区块链原理与实践

实验四

实验四Part1 Go语言基础&区块链中的典型密码算法

实验内容

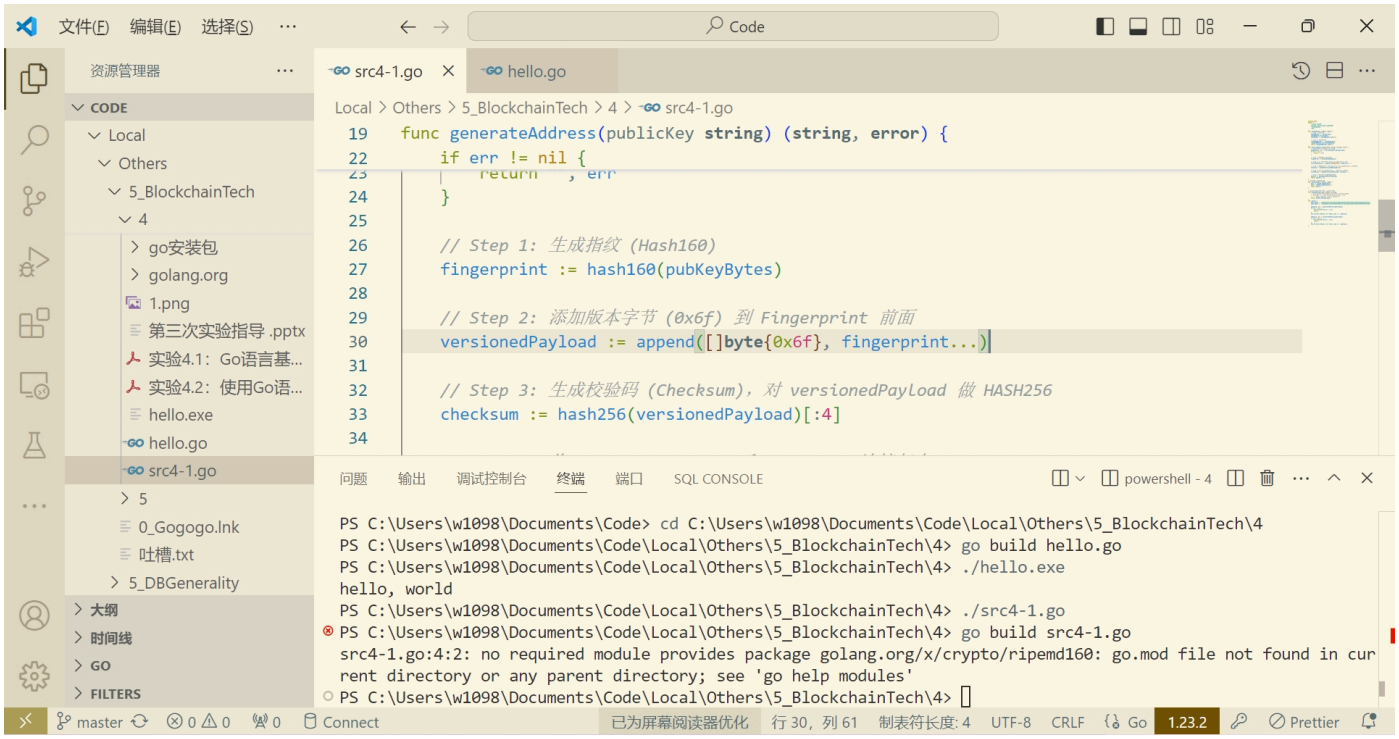
1. 配置Go语言环境
2. 比特币测试网地址的生成
3. Merkle Tree

实验步骤

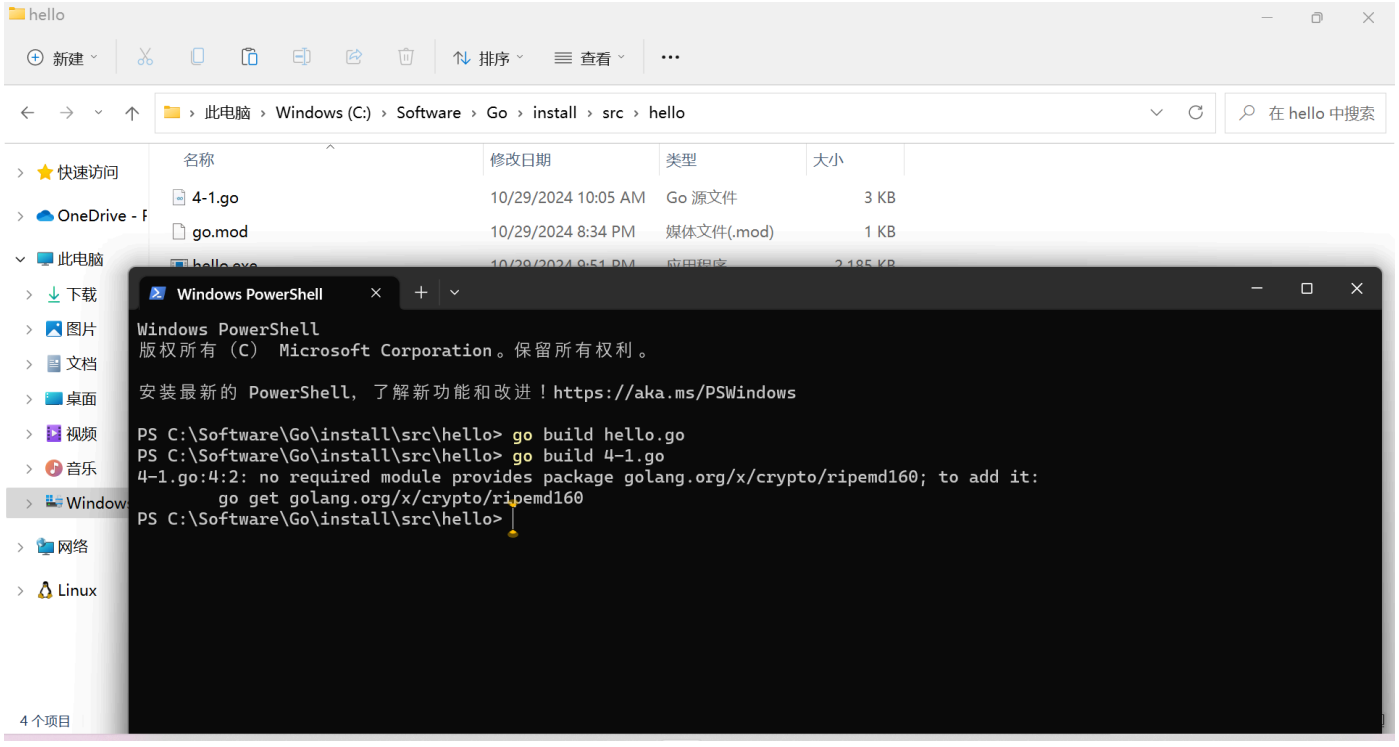
1. 下载msi并进行安装
2. 编写hello.go进行测试
3. 进行测试与调错

实验结果

go环境成功安装，hello.go正确执行：



比特币测试网地址的生成实验部分编写了完整代码，但ripemd160包无论是在线下载还是使用下发材料直接导入都无法进行安装，与助教学姐讨论许久后未果，无法产生执行结果，相关代码见实验报告末尾：



Merkle Tree实验部分的`compareMerkleTree`实现如下，完整代码见实验报告末尾：

```
func compareMerkleTree(tree1 *MerkleTree, tree2 *MerkleTree) int {
    for i := 0; i < len(tree1.leaves); i++ {
        if string(tree1.leaves[i].hash) != string(tree2.leaves[i].hash) {
            return i
        }
    }
```

```
    return -1 // 如果返回-1，表示没有找到差异
}
```

实验四Part2 使用Go语言构造区块链

实验内容

1. 配置Go语言环境
2. 比特币测试网地址的生成
3. Merkle Tree

实验步骤

1. 实验1-构建区块
2. 实验2-实现一条链
3. 实验3-添加工作量证明模块
4. 实验4-阅读代码：添加数据库

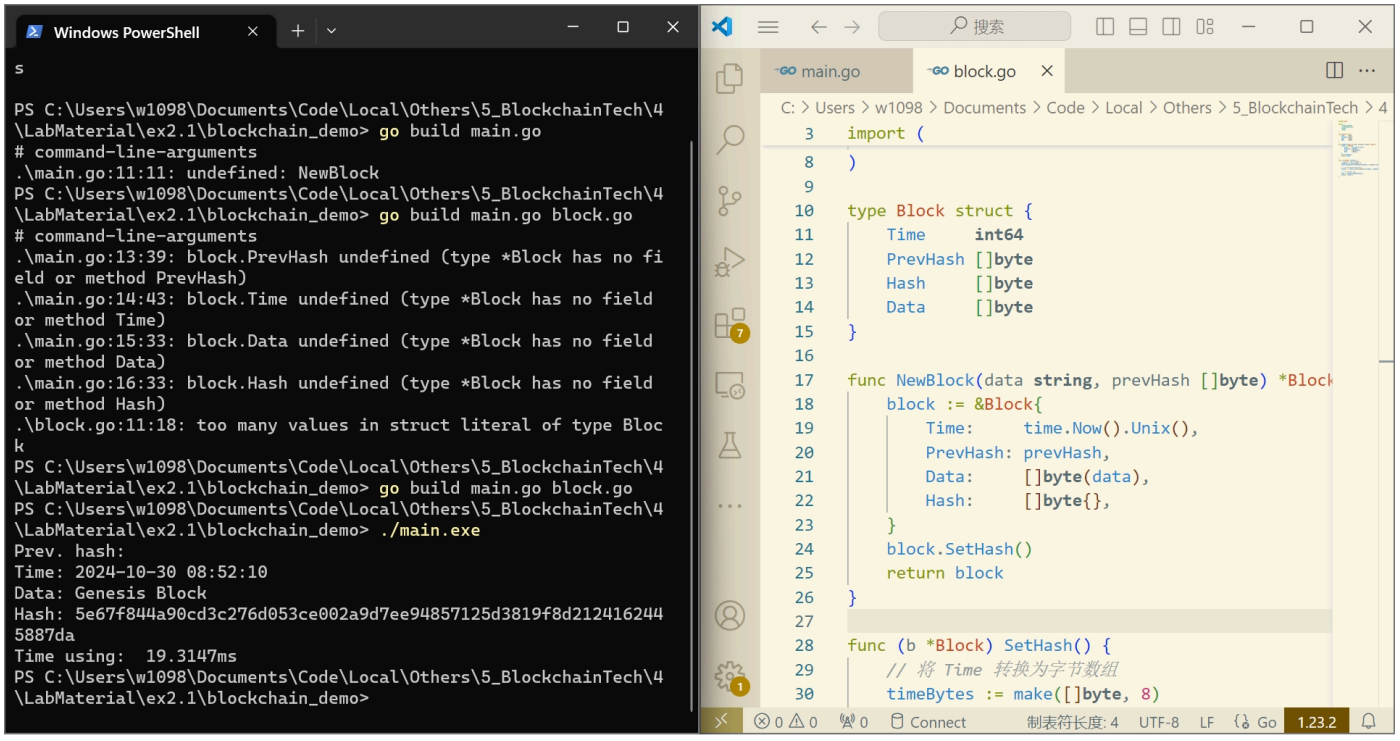
具体实现过程过于复杂，执行步骤详见接下来实验结果部分与实验报告最后的代码吧。

实验结果

实验1-构建区块

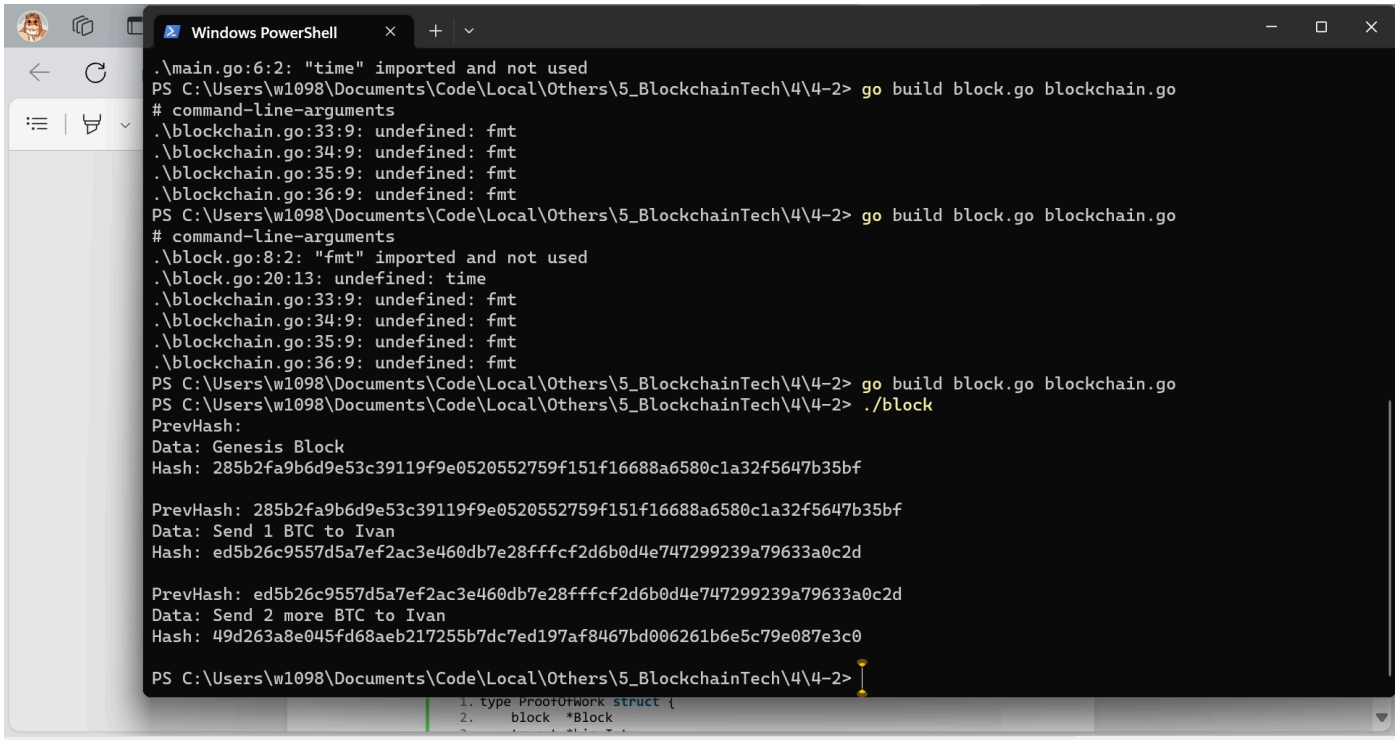
完善block.go文件中的`setHash`函数，共同编译block.go, main.go文件，测试区块。

测试通过！



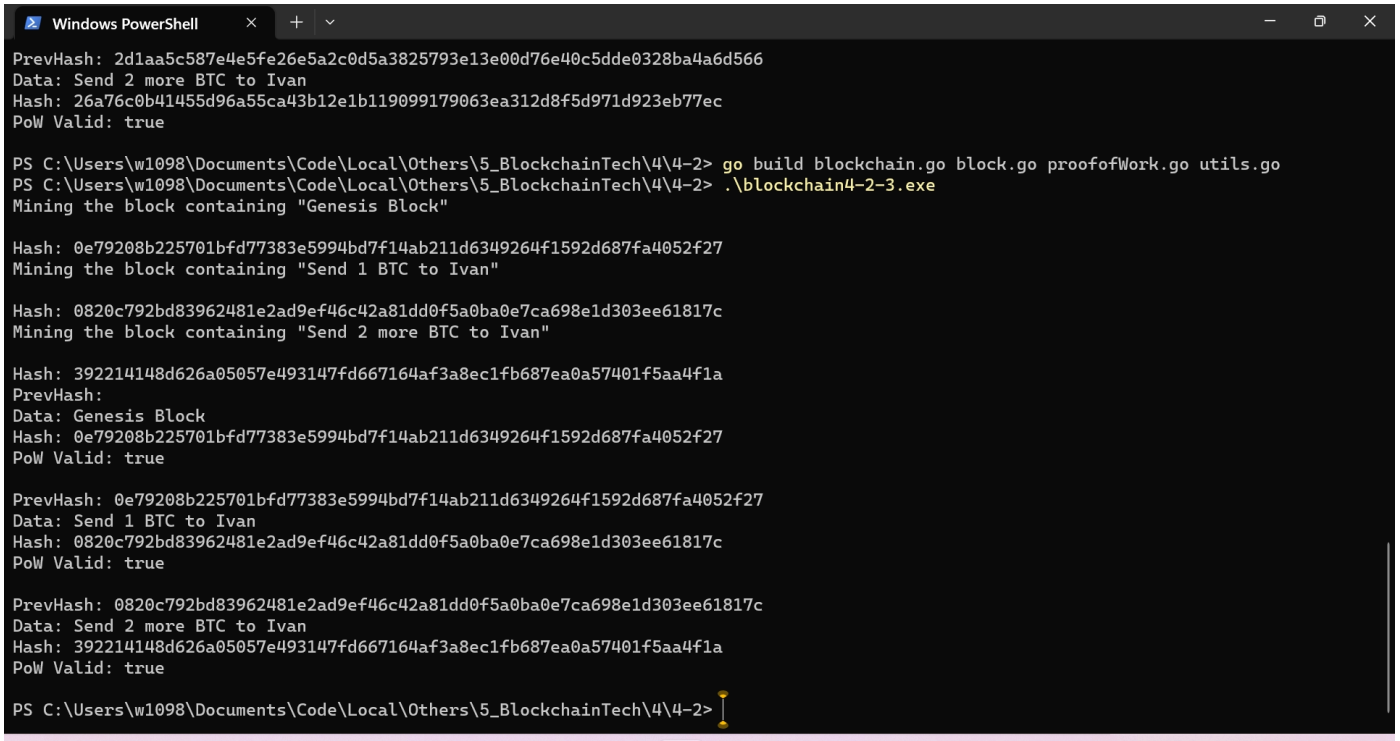
实验2-实现一条链

完善blockchain.go代码后，进一步编写测试main函数，共同编译block.go, blockchain.go文件，测试区块与链的链接。



实验3-添加工作量证明模块

在上一步的基础上添加新元素与新函数作为工作量证明的基础，完善proofofWork.go代码，编写新测试main函数，共同编译block.go, blockchain.go, proofofWork.go, 在已经连接好的区块链基础上测试工作量证明机制。



实验4-阅读代码：添加数据库

在上一步的基础上添加并整合blockchain.go（新版）和blockchain_interator.go回答问题如下：

- 为什么需要在block类中添加Serialize()和DeserializeBlock()两个函数？他们主要做了什么？描述一下NewBlockchain()和NewBlock()的执行逻辑。
- Blockchain类中的tip变量是做什么用的?
- 迭代器Interator是如何工作使得我们能够从数据库中遍历出区块信息的?

Serialize()和DeserializeBlock()的作用：
区块数据在代码中是以Block结构体的形式存储的，但数据库只能存储二进制数据（[]byte类型）。因此，在存储之前需要将Block结构体转换为字节数组，即Serialize()的作用。反之，从数据库读取区块时，需要将字节数据转换回Block结构体，即DeserializeBlock()的作用。就具体实现而言，他们使用了gob编解码工具。

NewBlockchain()和NewBlock()的执行逻辑：
NewBlockchain():
它会打开指定的数据库文件并检查数据库中是否已有区块链数据。
如果已经存在区块链数据（即非空），它会从数据库中读取最后一个区块的哈希值，并将该值作为链顶的哈希存储在tip变量中。如果数据库为空，则创建一个新的创世区块，并将创世区块写入数据库，同时将创世区块的哈希值作为链顶的tip存储。

NewBlock():
用于创建一个新的区块。每个新块都包含前一个区块的哈希以及块的内容。新区块的Hash值通过PoW算法生成，完成后会将区块的数据存储到数据库中。

Blockchain类中的tip变量:
tip是区块链的链顶指针，用来指向当前链中最后一个区块的哈希值。并借此快速找到区块链的链顶，避免遍历整个链以确定最新的区块。

Interator的工作原理：

`BlockchainIterator`会从链顶区块的哈希值开始，每次调用`Next()`方法，通过`currentHash`变量从数据库中读取对应的区块数据。读取到的区块数据会反序列化为`Block`结构体，并返回给调用者，同时将`currentHash`更新为该区块的`PrevHash`。通过不断调用`Next()`方法，迭代器会如同访问链表一样沿着区块链从链顶向创世区块逐个遍历，直到到达链的起点（`PrevHash`为空）为止。

实验收获

熟悉了go语言的语法，了解了区块链的底层工作逻辑与工作量证明机理，学习了区块链与数据库的关系

源代码

代码太长了，就使用链接的形式给出了。某些文件会在不同的实验阶段中被使用，所执行的代码片段并不相同，具体执行部分在注释中给出了。从邮件下载时可能因为文件存储位置的变化导致链接失效，但文件名仍然正确，见谅

Part1

hello.go

4-1.go

Merkel-Tree.go

Part2

block.go

blockchain.go

blockchain_interator.go

blockchain-4.go

main.go

proofofWork.go

utils.go