**Day3-区块链核心技术分布式账本**

**1. 任务介绍**

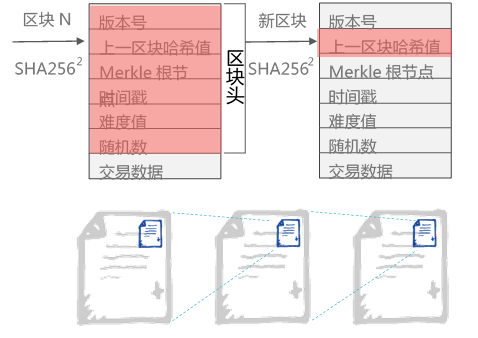
这次的任务是对fabric的账本的数据有一个初步的了解，包括账本的概念，构成账本的块的数据结构，构成块的交易的数据结构，然后通过一个程序真实的从一个账本中查看到块的数据以及交易的一些字段的信息。

账本是Fabric中所有状态转换的记录，具有有序和防篡改的特点。状态转换是参与各方提交链代码调用（交易）产生的结果。每个交易会产生一组资产键值对，这些键值对作为“创建”、“更新”或者“删除”提交给账本。

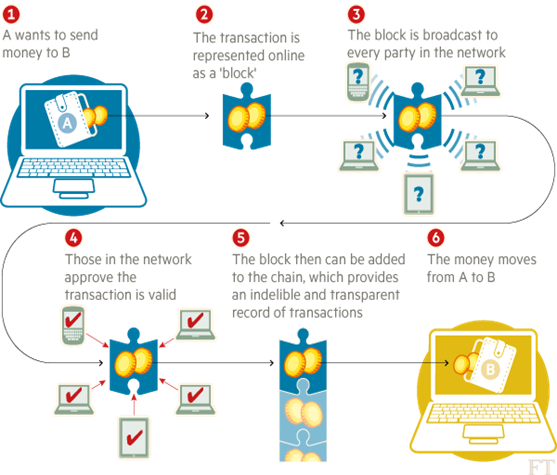
账本由一个区块链（链）构成，并将不可变的、有序的记录存放在区块中；同时包含一个状态数据库来记录当前的Fabric状态。每个通道中各有一个账本。各个节点对于它所属的每个通道，都会保存一份该通道的账本副本。

**2. 数据结构展示**

账本是由链构成，而链是由区块构成的，**区块(Block) + 链(Chain) = 区块链(Blockchain)。**



数据以区块（block）为单位产生和存储，并按照时间顺序连成链式（chain）数据结构。所有节点共同参与区块链系统的数据验证、存储和维护。新区块的创建需得到共识确认，并向各节点广播实现全网同步，之后就不能更改或删除。



* 区块的数据结构：

**type** Block **struct** {  
 Header \*BlockHeader **`protobuf:"bytes,1,opt,name=header" json:"header,omitempty"`** Data \*BlockData **`protobuf:"bytes,2,opt,name=data" json:"data,omitempty"`** Metadata \*BlockMetadata **`protobuf:"bytes,3,opt,name=metadata" json:"metadata,omitempty"`**}

**type** BlockHeader **struct** {  
 Number uint64 **`protobuf:"varint,1,opt,name=number" json:"number,omitempty"`** PreviousHash []byte **`protobuf:"bytes,2,opt,name=previous\_hash,json=previousHash,proto3" json:"previous\_hash,omitempty"`** DataHash []byte **`protobuf:"bytes,3,opt,name=data\_hash,json=dataHash,proto3" json:"data\_hash,omitempty"`**}

**type** BlockData **struct** {  
 Data [][]byte **`protobuf:"bytes,1,rep,name=data,proto3" json:"data,omitempty"`**}

**type** BlockMetadata **struct** {  
 Metadata [][]byte **`protobuf:"bytes,1,rep,name=metadata,proto3" json:"metadata,omitempty"`**}

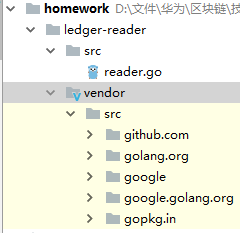
* 交易的数据结构：

**type** Transaction **struct** {  
 *// The payload is an array of TransactionAction. An array is necessary to  
 // accommodate multiple actions per transaction* Actions []\*TransactionAction **`protobuf:"bytes,1,rep,name=actions" json:"actions,omitempty"`**}

**type** TransactionAction **struct** {  
 *// The header of the proposal action, which is the proposal header* Header []byte **`protobuf:"bytes,1,opt,name=header,proto3" json:"header,omitempty"`** *// The payload of the action as defined by the type in the header For  
 // chaincode, it's the bytes of ChaincodeActionPayload* Payload []byte **`protobuf:"bytes,2,opt,name=payload,proto3" json:"payload,omitempty"`**}

**3. 任务执行**

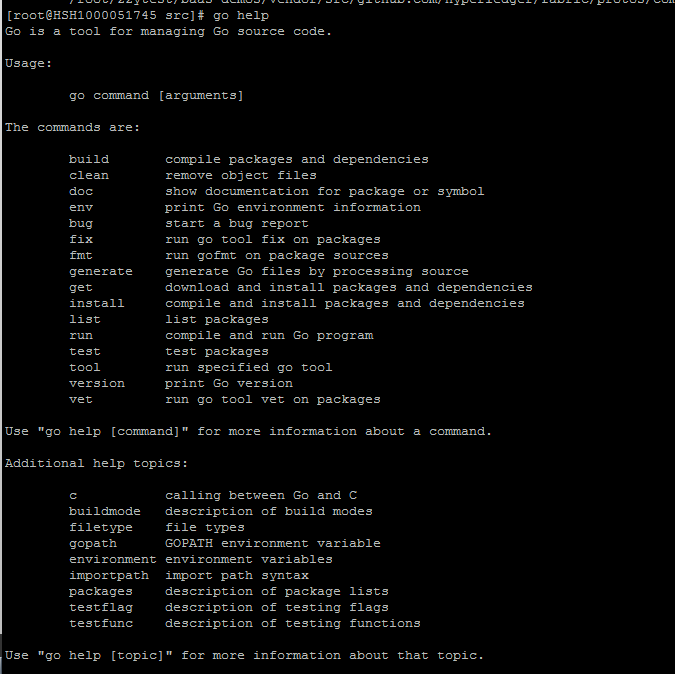
2.1 检查作业的代码结构，源码就一个reader的go文件，依赖库有很多



2.2 我们这里在Linux上面来运行和编译我们的代码，将作业的代码文件夹拷贝到linux机器上

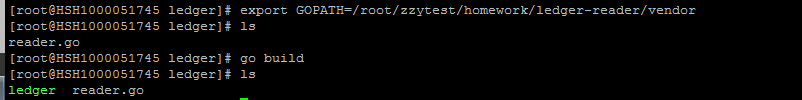


2.3 检查linux机器是否有go语言的编译环境，使用go help命令

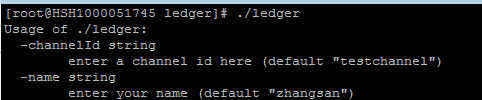


如果没有这个信息显示，点击<https://golang.org/dl/> 链接下载go语言安装包，并安装好，配置环境变量，然后使用命令显示上图结果。

2.4 配置gopath环境变量，使用go命令go build 编译代码



2．5 执行上图中编译的可执行文件



2.6 输入你的名字和你希望显示的通道名称完成打卡任务

**3.打卡任务**

完成如上操作并将结果截图显示。

截图中打印出区块数据信息如下所示：

