accounts包实现了以太坊客户端的钱包和账户管理。

账号的数据结构：

    type Account struct {

     Address common.Address `json:"address"` // Ethereum account address derived from the key

     URL URL `json:"url"` // Optional resource locator within a backend

    }

钱包interface，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 struct {

        URL() URL        // 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。

        Status() (string, error)    // 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。

        Open(passphrase string) error    //Open 初始化对钱包实例的访问。如果你open了一个钱包，你必须close它。

        Close() error    // Close 释放由Open方法占用的任何资源。

        Accounts() []Account    // 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。

        Derive(path DerivationPath, pin bool) (Account, error)    //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。

        SelfDerive(base DerivationPath, chain ethereum.ChainStateReader)    //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。

        SignHash(account Account, hash []byte) ([]byte, error)    // SignHash 请求钱包来给传入的hash进行签名。

        SignTx(account Account, tx \*types.Transaction, chainID \*big.Int) (\*types.Transaction, error)    // SignTx 请求钱包对指定的交易进行签名。

        SignHashWithPassphrase(account Account, passphrase string, hash []byte) ([]byte, error)    // SignHashWithPassphrase请求钱包使用给定的passphrase来签名给定的hash

        SignTxWithPassphrase(account Account, passphrase string, tx \*types.Transaction, chainID \*big.Int) (\*types.Transaction, error)    // SignHashWithPassphrase请求钱包使用给定的passphrase来签名给定的transaction

    }

后端Backend，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。

    type Backend struct {

        Wallets() []wallet    // Wallets获取当前能够查找到的钱包

        Subscribe(sink chan <- WalletEvent) event.Subscription    // 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。

    }

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。

以太坊账户定义，在accounts.keystore.key.go中定义

以太坊账户主要包含三条信息，ID，地址和公私钥对。

 type Key struct {

    ID uuid.UUID

    Address    common.Address

    PrivateKey    \*ecdsa.PrivateKey

}

以太坊创建账户的流程：

1，用户输入一个密码    （passphrase string）

2，内部通过椭圆曲线算法随机生成一个公私密钥对 （internal.ethapi.api newAccount方法）

3，对公钥hash得到地址

4，对密码使用scrypt算法加密,得到加密后的密码derivedKey

5，用derivedKey的对私钥使用AES-CTR算法加密，得到密文cipherText

6，对derivedKey和cipherText进行hash得到mac，这个mac实际上起到了签名的作用，在解密的时候去验证合法性，防止别人篡改

7，保存账号地址和加密过程中写死或随机生成的参数到json文件中，也就是就是上面的文件

创建账号的核心代码：（accounts.keystore.keystore\_passphrase.go）

中的EncryptKey方法

**func** EncryptKey(key \*Key, auth string, scryptN, scryptP int) ([]byte, error)

其中，key是加密的账号，包含ID，公私钥，地址

     auth是用户输入的密码

     scryptN,是scrypt算法中的N

     scryptP，scrypt算法中的P

derivedKey, err := scrypt.Key(authArray, salt, scryptN, *scryptR*, scryptP, *scryptDKLen*)

对用户名输入的密码使用scrypt加密，返回一个derivedKey