以太坊源码解析：accounts

2019-01-24 fatcat22 ethereum 原创 ethereum accounts 源码解析

> 本篇文章分析的源码地址为：https://github.com/ethereum/go-ethereum
> 分支：master
> commit id: 257bfff316e4efb8952fbeb67c91f86af579cb0a

引言

以太坊中，当你要挖矿或发送交易时，都需要一个账号和私钥。众所周知，与普通应用程序的账号密码不同，区块链中私钥一旦丢失，是没有“忘记密码”按钮的；如果被盗，你也不可能“修改密码”。所以私钥在安全性就非常重要。

accounts模块是以太坊的账户管理模块，其实也是一个命令行的钱包。它可以使用本地文件存储加密的私钥信息；也可以连接硬件钱包实现对私钥的创建与管理。在这篇文章里，我们将会对实现这些功能的源码进行介绍和分析。

accounts支持的钱包类型

在accounts中总共支持两大类共4种钱包类型。两大类包括keystore和usbwallet；其中keystore中的私钥存储可以分为加密的和不加密的；usbwallet支持ledger和trenzer两种硬件钱包。如下图所示：

下面我们依次对其作一个介绍。

keystore：本地文件夹

keystore类型的钱包其实是一个本地文件夹目录。在这个目录下可以存放多个文件，每个文件都存储着一个私钥信息。这些文件都是json格式，其中的私钥可以是加密的，也可以是非加密的明文。但非加密的格式已经被废弃了（谁都不想把自己的私钥明文存放在某个文件里）。

keystore的目录路径可以在配置文件中指定，默认路径是<DataDir>/keystore。每一个文件的文件名格式为：UTC--<created_at UTC ISO8601>--<address hex>。例如UTC--2016-03-22T12-57-55--7ef5a6135f1fd6a02593eedc869c6d41d934aef8。

keystore目录和目录内的文件是可以直接拷贝的。也就是说，如果你想把某个私钥转移到别的电脑上，你可以直接拷贝文件到其它电脑的keystore目录。拷贝整个keystore目录也是一样的。

HD：分层确定性（Hierarchical Deterministic）钱包

我们首先解释一下HD（Hierarchical Deterministic）的概念。这个概念的中文名称叫做“分层确定性”，我的理解是这是一种key的派生方式，它可以在只使用一个公钥（我们称这个公钥为主公钥，其对应的私钥称为主私钥）的情

Content

- 引言
- accounts支持的钱包类型
  - keystore：本地文件夹
  - HD：分层确定性（Hierarchical Deterministic）钱
- 目录结构
- 实现框架
- 总结
- Similar Posts
- Comments

况下，生成任意多个子公钥，而这些子公钥都是可以被主私钥控制的。HD的概念最早是从比特币的BIP-32提案中提出来的。

每一个key都有自己的路径，即是是一个派生的key，这一点和keystore类型是一样的。我们先来看一下HD账户的路径格式：

> m / purpose' / coin_type' / account' / change / address_index

这种路径规范不是一下子形成的。虽然BIP-32提出了HD的概念，但实现者自由度比较大，导致相互之间兼容性很差。因此在BIP-43中增加了 `purpose` 字段；而在BIP-44中对路径规范进行了大量的扩展，使其可以用在不同币种上[1]。

在BIP-43中推荐 `purpose` 的值为44'(0x8000002C)；而在BIPSLIP-44中为以太坊类型的 `coin_type` 为配的值为60'(0x8000003c)。所以我们在以太坊中可能看到形如 `m/44'/60'/0'/0` 这样的路径。

在accounts模块中共支持两种HD钱包：Ledger和Trenzer。它们都是非常有名的硬件钱包，有兴趣的朋友可以自己搜索一下，这是不作过多介绍。

目录结构

accounts模块下的源文件比较多，这里不一一说明，只挑一些比较重要的聊一下。

- accounts.go
  accounts.go定义了accounts模块对外导出的一些结构体和接口，包括Account结构体、Wallet接口和Backend接口。其中Account由一个以太坊地址和钱包路径组成；而各种类型的钱包需要实现Wallet和Backend接口来接入账入管理。

- hd.go
  hd.go中定义了HD类型的钱包的路径解析等函数。这个文件中的注释还解析了HD路径一些知识，值得一看。（但我认为它关于哪个BIP提案提出的哪个规范说得不对，比如注释中提到BIP-32定义了路径规范 `m / purpose' / coin_type' / account' / change / address_index` ，这应该是错误的，我们前面提到过，purpose是在BIP-43中提出的，而整个路径规范是在BIP-44中提出的）

- manager.go
  manager.go中定义了Manager结构及其方法。这是accounts模块对外导出的主要的结构和方法之一。其它模块（比如cmd/geth中）通过这个结构体提供的方法对钱包进行管理。

- url.go
  这个文件中的代码定义了代表以太坊钱包路径的URL结构体及相关函数。与hd.go中不同的是，URL结构体中保存了钱包的类型（scheme）和钱包路径的字符串形式的表示；而hd.go中定义了HD钱包路径的类型（非字符串类型）的解析及字符串转换等方法。
- keystore
  这是一个子目录，此目录下的代码实现了keystore类型的钱包。
  - account_cache.go
    此文件中的代码实现了accountCache结构体及方法。accountCache的功能是在内存中缓存keystore钱包目录下所有账号信息。无论keystore目录中的文件无何变动（新建、删除、修改），accountCache都可以在扫描目录时将变动更新到内存中。
  - file_cache.go
    此文件中的代码实现了fileCache结构体及相关代码。与account_cache.go类似，file_cache.go中实现了对keystore目录下所有文件的信息的缓存。accountCache就是通过fileCache来获取文件变动的信息，进而得到账号变动信息的。
  - key.go
    key.go主要定义了Key结构体及其json格式的marshal/unmarshal方式。另外这个文件中还定义了通过keyStore接口将Key写入文件中的函数。keyStore接口中定义了Key被写入文件的具体细节，在passphrase.go和plain.go中都有实现。
  - keystore.go
    这个文件里的代码定义了KeyStore结构体及其方法。KeyStore结构体实现了Backend接口，是

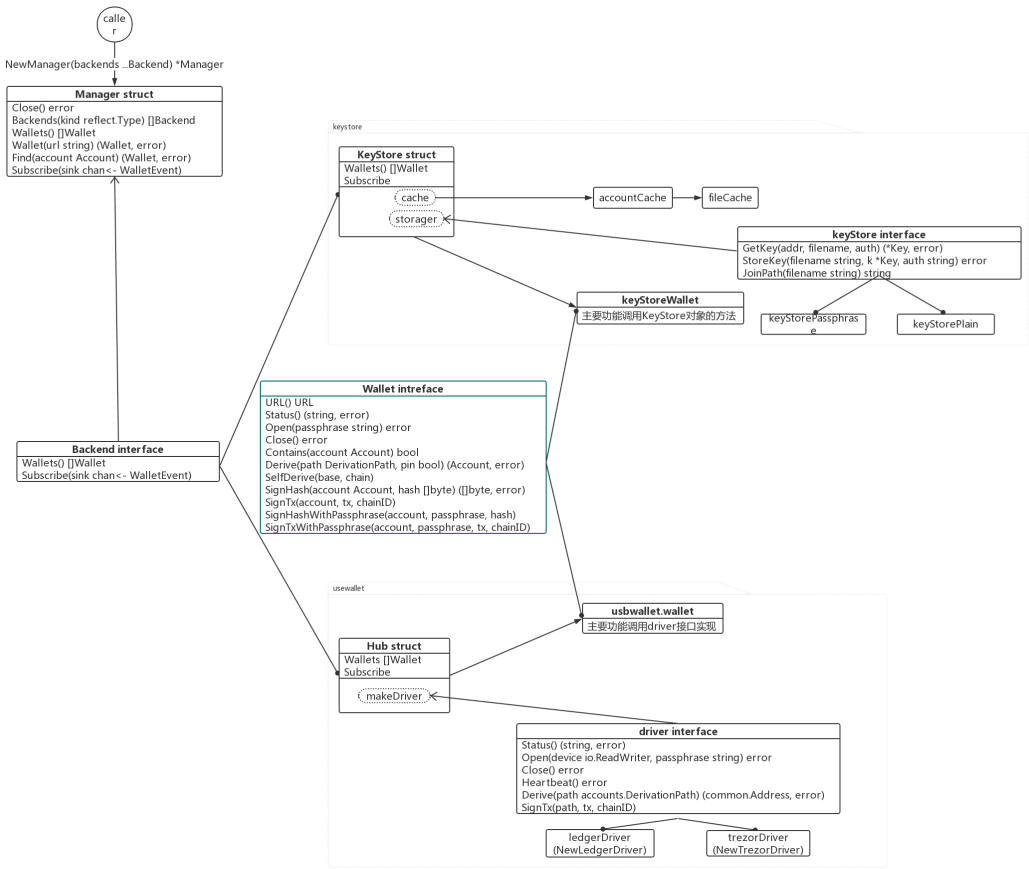
keystore类型的钱包的后端实现。同时它也实现了keystore类型钱包的大多数功能。

- passphrase.go
  passphrase.go中定义了keyStorePassphrase结构体及其方法。keyStorePassphrase结构体是对keyStore接口（在key.go文件中）的一种实现方式，它会要求调用者提供一个密码，从而使用aes加密算法加密私钥后，将加密数据写入文件中。
- plain.go
  这个文件中的代码定义了keyStorePlain结构体及其方法。keyStorePlain与keyStorePassphrase类似，也是对keyStore接口的实现。不同的是，keyStorePlain直接将密码明文存储在文件中。目前这种方式已被标记弃用且整个以太坊项目中都没有调用这个文件里的函数的地方，确实谁也不想将自己的私钥明文存在本地磁盘上。
- wallet.go
  wallet.go中定义了keystoreWallet结构体及其方法。keystoreWallet是keystore类型的钱包的实现，但其功能基本都是调用KeyStore对象实现的。
- watch.go
  watch.go中定义了watcher结构体及其方法。watcher用来监控keystore目录下的文件，如果文件发生变化，则立即调用account_cache.go中的代码重新扫描账户信息。但watcher只在某些系统下有效，这是文件的build注释：`// +build darwin,!ios freebsd linux,!arm64 netbsd solaris`

- usbwallet
  这是一个子目录，此目录下的代码实现了对通过usb接入的硬件钱包的访问，但只支持ledger和trezor两种类型的硬件钱包。
  - hub.go
    hub.go中定义了Hub结构体及其方法。Hub结构体实现了Backend接口，是usbwallet类型的钱包的后端实现。
  - ledger.go
    ledger.go中定义了ledgerDriver结构体及其方法。ledgerDriver结构体是driver接口的实现，它实现了与ledger类型的硬件钱包通信协议和代码。
  - trezor.go
    trezor.go中定义了trezorDriver结构体及其方法。与ledgerDriver类似，trezorDriver结构体也是driver接口的实现，它实现了与trezor类型的硬件钱包的通信协议和代码。
  - wallet.go
    wallet.go中定义了wallet结构体。wallet结构体实现了Wallet接口，是硬件钱包的具体实现。但它内部其实主要调用硬件钱包的driver实现相关功能。

实现框架

accounts模块中的代码虽然说不上多，但还是有些复杂的。它的复杂性主要来源于多种接口的使用，以及不同层次的概念。我们首先看一张框架图，然后再对难以理解的地方进行说明。



从图中可以看出，钱包类型分两种：keystore和usbwallet，它们都作为Backend接口被Manager对象管理。在不同的Backend内部，又有不同的Wallet接口的具体实现。调用者通过生成不同的Backend对象后，使用Manager进行统一管理。

这里我们要特别强调一下Wallet接口及其具体实现的含义，因为我自己曾发生误解。我曾误以为一个Wallet对象是用来管理多个账户的。但在accounts模块中，一个Wallet对象（如keyStoreWallet）仅仅只代表了一个账户，而不是多个。拿keyStoreWallet来说，它只代表了一个账户，也只代表了一个文件。

下面我们逐一对上面进行一下说明。在usbwallet类型的Backend中，`Hub`结构体代表了这种类型的后端钱包，它主要由函数NewLedgerHub或NewTrezorHub创建，根据不同的`driver`，它可以代表不同的硬件钱包。这也是图中`makeDriver`字段想要表达的意思。在ledger.go和trezor.go中，有各自的driver实现，它们使用不同的协议，通过device.Write和io.ReadAll对设备进行读写以实现通信。

在keystore类型的Backend中，`KeyStore`结构体代表了这种类型的后端钱包，它主要由函数NewKeyStore或NewPlaintextKeyStore创建，根据不同的`keyStore`，它可以代表加密或非加密的keystore类型的钱包。图中`storager`就是这个意思。非加密的存储比较简单且已经被弃用，我们不再多说。加密类型的存储是由`keyStorePassphrase`结构体实现的。它通过aes算法对私钥进行加密后写入文件中。

accountCache和fileCache用来缓存账户信息。fileCache在每次扫描keystore目录时，会与当前内存中保存的文件信息进行对比，找出新建、删除、修改的文件；accountCache利用这些信息更新内存中的账号信息。

总结

这篇文章主要介绍了以太坊accounts模块的实现架构。accounts模块是以太坊项目中的账号和私钥管理模块，它不仅存储私钥和账户信息，还用来对数据进行加密。

accounts模块的第一个概念是Backend，它代表的是不同的钱包类型。accounts内部有两种类型的Backend：本地目录（keystore）和硬件钱包（usbwallet）。本地目录的方式支持将私钥加密后存储在本地目录中；硬件钱包支持ledger和trezor两种。
accounts的第二个概念是Wallet，它内部的Wallet接口代表着对一个账号和私钥的管理，而不是“钱包”的概念。我觉得这有些歧义，改成”KeyPair”会更好。

以上是我对以太坊accounts模块的分析，如有错误还望不吝指正。

注释：

[1] 参考：http://www.huamo.online/2018/04/20/Ethereum源码阅读笔记-accounts-2-分层确定性钱包

Similar Posts

- 初识联盟链1: Fabric 是什么
- PBFT代码篇：fabric 中的 PBFT 实现
- 实用拜占庭容错算法（PBFT）
- 拜占庭将军问题
- 数据结构与算法：B树
- 以太坊源码解析：evm

上一篇 以太坊源码解析：trie下篇

下一篇 以太坊源码解析：共识算法之clique

Comments

Comments

Login

There are no comments posted yet. Be the first one!

Post a new comment

Enter text right here!

Comment as a Guest, or login:

Name

Displayed next to your comments.

Email

Not displayed publicly.

Website (optional)

If you have a website, link to it here.

Subscribe to None

Submit Comment