设计上的缺陷。 请注意您正在构建的标准的局限性。
  - Slider、Echidna 和 Manticore 可以帮助您发现偏离特定标准的情况。

工具选择备忘清单 {#tool-selection-cheatsheet}

组件	工具	示例
状态机	Echidna、Manticore	
访问控制	Slither、Echidna、Manticore	Slither 练习 2、Echidna 练习 2
算术运算	Manticore、Echidna	Echidna 练习 1、Manticore 练习 1 - 3
继承的正确性	Slither	Slither 练习 1
外部交互	Manticore、Echidna	
标准一致性	Slither、Echidna、Manticore	`slither-erc`

其他领域需要根据您的目标进行检查， 但以上囊括的大致重点领域对于所有智能合约系统来说都是一个良好的开端。

我们公开的审计包含了经过验证或测试的属性实例。 请考虑阅读以下报告的 `自动测试和验证部分`，以查看实际安全属性：

- 0x
- 平衡器

title: 针对初学者的智能合约指南
description: 关于编写和部署一个基于以太坊的简单智能合约的入门教程。
author: "elanh"
tags:

- "solidity"
- "hardhat"
- "alchemy"
- "智能合约"
- "入门指南"
- "部署"
  skill: 初学者
  lang: zh
  sidebar: true
  published: 2021-03-31

如果您是区块链开发的初学者，还不知道如何开始，或者您只是想了解怎样部署智能合约并与之进行交互，这篇教程就是为您准备的。 通过使用虚拟钱包 (Metamask)、Solidity、Hardhat 和 Alchemy（如果您不理解这些名词的含义，不用担心，后续我们会进行解释），我们将演示在 Ropsten 测试网络上创建并部署一个简单的智能合约。

在本教程的第 2 部分，我们将学习在智能合约部署后如何与之进行交互，第 3 部分将学习如何在 Etherscan 上发布智能合约。

如果您有任何问题，请随时在 Alchemy Discord 中提出！

步骤 1：连接到以太坊网络 {#step-1}

有多种方法可以向以太坊链发起连接请求。 为了简单起见，我们将使用一个 Alchemy 上的免费帐户。 Alchemy 是一个区块链开发者平台和 API，允许我们与以太坊进行通信，而无需运行我们自己的节点。 平台还有用于监测和分析的开发者工具，我们将在本教程中利用这些工具来了解我们智能合约部署中的情况。 如果您还没有 Alchemy 账户，您可以在这里免费注册。

步骤 2：创建应用程序（和应用程序接口密钥） {#step-2}

创建了 Alchemy 账户后，您可以通过创建应用程序来生成应用程序接口密钥。 我们可以用它向 Ropsten 测试网络发起请求。 如果您不熟悉测试网络，请查看此页面。

1. 在 Alchemy 仪表板中，将鼠标悬停在导航栏中的“应用程序”上，单击“创建应用程序”并前往此页面。

创建应用程序 Hello world

2. 将您的应用命名为“Hello World”，提供简短的描述，选择“Staging”作为环境（用于您的应用程序簿记），选择“Ropsten”网络。

创建应用程序视图 hello world

3. 点击“创建应用程序”即可！ 您的应用程序应该会出现在下面的表格中。

步骤 3：创建一个以太坊账户（地址） {#step-3}

我们需要一个以太坊帐户来发送和接收交易。 在本教程中，我们将使用 MetaMask——浏览器中的虚拟钱包，用来管理您的以太坊账户地址。 关于交易的详细信息。

您可以在这里免费下载并创建一个 Metamsk 账户。 创建账户时，或者如果您已经有一个账户时，确保切换到右上方的“Ropsten 测试网络”（这样我们就不会用实际货币进行交易）。

metask ropsten 示例

步骤 4：从水龙头添加以太币 {#step-4}

为了将我们的智能合约部署到测试网络，我们需要一些虚拟以太币。 要获得以太币，您可以访问 Ropsten 水龙头，输入您的 Ropsten 帐户地址，然后点击“发送 Ropsten 以太币”。 由于网络原因，您接收虚拟以太币可能需要一些时间。 您应该会很快在您的 MetaMask 帐户中看到以太币！

步骤 5：查看账户余额 {#step-5}

为了核实我们的余额，可以使用 Alchemy 的创作者工具发出 eth_getBalance 请求。 这将返回我们钱包中的以太币金额。 输入您的 Metamask 帐户地址并单击“发送请求”后，您应该会看到这样的响应：

```
{ "jsonrpc": "2.0", "id": 0, "result": "0x2B5E3AF16B1880000" }
```

> **注意：**结果以 wei 为单位，而非 ETH。 Wei 是以太坊最小的面额。 将 wei 转换为 ETH 的公式为：1 eth = 10^{18} wei。 因此，如果我们将 0x2B5E3AF16B1880000 转换为十进制，我们会得到 $5*10^{18}$，即 5 ETH。
>
> 呼！ 我们的虚拟货币到账了。

步骤 6：初始化我们的项目 {#step-6}

首先，需要为我们的项目创建一个文件夹。 导航到您的命令行，然后输入：

```
mkdir hello-world
cd hello-world
```

现在我们进入了项目文件夹，我们将使用 `npm init` 来初始化项目。 如果你尚未安装 npm，请遵循这些说明（我们还需要 Node.js，所以请一并下载！）。

```
npm init
```

如何回答安装中的问题并不重要，以下提供一个回答的样例供参考：

```
package name: (hello-world)
version: (1.0.0)
description: hello world smart contract
entry point: (index.js)
test command:
git repository:
keywords:
author:
license: (ISC)
About to write to /Users/.../.../.../hello-world/package.json:

{
  "name": "hello-world",
  "version": "1.0.0",
  "description": "hello world smart contract",
  "main": "index.js",
  "scripts": {
     "test": "echo \\"Error: no test specified\\" && exit 1"
  },
```

```
  "author": "",
  "license": "ISC"
}
```

批准 package.json，我们就可以进行下一步了！

步骤 7：Hardhat{#step-7}

安全帽是一个用于编译、部署、测试和调试以太坊软件的开发环境。 在部署到实时链上之前，它帮助开发者在本地构建智能合约和去中心化应用程序。

在我们的 `Hello-world` 项目中运行：

```
npm install --save-dev hardhat
```

查看此页面，了解更多有关安装说明的详细信息。

步骤 8：创建安全帽项目 {#step-8}

在我们的项目文件夹中运行：

```
npx hardhat
```

然后应该能看到一条欢迎消息和选项，用于选择您想要做的事情。 选择“创建一个空的 hardhat.config.js”：

```
888    888                      888 888               888
888    888                      888 888               888
888    888                      888 888               888
8888888888  8888b.  888d888 .d88888 88888b.   8888b.  888888
888    888     "88b 888P"  d88" 888 888 "88b     "88b 888
888    888 .d888888 888    888  888 888  888 .d888888 888
888    888 888  888 888    Y88b 888 888  888 888  888 Y88b.
888    888 "Y888888 888     "Y88888 888  888 "Y888888  "Y888

👷 Welcome to Hardhat v2.0.11 👷‍?

What do you want to do? …
Create a sample project
❯ Create an empty hardhat.config.js
Quit
```

这将生成一个 `hardhat.config.js` 文件，我们将用以配置项目的所有设置（步骤 13）。

步骤 9：添加项目文件夹 {#step-9}

为了使我们的项目有条理，我们将创建两个新的文件夹。 在您的命令行中导航到项目的根目录，然后输入：

```
mkdir contracts
mkdir scripts
```

- `contracts/` 是保存我们的 Hello world 智能合约代码文件的位置
- `scripts/` 是我们存放脚本的位置，用于部署我们的合约和与之交互。

步骤 10：编写合约 {#step-10}

您可能要问自己，天啊，到底什么时候才能写代码？？ 答案是，就是现在，步骤 10。

在您最喜欢的编辑器中打开 hello-world 项目（我们喜欢的是 VSCode）。 智能合约是用一种叫 Solidity 的语言编写的，我们将用它来编写我们的 HelloWorld.sol 智能合约。

1. 导航到“contracts”文件夹并创建一个名为 HelloWorld.sol 的新文件
2. 下面是我们将用于本教程的来自以太坊基金会的 Hello World 智能合约样例。 复制下面的内容并粘贴到 HelloWorld.sol 文件，并一定要阅读注释以了解该合约的工作内容：

```
// Specifies the version of Solidity, using semantic versioning.
// Learn more: https://solidity.readthedocs.io/en/v0.5.10/layout-of-source-files.html#pragma
pragma solidity ^0.7.0;

// Defines a contract named `HelloWorld`.
// 一个合约是函数和数据（其状态）的集合。 Once deployed, a contract resides at a specific address on the Ethereum blockchain. Learn more: https://solidity.readthedocs.io/en/v0.5.10/structure-of-a-contract.html
contract HelloWorld {

   // Declares a state variable `message` of type `string`.
   // 状态变量是其值永久存储在合约存储中的变量。 The keyword `public` makes variables accessible from outside a contract and creates a function that other contracts or clients can call to access the value.
   string public message;

   // Similar to many class-based object-oriented languages, a constructor is a special function that is only executed upon contract creation.
   // 构造器用于初始化合约的数据。 Learn more:https://solidity.readthedocs.io/en/v0.5.10/contracts.html#constructors
   constructor(string memory initMessage) {

      // Accepts a string argument `initMessage` and sets the value into the contract's `message` storage variable).
      message = initMessage;
   }

   // A public function that accepts a string argument and updates the `message` storage variable.
   function update(string memory newMessage) public {
      message = newMessage;
   }
}
```

这是一个非常简单的智能合约，创建时存储了一条消息，而且可以通过调用 `update` 功能来更新消息。

步骤 11：将 Metamask 和 Alchemy 连接至您的项目 {#step-11}

我们创建了 Metamask 钱包、Alchemy 帐户，并且编写了一个智能合约，现在是将这三者连起来的时候了。

从虚拟钱包发送的每笔交易都需要使用您独有的私钥签名。 为了给程序提供此项许可，我们可以安全地将私钥（和 Alchemy 应用程序接口密钥）存储在一个环境文件中。

> 如需了解更多关于发送交易的信息，请查看关于使用 web3 发送交易的 本教程。

首先，在项目目录中安装 dotenv 软件包：

```
npm install dotenv --save
```

然后在我们的项目根目录中创建 `.env` 文件，并将您的 Metamask 私钥和超文本传输协议 Alchemy 应用程序接口网址加入其中。

- 遵循这些说明导出您的私钥
- 请从下方获取超文本传输协议 Alchemy 应用程序接口网址

获取 alchemy 应用程序接口密钥

复制 Alchemy 应用程序接口网址

您的 `.env` 文件应该类似：

```
API_URL = "https://eth-ropsten.alchemyapi.io/v2/your-api-key"
PRIVATE_KEY = "your-metamask-private-key"
```

为了将这些变量和代码连接，我们将在步骤 13 中调用 `hardhat.config.js` 文件中的这些变量。

不要提交 `.env`！ 请确保永远不要与任何人共享或公开您的 `.env` 文件，因为这样做会泄露您的秘密。 如果您使用版本控制，请将您的 `.env` 添加到 gitignore 文件中。

步骤 12：安装 Ethers.js {#step-12-install-ethersjs}

Ethers.js 是一个软件库，通过以更加方便用户的方法打包标准 JSON RPC 方法，从而更容易与以太坊互动，并向以太坊提出请求。

安全帽使我们更容易将插件集成到工具和扩展功能中。 我们将利用 Ethers 插件完成合约部署（Ethers.js 有非常简洁的部署方法）。

在您的项目目录中输入：

```
npm install --save-dev @nomiclabs/hardhat-ethers "ethers@^5.0.0"
```

下一步中，我们还将在 `hardhat.config.js` 中使用 ethers。

步骤 13：更新 hardhat.config.js {#step-13-update-hardhatconfigjs}

到目前为止，我们已经添加了几个依赖库和插件，现在我们需要更新 `hardhat.config.js`，以便项目使用所有这些新的组件。

按如下所示更新您的 `hardhat.config.js` 代码：

```
require('dotenv').config();

require("@nomiclabs/hardhat-ethers");
const { API_URL, PRIVATE_KEY } = process.env;

/**
```

```
* @type import('hardhat/config').HardhatUserConfig
*/
module.exports = {
   solidity: "0.7.3",
   defaultNetwork: "ropsten",
   networks: {
      hardhat: {},
      ropsten: {
         url: API_URL,
         accounts: [`0x${PRIVATE_KEY}`]
      }
   },
}
```

步骤 14：编写合约 {#step-14-compile-our-contracts}

为了确保一切正常，我们来编译一下合约。 `compile` 任务是安全帽的内部任务之一。

在命令行中运行：

```
npx hardhat compile
```

您可能会看到关于 `SPDX license identifier not provided in source file` 的警告，但不用担心，希望其它的均看起来正常！ 如果遇到问题，您可以随时在 Alchemy cord 社区中发消息询问。

步骤 15：编写部署脚本 {#step-15-write-our-deploy-scripts}

合约已经写完，配置文件也准备妥当，现在是写合约部署脚本的时候了。

转到 `scripts/` 文件夹，创建一个名为 `deplos.js` 的新文件，在其中添加以下内容：

```
async function main() {
   const HelloWorld = await ethers.getContractFactory("HelloWorld");

   // Start deployment, returning a promise that resolves to a contract object
   const hello_world = await HelloWorld.deploy("Hello World!");
   console.log("Contract deployed to address:", hello_world.address);}

main()
  .then(() => process.exit(0))
  .catch(error => {
    console.error(error);
    process.exit(1);
  });
```

安全帽在合约教程中对这些代码的每一行均提供了很好的解释，我们在这里直接引用他们的解释。

```
const HelloWorld = await ethers.getContractFactory("HelloWorld");
```

ethers.js 中的 `ContractFactory` 是用于部署新智能合约的抽象对象。因此这里的 `HelloWorld` 是我们 hello world 合约实例的工厂。 使用 `hardhat-ethers` 插件时， `ContractFactory` 和 `Contract` 实例默认与第一个签名账户相连。

```
const hello_world = await HelloWorld.deploy();
```

调用 `ContractFactory` 代码中的 `deploy()` 函数会启动合约部署，然后返回解析为 `Contract` 的 `Promise`。这个对象包括我们智能合约中每个函数的对应调用方法。

步骤 16：部署合约 {#step-16-deploy-our-contract}

我们终于准备好部署我们的智能合约啦！ 导航到命令行后运行：

```
npx hardhat run scripts/deploy.js --network ropsten
```

您会看到类似以下所示的信息：

```
Contract deployed to address: 0x6cd7d44516a20882cEa2DE9f205bF401c0d23570
```

如果我们访问 Ropsten etherscan 并搜索我们的合约地址，应该能够看到它已经成功部署。 交易将类似以下：

etherscan 合约

`From` 地址应该和您的 Metamask 账户地址相同，而 To 地址会显示“Contract Creation”，但如果我们点击进入交易，我们会在 `To` 字段看到我们的合约地址：

etherscan 交易

恭喜！ 您刚刚在以太坊区块链上部署了一个智能合约 🎉

为了更深入了解到底发生了什么，我们转到 Alchemy 仪表板中的 Explorer 选项卡。 如果您有多个 Alchemy 应用程序，请确保按应用程序筛选，然后选择“Hello World”。 hello world 浏览器

在这里您会看到一系列的 JSON-RPC 调用，都是在我们调用 `.deploy()` 函数时，Hardhat/Ethers 替我们在后端完成的。 这里有两项重要调用，一个是 `eth_sendRawTransaction`，这是实际将我们的合约写入 Ropsten 链的请求， 另一个是 `eth_getTransactionByHash`，此为读取有关我们交易给定哈希值的请求（即 交易时的典型模式）。 如需了解更多关于发送交易的信息，请查看关于使用 Web3 发送交易的本教程

这是本教程第 1 部分的全部内容， 在第 2 部分中，我们将更新我们的初始信息，从而与我们的智能合约交互；在第 3 部分，我们将在 Etherscan 上发布我们的智能合约，使得每个人都会知道如何与之交互。

想了解更多有关 Alchemy 的信息吗？ 查看我们的网站。 不想错过更新？ 在这里订阅我们的新闻通讯！ 请务必关注我们的 Twitter 并加入我们的 Discord。

title: 如何实现ERC-721市场
description: 如何在一个去中心化的分类信息板上销售代币化的物品。
author: "Alberto Cuesta Cañada"
tags:

- "智能合约"
- "erc-721"
- "solidity"
- "代币"
  skill: 中等难度
  lang: zh
  sidebar: true

published: 2020-03-19
source: Hackernoon
sourceUrl: https://hackernoon.com/how-to-implement-an-erc721-market-1e1a32j9

在这篇文章中，我要向您展示如何针对以太坊区块链来开发一个类似于 Craigslist 的 物品分类信息网站。

在 Gumtree、Ebay 和 Craigslist 等网站没有出现的时候，分类信息板大多是由软木或纸制成的。 学校走廊、报纸、路灯、店面周围都有这种分类信息板。

所有这一切都随着互联网的出现而被改变了。 能够看到某种特殊的分类信息板的人数增加了许多数量级。 随着互联网技术的发展，这些信息板背后所代表的市场也变得更加有效率，同时也更容易扩展到全球范围。 Ebay 作为一个庞大的业务，其商业模式的源头就是这些实际存在的分类信息板。

区块链技术可以再一次重塑这种商业市场的形态，接下来让我来向您展示这一点是如何发生的。

货币化 {#monetization}

基于公共区块链技术的分类信息板的商业模式和 Ebay 及普通的公司有显著的不同。

首先，这种商业模式有一个去中心化的视角。 目前现存的平台都需要维护自己的服务器。 去中心化的平台是由其用户来维护的，因此其核心平台的成本从平台所有者的角度来讲会降至零。

中心化平台有前端界面，后台网站或者可以访问该去中心化平台的接口。 它同时也会有许多其他选择。 平台所有者可以限制客户的访问并强制每个人使用他们的接口，并收取费用。 平台所有者还可以决定是否开放访问权限（这是对普通用户的一种强权!），并让任何人构建与平台的接口。 或者平台所有者可以在这些极端情况中决定任何方法。

比我更有远见的商业领袖知道如何将其货币化。 我所看到的是，这与现状不同，并且可能是有利可图的。

此外，还可以从自动化和支付的角度来看问题。 在分类信息板中，有些东西可以非常有效地代币化，并产生交易。 代币化的资产很容易在区块链中转移。 高度复杂的支付方式可以在区块链中被轻松实现。

我在这里嗅到了一个商机。 通过区块链技术，可以轻松实现一个无需运行成本的分类信息板。复杂的支付路径被包含到了每笔交易里。 我相信有人会想出一个关于如何使用它的创意。

我很高兴来建造它。 接下来，让我们来看看代码。

实现 {#implementation}

前段时间我们启动了一个开源项目，其中包含了一些商业案例的示例实现和其他相关的资源，推荐您看一看。

这个以太坊分类信息板的代码在那里，请使用并研究它。 请注意，这个项目的代码尚未经过审核，您在把资金投入这个项目之前需要进行代码的尽职审查。

这个项目的基础并不复杂。 分类信息板中的所有广告是一个包含几个字段的结构体：

```
struct Trade {
  address poster;
  uint256 item;
  uint256 price;
  bytes32 status; // Open, Executed, Cancelled
}
```

因此，字段“Poster”是指发布这个广告的人。 “Item”是指需要售卖的物品。 “Price”字段是指该物品的价格。 “Status”字段是指这笔交易的执行状态，可以是“open”、“executed”、“cancelled”。

所有的这些交易将被报存到一个映射中。 因为 Solidity 中的一切事物似乎都是映射。 也是因为这种方式比较方便。

```
mapping(uint256 => Trade) public trades;
```

使用映射是因为我们需要在广告发布前，给每个广告赋予一个 id，同时在我们操作广告前，也需要知道广告的 id。 在智能合约或前端有多种处理方法。 请询问您是否需要一些提示。

接下来的问题是我们需要处理哪些物品，以及用于支付交易的货币是什么。

对于物品，我们只是要求它们实现ERC-721的接口。这实际上只是一种在区块链中表示现实世界物品的方式，

尽管它最适合数字资产。 我们将在构造函数中定制化我们自己的 ERC721 合约，这意味着我们分类信息板中的任何资产都需要事先被代币化

对于付款，我们将做类似的事情。 大多数区块链项目定义了自己的ERC-20加密货币。 其他一些人更喜欢使用像 DAI 这样的主流技术。 在这个分类信息板的应用中，您只需要在构造函数里决定您构建的货币是什么。 很容易。

```
constructor (
address _currencyTokenAddress, address _itemTokenAddress
) public {
 currencyToken = IERC20(_currencyTokenAddress);
itemToken = IERC721(_itemTokenAddress);
tradeCounter = 0;
}
```

目前我们到达这里了。 我们有广告、用于交易的商品和用来付款的货币。 制作一个广告意味着将同一个物品放在托管中，以表明您拥有它并且您没有发布它两次，可能是在不同的分类信息板上。

下面的代码正是这样做的。 将物品放在托管中，制作广告，做一些记账操作。

```
function openTrade(uint256 _item, uint256 _price)
  public
{
  itemToken.transferFrom(msg.sender, address(this), _item);
  trades[tradeCounter] = Trade({
    poster: msg.sender,
    item: _item,
    price: _price,
    status: "Open"
  });
  tradeCounter += 1;
  emit TradeStatusChange(tradeCounter - 1, "Open");
}
```

接受交易意味着选择某个广告（交易）、支付价格、接收物品。 以下的代码用来获取交易。 检查它是否可用。 支付该物品的费用。 获取物品。 更新广告信息。

```
function executeTrade(uint256 _trade)
  public
{
  Trade memory trade = trades[_trade];
  require(trade.status == "Open", "Trade is not Open.");
  currencyToken.transferFrom(msg.sender, trade.poster, trade.price);
  itemToken.transferFrom(address(this), msg.sender, trade.item);
  trades[_trade].status = "Executed";
  emit TradeStatusChange(_trade, "Executed");
}
```

最后，我们可以选择让卖家在买家接受之前退出交易。 在某些模型中，广告会在过期前保留一段时间。 您的选择，具体取决于您的市场设计。

该代码与用于执行交易的代码非常相似，只是没有货币交换。同时该物品返回到广告的发行方。

```
function cancelTrade(uint256 _trade)
  public
{
  Trade memory trade = trades[_trade];
  require(
    msg.sender == trade.poster,
    "Trade can be cancelled only by poster."
  );
  require(trade.status == "Open", "Trade is not Open.");
  itemToken.transferFrom(address(this), trade.poster, trade.item);
  trades[_trade].status = "Cancelled";
  emit TradeStatusChange(_trade, "Cancelled");
}
```

就是这样。 您已经浏览到了该代码实现的末尾。 令人惊讶的是，一些业务概念在用代码表达时是多么紧凑，这就是其中一个例子。 您可以在我们的代码库中查看完整的合约代码。

结论 {#conclusion}

分类信息板是一种易于在互联网技术的帮助下大规模扩张的常见市场结构，也是一种容易形成少数垄断赢家的非常流行的商业模式。

分类信息板也恰好是在区块链环境中容易进行复制的一种简单工具，具有可以挑战现有的巨头的非常具体的功能。

在本文中，我尝试将分类信息板的商业业务与技术实现结合起来进行讲解。 如果您拥有合适的技能，这些知识应该可以帮助您创建愿景和实施路线图。

与往常一样，如果您想构建任何有趣的东西并希望得到一些建议，请给我留言！ 我总是很乐意为您提供帮助。

title: 如何铸造非同质化代币（非同质化代币教程系列 2/3）
description: 本教程描述了如何使用我们的智能合约和 Web3 在以太坊区块链上铸造非同质化代币。
author: "Sumi Mudgil"
tags:

- "非同质化代币"
- "ERC-721"
- "Alchemy"

- "Solidity"
- "智能合约"
  skill: 初学者
  lang: zh
  sidebar: true
  published: 2021-04-22

Beeple：6900 万美元 3LAU：1100 万美元 Grimes：600 万美元

他们都使用 Alchemy 强大的应用程序接口铸造他们的非同质化代币。 在本教程中，我们将教您如何在 < 10 分钟内做到这一点。

“铸造非同质化代币”是在区块链上发布您的 ERC-721 代币的唯一方式。 使用非同质化代币教程系列第 1 部分中讲到的智能合约，我们可以灵活运用我们的 web3 技能铸造非同质化代币。 在本教程结束时，您将能够按照您的内心（和钱包）的愿望铸造出更多非同质化代币！

我们开始吧！

第 1 步：安装 web3 {#install-web3}

如果您跟随第一个教程创建了您的非同质化代币智能合约，那么您在使用 Ethers.js 方面已经有了经验。 Web3 与 Ethers 相似，也是一个库，用于更轻松地创建对以太坊区块链的请求。 在本教程中，我们将使用 Alchemy Web3，这是一个增强版 web3 库，提供自动重试和强大的 WebSocket 支持。

在您的项目主目录中运行：

```
npm install @alch/alchemy-web3
```

第 2 步：创建一个 mint-nft.js 文件 {#create-mintnftjs}

在您的脚本目录中，创建一个 mint-nft.js 文件并添加以下代码行：

```
require("dotenv").config()
const API_URL = process.env.API_URL
const { createAlchemyWeb3 } = require("@alch/alchemy-web3")
const web3 = createAlchemyWeb3(API_URL)
```

第 3 步：获取您的合约应用程序二进制接口 {#contract-abi}

我们的合约 ABI（应用程序二进制接口）是与我们的智能合约交互的接口。 您可以在此处了解更多关于合约应用程序二进制接口的信息。 安全帽自动为我们生成应用程序二进制接口，并将其保存在 MyNFT.json 文件中。 为了使用该接口，我们需要通过在我们的 mint-nft.js 文件中添加以下代码行来解析内容：

```
const contract = require("../artifacts/contracts/MyNFT.sol/MyNFT.json")
```

如果您想查看应用程序二进制接口，您可以将其发送到您的控制台：

```
console.log(JSON.stringify(contract.abi))
```

要运行 mint-nft.js 并查看发送到控制台的应用程序二进制接口，请导航至您的终端并运行

```
node scripts/mint-nft.js
```

第 4 步：使用星际文件系统为您的非同质化代币配置元数据 {#config-meta}

如果您还记得我们第 1 部分的教程，我们的 mintNFT 智能合约函数包含有一个 tokenURI 参数，该参数应解析为描述非同质化代币元数据的 JSON 文档，它是非同质化代币的核心，赋予非同质化代币可配置的属性，如名称、描述、图像和其他属性。

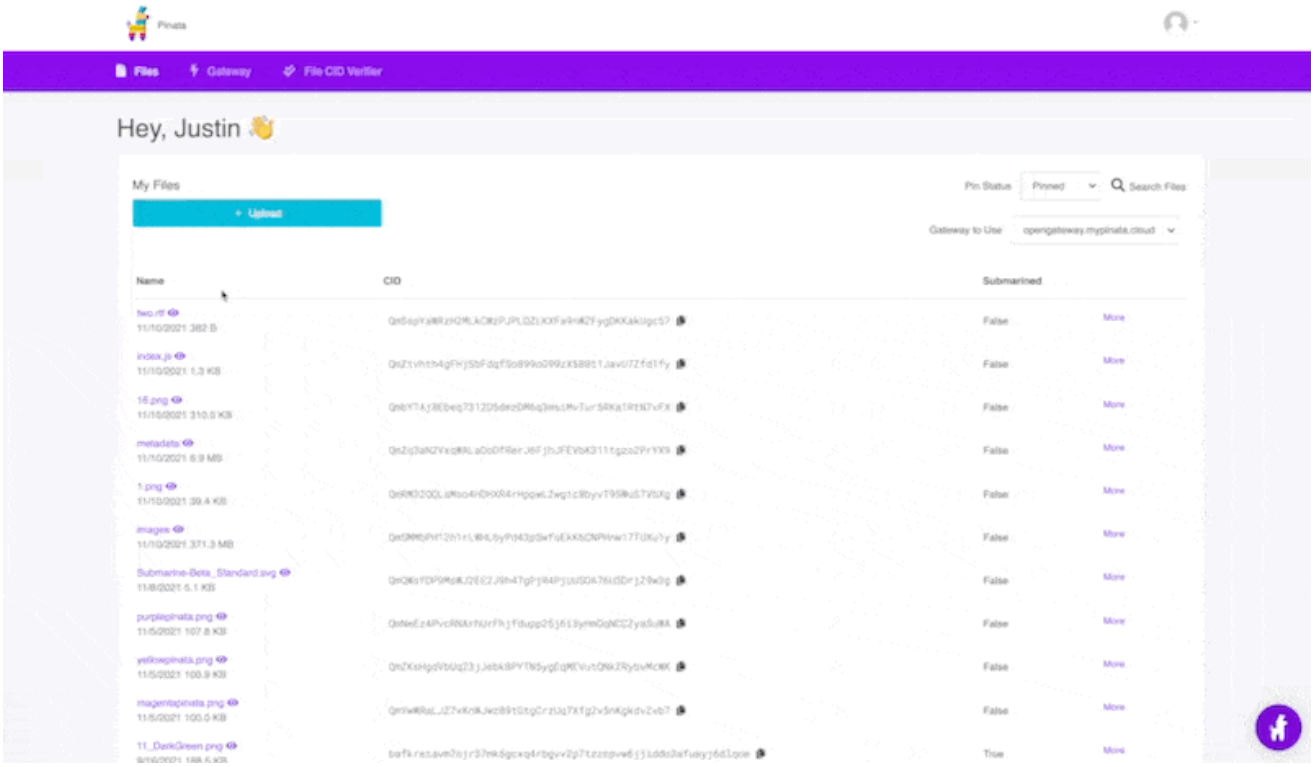
> *星际文件系统 (IPFS) 是一个分散协议和对等网络，用于在分布式文件系统中存储和共享数据。*

我们将使用 Pinata，一个方便的星际文件系统应用程序接口和工具包，用于存储我们的非同质化代币资产和元数据，以确保我们的非同质化代币真正去中心化。 如果您没有 Pinata 账户，请在此处注册一个免费账户，并完成电子邮件验证步骤。

创建账户后：

- 导航到“文件”页面，点击页面左上方蓝色的“上传”按钮。
- 将图像上传到 pinata——这将是您的非同质化代币的图像资产。 您可以随意为该资产命名。
- 上传之后，您将在“文件”页面的表格中看到文件信息。 您还会看到 CID 列。 您可以通过单击旁边的复制按钮来复制 CID。 您可以通过 `https://gateway.pinata.cloud/ipfs/<CID>` 查看您上传的信息。 例如，您可以在这里找到我们在星际文件系统上使用的图片。

为了方便更偏向视觉型的学习者理解，上述步骤总结如下：



现在，我们要再上传一份文件到 Pinata。 但在我们上传之前，需要先创建该文件！

在您的根目录中，建立一个名为 nft-metadata.json 的新文件，并添加以下 json 代码：

```
{
  "attributes": [
    {
      "trait_type": "Breed",
      "value": "Maltipoo"
```

```
    },
    {
      "trait_type": "Eye color",
      "value": "Mocha"
    }
  ],
  "description": "The world's most adorable and sensitive pup.",
  "image": "ipfs://QmWmvTJmJU3pozR9ZHFmQC2DNDwi2XJtf3QGyYiiagFSWb",
  "name": "Ramses"
}
```

可随意更改 json 中的数据。 您可以删除或添加到属性部分。 最重要的是，确保图像字段指向您的星际文件系统图像的位置——否则，您的非同质化代币将包含一张（非常可爱！）的狗狗照片。

编辑完 json 文件后，保存并上传到 Pinata，步骤与上传图像相同。

如何将您的 nft-metadata.json 上传至 Pinata

第 5 步：创建您的合约实例 {#instance-contract}

现在，为了与我们的合约进行交互，我们需要在代码中创建一个实例。 要做到这一点，我们需要合约地址，可以从部署或 Etherscan 中查找您用来部署合约的地址。

在 Etherscan 上查看您的合约地址

在上例中，我们的合约地址是 0x81c587EB0fE773404c42c1d2666b5f557C470eED。

接下来我们将使用 web3 的合约方法，用应用程序二进制接口和地址创建我们的合约。 在您的 mint-nft.js 文件中添加以下内容：

```
const contractAddress = "0x81c587EB0fE773404c42c1d2666b5f557C470eED"

const nftContract = new web3.eth.Contract(contract.abi, contractAddress)
```

第 6 步：更新 .env 文件 {#update-env}

现在，为了创建并发送交互到以太坊链，我们将使用您的以太坊公共账户地址来获得账户随机数（解释如下）。

将您的公钥添加到你的 .env 文件中——如果您完成了教程的第 1 部分，我们的 .env 文件现在应该如下所示：

```
API_URL = "https://eth-ropsten.alchemyapi.io/v2/your-api-key"
PRIVATE_KEY = "your-private-account-address"
PUBLIC_KEY = "your-public-account-address"
```

第 7 步：创建您的交易 {#create-txn}

首先，让我们定义一个名为 `mintNFT(tokenData)` 的函数并通过执行以下操作创建我们的交易：

1. 获取 *PRIVATE_KEY* 和 *PUBLIC_KEY*，来源为 `.env` 文件。
2. 接下来，我们需要弄清楚账户的随机数。 随机数规范用于跟踪从您的地址发送的交易数量——我们这样做是出于安全目的，以防止重放攻击。 要获得从您的地址发送的交易数量，我们要用到 getTransactionCount。
3. 最后，我们将使用以下信息设置我们的交易：

- `'from': PUBLIC_KEY`——我们的交易源于我们的公共地址
- `'to': contractAddress`——我们希望与之交互并发送交易的合约

- `'nonce': nonce`——账户随机数和从我们的地址发送的交易数量
- `'gas': estimatedGas`——预估完成交易所需的燃料
- `'data': nftContract.methods.mintNFT(PUBLIC_KEY, md).encodeABI()`——我们希望在这笔交易中进行的计算——铸造一个非同质化代币

您的 mint-nft.js 文件现在看起来应该像这样：

```
require('dotenv').config();
const API_URL = process.env.API_URL;
const PUBLIC_KEY = process.env.PUBLIC_KEY;
const PRIVATE_KEY = process.env.PRIVATE_KEY;

const { createAlchemyWeb3 } = require("@alch/alchemy-web3");
const web3 = createAlchemyWeb3(API_URL);

const contract = require("../artifacts/contracts/MyNFT.sol/MyNFT.json");
const contractAddress = "0x81c587EB0fE773404c42c1d2666b5f557C470eED";
const nftContract = new web3.eth.Contract(contract.abi, contractAddress);

async function mintNFT(tokenURI) {
  const nonce = await web3.eth.getTransactionCount(PUBLIC_KEY, 'latest'); //get latest
nonce

//the transaction
  const tx = {
    'from': PUBLIC_KEY,
    'to': contractAddress,
    'nonce': nonce,
    'gas': 500000,
    'data': nftContract.methods.mintNFT(PUBLIC_KEY, tokenURI).encodeABI()
  };
}
```

第 8 步：签署交易 {#sign-txn}

现在我们已经创建了我们的交易，我们需要签署它，以便将其发送出去。 在这里我们将使用到我们的私钥。

`web3.eth.sendSignedTransaction` 会给我们提供交易的哈希值，我们可以用它来确保我们的交易被开采出来，没有被网络丢弃。 您会注意到在交易签署部分，我们已经增加了一些错误检查，以便我们知晓交易是否成功通过。

```
require("dotenv").config()
const API_URL = process.env.API_URL
const PUBLIC_KEY = process.env.PUBLIC_KEY
const PRIVATE_KEY = process.env.PRIVATE_KEY

const { createAlchemyWeb3 } = require("@alch/alchemy-web3")
const web3 = createAlchemyWeb3(API_URL)

const contract = require("../artifacts/contracts/MyNFT.sol/MyNFT.json")
const contractAddress = "0x81c587EB0fE773404c42c1d2666b5f557C470eED"
const nftContract = new web3.eth.Contract(contract.abi, contractAddress)

async function mintNFT(tokenURI) {
```

```
  const nonce = await web3.eth.getTransactionCount(PUBLIC_KEY, "latest") //get latest
nonce

  //the transaction
  const tx = {
    from: PUBLIC_KEY,
    to: contractAddress,
    nonce: nonce,
    gas: 500000,
    data: nftContract.methods.mintNFT(PUBLIC_KEY, tokenURI).encodeABI(),
  }

  const signPromise = web3.eth.accounts.signTransaction(tx, PRIVATE_KEY)
  signPromise
    .then((signedTx) => {
      web3.eth.sendSignedTransaction(
        signedTx.rawTransaction,
        function (err, hash) {
          if (!err) {
            console.log(
              "The hash of your transaction is: ",
              hash,
              "\nCheck Alchemy's Mempool to view the status of your transaction!"
            )
          } else {
            console.log(
              "Something went wrong when submitting your transaction:",
              err
            )
          }
        }
      )
    })
    .catch((err) => {
      console.log(" Promise failed:", err)
    })
}
```

第 9 步：调用 mintNFT 并运行节点 mint-nft.js {#call-mintnft-fn}

还记得您上传到 Pinata 的 metadata.json 吗？ 从 Pinata 获取其哈希代码，并将以下内容作为参数传给函数 mintNFT `https://gateway.pinata.cloud/ipfs/<metadata-hash-code>`

如何获取哈希代码：

如何在 Pinata 上获得您的非同质化代币元数据哈希代码*How to get your nft metadata hashcode on Pinata*

> 仔细检查您复制的哈希代码是否链接到您的 **metadata.json**，方法是将 `https://gateway.pinata.cloud/ipfs/<metadata-hash-code>` 加载到独立窗口。 页面看起来应该类似于下面的截图：

您的页面应该显示 json 元数据*Your page should display the json metadata*

总之，您的代码应该看起来像这样：

```
require("dotenv").config()
const API_URL = process.env.API_URL
const PUBLIC_KEY = process.env.PUBLIC_KEY
const PRIVATE_KEY = process.env.PRIVATE_KEY

const { createAlchemyWeb3 } = require("@alch/alchemy-web3")
const web3 = createAlchemyWeb3(API_URL)

const contract = require("../artifacts/contracts/MyNFT.sol/MyNFT.json")
const contractAddress = "0x81c587EB0fE773404c42c1d2666b5f557C470eED"
const nftContract = new web3.eth.Contract(contract.abi, contractAddress)

async function mintNFT(tokenURI) {
  const nonce = await web3.eth.getTransactionCount(PUBLIC_KEY, "latest") //get latest
nonce

  //the transaction
  const tx = {
    from: PUBLIC_KEY,
    to: contractAddress,
    nonce: nonce,
    gas: 500000,
    data: nftContract.methods.mintNFT(PUBLIC_KEY, tokenURI).encodeABI(),
  }

  const signPromise = web3.eth.accounts.signTransaction(tx, PRIVATE_KEY)
  signPromise
    .then((signedTx) => {
      web3.eth.sendSignedTransaction(
        signedTx.rawTransaction,
        function (err, hash) {
          if (!err) {
            console.log(
              "The hash of your transaction is: ",
              hash,
              "\nCheck Alchemy's Mempool to view the status of your transaction!"
            )
          } else {
            console.log(
              "Something went wrong when submitting your transaction:",
              err
            )
          }
        }
      )
    })
    .catch((err) => {
      console.log("Promise failed:", err)
    })
}

mintNFT("ipfs://QmYueiuRNmL4MiA2GwtVMm6ZagknXnSpQnB3z2gWbz36hP")
```

现在，运行 `node scripts/mint-nft.js` 来部署您的非同质化代币。 几秒钟后，您应该在终端中看到这样的响应：

```
The hash of your transaction is:
0x10e5062309de0cd0be7edc92e8dbab191aa2791111c44274483fa766039e0e00

Check Alchemy's Mempool to view the status of your transaction!
```

接下来，访问您的 Alchemy 内存池，查看您的交易状态（待定、已开采还是被网络放弃）。 如果您的交易被放弃，可以通过 Ropsten Etherscan 搜索您的交易哈希值查看具体情况。

在 Etherscan 上查看您的非同质化代币交易哈希值*View your NFT transaction hash on Etherscan*

就是这样！ 您现在已经在以太坊区块链上部署和铸造了一个非同质化代币

使用 mint-nft.js，您可以随心所欲地（配合钱包）铸造尽可能多的非同质化代币！ 只要确保输入一个新的 tokenURI 来描述非同质化代币的元数据（否则，您将最终生成一堆相同但具有不同 ID 的 tokenURI）。

大概，您会想要展示一下您钱包中的非同质化代币——所以请务必查看第 3 部分：如何在您的钱包中查看您的非同质化代币！

title: 如何使用 Echidna 测试智能合约
description: 如何使用 Echidna 自动测试智能合约
author: "Trailofbits"
lang: zh
sidebar: true
tags:

- "solidity"
- "智能合约"
- "安全性"
- "测试"
- "模糊测试"

  skill: 高级
  published: 2020-04-10
  source: 构建安全的合约
  sourceUrl: https://github.com/crytic/building-secure-contracts/tree/master/program-analysis/echidna

---

安装 {#installation}

Echidna 可以通过 docker 或使用预编译的二进制程序安装。

通过 docker 安装 Echidna {#echidna-through-docker}

```
docker pull trailofbits/eth-security-toolbox
docker run -it -v "$PWD":/home/training trailofbits/eth-security-toolbox
```

最后一个命令在有权访问当前目录的 docker 中运行 eth-security-toolbox。 您可以从主机更改文件，并在 docker 中对文件运行工具

在 docker 中，运行：

```
solc-select 0.5.11
cd /home/training
```

通过二进制程序安装 {#binary}

https://github.com/crytic/echidna/releases/tag/1.4.0.0

基于属性的模糊测试简介 {#introduction-to-property-based-fuzzing}

Echidna 是一个模糊测试工具，我们在之前的博客中描述过（1、2、3）。

模糊测试 {#fuzzing}

模糊测试是一个在安全技术领域广为人知的技术。 它依靠生成或多或少数量的随机输入值来测试程序中的错误。 传统软件中的模糊测试工具（例如 AFL 或 LibFuzzer）是发现错误的有效工具。

除了完全随机生成输入值外，还有很多其他的技巧和策略来生成足够好的输入，包括：

- 从每次执行中获取反馈，并使用反馈来指导输入的生成。 例如，如果新生成的输入导致发现一条新的路径，那么，生成接近该路径的新输入是有意义的。
- 根据结构限制生成输入。 例如，如果您的输入包含一个带有校验和的报文头，则让模糊测试工具生成能够验证校验和的输入将会是很有意义的。
- 使用已知输入生成新输入：如果您有权访问一个有效输入的大型数据集， 则模糊测试工具可以从中生成新的输入，而不是从头开始生成。 这些通常称为 *种子*。

基于属性的模糊测试 {#property-based-fuzzing}

Echidna 属于一个特定的模糊测试工具系列：基于属性的模糊测试，很大程度上受到了 QuickCheck 的启发。 与尝试查找崩溃的经典模糊测试工具不同，Echedna 会视图去改变用户定义的不变量。

在智能合约中，不变量是 Solidity 函数，可以表示合约可能达到的任何错误或无效状态，包括：

- 不正确的访问控制：攻击者变成了合约的所有者。
- 不正确的状态机：合约暂停时代币仍然可以传输。
- 不正确的算法：用户可以余额不足的情况下获得无限的免费代币。

使用 Echidna 测试属性 {#testing-a-property-with-echidna}

我们来看看如何使用 Echidna 测试智能合约。 目标是以下智能合约 `token.sol`：

```
contract Token{
  mapping(address => uint) public balances;
  function airdrop() public{
      balances[msg.sender] = 1000;
  }
  function consume() public{
      require(balances[msg.sender]>0);
      balances[msg.sender] -= 1;
  }
  function backdoor() public{
      balances[msg.sender] += 1;
  }
}
```

我们假定此代币必须具有以下属性：

- 任何人最多可以持有 1000 代币

- 代币不能转移（它不是 ERC20 代币）

写入属性 {#write-a-property}

Echidna 的属性是 Solidity 函数。 一个属性必须：

- 没有任何参数
- 如果成功就返回 `true`
- 其名字以 `echidna` 开头

Echidna 将：

- 自动生成任意交易来测试属性。
- 报告导致属性返回 `false` 或引发错误的任何交易。
- 调用属性时丢弃负面影响（即，如果属性改变了一个状态变量，则在测试后会被丢弃）

以下属性会检查调用者持有的代币是否不超过 1000 个：

```
function echidna_balance_under_1000() public view returns(bool){
      return balances[msg.sender] <= 1000;
}
```

使用继承将合约与属性分开：

```
contract TestToken is Token{
      function echidna_balance_under_1000() public view returns(bool){
            return balances[msg.sender] <= 1000;
      }
  }
```

`token.sol` 实现了这个属性并继承了代币。

初始化合约 {#initiate-a-contract}

Echidna 需要一个无参 构造函数。 如果您的合约需要特定的初始化，则需要相应地改变构造函数。

Echidna 中有一些特定的地址：

- `0x00a329c0648769A73afAc7F9381E08FB43dBEA72` 用于调用构造函数。
- `0x10000`、`0x2000` 和 `0x00a329c0648769a73afAC7F9381e08fb43DBEA70` 用于随机调用其他函数。

在当前的示例中，我们不需要进行任何特定的初始化，因为我们的构造函数是空的。

运行 Echidna {#run-echidna}

用此命令启动 Echidna：

```
$ echidna-test contract.sol
```

如果 contract.sol 包含多个合约，您可以指定目标合约：

```
$ echidna-test contract.sol --contract MyContract
```

总结：测试属性 {#summary-testing-a-property}

下面总结了在我们的示例中运行 Echidna 的情况：

```
contract TestToken is Token{
    constructor() public {}
        function echidna_balance_under_1000() public view returns(bool){
          return balances[msg.sender] <= 1000;
        }
  }
```

```
$ echidna-test testtoken.sol --contract TestToken
...

echidna_balance_under_1000: failed!💥
  Call sequence, shrinking (1205/5000):
    airdrop()
    backdoor()

...
```

如果 `backdoor` 被调用，Echidna 会发现与该属性发生冲突。

过滤在模糊测试期间要调用的函数 {#filtering-functions-to-call-during-a-fuzzing-campaign}

我们来了解如何过滤要进行模糊测试的函数。 目标是以下智能合约：

```
contract C {
  bool state1 = false;
  bool state2 = false;
  bool state3 = false;
  bool state4 = false;

  function f(uint x) public {
    require(x == 12);
    state1 = true;
  }

  function g(uint x) public {
    require(state1);
    require(x == 8);
    state2 = true;
  }

  function h(uint x) public {
    require(state2);
    require(x == 42);
    state3 = true;
  }

 function i() public {
    require(state3);
```

```
    state4 = true;
  }

  function reset1() public {
    state1 = false;
    state2 = false;
    state3 = false;
    return;
  }

  function reset2() public {
    state1 = false;
    state2 = false;
    state3 = false;
    return;
  }

  function echidna_state4() public returns (bool) {
    return (!state4);
  }
}
```

这个小例子迫使 Echidna 找到特定的交易序列来改变一个状态变量。 这对一个模糊测试工具来说很困难（建议使用符号执行工具，比如 Manticore）。 我们可以运行 Echidna 来验证：

```
$ echidna-test multi.sol
...
echidna_state4: passed! 🥕
Seed: -3684648582249875403
```

过滤函数 {#filtering-functions}

Echidna 很难找到测试此合约的正确序列，因为两个重置函数（`reset1` 和 `reset2`）会将所有状态变量设置为 `false`。 但是，我们可以使用特殊的 Echidna 功能将重置函数列入黑名单，或者仅将 `f`、`g`、`h` 和 `i` 列入白名单。

要将函数列入黑名单，我们可以使用下面这个配置文件：

```
filterBlacklist: true
filterFunctions: ["reset1", "reset2"]
```

过滤函数的另一个方法是列出白名单里的函数。 为此，我们可以使用下面这个配置文件：

```
filterBlacklist: false
filterFunctions: ["f", "g", "h", "i"]
```

- 默认情况下，`filterBlacklist` 为 `true`。
- 只能通过名字进行过滤（不带参数）。 如果您有 `f()` 和 `f(uint256)` 两个函数，则过滤器 `"f"` 会匹配出这两个函数。

运行 Echidna {#run-echidna-1}

使用配置文件 `blacklist.yaml` 运行 Echidna：

```
$ echidna-test multi.sol --config blacklist.yaml
...
echidna_state4: failed!💥
  Call sequence:
    f(12)
    g(8)
    h(42)
    i()
```

Echidna 几乎立刻就可以找到伪造属性的交易序列。

总结：过滤函数 {#summary-filtering-functions}

在模糊测试期间，Echidna 可以将要调用的黑名单或白名单函数列入黑名单或白名单，方法是：

```
filterBlacklist: true
filterFunctions: ["f1", "f2", "f3"]
```

```
$ echidna-test contract.sol --config config.yaml
...
```

根据 `filterBlacklist` 布尔值的不同，Echidna 开始进行模糊测试时，要么将 `f1`、`f2` 和 `f3` 列入黑名单，要么只调用它们。

如何使用 Echidna 测试 Solidity 的断言 {#how-to-test-soliditys-assert-with-echidna}

在这个简短的教程中，我们将演示如何使用 Echidna 测试合约中的断言检查。 假设我们有这样一个合约：

```
contract Incrementor {
  uint private counter = 2**200;

  function inc(uint val) public returns (uint){
    uint tmp = counter;
    counter += val;
    // tmp <= counter
    return (counter - tmp);
  }
}
```

写一个断言： {#write-an-assertion}

我们要确保 `tmp` 返回其差值之后，小于或等于 `counter`。 我们可以写一个 Echidna 属性，但我们需要将 `tmp` 值保存某处。 相反，我们可以使用如下断言：

```
contract Incrementor {
  uint private counter = 2**200;

  function inc(uint val) public returns (uint){
    uint tmp = counter;
    counter += val;
    assert (tmp <= counter);
    return (counter - tmp);
  }
}
```

运行 Echidna {#run-echidna-2}

要启用断言失败测试，请创建 Echidna 配置文件 `config.yaml`：

```
checkAsserts: true
```

当我们在 Echidna 上运行这个合约时，我们会获得预期的结果：

```
$ echidna-test assert.sol --config config.yaml
Analyzing contract: assert.sol:Incrementor
assertion in inc: failed!💥
  Call sequence, shrinking (2596/5000):
    inc(21711016731996786641919559689128982722488122124807605757398297001483711807488)
    inc(72370055773322622139731865630429942408293740416025352524660990004945706024 96)
    inc(86844066927987146567678238756515930889952488499230423029593188005934847229952)

Seed: 1806480648350826486
```

正如您所见，Echidna 在 `inc` 函数中报告了一些断言失败。 每个函数可以添加多个断言，但 Echidna 无法判断哪个断言失败了。

使用断言的时机和方式 {#when-and-how-use-assertions}

断言可以用作显示属性的替代项，特别是如果要检查的条件与某些操作 `f` 的正确使用直接相关。 在某些代码之后添加断言，将强制在代码执行后立即执行检查：

```
function f(..) public {
    // some complex code
    ...
    assert (condition);
    ...
}
```

相反， 使用显式的 Echidna 属性将随机执行交易，无法轻松地强制要求何时对其进行检查。 但还是可以做以下变通的：

```
function echidna_assert_after_f() public returns (bool) {
    f(..);
    return(condition);
}
```

但是，也存在一些问题：

- 如果 `f` 被声明为 `internal` 或 `external` 则失败.
- 不清楚应该使用哪些参数来调用 `f`。
- 如果 `f` 恢复，属性将会失败。

一般来说，我们建议遵循 [John Regehr 关于如何使用断言的建议](#)：

- 在进行断言检查时不要强制任何负面影响。 例如： `assert(ChangeStateAndReturn() == 1)`
- 不要断言明显的语句。 例如， 在 `assert(var >= 0)` 中， `var` 被声明为 `uint`。

最后，请**不要使用** `require` 代替 `assert`，因为 Echidna 将无法检测到它（但合约仍将恢复）。

总结：断言检查 {#summary-assertion-checking}

下面总结了我们示例中 Echidna 的运行情况：

```
contract Incrementor {
  uint private counter = 2**200;

  function inc(uint val) public returns (uint){
    uint tmp = counter;
    counter += val;
    assert (tmp <= counter);
    return (counter - tmp);
  }
}
```

```
$ echidna-test assert.sol --config config.yaml
Analyzing contract: assert.sol:Incrementor
assertion in inc: failed!💥
  Call sequence, shrinking (2596/5000):
    inc(21711016731996786641919559689128982722488122124807605757398297001483711807488)
    inc(72370055773322622139731865630429942408293740416025352524660990004945706024 96)
    inc(86844066927987146567678238756515930889952488499230423029593188005934847229952)

Seed: 1806480648350826486
```

Echidna 发现，如果使用大参数多次调用此函数， `inc` 中的断言可能会失败。

收集和修改 Echidna 预料库 {#collecting-and-modifying-an-echidna-corpus}

我们来了解如何用 Echidna 收集和使用交易语料库。 目标是以下智能合约 `magic.sol`：

```
contract C {
  bool value_found = false;
  function magic(uint magic_1, uint magic_2, uint magic_3, uint magic_4) public {
    require(magic_1 == 42);
    require(magic_2 == 129);
    require(magic_3 == magic_4+333);
    value_found = true;
    return;
  }
```

```
  function echidna_magic_values() public returns (bool) {
    return !value_found;
  }

}
```

这个小例子迫使 Echidna 找到一系列交易来改变一个状态变量。 这对一个模糊测试工具来说很困难（建议使用一个符号执行工具，比如 Manticore）。 我们可以运行 Echidna 对此进行验证：

```
$ echidna-test magic.sol
...

echidna_magic_values: passed! 🎉

Seed: 2221503356319272685
```

然而，当我们运行该模糊测试活动时，我们仍然可以使用 Echidna 来收集语料库。

收集语料库 {#collecting-a-corpus}

为了启用语料库的收集，请创建一个语料目录：

```
$ mkdir corpus-magic
```

和一个 Echidna 配置文件 `config.yaml`：

```
coverage: true
corpusDir: "corpus-magic"
```

现在，我们可以运行工具并检查收集到的语料库：

```
$ echidna-test magic.sol --config config.yaml
```

Echidna 仍然找不到正确的 magic 值，但我们可以看一看它收集到的语料库。 例如，其中一个文件是：

```
[
  {
    "_gas'": "0xffffffff",
    "_delay": ["0x13647", "0xccf6"],
    "_src": "00a329c0648769a73afac7f9381e08fb43dbea70",
    "_dst": "00a329c0648769a73afac7f9381e08fb43dbea72",
    "_value": "0x0",
    "_call": {
      "tag": "SolCall",
      "contents": [
        "magic",
        [
          {
            "contents": [
              256,
```

```
 "93723985220345906694500679277863898678726808528711107336895287282192244575836"
            ],
            "tag": "AbiUInt"
          },
          {
            "contents": [256, "334"],
            "tag": "AbiUInt"
          },
          {
            "contents": [
              256,

 "68093943901352437066264791224433559271778087297543421781073458233697135179558"
            ],
            "tag": "AbiUInt"
          },
          {
            "tag": "AbiUInt",
            "contents": [256, "332"]
          }
        ]
      ]
    },
    "_gasprice'": "0xa904461f1"
  }
]
```

显然，此输入不会触发我们属性中的故障。 但是，在下一步，我们将看到如何对此进行修改。

为语料库生成种子 {#seeding-a-corpus}

Echidna 需要一些帮助才能处理 `magic` 函数。 我们将复制和修改输入以使用其合适的参数：

```
$ cp corpus/2712688662897926208.txt corpus/new.txt
```

我们将修改 `new.txt` 来调用 `magic(42,129,333,0)`。 现在，我们可以重新运行 Echidna：

```
$ echidna-test magic.sol --config config.yaml
...
echidna_magic_values: failed!💥
  Call sequence:
    magic(42,129,333,0)


Unique instructions: 142
Unique codehashes: 1
Seed: -7293830866560616537
```

这一次，我们立刻就能发现该属性发生了冲突。

查找消耗大量 gas 的交易 {#finding-transactions-with-high-gas-consumption}

我们来了解如何用 Echidna 查找有 gas 消耗的交易。 目标是以下智能合约：

```
contract C {
  uint state;

  function expensive(uint8 times) internal {
    for(uint8 i=0; i < times; i++)
      state = state + i;
  }

  function f(uint x, uint y, uint8 times) public {
    if (x == 42 && y == 123)
      expensive(times);
    else
      state = 0;
  }

  function echidna_test() public returns (bool) {
    return true;
  }

}
```

这里的 `expensive` 可能会消耗大量 gas。

目前，Echidna 总是需要一个属性来测试：这里 `echidna_test` 始终返回 `true`。 我们可以运行 Echidna 对此进行验证：

```
$ echidna-test gas.sol
...
echidna_test: passed! 🎉

Seed: 2320549945714142710
```

测量 gas 消耗 {#measuring-gas-consumption}

要使用 Echidna 计算 gas 消耗，请创建一个配置文件 `config.yaml`：

```
estimateGas: true
```

在这个例子中，我们还将减少交易序列的大小，以使结果更易于理解：

```
seqLen: 2
estimateGas: true
```

Run Echidna {#run-echidna-3}

创建好配置文件之后，我们就可以这样运行 Echidna：

```
$ echidna-test gas.sol --config config.yaml
...
echidna_test: passed! 🎉

f used a maximum of 1333608 gas
  Call sequence:
    f(42,123,249) Gas price: 0x10d5733f0a Time delay: 0x495e5 Block delay: 0x88b2

Unique instructions: 157
Unique codehashes: 1
Seed: -325611019680165325
```

- 显示的 gas 是由 HEVM 提供的估值。

过滤掉 gas 消耗减少的调用 {#filtering-out-gas-reducing-calls}

以上关于**在模糊测试活动期间过滤要调用的函数**的教程展示了如何从测试中删除一些函数。
这对于获得准确的 gas 消耗至关重要。 请考虑下面的示例：

```
contract C {
  address [] addrs;
  function push(address a) public {
    addrs.push(a);
  }
  function pop() public {
    addrs.pop();
  }
  function clear() public{
    addrs.length = 0;
  }
  function check() public{
    for(uint256 i = 0; i < addrs.length; i++)
      for(uint256 j = i+1; j < addrs.length; j++)
        if (addrs[i] == addrs[j])
          addrs[j] = address(0x0);
  }
  function echidna_test() public returns (bool) {
      return true;
  }
}
```

如果 Echidna 可以调用所有函数，它将无法轻松找到消耗大量 gas 的交易：

```
$ echidna-test pushpop.sol --config config.yaml
...
pop used a maximum of 10746 gas
...
check used a maximum of 23730 gas
...
clear used a maximum of 35916 gas
...
push used a maximum of 40839 gas
```

这是因为成本取决于 `addrers` 的大小，而随机调用往往会使数组几乎为空。 将 `pop` 和 `clear` 加入黑名单却给我们带来了更好的结果:

```
filterBlacklist: true
filterFunctions: ["pop", "clear"]
```

```
$ echidna-test pushpop.sol --config config.yaml
...
push used a maximum of 40839 gas
...
check used a maximum of 1484472 gas
```

总结：查找消耗大量 gas 的交易 {#summary-finding-transactions-with-high-gas-consumption}

Echidna 可以使用 `estimateGas` 配置选项找到消耗大量 gas 的交易：

```
estimateGas: true
```

```
$ echidna-test contract.sol --config config.yaml
...
```

模糊测试结束后，Echidna 将报告每个函数中消耗 gas 最多的交易。

title: 如何使用Manticore来发现智能合约漏洞
description: 如何使用Manticore来自动发现智能合约漏洞
author: Trailofbits
lang: zh
sidebar: true
tags:

- "solidity"
- "智能合约"
- "安全性"
- "测试"
- "形式化验证"
  skill: 高级
  published: 2020-01-13
  source: 构建安全合约

sourceUrl: https://github.com/crytic/building-secure-contracts/tree/master/program-analysis/manticore

---

本教程的目的是展示如何使用 Manticore 自动发现智能合约中的漏洞。

安装 {#installation}

Manticore 需要使用 python 3.6。 它可以通过 pip 安装或使用 docker。

使用 docker 的 Manticore {#manticore-through-docker}

```
docker pull trailofbits/eth-security-toolbox
docker run -it -v "$PWD":/home/training trailofbits/eth-security-toolbox
```

最后一个命令在一个可访问您当前目录的 docker 中运行 eth-security 工具箱。 您可以更改您主机中的文件，并从 docker 中运行文件上的工具

在 docker 中，运行：

```
cd /home/trufflecon/
```

使用 pip 的 Manticore {#manticore-through-pip}

```
pip3 install --user manticore
```

建议采用 solc 0.5.11。

运行脚本 {#running-a-script}

使用 python 3 运行一个 python 脚本：

```
python3 script.py
```

动态符号化执行简介 {#introduction-to-dynamic-symbolic-execution}

Nutshell 中的动态符号化执行 {#dynamic-symbolic-execution-in-a-nutshell}

动态符号化执行(DSE)是一种程序分析技术，用于探究具有高度语义意识的状态空间。 这项技术是基于 "程序路径"的发现 ，以一种称为 `path predicates` 的数学公式表示。 就概念来说，这种技术对路径预测的操作分为两步：

1. 它们是利用对程序输入的约束来构建的。
2. 它们被用来生成程序输入，从而导致相关路径的执行。

这种方法不会产生误报，因为所有被识别的程序状态都可以在具体执行过程中被触发。 例如，如果分析发现了一个整数溢出，就可以保证它是可重现的

路径预测示例 {#path-predicate-example}

为了了解 DSE 如何工作，请考虑以下示例：

```
function f(uint a){

  if (a == 65) {
      // A bug is present
  }

}
```

因为 f()包含两个路径，DSE 将构建两个不同的路径预测：

- 路径 1： `a == 65`
- 路径 2: `Not (a == 65)`

每条路径预测都是一个数学公式，可以传递给给所谓的SMT 求解器，它将尝试解方程式。 对于`路径1`，求解器会说，可以用`a=65`探索路径。 对于`路径2`，求解器可以给`a`指定一个 65 以外的任何值，例如`a=0`。

验证属性 {#verifying-properties}

Manticore 允许完全控制每个路径的所有执行情况。 因此，它允许您在几乎任何东西上添加任意限制。 这种控制允许在合约上创建财产。

请考虑下面的示例：

```
function unsafe_add(uint a, uint b) returns(uint c){
  c = a + b; // no overflow protection
  return c;
}
```

函数中只有一个要探索的路径：

- 路径 1: `c = a + b`

使用 Manticore，您可以对溢出进行检查，并对路径预测加以限制：

- `c = a + b AND (c < a OR c < b)`

如果有可能找到一个`a`和`b`的估值，对于这个估值，上面的路径预测是可行的，这意味着您发现了一个溢出。 例如，求解器可以生成输入`a = 10 , b = MAXUINT256`。

如果您考虑一个固定版本：

```
function safe_add(uint a, uint b) returns(uint c){
  c = a + b;
  require(c>=a);
  require(c>=b);
  return c;
}
```

与溢出检查相关的公式是：

- `c = a + b AND (c >= a) AND (c=>b) AND (c < a OR c < b)`

这个公式无法得以解决；在其他领域，这是一个在`safe_add`中，`c`总会增加的**证明**。

因此，DSE 是一个强大的工具，可以验证您代码的任意限制。

在 Manticore 下运行 {#running-under-manticore}

我们将看到如何探索使用 Manticore API 的智能合约。 目标是以下智能合约`example.sol`：

```
pragma solidity >=0.4.24 <0.6.0;

contract Simple {
    function f(uint a) payable public{
        if (a == 65) {
            revert();
        }
    }
}
```

运行一个独立的探索方式 {#run-a-standalone-exploration}

您可以通过以下命令直接在智能合约上运行 Manticore（`project`可以是一个 Solidity 文件，或者是项目目录）：

```
$ manticore project
```

您将会获得像这个一样的测试案例输出（顺序可能有变）：

```
...
... m.c.manticore:INFO: Generated testcase No. 0 - STOP
... m.c.manticore:INFO: Generated testcase No. 1 - REVERT
... m.c.manticore:INFO: Generated testcase No. 2 - RETURN
... m.c.manticore:INFO: Generated testcase No. 3 - REVERT
... m.c.manticore:INFO: Generated testcase No. 4 - STOP
... m.c.manticore:INFO: Generated testcase No. 5 - REVERT
... m.c.manticore:INFO: Generated testcase No. 6 - REVERT
... m.c.manticore:INFO: Results in /home/ethsec/workshops/Automated Smart Contracts Audit
- TruffleCon 2018/manticore/examples/mcore_t6vi6ij3
...
```

在没有额外信息的情况下，Manticore 将利用新的符号交易探索智能合约，直到它不再探索合约上的新途径为止。Manticore 不会在失败后执行新的交易（如恢复原状后）。

Manticore 将在一个`mcore_*`目录中输出信息。 除其他外，您将在这个目录中找到：

- `global.summary`：覆盖面和编译器警告
- `test_XXXXX.summary`：覆盖面、前一次的说明、每次测试案例的帐户余额
- `test_XXXXX.tx`：每个测试案例的交易详细列表

在这里，Manticore 发现了 7 个测试案例，它们对应于（文件名顺序可能会改变）：

	交易 0	交易 1	交易 2	结果
test_00000000.tx	合约创建	f(!=65)	f(!=65)	停止
test_00000001.tx	合约创建	回退函数		撤销
test_00000002.tx	合约创建			返回
test_00000003.tx	合约创建	f(65)		撤销
test_00000004.tx	合约创建	f(!=65)		停止
test_00000005.tx	合约创建	f(!=65)	f(65)	撤销
test_00000006.tx	合约创建	f(!=65)	回退函数	撤销

检索摘要 f(!=65)表示使用不同于 65 的任何值调用的调用的 f。

正如您可以注意到的那样，Manticore 为每个成功或撤销的交易生成一个独特的测试案例。

如果您想要快速的代码检查，请使用`--Quickmode`标志（它禁用 bug 检测器、gas 计算...）

通过 API 操纵智能合约 {#manipulate-a-smart-contract-through-the-api}

本节介绍如何通过 Manticore Python API 操纵智能合约的细节。 您可以使用 python 扩展名`*.py`创建新文件，并通过将 API 命令（下面将介绍其基础内容）添加到这个文件中来写入必要的代码，然后使用`$ python3 *.py`命令运行它。 您也可以直接在 python 控制台中执行下面的指令，使用`$python3`命令来运行控制台。

创建帐户 {#creating-accounts}

首先，您要通过以下命令启动一个新的区块链：

```
from manticore.ethereum importer MantocoreEVM

m = MantocoreEVM()
```

使用m.create_account创建一个非合约帐户：

```
user_account = m.create_account(balance=1000)
```

可以使用m.solidity_create_contract来部署一个 Solidity 合约：

```
source_code = '''
pragma solidity >=0.4.24 <0.6.0;
contract Simple {
    function f(uint a) payable public{
        if (a == 65) {
            revert();
        }
    }
}
'''
# Initiate the contract
contract_account = m.solidity_create_contract(source_code, owner=user_account)
```

概览 {#summary}

- 您可以通过m.create_account和m.solidity_create_contract创建用户和合约帐户。

执行交易 {#executing-transactions}

Manticore 支持两种类型的交易：

- 原始交易：已探索所有函数
- 命名交易：只探索一个函数

原始交易 {#raw-transaction}

使用m.transaction执行原始交易：

```
m.transaction(caller=user_account,
              address=contract_account,
              data=data,
              value=value)
```

调用者、地址、数据或交易的值可以是具体的或抽象的：

- mmmake_symbolic_value创建一个抽象值。
- mmmake_symbolic_buffer(size)创建一个符号字节数组。

例如

```
symbolic_value = m.make_symbolic_value()
symbolic_data = m.make_symbolic_buffer(320)
m.transaction(caller=user_account,
              address=contract_address,
              data=symbolic_data,
              value=symbolic_value)
```

如果数据是象征性的，Manticore 将在交易执行期间探索合约中的所有函数。 查看Hands on the Ethernaut CTF文章中的回退函数解释，对于理解函数选择的工作原理会有所帮助。

命名交易 {#named-transaction}

函数可以通过其的名称执行。 要使用 user_account 中的符号值以及 0 ether 执行 `f(uint var)`，请使用：

```
symbolic_var = m.make_symbolic_value()
contract_account.f(symbolic_var, caller=user_account, value=0)
```

如果没有指定交易的 `value`，则默认为 0。

概览 {#summary-1}

- 交易的参数可以是具体的或抽象的
- 原始交易将探索所有函数
- 函数可以通过其名称来调用

工作区 {#workspace}

`m.workspace` 目录用作所有生成的文件的输出目录：

```
print("Results are in {}".format(m.workspace))
```

终止探索 {#terminate-the-exploration}

要停止探索，请使用m.finalize()。 一旦这个方法被调用，就不应该再发送任何交易，而且 Manticore 会针对所探索的每一条路径生成测试案例。

总结：在 Manticore 下运行 {#summary-running-under-manticore}

将所有先前的步骤放在一起，我们就会得到：

```
from manticore.ethereum import ManticoreEVM

m = ManticoreEVM()

with open('example.sol') as f:
    source_code = f.read()

user_account = m.create_account(balance=1000)
contract_account = m.solidity_create_contract(source_code, owner=user_account)

symbolic_var = m.make_symbolic_value()
contract_account.f(symbolic_var)

print("Results are in {}".format(m.workspace))
m.finalize() # stop the exploration
```

以上所有代码都可以在 `example_run.py` 中找到。

获取投掷路径 {#getting-throwing-paths}

我们现在将为路径生成特定的输入，以在 `f()` 中引发异常。 目标仍为以下智能合约 `example.sol`。

```
pragma solidity >=0.4.24 <0.6.0;
contract Simple {
    function f(uint a) payable public{
        if (a == 65) {
            revert();
        }
    }
}
```

使用状态信息 {#using-state-information}

执行的每个路径都有其区块链的状态。 此状态要么是准备就绪，要么是被终止了，也就是说，它达到了 THROW 或 REVERT 指令状态。

- m.ready_states: 已准备就绪状态列表（他们没有执行 REVERT/INVALID）
- m.killed_states：终止状态列表
- m.all_states：所有状态

```
for state in m.all_states:
    # do something with state
```

您可以访问状态信息。 例如：

- `state.platform.get_balance(account.address)`：帐户余额
- `state.platform.transactions`：交易列表
- `state.platform.transactions[-1].return_data`：最后一笔交易返回的数据

最后一笔交易返回的数据是一个数组，可以用 ABI.deserialize 转换为一个值，例如：

```
data = state.platform.transactions[0].return_data
data = ABI.deserialize("uint", data)
```

如何生成测试案例 {#how-to-generate-testcase}

使用m.generate_testcase(state, name)生成测试案例：

```
m. generate_testcase(state, 'BugFound')
```

概览 {#summary-2}

- 您可以使用 m.all_states 对状态进行迭代
- `state.platform.get_balance(account.adds)` 返回帐户余额
- `state.platform.transactions` 返回交易列表
- `transaction.return_data` 是返回的数据
- `m.generate_testcase(state, name)` 为状态生成输入

总结：获取抛出路径 {#summary-getting-throwing-path}

```
from manticore.ethereum import ManticoreEVM

m = ManticoreEVM()
```

```
with open('example.sol') as f:
    source_code = f.read()

user_account = m.create_account(balance=1000)
contract_account = m.solidity_create_contract(source_code, owner=user_account)

symbolic_var = m.make_symbolic_value()
contract_account.f(symbolic_var)

## Check if an execution ends with a REVERT or INVALID
for state in m.terminated_states:
    last_tx = state.platform.transactions[-1]
    if last_tx.result in ['REVERT', 'INVALID']:
        print('Throw found {}'.format(m.workspace))
        m.generate_testcase(state, 'ThrowFound')
```

以上所有代码都可以在 `example_run.py` 中找到。

注意我们可以生成一个更简单的脚本，因为所有由 terminated_state 返回的状态在其结果中都有 REVERT 或 INVALID：这个例子只是为了演示如何操作 API。

添加限制 {#adding-constraints}

我们将看到如何对探索加以约束。 我们将作出这样的假设：`f()` 的文档指出，该函数从未在 `a == 65` 的情况下被调用，因此任何 `a == 65` 的错误都不是真正的错误。 目标仍为以下智能合约 `example.sol`：

```
pragma solidity >=0.4.24 <0.6.0;
contract Simple {
    function f(uint a) payable public{
        if (a == 65) {
            revert();
        }
    }
}
```

运算符 {#operators}

运算符模块使约束操作变得简便，除此之外，它还提供了其他功能：

- Operators.AND，
- Operators.OR，
- Operators.UGT（无符号大小），
- Operators.UGE（无符号大于或等于），
- Operators.ULT（无符号小于），
- Operators.ULE（无符号小于或等于）。

请使用以下代码导入模块：

```
from manticore.core.smtlib import Operators
```

`Operators.CONCAT` 用于将一个数组与一个值级联。 例如，一个交易的 return_data 需要转变为一个值，以便与另一个值进行检查对比：

```
last_return = operators.CONCAT(256,*last_return)
```

约束 {#state-constraint}

您可以在全局范围内或针对某个特定的状态使用约束。

全局约束 {#state-constraint}

使用`m.constraint(constraint)`来添加一个全局约束。 例如，您可以从一个符号地址调用合约，并将这个地址约束为特定的值：

```
symbolic_address = m.make_symbolic_value()
m.constraint(Operators.OR(symbolic == 0x41, symbolic_address == 0x42))
m.transaction(caller=user_account,
              address=contract_account,
              data=m.make_symbolic_buffer(320),
              value=0)
```

状态约束 {#state-constraint}

使用[state.constrain(constraint)]为一个特定的状态添加一个约束。它可以用来在状态检索后对其进行约束，以检查其上的某些属性。

检查约束 {#checking-constraint}

使用`solver.check(state.constracts)`来了解约束是否仍然可行。 例如，以下代码将 symbolic_value 限定为不等于 65 ，并检查状态是否仍然可行。

```
state.constrain(symbolic_var != 65)
if solver.check(state.constraints):
    # state is feasible
```

摘要：添加限制因素 {#summary-adding-constraints}

通过在前面的代码中添加约束，我们获得：

```
from manticore.ethereum import ManticoreEVM
from manticore.core.smtlib.solver import Z3Solver

solver = Z3Solver.instance()

m = ManticoreEVM()

with open("example.sol") as f:
    source_code = f.read()

user_account = m.create_account(balance=1000)
contract_account = m.solidity_create_contract(source_code, owner=user_account)

symbolic_var = m.make_symbolic_value()
contract_account.f(symbolic_var)

no_bug_found = True
```

```
## Check if an execution ends with a REVERT or INVALID
for state in m.terminated_states:
    last_tx = state.platform.transactions[-1]
    if last_tx.result in ['REVERT', 'INVALID']:
        # we do not consider the path were a == 65
        condition = symbolic_var != 65
        if m.generate_testcase(state, name="BugFound", only_if=condition):
            print(f'Bug found, results are in {m.workspace}')
            no_bug_found = False

if no_bug_found:
    print(f'No bug found')
```

以上所有代码都可以在`example_run.py`中找到。

title: 如何使用Slither发现智能合约漏洞
description: 如何使用Slither自动发现智能合约中的漏洞
author: Trailofbits
lang: zh
sidebar: true
tags:

- "solidity"
- "智能合约"
- "安全性"
- "测试"
- "静态分析"
  skill: 高级
  published: 2020-06-09
  source: 构建安全的合约
  sourceUrl: https://github.com/crytic/building-secure-contracts/tree/master/program-analysis/slither

---

如何使用 Slither {#how-to-use-slither}

本教程的目的是演示如何使用 Slither 自动查找智能合约中的漏洞。

- 安装
- 命令行使用方法
- 静态分析简介：静态分析简介
- API：Python API 说明

安装 {#installation}

Slither 需要 Python3.6 及以上版本。 它可以通过 pip 或使用 docker 来安装。

通过 pip 安装 Slither：

```
pip3 install --user slither-analyzer
```

通过 docker 安装 Slither：

```
docker pull trailofbits/eth-securitytoolbox
docker run-it -v "$PWD":/home/trufflecon trailofbits/eth-securitytoolbox
```

上文中最后一个命令在 docker 中运行 eth-security-toolbox 命令。该 eth-security-toolbox 命令需要能访问您的当前目录。 您可以从自己的主机更改文件，并在 docker 中运行针对文件这些工具。

在 docker 中，运行如下命令：

```
sol-select 0.5.11
cd /home/trufflecon/
```

运行脚本 {#running-a-script}

使用 python 3 运行如下 python 脚本：

```
python3 script.py
```

命令行 {#command-line}

命令行与用户定义的脚本。Slither 带有一组预定义的探测器，可以找到许多常见的漏洞。 从命令行调用 Slither 将运行所有的检测器，不需要使用者具有详细的静态分析知识。

```
slither project_paths
```

除了检测器之外，Slither 还通过其打印机和工具提供代码审查功能。

使用crytic.io以获得对私有探测器和 GitHub 集成功能的访问权限。

静态分析 {#static-analysis}

Slither 静态分析框架的功能和设计已在博客文章（1，2）和一篇学术论文中进行了描述。

静态分析有不同的风格。 您很可能意识到，像clang和gcc这样的编译器依赖于上述这些研究技术，但它也是（Infer、CodeClimate、FindBugs和基于形式化方法的一些工具，比如Frama-C和Polyspace。

我们不会在这里详尽地回顾静态分析技术和研究人员。 相反，我们将专注于了解 Slither 是如何工作的，以便您能更有效地利用它来发现漏洞和理解代码。

- 代码表示
- 代码分析
- 代码的中间表示

代码表示 {#code-representation}

与对单一的执行路径进行推理的动态分析相比，静态分析会对所有的执行路径进行一次性的推理。 为此，它依赖于一个不同的代码表示方式。 最常见的两个是抽象语法树(AST)和控制流图(CFG)。

抽象语法树(AST) {#abstract-syntax-trees-ast}

每次编译器解析代码时都会用到 AST 技术。 它可能是进行静态分析的最基本结构。

简而言之，AST 是一棵结构化的树，通常每片叶子包含一个变量或一个常量，而树的内部节点是操作符或控制流操作。 考虑以下代码：

```
function safeAdd(uint a, uint b) pure internal returns(uint){
    if(a + b <= a){
        revert();
    }
    return a + b;
}
```

相应的 AST 如图所示：

抽象语法树（AST）

Slither 使用由 solc 工具导出的 AST。

虽然构建简单，但 AST 是一个嵌套结构。 有时，这并不是最直观的代码分析方法。 例如，为了确定表达式 `a + b <= a` 所使用的运算，您必须首先分析 `<=`，然后分析 `+`。 一个常见的方法是使用所谓的访问者模式，它以递归方式在树上进行遍历。 Slither 在 `ExpressionVisitor` 中包含一个通用的访问者程序。

下面的代码使用 `ExpressionVisitor` 来检测一个表达式是否包含加法。

```
from slither.visitors.expression.expression import ExpressionVisitor
from slither.core.expressions.binary_operation import BinaryOperationType

class HasAddition(ExpressionVisitor):

    def result(self):
        return self._result

    def _post_binary_operation(self, expression):
        if expression.type == BinaryOperationType.ADDITION:
            self._result = True

visitor = HasAddition(expression) # expression is the expression to be tested
print(f'The expression {expression} has a addition: {visitor.result()}')
```

控制流图（CFG） {#control-flow-graph-cfg}

第二种最常见的代码表示是控制流图（CFG）。 顾名思义，它是一种基于图的表示方法，展现了所有的代码执行路径。 每个节点包含一条或多条指令。 图中的边代表控制流操作（if/then/else，循环，等等）。 我们上一个例子的 CFG 是：

控制流图（CFG）

大多数的代码分析技术都是建立在 CFG 的基础表示之上。

也存在许多其他的代码表示方法。 根据您想进行的代码分析的不同场景，每种代码表示方法都有其优点和缺点。

分析 {#analysis}

您可以在 Slither 工具中进行的最简单类型的分析是语法分析。

语法分析 {#syntax-analysis}

Slither 可以浏览代码的不同组成部分及其表示方法，使用类似模式匹配的方法找到代码内部不一致的地方和代码缺陷。

例如，以下检测器可以寻找与语法有关的问题。

- 状态变量映射：遍历所有的状态变量，并检查是否有任何变量映射来自于继承的合约（state.py#L51-L62）的变量。
- 不正确的 ERC20 接口：寻找不正确的 ERC20 函数签名（incorrect_erc20_interface.py#L34-L55）。

语义分析 {#semantic-analysis}

与语法分析相比，语义分析将更深入地分析代码的 "含义"。 这个系列包括一些宽泛的分析类型。 这些分析会产生更强大和有用的分析结果，但编写起来也更复杂。

语义分析常常被用于最先进的代码漏洞检测。

数据依赖性分析 {#fixed-point-computation}

如果变量 `variable_a` 的值受到变量 `variable_b` 的影响，那么就说变量 `variable_a` 是数据依赖于变量 `variable_b` 的。

在下面的代码中，变量 `variable_a` 是依赖于变量 `variable_b` 的。

```
// ...
variable_a = variable_b + 1;
```

Slither 内置了数据依赖分析功能，这要归功于它的中间表示法（在后面的部分讨论）。

在严格平等危险的检测器中可以找到一个数据依赖的使用例子。 这里 Slither 将对危险值进行严格平等比较 (incorrect_strict_equality.py#L86-L87)，同时，将通知用户应该使用 `>=` 或 `<=`（而不是 `==`）去阻止攻击者使合约进入代码陷阱。 另外，这个检测器会认为 `balanceOf(address)` 的调用结果值是一个潜在的危险 (incorrect_strict_equality.py#L63-L64)，同时，会使用数据依赖引擎来跟踪其使用情况。

定点计算 {#fixed-point-computation}

如果您的分析在 CFG 的节点中遍历并沿着 CFG 的边进行，您可能会看到一些已经访问过的节点。 例如，出现如下所示的循环代码：

```
for(uint i; i < range; ++){
    variable_a += 1
}
```

您的分析将需要知道何时停止。 这里有两种主要策略：1）在每个节点上迭代有限次数，2）通过计算所谓的*定点*。 一个定点基本上意味着分析此节点不会提供任何有意义的信息。

在代码可重入检测器中可以找到使用的定点的示例：Slither 探索这些节点，寻找外部调用、写入和读取存储。 一旦到达某个定点 (reentrancy.py#L125-L131)，分析器就会停止代码遍历，并通过不同的重入模式来分析结果，了解是否存在代码重入现象。 （reentrancy_benign.py、reentrancy_read_before_write.py、reentrancy_eth.py)。

使用高效的定点计算方法编写分析，需要很好地理解分析是如何传播其信息的。

代码的中间表示 {#intermediate-representation}

中间表示(IR)是一种比原始语言更适合进行静态分析的语言。 Slither 将 Solidity 转换为它自己的 IR：SlithIR。

如果您只是想编写基本的代码检查，那么理解 SlithIR 不是必须的。 但是，如果您打算编写更高级的语义分析，对 SlithIR 的理解将派上用场。 SlithIR和SSA打印器工具将帮助您理解代码是如何被翻译的。

API 基础知识 {#api-basics}

Slither 有一套 API，可让您探索合约的基本属性及其功能。

加载代码库：

```
from slither import Slither
slither = Slither('/path/to/project')
```

探索合约和相关的函数 {#exploring-contracts-and-functions}

一个`Slither`对象具有：

- `contracts (list(Contract)`：合约列表
- `contracts_derived (list(Contract)`：未被其他合约继承的合约列表（合约的子集）
- `get_contract_from_name (str)`：从合约名称返回一个合约实例

一个`Contract`对象具有：

- `name (str)`：合约名称
- `functions (list(Function))`：函数列表
- `modifiers (list(Modifier))`：函数列表
- `all_functions_called (list(Function/Modifier))`：合约可访问的所有内部函数列表
- `inheritance (list(Contract))`：继承的合约的列表
- `get_function_from_signature (str)`：通过函数签名返回一个函数
- `get_modifier_from_signature (str)`：从函数签名返回一个修饰符
- `get_state_variable_from_name (str)`：从其名称返回一个状态变量

一个`Function`或`Modifier`对象具有：

- `name (str)`：函数的名称
- `contract (contract)`：声明该函数的合约
- `nodes (list(Node))`：组成该函数/修饰符的 CFG 的节点列表
- `entry_point (Node)`：CFG 的入口点
- `variables_read (list(Variable))`：所读取变量的列表
- `variables_written (list(Variable))`：所写入变量的列表
- `state_variables_read (list(StateVariable))`：所读取状态变量的列表（所读取变量的子集）
- `state_variables_written (list(StateVariable))`：所写入状态变量的列表(所写入变量的子集)

---

title: 如何在钱包中查看您的非同质化代币（非同质化代币系列教程第 3/3 部分）
description: 本教程介绍了如何在 MetaMask 上查看所持有的非同质化代币！
author: "Sumi Mudgil"
tags:

- "非同质化代币"

- "ERC-721"
- "Alchemy"
- "MetaMask"
- "非同质化代币"
- "Solidity"
  skill: 初学者
  lang: zh
  sidebar: true
  published: 2021-04-22

本教程是非同质化代币教程系列的第 3/3 部分，在这里将教您查看我们新铸造的非同质化代币。 然而，您也可以把它当做在 MetaMask 上使用任何 ERC-721 代币的通用教程，无论是主网还是测试网。 如果您想学习如何在 Ethereum 上发行自己的非同质化代币，您可以查看第 1 部分：如何编写和部署非同质化代币智能合约！

恭喜！ 您已经来到我们非同质化代币系列教程中最短和最简单的部分——如何在虚拟钱包中查看您刚刚铸造的非同质化代币。 本示例中，我们将继续使用前两部分用过的 MetaMask。

作为一个前提条件，您应该已经在移动设备上安装了 MetaMask， 并且它应该包含您铸造非同质化代币的账户——您可以在 iOS 或 Android 免费获得该应用程序。

第 1 步：将你的网络设置为 Ropsten {#set-network-to-ropsten}

在应用程序顶端，按“钱包”按钮，然后会提示您选择网络。 由于我们的非同质化代币是在 Ropsten 网络上发行的，您将会选择 Ropsten 作为您的网络。

如何在 MetaMask Mobile 上设置 Ropsten 网络

第 2 步：将您的收藏添加到 MetaMask {#add-nft-to-metamask}

在您使用 Ropsten 网络时，选择右侧“收藏”选项卡，然后添加非同质化代币智能合约地址和您的非同质化代币的 ERC-721 代币 ID—— 您应该能够根据我们第 3 部分的教程，通过非同质化代币的交易哈希值在 Etherscan 网站上找到它。

如何查找您的交易哈希值和 ERC-721 代币 ID

您可能需要刷新几次才能查看您的非同质化代币——但一定会出现！

如何将您的非同质化代币上传到 MetaMask

恭喜！ 您已成功铸造了一个非同质化代币，现在可以查看了！ 我们迫不及待地想看看您将如何席卷非同质化代币世界！

title: 如何撰写和部署非同质化代币（非同质化代币教程系列 1/3）
description: 本教程是关于非同质化代币的系列教程的第一部分，将带您逐步了解如何使用以太坊和星际文件系统 (IPFS) 编写和部署非同质化代币（ERC-721 代币）智能合约。
author: "Sumi Mudgil"
tags:

- "非同质化代币"
- "ERC-721"
- "Alchemy"

- "Solidity"
- "智能合约"
  skill: 初学者
  lang: zh
  sidebar: true
  published: 2021-04-22

随着非同质化代币将区块链带入公众视野，现在是一个极好的机会，可以通过在以太坊区块链上发布自己的非同质化代币（ERC-721 代币）来了解炒作情况。

Alchemy 非常自豪能够推动非同质化代币领域的一些巨头，包括 Makersplace（最近在克利斯蒂拍卖行创造了 6900 万美元的数字艺术品销售记录）、Dapper Labs（NBA Top Shot & Crypto Kitties 的创作者）、OpenSea（世界上最大的非同质化代币市场）、Zora、Super Rare、NFTfi、Foundation、Enjin、Origin Protocol、Immutable 等。

在本教程中，我们将学习使用 MetaMask、Solidity、Hardhat、Pinata 和 Alchemy，在 Ropsten 测试网络上创建和部署 ERC-721 智能合约的全过程（如果您还不明白其中的含义，也不要着急——我们会加以解释！）。

在本教程的第二部分，我们将了解如何使用我们的智能合约来铸造非同质化代币；在第三部分，我们将说明如何在 MetaMask 上查看您的非同质化代币。

当然，如果您有任何问题，请随时联系 Alchemy Discord 或访问 Alchemy 的 NFT API 文档!

步骤 1：连接到以太坊网络 {#connect-to-ethereum}

有很多方法可以向以太坊区块链发出请求，但为了方便起见，我们将使用 Alchemy 上的免费帐户。Alchemy 是一个区块链开发平台，能够提供应用程序接口，让我们无需运行自己的节点，即可与以太坊公链进行通信。

在本教程中，我们将利用 Alchemy 平台的开发者工具进行监测和分析，以便了解智能合约部署的底层逻辑。 如果您还没有 Alchemy 账户，您可以在这里免费注册。

步骤 2：创建应用程序（和应用程序接口密钥） {#make-api-key}

创建了 Alchemy 账户后，您可以通过创建应用程序来生成应用程序接口密钥。 我们可以用它向 Ropsten 测试网络发起请求。 如果您想了解更多关于测试网络的信息，请查看本指南。

1. 在 Alchemy 仪表板中，将鼠标悬停在导航栏中的“应用程序”上，单击“创建应用程序”并前往此页面。

创建您的应用程序

2. 为您的应用程序命名（我们起名为“我的第一个非同质化代币！”），提供简短的描述，将环境设定为“Staging”（用于您的应用程序簿记），选择“Ropsten”网络。

配置并发布您的应用程序

3. 点击“创建应用程序”即可！ 您的应用程序应该会出现在下面的表格中。

步骤 3：创建一个以太坊账户（地址） {#create-eth-address}

我们需要一个以太坊帐户来发送和接收交易。 在本教程中，我们将使用 MetaMask——浏览器中的虚拟钱包，用来管理您的以太坊账户地址。 如果您想了解更多关于以太坊交易的运作方式，请查看以太坊基金会的这个页面。

您可以在这里免费下载并创建一个 MetaMask 账户。 创建账户时，或者如果您已经有一个账户时，确保切换到右上方的“Ropsten 测试网络”（这样我们就不会用实际货币进行交易）。

将 Ropsten 设为您的网络

步骤 4：从水龙头添加以太币 {#step-4-add-ether-from-a-faucet}

为了将我们的智能合约部署到测试网络，我们需要一些虚拟以太币。 要获取 ETH，您可以前往 FaucETH 并输入您的 Ropsten 账户地址，点击“Request funds”，然后在中选择“Ethereum Testnet Ropsten” 下拉菜单，最后再次单击“Request funds”按钮。 您应该会很快在您的 MetaMask 帐户中看到以太币！

步骤 5：查看账户余额 {#check-balance}

为了核实我们的余额，可以使用 Alchemy 的创作者工具发出 eth_getBalance 请求。 这将返回我们钱包中的以太币金额。 输入您的 Metamask 帐户地址并单击“发送请求”后，您应该会看到这样的响应：

```
`{"jsonrpc": "2.0", "id": 0, "result": "0xde0b6b3a7640000"}`
```

注：此结果以 wei 为单位，而非 ETH。 Wei 是以太坊最小的面额。 将 wei 转换为 ETH 的公式为 1 eth = 10^{18} wei。 因此，如果我们将 0xde0b6b3a7640000 转换为十进制，我们得到 $1*10^{18}$ wei，即 1 ETH。

呼！ 这里显示了我们所有的虚拟货币。

步骤 6：初始化我们的项目 {#initialize-project}

首先，需要为我们的项目创建一个文件夹。 导航到您的命令行，然后输入：

```
mkdir my-nft
cd my-nft
```

现在我们进入了项目文件夹，我们将使用 npm init 来初始化项目。 如果您还没有安装 npm，请遵循此处的说明（我们还需要 Node.js，所以请一并下载！）。

```
npm init
```

其实如何回答安装问题并不重要，以下提供一个回答的样例供参考：

```
package name: (my-nft)
version: (1.0.0)
description: My first NFT!
entry point: (index.js)
test command:
git repository:
keywords:
author:
license: (ISC)
About to write to /Users/thesuperb1/Desktop/my-nft/package.json:

{
  "name": "my-nft",
  "version": "1.0.0",
  "description": "My first NFT!",
  "main": "index.js",
  "scripts": {
```

```
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "",
  "license": "ISC"
}
```

批准 package.json，我们就可以开始了！

步骤 7：安装安全帽 (Hardhat) {#install-hardhat}

安全帽是一个用于编译、部署、测试和调试以太坊软件的开发环境。 在部署到实时链上之前，它帮助开发者在本地构建智能合约和去中心化应用程序。

在我们的 my-nft 项目内运行：

```
npm install --save-dev hardhat
```

查看此页面，了解更多有关安装说明的详细信息。

步骤 8：创建安全帽项目 {#create-hardhat-project}

在我们的项目文件夹中运行：

```
npx hardhat
```

然后应该能看到一条欢迎消息和选项，用于选择您想要做的事情。 选择“创建一个空的 hardhat.config.js”：

```
888    888                      888 888               888
888    888                      888 888               888
888    888                      888 888               888
8888888888  8888b.  888d888 .d88888 88888b.   8888b.  888888
888    888     "88b 888P"  d88" 888 888 "88b     "88b 888
888    888 .d888888 888    888  888 888  888 .d888888 888
888    888 888  888 888    Y88b 888 888  888 888  888 Y88b.
888    888 "Y888888 888     "Y88888 888  888 "Y888888  "Y888
👷 Welcome to Hardhat v2.0.11 👷‍
? What do you want to do? …
Create a sample project
❯ Create an empty hardhat.config.js
Quit
```

这将生成一个 hardhat.config.js 文件，我们将用以配置项目的所有设置（步骤 13）。

步骤 9：添加项目文件夹 {#add-project-folders}

为了使我们的项目有条理，我们将创建两个新的文件夹。 在您的命令行中导航到项目的根目录，然后输入：

```
mkdir contracts
mkdir scripts
```

- contracts/ 是我们保存非同质化代币智能合约代码的位置。
- scripts/ 是我们存放脚本的位置，用于部署我们的智能合约和与之交互。

步骤 10：编写合约 {#write-contract}

现在我们的环境已经配置完成，接下来将是更令人兴奋的步骤：*编写我们的智能合约代码！*

在您喜欢的编辑器中打开 my-nft 项目（我们喜欢用 VSCode）。 智能合约用一种名为 Solidity 的语言编写而成，我们将用它来编写我们的 MyNFT.sol 智能合约。

1. 导航到 `contracts` 文件夹并创建一个名为 MyNFT.sol 的新文件
2. 下面是我们的非同质化代币智能合约代码，基于 OpenZeppelin 库的 ERC-721 实现。 复制并粘贴以下内容到您的 MyNFT.sol 文件。

```
//Contract based on [https://docs.openzeppelin.com/contracts/3.x/erc721]
(https://docs.openzeppelin.com/contracts/3.x/erc721)
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

import "@openzeppelin/contracts/token/ERC721/ERC721.sol";
import "@openzeppelin/contracts/utils/Counters.sol";
import "@openzeppelin/contracts/access/Ownable.sol";
import "@openzeppelin/contracts/token/ERC721/extensions/ERC721URIStorage.sol";

contract MyNFT is ERC721URIStorage, Ownable {
    using Counters for Counters.Counter;
    Counters.Counter private _tokenIds;

    constructor() ERC721("MyNFT", "NFT") {}

    function mintNFT(address recipient, string memory tokenURI)
        public onlyOwner
        returns (uint256)
    {
        _tokenIds.increment();

        uint256 newItemId = _tokenIds.current();
        _mint(recipient, newItemId);
        _setTokenURI(newItemId, tokenURI);

        return newItemId;
    }
}
```

3. 因为我们要从 OpenZeppelin 合约库继承类，请在您的命令行中运行 `npm install @openzeppelin/contracts`，将该库安装到我们的文件夹中。

那么，这段代码究竟*做*了什么？ 让我们把它逐行分解。

在我们的智能合约的顶部，我们导入了三个 OpenZeppelin 智能合约类：

- @openzeppelin/contracts/token/ERC721/ERC721.sol 包含 ERC-721 标准的实现，我们的非同质化代币智能合约将继承此标准。 （要成为有效的非同质化代币，您的智能合约必须实现 ERC-721 标准的所有方法。）要更加详细地了解继承的 ERC-721 功能，请在这里查看接口定义。
- @openzeppelin/contracts/utils/Counters.sol 提供了只能以 1 为增量或减量的计数器。 我们的智能合约使用一个计数器来跟踪非同质化代币总铸币量，并在我们的新非同质化代币上设置唯一 ID。 （每个使用智能合约铸造的非同质化代币必须分配一个唯一 ID——在这里，我们的唯一 ID 仅由存在的非同质化代币总量决定。例如，我们用智能合约铸造的第一个非同质化代币的 ID 是“1”，第二个非同质化代币的 ID 是“2”，以此类推）

- @openzeppelin/contracts/access/Ownable.sol 在我们的智能合约上设置了访问控制，因此只有智能合约的所有者（您）可以铸造非同质化代币。 （注意：包含访问控制完全是一种偏好。 如果您希望任何人都能使用您的智能合约铸造非同质化代币，请删除第 10 行的 Ownable 和第 17 行的 onlyOwner）。

在导入声明后，是我们自定义的非同质化代币智能合约。合约非常短，只包含一个计数函数、一个构造函数和独立函数！ 这要归功于我们继承的 OpenZepelin 合约，它们实现了我们创建非同质化代币所需的大多数方法。例如 `owerOf` 函数，其作用是返回非同质化代币所有者，以及 `transferFrom` 函数，其作用是将非同质化代币的所有权从一个账户转移到另一个账户。

在我们的 ERC-721 构造函数中，您会注意到我们传递 2 个字符串，“MyNFT”和“NFT”。 第一个变量是智能合约的名称，第二个变量是其符号。 您可以随意命名这些变量！

最后，我们的函数 `mintNFT(address recipient, string memory tokenURI)` 允许我们铸造 NFT！ 您将注意到，这个函数包含两个变量：

- `address recipient` 指定了接收新铸造非同质化代币的地址
- `string memory tokenURI` 是一个可以解析为 JSON 文档的字符串，该文档用于描述非同质化代币的元数据。 非同质化代币的元数据是它的核心所在，使它拥有可配置的属性，例如名称、描述、图像和其他属性。 在本教程第二部分，我们将描述如何配置元数据。

`mintNFT` 调用了继承的 ERC-721 库中的一些方法，最终返回一个数字，代表新铸造非同质化代币的 ID。

步骤 11：将 Metamask 和 Alchemy 连接至您的项目 {#connect-metamask-and-alchemy}

现在，我们创建了 Metamask 钱包、Alchemy 帐户，并且编写了一个智能合约，现在是将这三者连起来的时候了。

从虚拟钱包发送的每笔交易都需要使用您独有的私钥签名。 为了给程序提供此项许可，我们可以安全地将私钥（和 Alchemy 应用程序接口密钥）存储在一个环境文件中。

如需了解更多关于发送交易的信息，请查看关于使用 web3 发送交易的 本教程。

首先，在项目目录中安装 dotenv 软件包：

```
npm install dotenv --save
```

然后，在我们项目的根目录中创建一个 `.env` 文件，并将您的 MetaMask 私钥和 HTTP Alchemy API URL 添加到其中。

- 遵循这些说明，从 MetaMask 导出您的私钥
- 请从下方获取超文本传输协议 Alchemy 应用程序接口网址并将其复制到剪贴板

复制您的 Alchemy 应用程序接口网址

您的 `.env` 现在应该如下所示：

```
API_URL="https://eth-ropsten.alchemyapi.io/v2/your-api-key"
PRIVATE_KEY="your-metamask-private-key"
```

为了将这些变量和代码连接，我们将在步骤 13 中调用 hardhat.config.js 文件中的这些变量。

不要提交 `.env`！ 请确保永远不要与任何人共享或公开您的 `.env` 文件，因为这样做会泄露您的秘密。 如果您使用版本控制，请将您的 `.env` 添加到 gitignore 文件中。

步骤 12：安装 Ethers.js {#install-ethers}

Ethers.js 是一个软件库，通过以更加方便用户的方法打包标准 JSON RPC 方法，从而更容易与以太坊互动，并向以太坊提出请求。

安全帽使我们更容易将插件集成到工具和扩展功能中。 我们将利用 Ethers 插件完成合约部署（Ethers.js 有非常简洁的部署方法）。

在您的项目目录中输入：

```
npm install --save-dev @nomiclabs/hardhat-ethers ethers@^5.0.0
```

下一步中，我们还将在 hardhat.config.js 中使用 ethers。

步骤 13：更新 hardhat.config.js {#update-hardhat-config}

到目前为止，我们已经添加了几个依赖库和插件，现在我们需要更新 hardhat.config.js，以便项目使用所有这些新的组件。

按如下所示更新您的 hardhat.config.js 代码：

```
/**
* @type import('hardhat/config').HardhatUserConfig
*/
require('dotenv').config();
require("@nomiclabs/hardhat-ethers");
const { API_URL, PRIVATE_KEY } = process.env;
module.exports = {
   solidity: "0.8.1",
   defaultNetwork: "ropsten",
   networks: {
      hardhat: {},
      ropsten: {
         url: API_URL,
         accounts: [`0x${PRIVATE_KEY}`]
      }
   },
}
```

步骤 14：编写合约 {#compile-contract}

为了确保一切正常，我们来编译一下合约。 编译任务是安全帽的内部任务之一。

在命令行中运行：

```
npx hardhat compile
```

您可能会看到关于“源文件中未提供 SPDX 许可证识别码”的警告，但不用担心，希望其它的均看起来正常！ 如果遇到问题，您可以随时在 Alchemy cord 社区中发消息询问。

步骤 15：编写部署脚本 {#write-deploy}

合约已经写完，配置文件也准备妥当，现在是写合约部署脚本的时候了。

转到 `scripts/` 文件夹，创建一个新的文件名为 `deploy.js` ，在其中添加以下内容：

```
async function main() {
  const MyNFT = await ethers.getContractFactory("MyNFT")

  // Start deployment, returning a promise that resolves to a contract object
  const myNFT = await MyNFT.deploy()
  await myNFT.deployed()
  console.log("Contract deployed to address:", myNFT.address)
}

main()
  .then(() => process.exit(0))
  .catch((error) => {
    console.error(error)
    process.exit(1)
  })
```

安全帽在合约教程中对这些代码的每一行均提供了很好的解释，我们在这里直接引用他们的解释。

```
const MyNFT = await ethers.getContractFactory("MyNFT");
```

ethers.js 中的 ContractFactory 是用于部署新智能合约的抽象对象。因此这里的 MyNFT 是我们非同质化代币合约实例的工厂。 使用 hardhat-ethers 插件时，ContractFactory 和合约实例默认与第一个签名账户相连。

```
const myNFT = await MyNFT.deploy();
```

调用 ContractFactory 代码中的 deploy() 函数会启动合约部署，然后返回解析为合约的 Promise。 这个对象包括我们智能合约中每个函数的对应调用方法。

步骤 16：部署合约 {#deploy-contract}

我们终于准备好部署我们的智能合约啦！ 返回项目目录的根目录，在命令行中运行：

```
npx hardhat --network ropsten run scripts/deploy.js
```

您会看到类似以下所示的信息：

```
Contract deployed to address: 0x81c587EB0fE773404c42c1d2666b5f557C470eED
```

如果我们去 Ropsten etherscan 并搜索我们的合约地址，我们应该能够看到它已经部署成功。 如果您无法立即看到它，请稍等片刻，因为它可能需要一些时间。 交易将类似以下：

在 Etherscan 上查看您的交易地址

发送地址应匹配您的 MetaMask 帐户地址，收件地址将显示“合约创建”。 如果我们点击进入交易，我们将在“To”字段中看到我们的合约地址：

在 Etherscan 上查看您的合约地址

太棒了！ 您刚刚在以太坊区块链上部署了一个非同质化代币智能合约！

为了更深入了解到底发生了什么，我们转到 Alchemy 仪表板中的 Explorer 选项卡。 如果您有多个 Alchemy 应用程序，请确保按应用程序筛选，然后选择“MyNFT”。

使用 Alchemy 的浏览器仪表板查看“后端”调用

在这里您会看到一系列的 JSON-RPC 调用，都是在我们调用 .deploy() 函数时，Hardhat/Ethers 替我们在后端完成的。 这里有两项重要调用，一个是 eth_sendRawTransaction，这是实际将我们的合约写入 Ropsten 链的请求，另一个是 eth_getTransactionByHash，此为读取有关我们交易给定哈希值的请求（即发送交易时的典型模式）。 如需了解更多关于发送交易的信息，请查看关于使用 Web3 发送交易的本教程。

以上即为本教程第 1 部分的全部内容。 在第 2 部分，我们将通过铸造非同质化代币与我们的智能合约进行交互，在第 3 部分，我们将向您演示如何在以太坊钱包中查看您的非同质化代币！

title: 通过solidity与其他合约进行交互
description: 如何对已经存在的合约进行智能合约的部署，并与其进行交互
author: "jdourlens"
tags:

- "智能合约"
- "solidity"
- "remix"
- "工厂"
- "部署"
- "可组合性"

skill: 高级
lang: zh
sidebar: true
published: 2020-04-05
source: 以太坊开发团队
sourceUrl: https://ethereumdev.io/interact-with-other-contracts-from-solidity/
address: "0x19dE91Af973F404EDF5B4c093983a7c6E3EC8ccE"

在之前的教程中，我们学习到了关于 如何部署您的第一个智能合约的内容。在此基础上，还进一步学习了一些特性，比如使用修饰符控制访问权限、Solidity 中的错误处理等。 在本教程中，我们将学习如何基于一个已有的合约进行智能合约的部署，并与其交互。

我们将会通过创建工厂的方式编写一个允许任何人拥有自己的 `Counter` 智能合约的合约，这个合约的名称将为 `CounterFactory`。 首先，我们给大家展示一下初始 `Counter` 智能合约的代码。

```
pragma solidity 0.5.17;

contract Counter {

    uint256 private _count;
    address private _owner;
    address private _factory;


     modifier onlyOwner(address caller) {
        require(caller == _owner, "You're not the owner of the contract");
        _;
```

```
    }

    modifier onlyFactory() {
        require(msg.sender == _factory, "You need to use the factory");
        _;
    }

     constructor(address owner) public {
        _owner = owner;
        _factory = msg.sender;
    }

     function getCount() public view returns (uint256) {
        return _count;
    }

    function increment(address caller) public onlyFactory onlyOwner(caller) {
        _count++;
    }

}
```

注意，我们稍微修改了合约代码，来记录工厂的地址和合约所有者的地址。 当您从另一个合约来调用这个合约的代码的时，msg.sender 将引用合约工厂的地址。 这是**要理解的一个要点**，因为使用合约来与其他合约进行交互是一种常见做法。 因此，在复杂的情况下，您需要特别注意谁是发送者。

为此，我们也添加了修饰符`onlyFactory`，以确保改变状态的函数只能由将传递原始调用者作为参数的工厂调用。

在我们将管理所有其他 Counters 的新`CounterFactory`中，我们会添加一个映射，该映射会将所有者关联到其 counter 合约地址：

```
mapping(address => Counter) _counters;
```

在以太坊中，映射等同于 javascript 中的对象，它们允许将类型 A 的值映射到类型 B 的值。在本例中，我们将所有者的地址与 Counter 的实例进行映射。

为某人实例化一个新的 Counter 的代码将类似于如下：

```
  function createCounter() public {
      require (_counters[msg.sender] == Counter(0));
      _counters[msg.sender] = new Counter(msg.sender);
  }
```

首先，我们会检查一下此人是否已经拥有了一个 counter。 如果他未拥有 counter，我们会将他的地址传给`Counter`构造函数来实例化一个新的 counter，并将新创建的实例分配至映射。

获取一个特定 Counter 的计数的代码将类似于如下：

```
function getCount(address account) public view returns (uint256) {
    require (_counters[account] != Counter(0));
    return (_counters[account].getCount());
}

function getMyCount() public view returns (uint256) {
    return (getCount(msg.sender));
}
```

第一个函数会检查对于某个给定的地址 Counter 合约是否存在，然后从实例中调用 `getCount` 方法。 第二个函数 `getMyCount` 只是将 msg.sender 直接传递到 `getCount` 函数的方法。

`increment` 函数非常相似，但是会将原始交易发送者传递到 `Counter` 合约。

```
function increment() public {
      require (_counters[msg.sender] != Counter(0));
      Counter(_counters[msg.sender]).increment(msg.sender);
  }
```

注意，如果被调用次数过多，counter 可能会遇到溢出的问题。 您应尽可能多地使用SafeMath 库来防止出现这种可能的情况。

要部署合约，您需要同时提供 `CounterFactory` 和 `Counter` 的代码。 针对 Remix 中的示例进行部署时，您需要选择 CounterFactory。

下面是完整的代码：

```
pragma solidity 0.5.17;

contract Counter {

    uint256 private _count;
    address private _owner;
    address private _factory;


     modifier onlyOwner(address caller) {
        require(caller == _owner, "You're not the owner of the contract");
        _;
    }

    modifier onlyFactory() {
        require(msg.sender == _factory, "You need to use the factory");
        _;
    }

     constructor(address owner) public {
        _owner = owner;
        _factory = msg.sender;
    }

     function getCount() public view returns (uint256) {
        return _count;
    }
```

```
    function increment(address caller) public onlyFactory onlyOwner(caller) {
        _count++;
    }

}

contract CounterFactory {

    mapping(address => Counter) _counters;

    function createCounter() public {
        require (_counters[msg.sender] == Counter(0));
        _counters[msg.sender] = new Counter(msg.sender);
    }

    function increment() public {
        require (_counters[msg.sender] != Counter(0));
        Counter(_counters[msg.sender]).increment(msg.sender);
    }

    function getCount(address account) public view returns (uint256) {
        require (_counters[account] != Counter(0));
        return (_counters[account].getCount());
    }

    function getMyCount() public view returns (uint256) {
        return (getCount(msg.sender));
    }

}
```

编译完成后，您将在 Remix 的部署部分选择要部署的工厂。

选择要在Remix中部署的工厂。

然后，您可以运行工厂合约并且观察值的变化情况。 如果希望通过其他地址来调用智能合约，您需要在 Remix 的帐户选择中更改地址。

title: 去中心化金融 (DeFi)
description: 以太坊上的 DeFi 简介
lang: zh
template: use-cases
emoji: "💸"
sidebar: true
image: ../../../../assets/use-cases/defi.png
alt: 由乐高积木拼成的 ETH 徽标。
sidebarDepth: 2
summaryPoint1: 当前金融体系的全球性、开放性替代方案。
summaryPoint2: 允许您借贷、储蓄、投资、交易等等的产品。

summaryPoint3: 基于任何人都可以用来编程的开源技术。

DeFi 是专为互联网时代构建的开放式全球金融系统，可替代不透明、控制严格并由几十年前的基础设施和流程支撑的系统。 让您可以控制和了解自己的资金。 让您有机会接触全球市场，并可以替代当地货币或银行业务。 DeFi 产品向任何有互联网连接的人开放金融服务，产品主要由用户创造和维护。 迄今为止，价值数百亿美元的加密货币已经通过 DeFi 应用程序流动，而且每天都在增长。

什么是 DeFi? {#what-is-defi}

DeFi 是金融产品和服务的统称，任何可以使用以太坊（具有互联网连接）的人都可以访问这些产品和服务。 有了 DeFi，市场始终开放，没有集中管理机构可以阻止付款或拒绝访问任何内容。 以前有可能出现人为错误的缓慢服务，替代为由任何人都可以检查和审查的代码来处理，变得自动和安全。

这是一个蓬勃发展的加密经济，您可以在那里放贷、借款、做多/做空、赚取利息等等。 精通加密技术的阿根廷人已经使用 DeFi 来逃避严重的通货膨胀。 公司已经开始实时向员工发放工资。 有些人甚至在不需要任何个人身份证明的情况下，就已经获得或偿还了价值数百万美元的贷款。

DeFi 与传统金融 {#defi-vs-tradfi}

了解 DeFi 潜力的最佳方法是了解目前存在的问题。

- 有些人无法设立银行帐户或使用金融服务。
- 无法获得金融服务会阻碍人们就业。
- 金融机构可能会阻止您获得报酬。
- 金融服务的一个隐性收费就是个人数据。
- 政府和中央机构可以随意关闭市场。
- 交易时间通常限于特定时区的营业时间。
- 由于内部的人工流程，资金转移可能需要几天时间。
- 金融服务存在溢价，因为中介机构需要分成。

对比 {#defi-comparison}

DeFi 去中心化金融	传统金融
您持有您的钱。	资金由机构持有。
您可以控制自己的资金流向和使用方式。	您必须相信机构不会错误地管理资金，比如借给风险借款人。
资金转移在几分钟内完成。	如果人工处理，支付可能需要几天时间。
匿名交易。	金融活动与您的身份紧密相连。
DeFi 对任何人开放。	您必须申请使用金融服务。
交易时间 24 小时不间断。	根据人工作息时间制定交易时间。
建立在透明基础上 - 任何人都可以查看产品数据并检查系统运行状况。	金融机构是闭门造车：您不能要求查看他们的贷款历史，管理资产的记录，等等。

探索 DeFi 应用

从比特币开始... {#bitcoin}

从某种程度来讲，比特币是第一款 DeFi 应用程序。 比特币让您真正拥有和掌控价值，并可将其发送到世界任何地方。 为实现这一点，它提供了一种方式，让众多互不信任的人在无需可信赖中介的情况下，就账户的账本达成一致。 比特币向所有人公开，任何人都无权更改其规则。 比特币的规则（例如稀缺性和开放性），都写入了技术中。 与传统金融不同，在传统金融中，政府可以通过印钞让您的储蓄贬值，而企业可以关闭市场。

以太坊便以此为基础。 就像比特币一样，规则不会因你而改变，而且每个人都可以得到。 使用智能合约技术还可以让这种数字货币程序化，使它的功能不仅限于储蓄和交易。

可编程货币 {#programmable-money}

这听起来很奇怪……“我为什么会想对钱进行编程”？ 然而，这只是以太坊代币的一个默认功能。 任何人都可以对逻辑编程，以生成支付。 这样，您就可以将比特币的控制权和安全性与传统金融机构提供的服务相结合。 然后，您就可以用加密货币做一些比特币做不到的事情，比如借贷、预约付款、投资指数基金等等。

如果您不熟悉以太坊，请尝试我们推荐的 DeFi 应用程序。

探索 DeFi 应用程序

您可以用 DeFi 做什么？ {#defi-use-cases}

大多数金融服务都有去中心化的替代方案。 但以太坊也为打造全新的金融产品开辟了机会。 这个清单还在不断增长。

- 向世界各地汇款
- 在全球范围内流转资金
- 获取稳定货币
- 抵押借款
- 无抵押借款
- 开始加密货币存款
- 交易代币
- 扩充您的投资组合
- 为您的灵感提供资金支持
- 购买保险
- 管理您的投资组合

向世界各地快速汇款 {#send-money}

作为一个区块链，以太坊设计用于让用户在全球范围内安全地发送交易。 与比特币一样，以太坊使得向世界各地发送资金就像发送电子邮件一样容易。 只需从钱包中输入收款人的 ENS 名称（如 bob.eth）或他们的账户地址，您的付款（通常）将在几分钟内直接到达对方账户。 要发送或接收付款，您将需要一个钱包。

查看支付 dapp

在全球范围内流转资金... {#stream-money}

您也可以通过以太坊汇款。 这可以让您秒速支付某人的工资，让他们可以在需要时获得资金。 或者立刻租用一些物品，如储物柜或电动滑板车。

而且，如果您因为 ETH 的价值可能发生变化而不想发送 ETH，以太坊上还有其它可供选择的货币：稳定币。

获取稳定币 {#stablecoins}

对很多金融产品和一般支出来说，数字货币的波动都是问题。 DeFi 社区已经通过稳定币解决了这个问题。 它们的价值始终与其他资产挂钩，通常是像美元这样的通用货币。

Dai 或 USDC 等代币的价值和美元的差距通常保持在几美分以内。 这使他们能够完美地用于收入或零售。 在政府发行的货币出现巨大不确定性的时候，拉丁美洲的许多人都曾使用稳定币作为保护储蓄的一种方式。

关于稳定币的更多信息

借款 {#lending}

向去中心化供应商借钱主要有两种情况。

- 点对点，意味着借款人将直接从特定贷款人那里借款。
- 基于资金池，贷款人向借款人可以借贷的资金池提供资金（流动性）。

查看借款 dapp

选择去中心化贷款人具有许多优势...

隐私借贷 {#borrowing-privacy}

今天，贷款和借钱都围绕着相关个人进行。 在放贷前，银行需要了解您是否有能力偿还贷款。

去中心化借贷无需任何一方表明自己的身份即可进行。 但是，借款人必须提供抵押品，如果他们的贷款没有偿还，贷款人将自动获得抵押品。 有的贷款人甚至接受 NFT 作为抵押品。 NFT 是指绘画等独特资产。 关于 NFT 的更多信息

这允许您在不进行征信调查或透露私人信息的情况下借钱。

获得全球资金 {#access-global-funds}

当您选择去中心化贷款人时，可以使用来自全球各地的存款，而不仅仅是选定银行或机构保管的资金。 这使人们更容易获得贷款，并提高利率。

税收优惠 {#tax-efficiencies}

借贷可以让您获得所需资金，而无需出售您的 ETH（该行为需要征税）。 相反，您可以使用 ETH 作为抵押，借入稳定币。 这不仅为您提供所需的现金流，您还可以继续持有自己的 ETH。 当您需要现金时，稳定币是更好的代币，因为它们不会像 ETH 一样波动。 关于稳定币的更多信息

闪电贷 {#flash-loans}

闪电贷是一种实验性更强的去中心化借贷形式，让您在没有抵押物或未提供任何个人信息的情况下借贷。

目前，非技术人员还不能广泛使用它们，但它们暗示了将来每个人都可以使用。

它的工作原理是，在一次交易中放贷和还款。 如果无法偿还，资金将返还给贷款方，就像什么都没有发生过一样。

经常使用的资金放入流动资金池（用于借贷的大资金池）。 如果资金在给定时刻没有使用，这就创造了一个机会，有人可以借用这些资金用于开展业务，并几乎在借入资金的同时全额偿还。

这意味着必须在专门编写的交易中包含大量逻辑。 举一个简单的例子，有人可以使用闪电贷按特定价格借入尽可能多的资产，以便在价格更高的其他交易所出售资产。

所以在该笔交易中会发生以下情况：

- 以 $1.00 的价格从 A 交易所中借入 X 数量的 $asset
- 在 B 交易所按 $1.10 的价格出售 X 数量的 $asset
- 在 A 交易所偿还贷款
- 赚到扣除交易费后的收益

如果交易所 B 的供应突然下降，而用户无法购买足够的资金来支付原始贷款，交易就会失败。

要想在传统金融世界实现上述操作，您需要大量的资金。 这些赚钱的策略仅供已经拥有财富的人士使用。 闪电贷款是一个未来的例子，在这里，拥有金钱不一定是赚钱的先决条件。

关于闪电贷的更多信息

开始使用加密货币存款 {#saving}

借款 {#lending}

您可以通过借出加密货币来赚取利息，并看到资金的实时增长。 现在的利率比当地银行可能高出许多（如果有幸能够使用当地的银行系统）。 下面是一个示例：

- 您借出 100 Dai 稳定币，借给类似 Aave 的产品。
- 您会收到 100 个 Aave Dai (aDai)，代表您已借出的 Dai。
- 您的 aDai 将根据利率增加，您可以看到钱包里的余额在增长。 根据 APR，您的钱包余额在几天甚至几小时后会显示为 100.1234 这样的数字。
- 您可以随时提取与 aDai 余额等额的普通 Dai。

查看借款 dapp

无损彩票 {#no-loss-lotteries}

像 PoolTogether 这样的无损彩票是一种有趣而创新的省钱方法。

- 您使用 100 Dai 购买了 100 张彩票。
- 您将收到 100 个 plDai，代表您的 100 张彩票。
- 如果您有一张彩票中奖，您的 plDai 余额将按奖池金额增加。
- 如果未中奖，您的 100 plDai 就会滚到进入下周的抽奖。
- 您可以随时提取与您的 aDai 余额相等的普通 Dai。

奖池由所有借出彩票存款产生的利息汇聚而成，类似上面的借贷案例。

尝试 PoolTogether

兑换代币 {#swaps}

以太坊上有数千种代币。 去中心化交易所 (DEX) 可让您随时交易不同的代币。 您的资产永远由您掌控。 这就像前往其他国家时要兑换货币。 但 DeFi 版本从来不关闭。 市场全年全天候开放，技术支持可保证始终有人接受交易。

例如，如果您想使用无损彩票 PoolTogether（如上所述），则需要类似 Dai 或 USDC 的代币。 DEX 允许您将 ETH 换成这些代币，并在完成交易后再换回来。

查看代币兑换

高级交易 {#trading}

对于喜欢加强控制的交易者来说，还有更多高级选项。 限价单、永续合约、保证金交易等都可实现。 通过去中心化交易，您可以获得全球流动性，市场永远不会关闭，而且您始终控制着自己的资产。

使用中心化交易所时，您必须在交易前存入资产，并相信交易所会妥善处理这些资产。 当您存入资产时，它们会面临风险，因为中心化交易所对黑客来说是诱人目标。

查看更多交易 dapp

增长您的投资组合 {#investing}

以太坊上有一些基金管理产品，它们会尝试根据您选择的策略为您的投资组合增值。 自动完成而且对每个人都开放，没有管理人员从利润中抽成。

一个很好的示例是 DeFi 指数基金 (DPI)。 这是一个自动调整的基金，以确保您的组合总是包含市值最高的 DeFi 代币 。 您无需管理任何细节，并且可以随时从基金中提款。

查看投资 dapp

为您的想法提供资金 {#crowdfunding}

以太坊是一个理想的众筹平台：

- 潜在的资助者可以来自任何地方 - 以太坊及其代币向世界上任何地方的任何人开放。
- 公开透明，募捐者可以证明已经筹集了多少资金。 您甚至可以在以后的工作中追踪资金的使用情况。
- 募捐者可以设置自动退款，例如，如果有未达到特定的最后期限和最低金额，就可以自动退款。

查看更多众筹 dapp

二次方融资 {#quadratic-funding}

以太坊是开源软件，到目前为止，很多工作都由社区资助。 这催生了一种有趣的新型募资模式：二次方融资。 这有可能改进我们在未来为各种公共产品募资的方式。

二次方募资确保获得最多资金的项目是那些具有最独特需求的项目。 换句话说，就是那些能够改善大多数人生活的项目。 其运作方式如下：

1. 有一个相应的捐助资金库。
2. 开始一轮公共融资。
3. 人们可以通过捐资来表明对一个项目的需求。
4. 一旦这轮结束，匹配池的资金就会分配给项目。 那些需求最独特的项目会从匹配池中获得最高金额。

这意味着获得 100 笔 1 美元捐款的项目 A 最终可能比获得 1 笔 1 万美元捐款的项目 B 获得更多资金（取决于匹配池的大小）。

关于二次方融资的更多信息

保险 {#insurance}

去中心化保险旨在使保险更便宜、支付速度更快且更加透明。 随着自动化程度的提高，保险的价格更加低廉，赔付的速度也快了很多。 用来决定您的索赔的数据完全透明。

就像任何软件一样，以太坊产品也可能出现漏洞。 因此，目前该领域中的许多保险产品都致力于保护用户免受资金损失。 然而，有一些项目开始为日常生活建立保障。 Etherisc 的作物保护就是一个很好的例子，其目的是保护肯尼亚的小农户对抗干旱和洪水侵袭。 去中心化保险可以为那些经常被传统保险拒之门外的农民提供更便宜的保险。

查看保险 dapp

聚集者和投资组合管理 {#aggregators}

拥有了这么多种服务，您需要一种方法来跟踪跟踪所有投资、贷款和交易。 有许多产品可以让您集中协调所有 DeFi 活动。 这就是 DeFi 开放架构的魅力所在。 团队可以创建界面，您不仅可以看到各个产品中的余额，也可以使用其功能。 当您探索更多的 DeFi 时，可能会发现这个很有用。

查看投资组合 dapp

DeFi 如何运作？ {#how-defi-works}

DeFi 使用加密货币和智能合约提供服务，无需中介。 在现代金融体系下，金融机构充当交易的担保人。 因为您的资产通过这些机构流通，从而为这些机构赋予了巨大的权力。 世界上还有数十亿人甚至无法使用银行账户。

在 DeFi，智能合约取代了交易中的金融机构。 智能合约是一种以太坊帐户，可以持有资金，并可以根据某些条件发送/退还资金。 智能合约上线时，没有人可以改变其有效期，它会始终按程序运行。

一份旨在发放补贴或零用钱的合同可以被编程为：每周五从 A 帐户向 B 帐户汇款。 只有帐户 A 拥有所需资金，它才会这样做。 没有人可以改变智能合约并将帐户 C 添加为收款人以窃取资金。

合约也是公开的，任何人都可以检查和审核。 这意味着不良合约往往会很快受到社会监督。

这确实意味着，目前需要信任以太坊社区中能够读取代码的技术员。 基于开源的社区有助于控制开发者，但是随着时间的推移，这种需求将逐渐减少，因为智能合约变得更易于阅读。并且开发了其他方法来证明代码的可信赖性。

以太坊与 DeFi {#ethereum-and-defi}

以太坊为 DeFi 提供了良好基础，原因如下：

- 没有人拥有以太坊或部署在以太坊上的智能合约 - 这让每个人都有机会使用 DeFi。 这也意味着任何人都不能改变规则。
- DeFi 产品都使用相同的语言：以太坊。 这意味着许多产品能够无缝合作。 您可以在一个平台上借出代币，并通过完全不同的应用程序在不同的市场上交换带利息的代币。 这就像能够在银行兑现忠诚度积分。
- 代币和加密货币内置在以太坊这个共享账本上的 - 记录交易和所有权是以太坊的事情。
- 以太坊可实现完全财务自由 - 大多数产品永远不会保管您的资金，让您掌控一切。

您可以把 DeFi 看成分层结构：

1. 区块链 - 以太坊包含交易历史和帐户状态。
2. 资产 - ETH 和其他代币。
3. 协议 - 智能合同 提供功能，例如，允许去中心化资产借贷的服务。
4. 应用程序 - 我们用来管理和访问协议的产品。

构建 DeFi {#build-defi}

DeFi 是一场开源行动。 DeFi 协议和应用都是开放的，您可以自行检查、分叉和创新。 由于这个分层堆栈（他们都共享相同的基础区块链和资产），协议可以混合和匹配以解锁独特的组合机会。

关于构建 dapp 的更多信息

延伸阅读 {#futher-reading}

DeFi 数据 {#defi-data}

- DeFi Pulse
- DeFi Prime
- DeFi Llama

DeFi 文章 {#defi-articles}

- 什么是去中心化金融？ – *ETHHub，定期更新*
- DeFi 初学者指南 - *Sid Coelho-Prabhu，2020 年 1 月 6 日*

视频 {#videos}

- Finematics - 去中心化金融教育 - *关于 DeFi 的视频*
- The Defiant - _ DeFi 基础：在这个偶尔令人困惑的空间里，您开始需要了解的一切。_
- Whiteboard Crypto _ - 什么是 DeFi？_

社区 {#communities}

- DeFi Llama Discord 服务器
- DeFi Pulse Discord 服务器

---

title: 使用 Create-eth-app 启动您的 dapp 前端开发
description: 如何使用 Create-eth-app 及其功能的概述
author: "Markus Waas"
tags:

- "create-eth-app"
- "前端"
- "javascript"
- "ethers.js"
- "图表"
- "aave"
- "compound"
- "uniswap"
- "sablier"

  skill: 初学者
  lang: zh
  sidebar: true
  published: 2020-04-27
  source: soliditydeveloper.com
  sourceUrl: https://soliditydeveloper.com/create-eth-app

上一次，我们了解 Solidity 大的框架时，已经提到了 create-eth-app 。 现在，您将了解如何使用它，它集成了哪些功能以及如何对其进行扩展等内容。 这个应用程序由 Sablier创始人 Paul Razvan Berg 启动，它将启动您的前端开发，并且具有多个可选集成供您选择。

安装 {#installation}

安装需要 Yarn 0.25 或更高版本 (`npm install yarn --global`)。 安装就像运行程序一样简单：

```
yarn create eth-app my-eth-app
cd my-eth-app
yarn react-app:start
```

这是正后台使用 create-react-app。 要查看您的应用程序，请打开 `http://localhost:3000/`。 当您准备好部署到生产环境中时，使用 yarn build 创建一个缩小的捆绑包。 一个简单的托管它的方法是使用 Netlify。 您可以创建一个 GitHub 存储库，将其添加到 Netlify，设置构建命令，您就完成了！ 您的应用程序将被托管并可供所有人使用。 所有这些都是免费的。

功能 {#features}

React 和 create-react-app {#react--create-react-app}

首先，我们来了解应用程序的核心：React 和 *create-react-app* 带来的所有附加功能。 如果您不想集成以太坊，那么仅使用它是一个很好的选择。 React 本身使构建交互式 UI 变得非常容易。 它可能不像 Vue 那样方便初学者，但仍然被广泛使用。 它具有更多的功能，最重要的是还有数千个附加库可供选择。 *create-react-app* 也使它非常容易开始使用，具有的功能包括：

- React、JSX、ES6、TypeScript、Flow 语法支持。
- ES6 之外的语言附加功能，如对象扩展运算符。
- 自动添加前缀的 CSS，所以您不需要 -webkit- 或其他前缀。
- 快速交互式单元测试运行程序，内置对覆盖率报告的支持。
- 对常见错误发出警告的实时开发服务器。
- 一个用于捆绑 JS、CSS 和图片的构建脚本，带有哈希值和资源映射。

尤其是 *create-eth-app* 正在使用新的 hooks effect。 这是一种编写强大而又非常小巧的所谓功能组件的方法。 关于如何在 *create-eth-app* 中使用 Apollo，请看下面关于 Apollo 的部分。

Yarn Workspace {#yarn-workspaces}

Yarn Workspace允许您拥有多个包， 但可以从根目录对他们进行管理，而且可以使用 `yarn install` 一次性安装所有依赖项。 这对于较小的附加包，例如智能合约地址/ABI 管理（关于您在哪里部署了哪些智能合约以及如何与它们通信的信息）或图集成等，尤其有意义，这两个包都是 `create-eth-app` 的一部分。

ethers.js {#ethersjs}

虽然 Web3 仍被广泛使用，但 ethers.js 作为一种替代方案，在过去一年中获得了更多的关注，并且成为集成到 *create-eth-app* 的一员。 您可以使用这个操作，将它更改为 Web3，或者考虑升级为 ethers.js v5 ，该版本即将完成测试阶段。

图表 {#the-graph}

与 Restful API 相比，GraphQL 是处理数据的另一种方式。 与 Resful Api 相比 它们有几个优势，特别是对于去中心化的区块链数据来说更是如此。 如果您对这背后的原因感兴趣，可以看看 GraphQL 将为去中心化网络提供动力。

通常您会直接从您的智能合约中获取数据。 想要读取上次交易的时间吗？ 只需调用 `MyContract.methods.latestTradeTime().call()`，它可以从以太坊的节点获取数据到您的 Dapp 中。 但如果您需要数百个不同的数据点，该怎么办？ 这将导致数以百计的数据被提取到节点，每次都需要一个 RTT，使您的 Dapp 变得缓慢而低效。 一个变通的办法是在您的合约中设置一个取数器调用函数，一次性返回多个数据。 但这并不总是理想的。

然后您可能对历史数据也感兴趣。 您不仅想知道上次交易的时间，还想知道自己做过的所有交易的时间。 使用 *create-eth-app* 子图包，阅读文档并使其适合您自己的合约。 如果您正在寻找受欢迎的智能合约，甚至可能已经有了一个子图。 可以查看子图浏览器。

您有了一个子图后，您可以在 Dapp 中编写一个简单的查询来检索所有重要的区块链数据，包括您需要的历史数据，只需获取一次即可。

Apollo {#apollo}

由于 Apollo Boost 集成，您可以轻松将图集成到您的 React Dapp 中。 特别是在使用 React hooks 和 Apollo 时，获取数据就像在您的组件中写一个 GraphQl 查询一样简单：

```
const { loading, error, data } = useQuery(myGraphQlQuery)

React.useEffect(() => {
  if (!loading && !error && data) {
    console.log({ data })
  }
}, [loading, error, data])
```

模板 {#templates}

在顶部，您可以从几个不同的模板中进行选择。 到目前为止，您可以使用 Aave、Comp、Unifap 或 sablier 集成。 它们都增加了重要的服务智能合约地址以及预先制作的子图集成。 只需将模板添加到创建命令，例如 `yarn create eth-app my-eth-app --with-template aave`。

Aave {#aave}

Aave 是一个去中心化的货币借贷市场。 存款人向市场提供流动性以赚取被动收入，而借款人则可以利用抵押物进行借贷。 Aave 的一个独特功能是那些 闪电贷 允许您在没有任何抵押品的情况下借钱， 只要您在一次交易中返还贷款。 例如，这对于在套利交易中为您提供额外的现金很有用。

为您赢得利益的交易代币被称为 *aTokens*。

当您选择将 Aave 与 *create-eth-app* 集成时，您将获得子图集成。 Aave 使用 The Graph，并且已经在 Ropsten and Mainnet in raw or formatted form。

Aave 闪电贷备忘录 – “是啊，如果我的闪电贷可以保留超过 1 笔交易，那就太好了”

Compound {#compound}

Compound 类似于 Aave。 集成中已包含新的 Compound v2 Subgraph。 在这里，赚取利益的代币竟然被称为* cTokens*。

Uniswap {#uniswap}

Uniswap 是一个去中心化的交易所 (DEX)。 流动性供应商可以通过为交易双方提供所需的代币或 ETH 来赚取费用。 它正在被广泛使用，因此对于非常多的各种代币来说，它的流动性是最高的当中的一个。 您可以轻松地将其集成到您的 Dapp 中，例如允许用户将其 ETH 换成 DAI 。

遗憾的是，在撰写本文时，集成仅针对 Uniswap v1，而不是 刚刚发布的 v2。

Sablier {#sablier}

Sablier 允许用户进行流支付。 完成最初的设置后，您实际上是不断地得到您的货币，而不是在某个支付日，并且不需要进行进一步的管理。 该集成包含了它自己的子图。

接下来是什么？ {#whats-next}

如果您对 *create-eth-app* 有任何疑问，请访问 Sablier 社区服务器，在那里您可以与 *create-eth-app* 的作者取得联系。 作为接下来的第一步，您可能希望集成一个 UI 框架，例如 Material UI，为您实际需要的数据编写 GraphQL 查询并设置部署。

title: 使用事件记录智能合约中的数据
description: 智能合约事件的简介，以及如何利用事件来记录数据
author: "jdourlens"
tags:

- "智能合约"
- "remix"
- "solidity"
- "事件"
  skill: 中级
  lang: zh
  sidebar: true
  published: 2020-04-03
  source: 以太坊开发团队
  sourceUrl: https://ethereumdev.io/logging-data-with-events/
  address: "0x19dE91Af973F404EDF5B4c093983a7c6E3EC8ccE"

在 solidity 中，事件是智能合约可触发的调度信号。 去中心化应用或其他任何连接到以太坊 JSON-PRC API 的程序，都可以监听这些事件，并执行相应操作。 可以建立事件的索引，以便稍后可以搜索到事件历史记录。

在撰写这篇文章之时，以太坊区块链上最常见的事件是由 ERC20 代币转账时触发的 Transfer 事件。

```
event Transfer(address indexed from, address indexed to, uint256 value);
```

事件签名在合约代码内声明，并且可以使用 emit 关键字来触发。 例如，transfer 事件记录了谁发起了转账(*from*)，转账给谁(*to*)，以及转账的代币数转账(*value*)。

我们再次回到 Counter 智能合约，决定在每次值发生变化时进行记录。 由于这个合约不是为了部署，而是作为基础，通过扩展来构建另一个合约：因此它被称为抽象合约。 在我们 counter 示例中，它将类似于如下：

```
pragma solidity 0.5.17;
```

```
contract Counter {

    event ValueChanged(uint oldValue, uint256 newValue);

    // Private variable of type unsigned int to keep the number of counts
    uint256 private count = 0;

    // Function that increments our counter
    function increment() public {
        count += 1;
        emit ValueChanged(count - 1, count);
    }

    // Getter to get the count value
    function getCount() public view returns (uint256) {
        return count;
    }

}
```

注意：

- **第 5 行**：我们声明了事件及其包含的内容、旧值以及新值。
- **第 13 行**：当我们增加 count 变量的值时，我们会触发事件。

如果我们现在部署合约并调用 increment 函数，如果您在名为 logs 的数组内单击新交易，我们将看到 Remix 会自动显示它。

Remix截屏

日志在调试智能合约时非常有用，另一方面，如果您构建一个不同人使用的应用，并且使分析更容易跟踪和了解您的智能合约的使用情况，那么日志也是非常重要的手段。 交易生成的日志会显示常见的区块浏览器中，并且，举例来说，您也可以使用它们来创建链外脚本，用于侦听特定的事件，并且这些事件发生时采取相应操作。

title: 离线数据完整性的默克尔证明
description: 在链上确保链下数据的完整性
author: Ori Pomerantz
tags:

- "默克尔"
- "完整性"
- "存储"

skill: 高级
lang: zh
sidebar: true
published: 2021-12-30

简介 {#introduction}

理想的情况下，我们想要将所有东西存储在以太坊存储器中。这些数据存储于数以千计的计算机中 且有极高的可用性（数据不会被审查）和完整性（不能在未经授权的情况下修改数据）， 但存储一个 32 字节的词一般需要花费 20,000 燃料。 在撰写此教程时，该费用 等于 6.60 美元。 每字节 21 美分对许多应用程序来说都过于昂贵。

为了解决这个问题，以太坊生态系统开发了许多以去中心化方式存储数据的 代替方法。 通常情况下， 需要在可用性与价格之间进行取舍。 然而，完整性通常可以得到保证。

本篇文章中，您将学习**如何**在以下情况下确保数据的完整性：不将数据存储在区块链上，而是使用 Merkle 证明。

工作原理 {#how-does-it-work}

理论上，我们只需要存储链上数据的哈希值，然后将所有数据发送到需要这些数据的交易中。 但是，这会非常昂贵。 1 字节的数据通过交易发送的成本约 16 个燃料，目前约 0.5 美分，或每千字节约 5 美元。 每兆字节 5,000 美元，这对于许多应用程序来说仍然太昂贵，而且还没有加上对数据进行哈希计算的费用。

解决办法是反复对不同的数据子集进行哈希计算。对于不需要发送的数据，只要发送哈希值即可。 默克尔树是一种树型数据结构，每个节点值都是其下方节点的哈希值：

默克尔树

根哈希是唯一需要存储在链上的部分。 为了证明一个特定值，您需要提供所有与之合并才能获取根的哈希值。 例如，要证明 `C`，您需要提供 `D`、`H(A-B)` 和 `H(E-H)`。

C 值证明

实现 {#implementation}

此处提供示例代码。

链下代码 {#off-chain-code}

在这篇文章中，我们使用 JavaScript 进行链下计算。 大多数去中心化应用程序的 JavaScript 中都有链下组件。

创建默克尔根 {#creating-the-merkle-root}

首先，我们需要向区块链提供默克尔根。

```
const ethers = require("ethers")
```

我们使用源自以太币包的哈希函数。

```
// The raw data whose integrity we have to verify. The first two bytes a
// are a user identifier, and the last two bytes the amount of tokens the
// user owns at present.
const dataArray = [
  0x0bad0010, 0x60a70020, 0xbeef0030, 0xdead0040, 0xca110050, 0x0e660060,
  0xface0070, 0xbad00080, 0x060d0091,
]
```

例如，将每条数据编码为单个 256 位整数，并以非 JSON 的格式存储。 然而，这意味着在提取合约中的数据时，可以大大减少处理工作，从而显著降低了燃料成本。 您可以在链上读取 JSON 格式的数据，但请尽量避免这么做。

```
// The array of hash values, as BigInts
const hashArray = dataArray
```

在此例中，我们的数据开始就是 256 位，因此不需要处理。 如果我们使用更复杂的数据结构，如字符串，我们需要确保先取数据的哈希值并获取哈希数组。 请注意，还有一个原因是我们不关心用户是否知道其他用户的信息。 否则，我们将不得不进行散列计算，使用户 1 不知道用户 0 的值，用户 2 不知道用户 3 的值，等等。

```
const pairHash = (a, b) =>
  BigInt(ethers.utils.keccak256("0x" + (a ^ b).toString(16).padStart(64, 0)))
```

以太币哈希函数的预期结果是一个带十六进制数字的 JavaScript 字符串，如 `0x60A7`，并用另一个相同结构的字符串响应。 然而，对于代码的其他部分，使用 `BigInt` 类型更为容易，所以我们将转换为一个十六进制字符串，然后再转回 BigInt。

此函数是对称的（a xor b 的哈希值）。 这意味着在检查默克尔证明时，我们不必烦恼是将证明中的值放在计算值之前还是之后。 默克尔证明在链上进行检查，所以链上的步骤越少越好。

```
// The value to denote that a certain branch is empty, doesn't
// have a value
const empty = 0n
```

如果证明值的数量不是 2 的整数次方，我们需要处理空白分支。 该程序的处理方式是将 0 作为占位符。

缺少分支的默克尔树

```
// Calculate one level up the tree of a hash array by taking the hash of
// each pair in sequence
const oneLevelUp = (inputArray) => {
  var result = []
  var inp = [...inputArray] // To avoid over writing the input

  // Add an empty value if necessary (we need all the leaves to be
  // paired)
  if (inp.length % 2 === 1) inp.push(empty)

  for (var i = 0; i < inp.length; i += 2)
    result.push(pairHash(inp[i], inp[i + 1]))

  return result
} // oneLevelUp
```

这个函数在默克尔树中通过对当前层级的数值进行散列计算来上升一个层级。 请注意，这不是最有效的代码。我们本来可以避免重复输入并在循环中适当地加上 `hashEmpty`，但此代码是针对可读性的优化。

```
const getMerkleRoot = (inputArray) => {
  var result

  result = [...inputArray]

  // Climb up the tree until there is only one value, that is the
  // root.
  //
  // If a layer has an odd number of entries the
```

```
  // code in oneLevelUp adds an empty value, so if we have, for example,
  // 10 leaves we'll have 5 branches in the second layer, 3
  // branches in the third, 2 in the fourth and the root is the fifth
  while (result.length > 1) result = oneLevelUp(result)

  return result[0]
}
```

要获得根，向上升到只有一个剩余值。

创建一个默克尔证明 {#creating-a-merkle-proof}

默克尔证明是将证明可返回默克尔根的值一起进行散列计算。 要证明的数值往往来自其他数据，因此这里更愿意单独提供，而不是作为代码的一部分。

```
// A merkle proof consists of the value of the list of entries to
// hash with. Because we use a symmetrical hash function, we don't
// need the item's location to verify the proof, only to create it
const getMerkleProof = (inputArray, n) => {
    var result = [], currentLayer = [...inputArray], currentN = n

    // Until we reach the top
    while (currentLayer.length > 1) {
        // No odd length layers
        if (currentLayer.length % 2)
            currentLayer.push(empty)

        result.push(currentN % 2
               // If currentN is odd, add with the value before it to the proof
            ? currentLayer[currentN-1]
               // If it is even, add the value after it
            : currentLayer[currentN+1])
```

我们对 `(v[0],v[1])`、`(v[2],v[3])` 等进行散列计算。 因此，对于偶数值，我们需要下一个值，而奇数值则需要上一个值。

```
        // Move to the next layer up
        currentN = Math.floor(currentN/2)
        currentLayer = oneLevelUp(currentLayer)
    }   // while currentLayer.length > 1

    return result
}   // getMerkleProof
```

链上代码 {#off-chain-code}

最后，我们有核查证明的代码。 链上代码用 Solidity 编写而成。 优化在这里更为重要，因为燃料相对昂贵。

```
//SPDX-License-Identifier: Public Domain
pragma solidity ^0.8.0;

import "hardhat/console.sol";
```

编写此代码时，我使用的是安全帽开发环境，它使我们在开发过程中可以获得 Solidity 的控制台输出。

```
contract MerkleProof {
    uint merkleRoot;

    function getRoot() public view returns (uint) {
      return merkleRoot;
    }

    // Extremely insecure, in production code access to
    // this function MUST BE strictly limited, probably to an
    // owner
    function setRoot(uint _merkleRoot) external {
      merkleRoot = _merkleRoot;
    }   // setRoot
```

为默克尔根设置和获取函数。 在生产系统中，让每个人都更新 Merkle 根是一个*非常糟糕的主意*。 这里这样做是为了简化示例代码。 **不要在一个数据完整性非常重要的系统上执行。**

```
    function pairHash(uint _a, uint _b) internal pure returns(uint) {
      return uint(keccak256(abi.encode(_a ^ _b)));
    }
```

此函数会生成一个配对哈希值。 它只是 Solidity 中 `pairHash` 函数的 JavaScript 代码。

注意：这是另一个优化可读性的情况。 基于函数定义，也许可以将数据存储为 `bytes32` 类型值并避免转换。

```
    // Verify a Merkle proof
    function verifyProof(uint _value, uint[] calldata _proof)
        public view returns (bool) {
      uint temp = _value;
      uint i;

      for(i=0; i<_proof.length; i++) {
        temp = pairHash(temp, _proof[i]);
      }

      return temp == merkleRoot;
    }

}  // MarkleProof
```

在数学符号中，默克尔验证看起来像这样：`H(proof_n, H(proof_n-1, H(proof_n-2, proof_n-2, ... H(proof_1, H(proof_0, value))...)))`。 此代码实现了默克尔证明。

默克尔证明和卷叠很难混淆 {#merkle-proofs-and-rollups}

默克尔与卷叠很难很好地配合。 原因在于，卷叠会将所有交易数据写于 L1 上，但在 L2 进行处理。 发送交易默克尔证明的成本平均达 638 个燃料/层（目前，调用数据中一字节花费 16 个燃料（若不为零），如果为零，则需花费 4 个燃料）。 如果我们的数据包含 1024 个字，默克尔证明需要 10 层，或者总共 6380 个燃料。

例如在 Optimism 中，写入 L1 的燃料成本约为 100 gwei，写入 L2 的成本为 0.001 gwei（这里按正常价格计，可能因为网络拥挤而提高）。 所以对于 L1 上 1 个燃料的成本，我们可能需要在 L2 上花费 10 万个燃料的处理费用。 假定我们不用重写存储器，这意味着我们可以用一个 L1 燃料的价格将五个字写入 L2 上的存储器中。 对于单个默克尔证明，我们可以将全部 1024 个字写入存储器（假定可以在链上计算出来，而非在交易中提供），并且还剩下大部分燃料。

结论 {#conclusion}

在实际工作中，您可能永远不能独立实现默克尔树。 有很多广为人知并经过审核的软件库可供使用，一般来说，最好不要自己独立实现密码原语。 但我希望你们现在能够更好地理解默克尔证明，并能够决定何时值得使用。

请注意，虽然默克尔证明能维持*完整性*，但并不维持*可用性*。 虽然任何人都不能拿走您的资产可以让您安心，但如果数据存储方决定禁止您访问数据资产，您也不能构建一个默克尔树来访问它们。 所以默克尔树最好和某种去中心化存储器一起使用，例如星际文件系统。

title: 如何通过Geth运行轻节点
description: 如何使用 Geth 下载、安装和运行轻客户端。
author: "Brian Gu"
tags:

- "客户端"
- "geth"
- "节点"
  skill: 入门
  lang: zh
  sidebar: true
  published: 2022-03-04

您可能会对运行以太坊节点感兴趣。 最简单的实现方式就是下载、安装和运行 Geth。 通过 Geth，我们只需数分钟便能设置并运行一个轻节点。

轻量级客户端需要不到 400MB 的存储空间，同时仍允许与以太坊状态进行完全交互。 轻客户端从远程对等点检索数据，因此与其他同步模式相比，某些查询可能需要更长的时间来响应。

有关不同同步模式之间差异的说明，请阅读我们的节点和客户端开发者文档。

安装并运行 {#install-and-run}

首先，安装 Geth。

安装 Geth 后，您可以通过在终端窗口中运行以下命令以“light”模式运行以太坊节点：

```
geth --syncmode light
```

一旦启动，Geth 将开始连接到以太坊上的其他节点——称为“对等节点”。 连接到对等节点的过程可能需要一段时间。

当您的 Geth 节点有足够的 peers 时，它将从链上的新块中导入标头。

当新的区块头不再有“age”时，Geth 将同步到链的头部。

停止和重启节点 {#stopping-and-restarting-your-node}

您可以随时按 `CTRL`+`C` 停止您的节点。

重新启动节点时，Geth 将需要几分钟来下载自上次运行节点以来创建的块头。

启用 HTTP-RPC 服务器 {#enable-the-http-rpc-server}

启用 HTTP-RPC 服务器可让您将以太坊节点连接到其他软件，如钱包、浏览器扩展或自定义软件库。

您可以在启动 Geth 时通过运行以下命令来启用 HTTP-RPC 服务器：

```
geth --syncmode light --http
```

启用后，运行 `curl http://127.0.0.1:8545`。 这应该不会报告错误。

允许远程连接 {#allow-remote-connections}

要允许远程主机连接到您的节点，请使用以下命令启动 Geth：

```
geth --syncmode light --http --http.addr 0.0.0.0
```

注意：这假设没有进程阻止对您的 localhost 的请求，例如防火墙。

Geth JavaScript 控制台 {#geth-javascript-console}

Geth 有一个内置的 JavaScript 控制台和一个名为 web3js 的 JavaScript API，您可以使用它与您的节点进行交互。

要使用 JavaScript 控制台运行：

```
geth attach
```

该控制台允许与以太坊直接交互。 例如，运行 `eth.blockNumber` 命令将打印最新的已知区块编号。

完整 web3js 文档

主网和测试网 {#mainnet-and-testnets}

Geth 默认在 以太坊主网 上运行你的节点。

通过在终端中运行以下命令之一，也可以使用 Geth 在 公共测试网络 之一上运行节点：

```
geth --syncmode light --ropsten
geth --syncmode light --rinkeby
geth --syncmode light --goerli
```

区块链和 EVM 的数据存储在哪里？ {#where-is-the-blockchain-and-evm-data-stored}

Geth 用于存储原始区块链数据的目录取决于您的操作系统。 运行 Geth 后，查找如下所示的信息：

```
INFO [11-18|14:04:47] Allocated cache and file handles
database=/Users/bgu/Library/Ethereum/testnet/geth/lightchaindata cache=768 handles=128
```

`“database=”`后面的路径应该会告诉您区块链数据存储在您的机器上的何处。 如果您运行的是完整节点，则此目录将包含有关已提交到区块链的每个区块的所有数据。 因为我们正在运行轻节点，此目录仅包含区块头。

这里值得强调的是，这是区块链能够维持活跃度的最低限度。 区块链的完整内容和 EVM 状态存储在以太网络中的每个完整节点上，存储在与您计算机上的目录非常相似的目录中。

延伸阅读 {#further-reading}

- 详细了解不同的网络。
- 运行全节点

---

title: 如何通过刷写MicroSD卡将您的Raspberry Pi 4变为一个节点
description: 刷写您的树莓派 4，插入以太网电缆，连接固态硬盘并打开设备电源，将树莓派 4 转变为运行执行层或共识层的完整以太坊节点（信标链/验证者）
author: "EthereumOnArm"
tags:

- "客户端"
- "执行层"
- "共识层"
- "节点"
  lang: zh
  sidebar: true
  skill: 中级
  published: 2020-05-07
  source: r/ethereum
  sourceUrl: https://www.reddit.com/r/ethereum/comments/gf3nhg/ethereum_on_arm_raspberry_pi_4_images_release/

TL;DR：刷写您的树莓派 4，插入以太网电缆，连接固态硬盘并打开设备电源，将树莓派 4 转变为运行执行层或共识层的完整以太坊节点（信标链/验证者）

了解有关以太坊升级的信息

首先是一些背景。 如您所知，运行 Raspberry Pi 4 镜像时，我们已经遇到了一些内存问题[1]，因为 Raspbian 操作系统仍然是 32 位操作系统[2]（至少用户区是如此）。 虽然我们更愿意坚持使用官方操作系统，但我们得出的结论是，为了解决这些问题，我们需要迁移原生的 64 位操作系统。

此外，共识客户端不支持 32 位二进制文件，因此使用 Raspbian 会阻止树莓派 4 运行共识层节点（以及质押的可能性）。

因此，几经测试，我们现将发布基于 64 位 Ubuntu 20.04 [3] 的两个不同映像：执行层版和共识层版。

基本上，两者都是相同的映像，并且包含基于 Raspbian 的映像的相同功能。 但它们被设置为默认运行执行层或共识层软件。

程序负责所有必要的步骤，从设置环境和格式化 SSD 磁盘到安装和运行以太坊软件，以及启动区块链同步。

主要特性 {#main-features}

- 基于 Ubuntu 20.04 64 位
- 自动完成 USB 磁盘分区和格式化
- 在 Armbian 工作的基础上增加了交换内存（ZRAM 内核模块 + 一个交换文件）[7]。

- 根据 MAC 哈希值将主机名改为像“ethnode-e2a3e6fe”这样的名称。
- 将软件作为 systemd 服务运行并开始同步区块链
- 包括一个用于安装和升级以太坊软件的 APT 存储库
- 包括一个基于 Grafana/Prometheus 的监控仪表板

包含的软件 {#software-included}

两个映像都包含相同的软件包，它们之间唯一的区别是执行版默认运行 Geth，而共识版则默认运行 Prysm 信标链。

执行客户端 {#execution-clients}

- Geth[8]：1.9.13 (官方二进制文件)
- Parity[9]：2.7.2（交叉编译）。
- Nethermind [10]：1.8.28（交叉编译）。
- Hyperledger Besu[11]：1.4.4（已编译）。

共识客户端 {#consensus-clients}

- Prysm[12]：1.0.0-alpha6（官方二进制文件）。
- Lighthouse[13]：0.1.1（已编译）。

以太坊框架 {#ethereum-framework}

- Swarm[14]：0.5.7（官方二进制文件）。
- Raiden Network[15]：0.200.0~rc1（官方二进制文件）。
- IPFS[16]：0.5.0（官方二进制文件）。
- Statusd[17]：0.52.3（已编译）。
- Vipnode[18]：2.3.3（官方二进制文件）。

安装指南和使用方法 {#installation-guide-and-usage}

推荐的硬件和设置 {#recommended-hardware-and-setup}

- Raspberry 4（B 型）- 4GB
- MicroSD 卡（最小 16 GB，10 级）
- SSD USB 3.0 磁盘（见存储部分）
- 电源
- 以太网电缆
- 30303 端口转发（执行层）和 13000 端口转发（共识层）[4]
- 带散热片和风扇的机箱（可选，但强烈推荐）
- USB 键盘、显示器和 HDMI 电缆（微型 HDMI）（可选）。

存储 {#storage}

你将需要一个固态硬盘来运行以太坊客户端（没有固态硬盘，就绝对没有机会同步以太坊区块链）。 有两种选择：

- 使用 USB 便携式 SSD 磁盘，如三星 T5 便携式 SSD。
- 使用带有 SSD 磁盘的 USB3.0 外置硬盘盒。 在我们的例子中，我们使用了 Inateck 2.5 硬盘盒 FE2011。 请确保购买带有 UAS 兼容芯片的机箱，特别是这些芯片之一：JMicron（JMS567 或 JMS578）或 ASMedia（ASM1153E）。

在两种情况下，都需避免使用劣质 SSD 磁盘，因为它是节点的关键组成部分，可能会极大地影响性能（及同步时间）。

请记住，需要将磁盘插入 USB 3.0 端口（蓝色）

映像下载和安装 {#image-download-and-installation}

1. 下载执行层和共识层映像 {#1-download-execution-or-consensus-images}

下载执行层映像

sha256 7fa9370d13857dd6abcc8fde637c7a9a7e3a66b307d5c28b0c0d29a09c73c55c

下载共识层映像

sha256 74c0c15b708720e5ae5cac324f1afded6316537fb17166109326755232cd316e

2. 刷写映像 {#2-flash-the-image}

在您的台式机/笔记本电脑中插入 microSD，然后下载文件（例如执行层）。

```
wget https://ethraspbian.com/downloads/ubuntu-20.04-preinstalled-server-arm64+raspi-eth1.img.zip
```

注意：如果您不习惯使用命令行，或者运行的是 Windows，则可以使用Etcher。

打开终端并检查您的 MicroSD 设备名称是否正在运行：

```
sudo fdisk -l
```

您应会看到一个名为 mcblk0 或 sdd 的设备。 解压缩并刷写镜像：

```
unzip ubuntu-20.04-preinstalled-server-arm64+raspi-eth1.img.zip
sudo dd bs=1M if=ubuntu-20.04-preinstalled-server-arm64+raspi-eth1.img of=/dev/mmcblk0 && sync
```

3. 将 MicroSD 插入 Raspberry Pi 4。 连接以太网电缆并插入 USB SSD 硬盘（确保您使用的是蓝色端口）。 {#3-insert-the-microsd-into-the-raspberry-pi-4-connect-an-ethernet-cable-and-attach-the-usb-ssd-disk-make-sure-you-are-using-a-blue-port}

4. 打开设备电源 {#4-power-on-the-device}

Ubuntu 操作系统将在一分钟内启动，但**您将需要等待大约 10 分钟**，以便让脚本执行必要的任务，来将设备变成以太坊节点并重新启动 Raspberry。

根据镜像，您将运行：

- 执行客户端：Geth 作为同步区块链的默认客户端
- 共识客户端：Prysm 作为同步信标链的默认客户端（Prater 测试网）。

5. 登录 {#5-log-in}

您可以通过 SSH 或使用控制台登录（如果连接了显示器和键盘）。

```
User: ethereum
Password: ethereum
```

第一次登录时会提示您更改密码，因此需要登录两次。

6. 为 Geth 打开 30303 端口，如果您正在运行 Prysm 信标链，则打开 13000 端口。 如果不知道如何执行此操作，请搜索 "端口转发"，后跟您的路由器型号。 {#6-open-30303-port-for-geth-and-13000-if-you-are-running-prysm-beacon-chain-if-you-dont-know-how-to-do-this-google-port-forwarding-followed-by-your-router-model}

7. 获取控制台输出 {#7-get-console-output}

您可以通过输入以下命令查看后台发生了什么：

```
sudo tail -f /var/log/syslog
```

恭喜！ 您现在正在您的 Raspberry Pi 4 上运行一个完整的以太坊节点。

同步区块链 {#syncing-the-blockchain}

现在您需要等待区块链同步。 对于执行层，这将需要几天的时间，具体取决于若干因素，但预计最多需要约 5-7 天。

如果运行的是共识层 Prater 测试网，则可以预期约 1-2 天的信标链同步时间。 请记住，您需要稍后设置验证器才能启动权益质押过程。 [如何运行共识层验证器]

监测仪表板 {#monitoring-dashboards}

在这第一个版本中，包括了 3 个基于 Prometheus 的监测仪表板[5] / Grafana[6]，以便监测节点和客户端的数据（Geth 和 Besu）。 您可以通过您的 Web 浏览器访问：

```
URL: http://your_raspberrypi_IP:3000
User: admin
Password: ethereum
```

切换客户端 {#switching-clients}

所有客户端都作为 systemd 服务运行。 这一点很重要，因为如果出现问题，系统将自动重启进程。

Geth 和 Prysm 信标链默认运行 (取决于您正在同步的内容，执行层或共识层)，因此，如果您想切换到其他客户端 (例如从 Geth 到 Nethermind)，您需要先停止并禁用 Geth ，并启用并启动另一个客户端：

```
sudo systemctl stop geth && sudo systemctl disable geth
```

启用和运行执行客户端的命令：

```
sudo systemctl enable besu && sudo systemctl start besu
sudo systemctl enable nethermind && sudo systemctl start nethermind
sudo systemctl enable parity && sudo systemctl start parity
```

共识客户端：

```
sudo systemctl stop prysm-beacon && sudo systemctl disable prysm-beacon
sudo systemctl start lighthouse && sudo systemctl enable lighthouse
```

更改参数 {#changing-parameters}

客户端的配置文件位于/etc/ethereum/目录中。 您可以编辑这些文件，并重启 systemd 服务，以使更改生效。 唯一的例外是 Nethermind，它还有一个在以下位置的主网配置文件：

```
/etc/nethermind/configs/mainnet.cfg
```

区块链客户端的数据存储在以太坊主帐户上，如下所示（注意目录名称前的点）。

执行层 {#execution-layer}

```
/home/ethereum/.geth
/home/ethereum/.parity
/home/ethereum/.besu
/home/ethereum/.nethermind
```

共识层 {#consensus-layer}

```
/home/ethereum/.eth2
/home/ethereum/.eth2validators
/home/ethereum/.lighthouse
```

Nethermind 和 Hyperledger Besu {#nethermind-and-hyperledger-besu}

这两个出色的执行层客户端已经成为 Geth 和 Parity 的绝佳替代方案。 网络的多样性越多越好，所以您可以试一试，为网络健康作出贡献。

两者都需要进一步的测试，所以请随时使用它们并报告您的反馈。

如何运行共识验证器（质押） {#validator}

Prater 测试网信标链同步后，您便可以在同一设备中运行验证器。 您需要遵循这些参与步骤。

第一次，您需要通过运行 "验证器 "二进制文件手动创建一个帐户，并设置一个密码。 一旦您完成了这一步，就可以将密码添加到`/etc/ethereum/prysm-validator.conf`，并将验证器作为一个 systemd 服务启动。

感谢反馈 {#feedback-appreciated}

我们投入了大量的工作，试图将 Raspberry Pi 4 设置为一个完整的以太坊节点，因为我们知道这个设备的庞大用户群可能对网络产生非常积极的影响。

请考虑到这是基于 Ubuntu 20.04 的第一个映像，所以可能会有一些错误。 如果确实出现了错误，请在 GitHub 上开一个 issue 或在 Twitter 上联系我们。

参考： {#references}

1. Geth 反复出现 SIGSEGV 崩溃
2. https://github.com/diglos/ethereumonarm
3. https://ubuntu.com/download/raspberry-pi
4. https://wikipedia.org/wiki/Port_forwarding
5. https://prometheus.io
6. https://grafana.com
7. https://forum.armbian.com/topic/5565-zram-vs-swap/
8. https://geth.ethereum.org
9. https://github.com/openethereum/openethereum * **注意，OpenEthereum 已经废弃并停止维护。**请谨慎使用，最好切换至其他客户端实现。
10. https://nethermind.io
11. https://www.hyperledger.org/projects/besu
12. https://github.com/prysmaticlabs/prysm
13. https://lighthouse.sigmaprime.io
14. https://ethersphere.github.io/swarm-home
15. https://raiden.network
16. https://ipfs.io
17. https://status.im
18. https://vipnode.org

---

title: 智能合约安全清单
description: 编写安全智能合约的推荐工作流程
author: "Trailofbits"
tags:

- "智能合约"
- "安全性"
- "solidity"
  skill: 中等难度
  lang: zh
  sidebar: true
  published: 2020-09-07
  source: 构建安全的合约
  sourceUrl: https://github.com/crytic/building-secure-contracts/blob/master/development-guidelines/workflow.md

---

智能合约开发清单 {#smart-contract-development-checklist}

这是一个高级别的过程，我们建议在您编写智能合约时遵循。

检查已知的安全问题：

- 使用 Slither 审核您的合约。 该程序有 40 多个内置检测器，可以应对常见的脆弱性。 在每次签入时使用新代码运行它，确保它获得一个干净的报告（或使用分类模式来消除某些问题）。

- 使用 Crytic 审核您的合约。 它可以检查 Slither 未涉及的 50 个问题。 Crytic 还可以帮助您的团队成员保持联系，通过在 GitHub 上的拉取请求轻松显示安全问题。

考虑您的智能合约的特殊功能：

- 您的合约是否可以升级？ 使用 `slither-check-upgradeability` 或 Crytic 检查您的可升级代码是否有缺陷。 我们记录了 17 种升级可能出现的问题。
- 您的合约是否声明符合 ERC 的要求？ 使用 `slither-check-erc` 对它们进行审核。 这个工具能立即识别出六个常见规范的偏差。
- 您在 Truffle 中设有单元测试吗？ 将它们添加到 `slither-prop`。 它可以根据您的具体代码自动生成一套强大的安全属性，用于 ERC20 功能。
- 您是否与第三方代币集成？ 在依赖外部合约之前，请先查看我们的 代币集成清单。

目视检查代码的关键安全功能。

- 查看 Slither 的 inheritance-graph printer。 避免以外屏蔽和 C3 线性化问题。
- 查看 Slither 的 function-summary printer。 它报告功能可见性和访问控制。
- 查看 Slither 的 vars-and-auth printer。 它报告了对状态变量的访问控制。

记录关键安全属性，并使用自动化测试生成器对其进行评估：

- 学习 记录代码的安全属性。 一开始是艰难的，但它是实现良好结果的最重要的活动。 这也是使用本教程中任何高级技术的先决条件。
- 在 Solidity 中定义安全属性，用于 Echidna 和 Manticore。 重点是您的状态机、访问控制、算术运算、外部交互和标准一致性。
- 使用 Slither 的 Python API 定义安全属性。 重点是继承、变量依赖项、访问控制和其他结构问题。
- 在每次提交时，使用 Crytic 运行您的属性测试。 Crytic 可以使用和评估安全属性测试，因此团队中的每个人都可以轻松地看到他们通过 GitHub。 测试失败会阻止提交。

最后，请注意自动化工具无法轻易找到的问题：

- 缺乏隐私：当您的事务在池中排队时，其他所有人都可以看到您的事务
- 抢先提前交易
- 加密操作
- 与外部 DeFi 组件的风险互动

寻求帮助 {#ask-for-help}

以太坊的办公时间为每周二下午。 这些 1 小时的 1 对 1 会议是一个机会，您可以向我们询问有关安全的任何问题，使用我们的工具进行故障排除，并获得专家对您当前方法的反馈。 我们将帮助您消化本指南。

加入我们的 Slack: Empire Hacking。 如果您有任何问题，可以随时在 #crytic 和 #ethereum 频道上联系我们。

title: 使用 ethers.js 发送代币
description: 使用 ethers.js 发送代币的初学者入门指南。
author: Kim YongJun
tags:

- "ETHERS.JS"
- "ERC-20"
- "代币"
  skill: 初学者
  lang: zh

sidebar: true
published: 2021-04-06

使用 ethers.js(5.0) 发送代币 {#send-token}

在本教程中，您将学习如何： {#you-learn-about}

- 导入 ethers.js
- 进行代币转账
- 根据网络流量情况设置燃料价格

入门指南 {#to-get-started}

开始前，我们必须先将 ethers.js 库导入我们的 javascript 中 包括 ethers.js(5.0)

安装 {#install-ethersjs}

```
/home/ricmoo> npm install --save ethers
```

浏览器版 ES6

```
<script type="module">
  import { ethers } from "https://cdn.ethers.io/lib/ethers-5.0.esm.min.js"
  // Your code here...
</script>
```

浏览器版 ES3(UMD)

```
<script
  src="https://cdn.ethers.io/lib/ethers-5.0.umd.min.js"
  type="application/javascript"
></script>
```

参数 {#param}

1. `contract_address`：代币合约地址（您想要转账的代币不是以太币时所需的合约地址）
2. `send_token_amount`：您想要发送给接收人的金额
3. `to_address`：接收人的地址
4. `end_account`：发送人的地址
5. `private_ke`：发送者签署交易和实际进行代币转账的私钥

注意 {#notice}

`signTransaction(tx)` 已删除，因为 `sendTransaction()` 在内部执行。

发送程序 {#procedure}

1. 连接到网络（测试网） {#connect-to-network}

设置提供商 (Infura) {#set-provider}

连接到 Ropsten 测试网

```
window.ethersProvider = new ethers.providers.InfuraProvider("ropsten")
```

2. 创建钱包 {#create-wallet}

```
let wallet = new ethers.Wallet(private_key)
```

3. 连接钱包到网络 {#connect-wallet-to-net}

```
let walletSigner = wallet.connect(window.ethersProvider)
```

4. 获取当前燃料价格 {#get-gas}

```
window.ethersProvider.getGasPrice() // gasPrice
```

5. 定义交易 {#define-transaction}

以下定义的变量取决于 `send_token()`

交易参数 {#transaction-params}

1. `send_accoun`：代币发送人的地址
2. `to_address`：代币接收人的地址
3. `send_token_amount`：您想要发送给接收人的金额
4. `gas_limit`：燃料限制
5. `gas_limit`：燃料价格

[请参阅下文了解如何使用]

```
const tx = {
  from: send_account,
  to: to_address,
  value: ethers.utils.parseEther(send_token_amount),
  nonce: window.ethersProvider.getTransactionCount(send_account, "latest"),
  gasLimit: ethers.utils.hexlify(gas_limit), // 100000
  gasPrice: gas_price,
}
```

6. 转账 {#transfer}

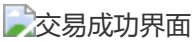
```
walletSigner.sendTransaction(tx).then((transaction) => {
  console.dir(transaction)
  alert("Send finished!")
})
```

使用方法 {#how-to-use}

```
let private_key =
  "41559d28e936dc92104ff30691519693fc753ffbee6251a611b9aa1878f12a4d"
let send_token_amount = "1"
let to_address = "0x4c10D2734Fb76D3236E522509181CC3Ba8DE0e80"
let send_address = "0xda27a282B5B6c5229699891CfA6b90OA716539E6"
let gas_limit = "0x100000"
let wallet = new ethers.Wallet(private_key)
let walletSigner = wallet.connect(window.ethersProvider)
let contract_address = ""
window.ethersProvider = new ethers.providers.InfuraProvider("ropsten")

send_token(
  contract_address,
  send_token_amount,
  to_address,
  send_address,
  private_key
)
```

成功！ {#success}


交易成功界面

send_token() {#send-token-method}

```
function send_token(
  contract_address,
  send_token_amount,
  to_address,
  send_account,
  private_key
) {
  let wallet = new ethers.Wallet(private_key)
  let walletSigner = wallet.connect(window.ethersProvider)

  window.ethersProvider.getGasPrice().then((currentGasPrice) => {
    let gas_price = ethers.utils.hexlify(parseInt(currentGasPrice))
    console.log(`gas_price: ${gas_price}`)

    if (contract_address) {
      // general token send
      let contract = new ethers.Contract(
        contract_address,
        send_abi,
        walletSigner
      )

      // How many tokens?
      let numberOfTokens = ethers.utils.parseUnits(send_token_amount, 18)
      console.log(`numberOfTokens: ${numberOfTokens}`)

      // Send tokens
```

```
      contract.transfer(to_address, numberOfTokens).then((transferResult) => {
        console.dir(transferResult)
        alert("sent token")
      })
    } // ether send
    else {
      const tx = {
        from: send_account,
        to: to_address,
        value: ethers.utils.parseEther(send_token_amount),
        nonce: window.ethersProvider.getTransactionCount(
          send_account,
          "latest"
        ),
        gasLimit: ethers.utils.hexlify(gas_limit), // 100000
        gasPrice: gas_price,
      }
      console.dir(tx)
      try {
        walletSigner.sendTransaction(tx).then((transaction) => {
          console.dir(transaction)
          alert("Send finished!")
        })
      } catch (error) {
        alert("failed to send!!")
      }
    }
  })
}
```

---

title: 使用 Web3 发送交易
description: "这是使用 web3 发送以太坊交易的初学者友好指南。 要向以太坊区块链发送交易，主要有三个步骤：创建、签署和广播。 我们将对三个方面进行讨论。"
author: "Elan Halpern"
tags:

- "交易"
- "web3.js"
- "alchemy"

  skill: 初学者
  lang: zh
  sidebar: true
  published: 2020-11-04
  source: Alchemy 文档
  sourceUrl: https://docs.alchemy.com/alchemy/tutorials/sending-txs

---

这是使用 web3 发送以太坊交易的初学者友好指南。 要向以太坊区块链发送交易，主要有三个步骤：创建、签署和广播。 我们将对这三个方面进行讨论，希望能回答您可能遇到的所有问题！ 在本教程中，我们将使用 Alchemy 将我们的交易发送到以太坊链。 您可以在这里创建一个免费的 Alchemy 帐户。

注意：如果您想集成签名，本指南适用于在您的应用的*后端*上签署您的交易 您在前端的交易，请查看将 Web3 与浏览器提供商集成。

基本概念 {#the-basics}

像大多数区块链开发人员刚开始的时候一样，您可能已经对如何发送交易（应该非常简单）进行了一些研究，然后阅读了大量的指南，发现每个人有不同的解读，让您有点不知所措和困惑。 如果您已上了那条船，就不要担心，我们在某些时候都会这样！ 所以，在开始之前，让我们弄清楚一些事情：

1. Alchemy 不会存储您的私钥 {#alchemy-does-not-store-your-private-keys}

- 这意味着 Alchemy 无法代表您签署和发送交易。 这样做的原因是出于安全考虑。 Alchemy 绝不会要求您分享您的私钥，您也绝不应该与托管节点（或任何人）分享您的私钥。
- 您可以使用 Alchemy 的核心 API 从区块链中读取数据，但要写入它，您需要在通过 Alchemy 发送交易之前使用其他东西对交易进行签名 (这对于任何其他 节点服务)。

2. 什么是 "签名者"？ {#what-is-a-signer}

- 签名者将使用您的私钥为您签署交易。 在本教程中，我们将使用 Alchemy web3 来签署我们的交易，但您也可以使用任何其他 web3 库。
- 在前端，metamask 就是一个签名者的好例子，它将代表您签署和发送交易。

3. 为什么我需要在我的交易上签名？ {#why-do-i-need-to-sign-my-transactions}

- 每个想要在以太坊网络上发送交易的用户都必须在交易上签名（使用他们的私钥），以验证交易的来源是其所声称的那个人。
- 保护这个私钥非常重要，因为拥有这个私钥就可以完全控制您的以太坊帐户，允许您（或任何有权限的人）代表您进行交易。

4. 如何保护我的私钥？ {#how-do-i-protect-my-private-key}

- 有许多方法来保护您的私钥，并使用它来发送交易。 在本教程中，我们将使用一个.env 文件。 然而，你也可以使用一个单独的存储私钥的服务提供器，使用一个密钥库文件，或其他选项。

5. `eth_sendTransaction` 和 `eth_sendRawTransaction` 之间有什么区别？ {#difference-between-send-and-send-raw}

`eth_sendTransaction` 和 `eth_sendRawTransaction` 都是 Ethereum API 函数，用于将交易广播到 Ethereum 网络，以便将其添加到未来的区块中。 它们在处理交易签名的方式上有所不同。

- `eth_sendTransaction` 用于发送*未签名的*交易，这意味着您发送到的节点必须管理您的私钥，以便它能够在将交易广播到链中之前对交易进行签名。 由于 Alchemy 不持有用户的私钥，因此他们不支持这种方法。
- `eth_sendRawTransactions` 用于广播已经签名的交易。 这意味着您首先必须使用 `signTransaction(tx, private_key)`，然后将结果发送到 `eth_sendRawTransaction`。

当使用 web3 时，通过调用函数 web3.eth.sendSignedTransaction 来访问`eth_sendRawTransaction`。

这就是我们将在本教程中使用的内容。

6. Web3 库是什么？ {#what-is-the-web3-library}

- Web3.js 是一个围绕标准 JSON-RPC 调用的封装库，在以太坊开发中使用相当普遍。
- 有许多针对不同语言的 web3 库。 在本教程中，我们将使用 Alchemy Web3，它是用 JavaScript 编写的。 您可以在这里查看其他选项 。

好了，现在我们把这些问题都解决了，让我们继续学习教程。 随时在 Alchemy discord 中提问！

注意： 本指南需要安装 Alchemy 帐户、以太坊地址或 MetaMask 钱包、NodeJs 和 npm。 如果没有，按以下步骤操作：

1. 创建一个免费的 Alchemy 帐户
2. 创建 MetaMask 帐户 (或获取以太坊地址)
3. 按照这些步骤安装 NodeJs 和 NPM

发送交易的步骤 {#steps-to-sending-your-transaction}

1. 在 Rinkeby 测试网上创建一个 Alchemy 程序 {#create-an-alchemy-app-on-the-rinkeby-testnet}

导航到您的 Alchemy 仪表板并创建一个新的应用程序，选择 Rinkeby（或任何其他测试网）作为您的网络。

2. 从 Rinkeby faucet 请求 ETH {#request-eth-from-rinkeby-faucet}

按照 Alchemy Rinkeby 水龙头 上的说明接收 ETH。 确保包含您的 **Rinkeby** 以太坊地址 (来自 MetaMask) 而不是其他网络。 按照说明操作后，请仔细检查您是否已在钱包中收到 ETH。

3. 创建一个新的项目目录，并使用 `cd` 命令进入该目录。 {#create-a-new-project-direction}

从命令行（macs 终端）创建一个新的项目目录并导航到这个目录：

```
mkdir sendtx-example
cd sendtx-example
```

4. 安装 Alchemy Web3（或任何 web3 库）。 {#install-alchemy-web3}

在您的项目目录中运行以下命令，以安装 Alchemy Web3。

```
npm install @alch/alchemy-web3
```

5. 安装 dotenv {#install-dotenv}

我们将使用一个 .env 文件来安全存储我们的 API 密钥和私钥。

```
npm install dotenv --save
```

6. 创建 .env 文件 {#create-the-dotenv-file}

在您的项目目录中创建一个 `.env` 文件并添加以下内容 (替换 "`your-api-url`" 和 "`your-private-key`")

- 要找到您的 Alchemy API URL，请导航到您刚刚在仪表板上创建的应用程序详细信息页面。 点击右上角的"查看密钥"，然后获取 HTTP URL。
- 要使用 MetaMask 查找您的私钥，请查看此 指南。

```
API_URL = "your-api-url"
PRIVATE_KEY = "your-private-key"
```

不要提交 `.env`！ 请确保永远不要与任何人共享或公开您的 `.env` 文件，因为这样做会泄露您的秘密。 如果您使用版本控制，请将您的 `.env` 添加到 gitignore 文件中。

7. 创建 `sendTx.js` 文件 {#create-sendtx-js}

很好，我们在 .env 文件中保护了我们的敏感数据，让我们现在开始编码吧。 对于我们的发送交易示例，我们将把 ETH 发送回 Rinkeby faucet。

创建一个 `sendTx.js` 文件，这是我们将配置和发送我们的示例交易的地方，并在文件中添加以下几行代码:

```
async function main() {
    require('dotenv').config();
    const { API_URL, PRIVATE_KEY } = process.env;
    const { createAlchemyWeb3 } = require("@alch/alchemy-web3");
    const web3 = createAlchemyWeb3(API_URL);
    const myAddress = '0x610Ae88399fc1687FA7530Aac28eC2539c7d6d63' //TODO: replace this
address with your own public address

    const nonce = await web3.eth.getTransactionCount(myAddress, 'latest'); // nonce
starts counting from 0

    const transaction = {
     'to': '0x31B98D14007bDEe637298086988A0bBd31184523', // faucet address to return eth
     'value': 1000000000000000000, // 1 ETH
     'gas': 30000,
     'nonce': nonce,
     // optional data field to send message or execute smart contract
    };

    const signedTx = await web3.eth.accounts.signTransaction(transaction, PRIVATE_KEY);

    web3.eth.sendSignedTransaction(signedTx.rawTransaction, function(error, hash) {
    if (!error) {
      console.log("🎉 The hash of your transaction is: ", hash, "\n Check Alchemy's
Mempool to view the status of your transaction!");
    } else {
      console.log("❗Something went wrong while submitting your transaction:", error)
    }
   });
}

main();
```

请务必将 **line 6** 上的地址替换为您自己的公共地址。

现在，在我们开始运行这段代码之前，让我们先谈谈这里的一些组件。

- `nonce`：nonce 规范用于跟踪从您的地址发送的交易数量。 为了安全起见，我们需要这样做，以防止重放攻击。 要获得从您的地址发送的交易数量，我们使用 getTransactionCount。
- `transaction`：交易对象有几个方面需要我们指定
  - `to`：这是我们要发送 ETH 的地址。 在这种情况下，我们将 ETH 发送回我们最初请求的 Rinkeby faucet。
  - `value`：这是我们希望发送的金额，以 wei 为单位，10^18 wei = 1 ETH。
  - `gas`：有多种方法来确定与您的交易相关的 gas 的正确数量。 Alchemy 甚至有一个矿工费出价的 webhook，当矿工费出价落在某个阈值内时，它可以通知您。 对于主网交易，检查 gas 估算器如 ETH Gas Station以确定要包含的正确 gas 数量是一种很好的做法。 21000 是以太坊操作所使用的最小 gas 数量，所以为了确保我们的交易得以执行，我们在这里输入 30000。
  - `nonce`：参见上面的 nonce 定义。 Nonce 从零开始计数。
  - [OPTIONAL] data：用于在您的转账中发送附加信息，或调用智能合约，余额转账不需要，请查看下面的注释。
- `signedTx`：要签署交易对象，我们将使用 `signTransaction` 方法和我们的 `PRIVATE_KEY`
- `sendSignedTransaction`：一旦我们有一个已签名的交易。 我们可以通过使用 `sendSignedTransaction` 将其发送出去，以包含在后续区块中

关于数据的注释 有两种主要类型的交易可以在以太坊中发送。

- Balance transfer：将 eth 从一个地址发送到另一个地址。 不需要数据字段，但是，如果您想在交易中发送附加信息，您可以在此字段中包含 HEX 格式的信息。
  - 例如，假设我们想将 IPFS 文档的哈希写入以太坊链，以便为其提供不可变的时间戳。 我们的数据字段应该看起来像 data: web3.utils.toHex('IPFS hash')。 现在任何人都可以查询该链并查看该文档是何时添加的。
- Smart contact transaction：在链上执行一些智能合约代码。 在这种情况下，数据字段应包含您希望执行的智能功能以及任何参数。
  - 有关实际示例，请查看此 Hello World 教程。

8. 使用 `节点 sendTx.js` 运行代码 {#run-the-code-using-node-sendtx-js}

返回到您的终端或命令行并运行：

```
node index.js
```

9. 在内存池中查看您的交易 {#see-your-transaction-in-the-mempool}

在您的 Alchemy 仪表板上打开 内存池页面，并通过您创建的应用程序进行过滤，以找到您的交易。 我们可以在这里观看交易从待处理状态转换到被挖矿状态（如果成功），或者如果失败，则从待处理状态转换到被丢弃状态。 确保将其保持在“全部”，这样您就能捕捉到“被挖矿”、“待处理”和“被丢弃”的交易。 您还可以通过寻找发送到地址 `0x31b98d14007bdee6372980898a0bbd31184523` 的交易来搜索您的交易。

要在找到交易后查看交易的详细信息，请选择 tx 哈希值，您将看到这样的视图：

内存池监视器截图

在视图里，您可以通过点击红色圆圈圈出的图标，在 Etherscan 上查看您的交易！

太好了！ 您刚刚使用 Alchemy 发送了您的第一笔以太坊交易 🎉

如有关于本指南的反馈和建议，请在 Alchemy 的 Discord 上给 Elan 留言！

最初发表于 https://docs.alchemyapi.io/tutorials/sending-transactions-using-web3-and-alchemy。

title: 设置 web3.js 以用 JavaScript 操作 Ethereum 区块链。
description: 如何使用智能合约与使用 Solidity 语言的代币进行交互
author: "jdourlens"
tags:

- "web3.js"
- "javascript"

skill: 初学者
lang: zh
sidebar: true
published: 2020-04-11
source: EthereumDev
sourceUrl: https://ethereumdev.io/setup-web3js-to-use-the-ethereum-blockchain-in-javascript/
address: "0x19dE91Af973F404EDF5B4c093983a7c6E3EC8ccE"

在本教程中，我们将了解如何开始使用 web3.js 与以太坊区块链进行交互。 Web3.js 可用于前端和后端，从区块链读取数据或进行交易，甚至部署智能合约。

第一步是将 web3.js 加入到您的项目中。 要在网页中使用它，您可以使用 JSDeliver 等 CDN 直接导入库。

```
<script src="https://cdn.jsdelivr.net/npm/web3@latest/dist/web3.min.js"></script>
```

如果您更喜欢在后端或使用 build 的前端项目中安装库，可以使用 npm 进行安装：

```
npm install web3 --save
```

然后将 Web3.js 导入到 Node.js 脚本或 Browserify 前端项目，您可以使用下面的 JavaScript 代码行：

```
const Web3 = require("web3")
```

现在我们在项目中加入了库，我们需要对其进行初始化。 如果您的项目需要能够与区块链通信。 大部分以太坊库通过 RPC 的调用与节点进行通信。 为了启动我们的 web3 服务提供器，我们会实例化一个 Web3 实例，将服务提供器的 URL 作为构造函数传递。 如果您有一个节点或 ganache 实例在您的计算机上运行，它看起来将是这样：

```
const web3 = new Web3("http://localhost:8545")
```

如果您想直接访问托管节点，您可以使用 Infura 或 Cloudfare 提供的免费节点：

```
const web3 = new Web3("https://cloudflare-eth.com")
```

为了测试我们是否正确配置了 web3 实例，我们将尝试使用 `getBlockNumber` 函数检索最新的区块编号。 此函数接受回调作为参数，并以整数形式返回区块编号。

```
var Web3 = require("web3")
const web3 = new Web3("https://cloudflare-eth.com")

web3.eth.getBlockNumber(function (error, result) {
  console.log(result)
})
```

如果您执行这个程序，它只会打印最新的区块编号：区块链的顶部。 您还可以使用 await/async 函数调用来避免在代码中的嵌套回调：

```
async function getBlockNumber() {
  const latestBlockNumber = await web3.eth.getBlockNumber()
  console.log(latestBlockNumber)
  return latestBlockNumber
}

getBlockNumber()
```

您可以在官方 web3 文档中查看 web3 实例上可用的所有函数。

大多数 Web3 库都是异步的，因为在后台，该库对返回结果的节点进行 JSON RPC 调用。

如果您在浏览器中运行， 一些钱包直接注入一个 web3 实例，并且您应该尽可能地尝试使用它，特别是如果您计划与用户的以太坊地址交互以进行交易。

下面是检测 MetaMask 钱包是否可用的代码段，如果可用，请尝试启用它。 稍后它将允许您读取用户的余额，并使它们能够验证您想让它们在以太坊区块链上进行的交易：

```
if (window.ethereum != null) {
  state.web3 = new Web3(window.ethereum)
  try {
    // Request account access if needed
    await window.ethereum.enable()
    // Acccounts now exposed
  } catch (error) {
    // User denied account access...
  }
}
```

像 Ethers.js 这样的 web3.js 的替代品确实存在，但我们将把所有 JavaScript 教程都集中在 web3.js 上，因为它是在浏览器中与以太坊交互的官方库。 在下一个教程中，我们将了解如何轻松监听区块链上的新传入区块并查看它们包含的内容。

title: 智能合约安全准则
description: 构建您的dapp时要考虑的安全准则清单
author: "Trailofbits"
tags:

- "solidity"
- "智能合约"

- "安全性"
  skill: 中等难度
  lang: zh
  sidebar: true
  published: 2020-09-06
  source: 构建安全的合约
  sourceUrl: https://github.com/crytic/building-secure-contracts/blob/master/development-guidelines/guidelines.md

---

遵循这些高级建议，构建更安全的智能合约。

设计准则 {#design-guidelines}

编写代码之前，务必先讨论行智能合约的设计。

文档和规范 {#documentation-and-specifications}

文档可以在不同级别编写，在执行合约时应进行更新：

- **系统的简要英文描述**，描述合约的行为和所使用的代码库
- **数据模型和架构图**，包括合约的交互概述图和系统的状态转换图。 可以使用Slither 打印机生成相关图示。
- **完整的代码文档**, 可以为 Solidity 使用Natspec 格式。

链上计算与链下计算 {#on-chain-vs-off-chain-computation}

- **保持尽可能多的链下代码。** 保持链上代码层的最小化。 用链下代码的方式对数据进行预处理，使链上验证变得简单。 是否需要已排序列表？ 在链下对列表进行排序，然后在链上只检查其顺序。

升级 {#upgradeability}

我们在我们的博客文章中讨论了不同的升级解决方案。 在编写任何代码之前，请慎重选择是否支持可升级性。 该决定将影响您如何构建我们的代码。 一般来说，我们建议：

- **优先考虑合约迁移而不是可升级性。**迁移系统有许多与可升级系统相同的优点，不存在缺陷。
- **使用数据分离模式而不是 delegatecallproxy 模式。**如果您的项目有明确的抽象分离，则使用数据分离的可升级性只需要进行一些调整。 delegatecallproxy 需要 EVM 专业知识，并且非常容易出错。
- **在部署前记录迁移/升级程序。**如果您不得不在没有任何指导的情况下在压力下做出反应，您就会犯错。 提前写好要遵循的程序。 其中应包括：
  - 启动新的智能合约的调用
  - 密钥存放在哪里以及如何获取它们
  - 如何检查部署！ 开发和测试部署后的脚本。

实施指南 {#implementation-guidelines}

力求简洁。 尽可能使用最简单的解决方案来实现您的目的。 所有的团队成员都应当能够理解解决方案。

功能组成 {#function-composition}

使用便于检查的代码库架构， 避免选择不利于正确性验证的架构。

- 通过多个合同或将相似的功能组合在一起（例如，身份验证、算术等）来**拆分您的系统逻辑**。
- **编写小的函数，并且目的明确。**这将便于审查并允许对单个组件进行测试。

继承 {#inheritance}

- **使继承易于管理。**继承应用于划分逻辑，但是，您的项目应以最小化继承树的深度和宽度为目标。
- **使用 Slother 的inheritance printer检查合约层级结构。**inheritance printer 可帮助您审查层级结构的大小。

事件 {#events}

- **记录所有关键操作。**事件有助于在开发过程中调试合约，并在部署后对其进行监控。

规避已知漏洞 {#avoid-known-pitfalls}

- **了解最常见的安全问题。**有大量的用于常见问题的网络资源，例如Ethernaut CTF、Capture the Ether或Not so smart contracts。
- **了解Solidity 文档中的警告部分，** 警告部分将通知您该语言的非明显行为。

依赖关系 {#dependencies}

- **使用经过充分测试的库。**从经过充分测试的库中导入代码将降低编写错误代码的可能性。 如果您打算编写一个 ERC20 合约，请使用OpenZepelin。
- **使用依赖关系管理器，不要直接复制粘贴代码。** 如果依赖于外部来源，请保持与原始来源的的同步。

测试和验证 {#testing-and-verification}

- **编写详尽的单元测试。**全面的测试套件对于构建高质量的软件至关重要。
- **编写Slither、Echidna和Manticore自定义检查和属性。**自动化工具将帮助确保您的合约安全。 查看本指南的其余部分，了解如何编写高效的检查和属性。
- **使用crytic.io。**Crytic 与 Github 集成，提供对私有 Slither 探测器的访问，并从 Echidna 运行自定义属性检查。

Solidity {#solidity}

- **使用 Solidity 0.5，而不是 0.4 和 0.6。**在我们看来，相较于 0.4，Solidity 0.5 更加安全，具有更好的内部实践。 Solidity 0.6 对于发布级产品而言过于不稳定，还需要时间才能趋于成熟。
- **使用稳定版本进行编译；使用最新版本进行警告检查。** 保证代码中不存在最新版本编译器报告的问题。 然而，Solidity 的发布周期很快，编译器中存在错误，所以我们不推荐最新版本进行部署（请参阅 Slither 的 solc 版本建议）。
- **不要使用内联汇编。**使用汇编需要 EVM 专业知识。 如果对黄皮书没有很好的*理解*，请不要写 EVM 代码。

部署准则 {#deployment-guidelines}

合约开发和部署完成后：

- **监测合约。**关注日志，在合约或钱包发生异常时随时做出反应。
- **将联系信息添加到blockchain-security-contacts。**如果发现安全缺陷，此列表可以帮助第三方与您联系。
- **保护特权用户的钱包。** 如果使用硬件钱包存储密钥，请按照最佳实践进行操作。

- **制定事故响应计划。** 以合约可能发生错误为前提进行考量。 即使合约本身没有错误，攻击者也可能有机会控制合约拥有者的密钥。