accounts包实现了以太坊客户端的钱包和账户管理。以太坊的钱包提供了keyStore模式和usb两种钱包。同时以太坊的 合约的ABI的代码也放在了account/abi目录。 abi项目好像跟账户管理没有什么关系。 这里暂时只分析了账号管理的接口。 具体的keystore和usb的实现代码暂时不会给出。

账号是通过数据结构和接口来定义了

数据结构

账号

```
// Account represents an Ethereum account located at a specific location defined
// by the optional URL field.
// 一个账号是20个字节的数据。 URL是可选的字段。
type Account struct {
    Address common.Address `json:"address"` // Ethereum account address derived fro
m the key
    URL     URL            `json:"url"`     // Optional resource locator within a b
ackend
}

const (
    HashLength    = 32
    AddressLength = 20
)
// Address represents the 20 byte address of an Ethereum account.
type Address [AddressLength]byte
```

钱包。钱包应该是这里面最重要的一个接口了。 具体的钱包也是实现了这个接口。
钱包又有所谓的分层确定性钱包和普通钱包。

```
// Wallet represents a software or hardware wallet that might contain one or more
// accounts (derived from the same seed).
// Wallet 是指包含了一个或多个账户的软件钱包或者硬件钱包
type Wallet interface {
    // URL retrieves the canonical path under which this wallet is reachable. It is
    // user by upper layers to define a sorting order over all wallets from multipl
e
    // backends.
    // URL 用来获取这个钱包可以访问的规范路径。 它会被上层使用用来从所有的后端的钱包来排序
。
    URL() URL
```

```
    // Status returns a textual status to aid the user in the current state of the
    // wallet. It also returns an error indicating any failure the wallet might hav
e
    // encountered.
    // 用来返回一个文本值用来标识当前钱包的状态。 同时也会返回一个error用来标识钱包遇到的
任何错误。
    Status() (string, error)

    // Open initializes access to a wallet instance. It is not meant to unlock or
    // decrypt account keys, rather simply to establish a connection to hardware
    // wallets and/or to access derivation seeds.
    // Open 初始化对钱包实例的访问。这个方法并不意味着解锁或者解密账户，而是简单地建立与硬
件钱包的连接和/或访问衍生种子。
    // The passphrase parameter may or may not be used by the implementation of a
    // particular wallet instance. The reason there is no passwordless open method
    // is to strive towards a uniform wallet handling, oblivious to the different
    // backend providers.
    // passphrase参数可能在某些实现中并不需要。 没有提供一个无passphrase参数的Open方法的
原因是为了提供一个统一的接口。
    // Please note, if you open a wallet, you must close it to release any allocate
d
    // resources (especially important when working with hardware wallets).
    // 请注意，如果你open了一个钱包，你必须close它。不然有些资源可能没有释放。 特别是使用
硬件钱包的时候需要特别注意。
    Open(passphrase string) error

    // Close releases any resources held by an open wallet instance.
    // Close 释放由Open方法占用的任何资源。
    Close() error

    // Accounts retrieves the list of signing accounts the wallet is currently awar
e
    // of. For hierarchical deterministic wallets, the list will not be exhaustive,
    // rather only contain the accounts explicitly pinned during account derivation
.
    // Accounts用来获取钱包发现了账户列表。 对于分层次的钱包， 这个列表不会详尽的列出所有
的账号， 而是只包含在帐户派生期间明确固定的帐户。
    Accounts() []Account

    // Contains returns whether an account is part of this particular wallet or not
.
    // Contains 返回一个账号是否属于本钱包。
    Contains(account Account) bool

    // Derive attempts to explicitly derive a hierarchical deterministic account at
    // the specified derivation path. If requested, the derived account will be add
ed
```

```
    // to the wallet's tracked account list.
    // Derive尝试在指定的派生路径上显式派生出分层确定性帐户。 如果pin为true，派生帐户将被添加到钱包的跟踪帐户列表中。
    Derive(path DerivationPath, pin bool) (Account, error)

    // SelfDerive sets a base account derivation path from which the wallet attempts
    // to discover non zero accounts and automatically add them to list of tracked
    // accounts.
    // SelfDerive设置一个基本帐户导出路径，从中钱包尝试发现非零帐户，并自动将其添加到跟踪帐户列表中。
    // Note, self derivaton will increment the last component of the specified path
    // opposed to decending into a child path to allow discovering accounts starting
    // from non zero components.
    // 注意，SelfDerive将递增指定路径的最后一个组件，而不是下降到子路径，以允许从非零组件开始发现帐户。
    // You can disable automatic account discovery by calling SelfDerive with a nil
    // chain state reader.
    // 你可以通过传递一个nil的ChainStateReader来禁用自动账号发现。
    SelfDerive(base DerivationPath, chain ethereum.ChainStateReader)

    // SignHash requests the wallet to sign the given hash.
    // SignHash 请求钱包来给传入的hash进行签名。
    // It looks up the account specified either solely via its address contained within,
    // or optionally with the aid of any location metadata from the embedded URL field.
    //它可以通过其中包含的地址（或可选地借助嵌入式URL字段中的任何位置元数据）来查找指定的帐户。
    // If the wallet requires additional authentication to sign the request (e.g.
    // a password to decrypt the account, or a PIN code o verify the transaction),
    // an AuthNeededError instance will be returned, containing infos for the user
    // about which fields or actions are needed. The user may retry by providing
    // the needed details via SignHashWithPassphrase, or by other means (e.g. unlock
    // the account in a keystore).
    // 如果钱包需要额外的验证才能签名(比如说 需要密码来解锁账号， 或者是需要一个PIN 代码来验证交易。)
    // 会返回一个AuthNeededError的错误，里面包含了用户的信息，以及哪些字段或者操作需要提供。
    // 用户可以通过 SignHashWithPassphrase来签名或者通过其他手段(在keystore里面解锁账号)
    SignHash(account Account, hash []byte) ([]byte, error)

    // SignTx requests the wallet to sign the given transaction.
    // SignTx 请求钱包对指定的交易进行签名。
    // It looks up the account specified either solely via its address contained within,
```

```
    // or optionally with the aid of any location metadata from the embedded URL fi
eld.
    //
    // If the wallet requires additional authentication to sign the request (e.g.
    // a password to decrypt the account, or a PIN code o verify the transaction),
    // an AuthNeededError instance will be returned, containing infos for the user
    // about which fields or actions are needed. The user may retry by providing
    // the needed details via SignTxWithPassphrase, or by other means (e.g. unlock
    // the account in a keystore).
    SignTx(account Account, tx *types.Transaction, chainID *big.Int) (*types.Transa
ction, error)

    // SignHashWithPassphrase requests the wallet to sign the given hash with the
    // given passphrase as extra authentication information.
    // SignHashWithPassphrase请求钱包使用给定的passphrase来签名给定的hash
    // It looks up the account specified either solely via its address contained wi
thin,
    // or optionally with the aid of any location metadata from the embedded URL fi
eld.
    SignHashWithPassphrase(account Account, passphrase string, hash []byte) ([]byte
, error)

    // SignTxWithPassphrase requests the wallet to sign the given transaction, with
 the
    // given passphrase as extra authentication information.
    // SignHashWithPassphrase请求钱包使用给定的passphrase来签名给定的transaction
    // It looks up the account specified either solely via its address contained wi
thin,
    // or optionally with the aid of any location metadata from the embedded URL fi
eld.
    SignTxWithPassphrase(account Account, passphrase string, tx *types.Transaction,
 chainID *big.Int) (*types.Transaction, error)
}
```

后端 Backend

```
// Backend is a "wallet provider" that may contain a batch of accounts they can
// sign transactions with and upon request, do so.
// Backend是一个钱包提供器。 可以包含一批账号。他们可以根据请求签署交易，这样做。
type Backend interface {
    // Wallets retrieves the list of wallets the backend is currently aware of.
    // Wallets获取当前能够查找到的钱包
    // The returned wallets are not opened by default. For software HD wallets this
    // means that no base seeds are decrypted, and for hardware wallets that no act
ual
    // connection is established.
    // 返回的钱包默认是没有打开的。
```

```
    // The resulting wallet list will be sorted alphabetically based on its interna
l
    // URL assigned by the backend. Since wallets (especially hardware) may come an
d
    // go, the same wallet might appear at a different positions in the list during
    // subsequent retrievals.
    //所产生的钱包列表将根据后端分配的内部URL按字母顺序排序。 由于钱包（特别是硬件钱包）可能会打开和关闭，所以在随后的检索过程中，相同的钱包可能会出现在列表中的不同位置。
    Wallets() []Wallet

    // Subscribe creates an async subscription to receive notifications when the
    // backend detects the arrival or departure of a wallet.
    // 订阅创建异步订阅，以便在后端检测到钱包的到达或离开时接收通知。
    Subscribe(sink chan<- WalletEvent) event.Subscription
}
```

manager.go

Manager是一个包含所有东西的账户管理工具。 可以和所有的Backends来通信来签署交易。

数据结构

```
// Manager is an overarching account manager that can communicate with various
// backends for signing transactions.
type Manager struct {
    // 所有已经注册的Backend
    backends map[reflect.Type][]Backend // Index of backends currently registered
    // 所有Backend的更新订阅器
    updaters []event.Subscription       // Wallet update subscriptions for all back
ends
    // backend更新的订阅槽
    updates  chan WalletEvent           // Subscription sink for backend wallet cha
nges
    // 所有已经注册的Backends的钱包的缓存
    wallets  []Wallet                   // Cache of all wallets from all registered
 backends
    // 钱包到达和离开的通知
    feed event.Feed // Wallet feed notifying of arrivals/departures
    // 退出队列
    quit chan chan error
    lock sync.RWMutex
}
```

创建Manager

```
// NewManager creates a generic account manager to sign transaction via various
// supported backends.
func NewManager(backends ...Backend) *Manager {
    // Subscribe to wallet notifications from all backends
    updates := make(chan WalletEvent, 4*len(backends))

    subs := make([]event.Subscription, len(backends))
    for i, backend := range backends {
        subs[i] = backend.Subscribe(updates)
    }
    // Retrieve the initial list of wallets from the backends and sort by URL
    var wallets []Wallet
    for _, backend := range backends {
        wallets = merge(wallets, backend.Wallets()...)
    }
    // Assemble the account manager and return
    am := &Manager{
        backends: make(map[reflect.Type][]Backend),
        updaters: subs,
        updates:  updates,
        wallets:  wallets,
        quit:     make(chan chan error),
    }
    for _, backend := range backends {
        kind := reflect.TypeOf(backend)
        am.backends[kind] = append(am.backends[kind], backend)
    }
    go am.update()

    return am
}
```

update方法。 是一个goroutine。会监听所有backend触发的更新信息。 然后转发给feed.

```
// update is the wallet event loop listening for notifications from the backends
// and updating the cache of wallets.
func (am *Manager) update() {
    // Close all subscriptions when the manager terminates
    defer func() {
        am.lock.Lock()
        for _, sub := range am.updaters {
            sub.Unsubscribe()
        }
        am.updaters = nil
        am.lock.Unlock()
    }()
```

```
    // Loop until termination
    for {
        select {
        case event := <-am.updates:
            // Wallet event arrived, update local cache
            am.lock.Lock()
            switch event.Kind {
            case WalletArrived:
                am.wallets = merge(am.wallets, event.Wallet)
            case WalletDropped:
                am.wallets = drop(am.wallets, event.Wallet)
            }
            am.lock.Unlock()

            // Notify any listeners of the event
            am.feed.Send(event)

        case errc := <-am.quit:
            // Manager terminating, return
            errc <- nil
            return
        }
    }
}
```

返回backend

```
// Backends retrieves the backend(s) with the given type from the account manager.
func (am *Manager) Backends(kind reflect.Type) []Backend {
    return am.backends[kind]
}
```

订阅消息

```
// Subscribe creates an async subscription to receive notifications when the
// manager detects the arrival or departure of a wallet from any of its backends.
func (am *Manager) Subscribe(sink chan<- WalletEvent) event.Subscription {
    return am.feed.Subscribe(sink)
}
```

对于node来说。是什么时候创建的账号管理器。

```
// New creates a new P2P node, ready for protocol registration.
```

```go
func New(conf *Config) (*Node, error) {
    ...
    am, ephemeralKeystore, err := makeAccountManager(conf)




func makeAccountManager(conf *Config) (*accounts.Manager, string, error) {
    scryptN := keystore.StandardScryptN
    scryptP := keystore.StandardScryptP
    if conf.UseLightweightKDF {
        scryptN = keystore.LightScryptN
        scryptP = keystore.LightScryptP
    }

    var (
        keydir    string
        ephemeral string
        err       error
    )
    switch {
    case filepath.IsAbs(conf.KeyStoreDir):
        keydir = conf.KeyStoreDir
    case conf.DataDir != "":
        if conf.KeyStoreDir == "" {
            keydir = filepath.Join(conf.DataDir, datadirDefaultKeyStore)
        } else {
            keydir, err = filepath.Abs(conf.KeyStoreDir)
        }
    case conf.KeyStoreDir != "":
        keydir, err = filepath.Abs(conf.KeyStoreDir)
    default:
        // There is no datadir.
        keydir, err = ioutil.TempDir("", "go-ethereum-keystore")
        ephemeral = keydir
    }
    if err != nil {
        return nil, "", err
    }
    if err := os.MkdirAll(keydir, 0700); err != nil {
        return nil, "", err
    }
    // Assemble the account manager and supported backends
    // 创建了一个KeyStore的backend
    backends := []accounts.Backend{
        keystore.NewKeyStore(keydir, scryptN, scryptP),
    }
    // 如果是USB钱包。 需要做一些额外的操作。
```

```
    if !conf.NoUSB {
        // Start a USB hub for Ledger hardware wallets
        if ledgerhub, err := usbwallet.NewLedgerHub(); err != nil {
            log.Warn(fmt.Sprintf("Failed to start Ledger hub, disabling: %v", err))
        } else {
            backends = append(backends, ledgerhub)
        }
        // Start a USB hub for Trezor hardware wallets
        if trezorhub, err := usbwallet.NewTrezorHub(); err != nil {
            log.Warn(fmt.Sprintf("Failed to start Trezor hub, disabling: %v", err))
        } else {
            backends = append(backends, trezorhub)
        }
    }
    return accounts.NewManager(backends...), ephemeral, nil
}
```