Geth-query报错总结

> 本文主要介绍Geth-query工具编写过程中的一些问题，特此记录。

并发程序文件夹创建失败

对于对并发程序对于新建一个文件夹，正常逻辑是首先检查一个文件夹是否存在，不存在就创建一个文件夹。但是可能存在一种情况：在检查csv_data目录的时候可能会在检查`os.IsNotExist(err)`时候文件不存在，但是`os.Mkdir(absDir, 0777)`的时候该`csv_data`文件已经由别的协程创建成功，因此writer指针为空。

```
    if _, err := os.Stat(absDir); os.IsNotExist(err) {
        err = os.Mkdir(absDir, 0777)
        if err != nil {
            Logger
            return nil, err
        }
    }

    preDir := absDir + "/" + prefix
    if _, err := os.Stat(preDir); os.IsNotExist(err) {
        err = os.Mkdir(preDir, 0777)
        if err != nil {
            return nil, err
        }
    }

    folderDir := fmt.Sprintf("%v/%v", preDir,
folderSeq*FILE_NUMBER_IN_FOLDER*BLOCK_NUMBER_IN_FILE)
    if _, err := os.Stat(folderDir); os.IsNotExist(err) {
        err = os.Mkdir(folderDir, 0777)
        if err != nil {
            return nil, err
        }
    }
```

for range语句循环变量为值传递

对于情况1能够将`transfers`的地址加入切片，情况2将循环局部变量的地址加入切片，情况3新建一个对象将`transferTmp`的值赋值给新对象，并且将新对象的地址加入切片。

```
        for _, transferTmp := range transfers {
            transfer := (*valueTransfer)(unsafe.Pointer(&transferTmp))
            // 1.
            // data.transfers = append(data.transfers, (*valueTransfer)
(unsafe.Pointer(&transfers[i])))
            // Logger.Infof("value: %v",
data.transfers[len(data.transfers)-1].value)
            // 2.
            // data.transfers = append(data.transfers, (*valueTransfer)
(unsafe.Pointer(&transfer)))
            data.transfers = append(data.transfers,
&valueTransfer{transfer.depth, transfer.transactionHash, transfer.src,
transfer.srcBalance, transfer.dest, transfer.destBalance, transfer.value,
transfer.kind, transfer.snapshot})
        }
```

go 指针内存释放问题

1. 对于全局变量要通过指针保存局部变量的值，最好的办法是将局部变量指针指向的局部地址内容重新new一边，然后赋给全局变量指针。因为局部变量的值在后面可能发生变化。使用全局变量的指针保存局部变量的指针最后全局变量保存的值具有很大的随机性(**全局变量保存局部变量的某一成员指针，局部变量也会被gc释放**)。

```
        evm.Interpreter().(*EVMInterpreter).GetConfig().Tracer.
(*StructLogger).CaptureTransferState(evm.StateDB.(*state.StateDB), evm.depth,
common.Hash{}, caller.Address(),
            to.Address(), big.NewInt(0).Set(value), "CALL", evm.StateDB.
(*state.StateDB).GetNextRevisionId())
```

上面一段代码中直接保存value的地址，该value地址的值后面可能会发生变化，尽管有指针副本一直指向该处，内存不会释放，但是其值已经改变。具体错误现象是`transfer`中的value，只有transactions中的value和gas fee的value是正确的。其余内部交易的value值都是错误的。因为stack中每次push的value值都是从`integer = interpreter.intPool.get()`中获取的，该pool中的value值是在动态改变。

2. 局部指针变量指向一块较大内存，通过全局指针保存。导致一直有指针指向局部分配的较大内存地址，因此大内存一直不能被go的gc释放，导致内存占用过高。

> 上述问题是由于captureState函数中新建了一些对象，然后这些对象的指针被保存在全局变量的结构体中，导致go gc工具在释放内存时由于captureState中的对象一直有全局对象指针指向其值，导致其内存一直得不到释放。stack和memory都是一个1024字节的数组，因此其内存一直就是成M的增加，当碰到6810086中的`0x68b71d202dc52ad80812b563f3f6b0aaf1f19c04c1260d13055daad5b88a36a8`交易时，其trace数量为100W，导致其内存分配巨大而一直得不到释放。只需将保存stack和memory的部分注释掉即可，因为该部分只要debug打开，该部分会默认加入到structlog的全局变量中。

其它问题

1. reverTransfer 中的revertNum初始值应该为-1， 避免transfer全部revert掉之后初始值0，还剩余一个transfer。
2. selfdestruct中转账的源地址为合约地址，目的地址为stack.pop()出来的地址，注意stack.pop()动作不要为了获取目的地址而执行两遍。