**课程报告**

|  |  |
| --- | --- |
| **课程名称：** | **企业软件项目实训** |
| **学生姓名：** | **刘嘉文** |
| **学生学号：** | **201630665168** |
| **学生专业：** | **软件工程** |
| **开课学期：** | **2018-2019第二学期** |

**软件学院**

**2019年6月**

1. **学习目的**
2. 了解区块链的架构及原理
3. FISCO-BCOS区块链系统的特性及使用方法
4. 以太坊相关基础
5. **学习用软件及系统**
6. Oracle VM虚拟机
7. FISCO-BCOS区块链系统
8. Spring Boot Starter
9. **学习内容**
10. **区块链技术原理：**

区块链（Blockchain）是比特币的一个重要概念，货币联合清华大学五道口金融学院互联网金融实验室、新浪科技发布的《2014—2016全球比特币发展研究报告》提到区块链是比特币的底层技术和基础架构。本质上是一个去中心化的数据库，同时作为比特币的底层技术。区块链是一串使用密码学方法相关联产生的数据块，每一个数据块中包含了一次比特币网络交易的信息，用于验证其信息的有效性（防伪）和生成下一个区块。

首先，**比特币**是什么：

比特币是存储在计算机里一枚货币，计算机存储的本质就是一串数字，那么存储比特币的这串数字为什么就可以等同于真正的货币呢？而且比特币让人感觉很像黄金，不管这黄金从什么渠道获取（从沙子里掏的，还是自己购买的），人类都会认同它的财富价值。

说到比特币很多文章总结它的特点，例如：去中心化啊，记账系统，安全性好，不过它有什么特点，大家现在几乎都公认比特币可以等同于货币，但是比特币这种货币是有别于现实社会中的各国央行发行发行的法定货币，法定货币的一大重要特点就是需要一个权威的机构（一般是指代表国家的央行）来发行货币，法定货币的价值是由老百姓认同国家的权威性来体现的，不管怎么说法定货币是一种中心化的发行货币机制，法定货币都会受到国家这个中心所控制和约束。比特币的设计理念不同，它想通过一种没有任何权威中心所左右的方式发行货币，让货币只包含经济价值的本质属性。

回到区块链。

**区块链技术原理**

如今的区块链技术概括起来是指通过去中心化和去信任的方式集体维护一个可靠数据库的技术。其实，区块链技术并不是一种单一的、全新的技术，而是多种现有技术（如加密算法、P2P文件传输等）整合的结果，这些技术与数据库巧妙地组合在一起，形成了一种新的数据记录、传递、存储与呈现的方式。简单的说，区块链技术就是一种大家共同参与记录信息、存储信息的技术。过去，人们将数据记录、存储的工作交给中心化的机构来完成，而区块链技术则让系统中的每一个人都可以参与数据的记录、存储。区块链技术在没有中央控制点的分布式对等网络下，使用分布式集体运作的方法，构建了一个P2P的自组织网络。通过复杂的校验机制，区块链数据库能够保持完整性、连续性和一致性，即使部分参与人作假也无法改变区块链的完整性，更无法篡改区块链中的数据。区块链技术涉及的关键点包括：、去中心化、集体维护、时间戳、可靠数据库、去信任、非对称加密等。

区块链技术重新定义了网络中信用的生成方式：在系统中，参与者无需了解其他人的背景资料，也不需要借助第三方机构的担保或保证，区块链技术保障了系统对价值转移的活动进行记录、传输、存储，其最后的结果一定是可信的。

区块链技术原理的来源可归纳为一个数学问题：拜占庭将军问题。拜占庭将军问题延伸到互联网生活中来，其内涵可概括为：在互联网大背景下，当需要与不熟悉的对手方进行价值交换活动时，人们如何才能防止不会被其中的恶意破坏者欺骗、迷惑从而做出错误的决策。进一步将拜占庭将军问题延伸到技术领域中来，其内涵可概括为：在缺少可信任的中央节点和可信任的通道的情况下，分布在网络中的各个节点应如何达成共识。区块链技术解决了闻名已久的拜占庭将军问题——它提供了一种无需信任单个节点、还能创建共识网络的方法。

1. **联盟链和公有链的异同：**

**公有链：**

公有链，顾名思义，从字面上理解，就是“公有”，它是指对全世界所有人开放的，任何人都可以读取数据、发送交易且交易能够获得有效确认的共识区块链。

公有链是指全世界任何人都可以随时进入系统中读取数据、发送可确认交易、竞争记账的区块链。公有链通常被认为是完全去中心化的，因为没有任何人或机构可以控制或者篡改其中数据的读写。

公有链一般会通过代币机制鼓励参与者竞争记账，来确保数据的安全性。比特币、以太币都是典型的公有链。

公有链上的各个节点可以自由加入和退出网络，并参加链上数据的读写，读写时以扁平的拓扑结构互联互通，网络中不存在任何中心化的服务端节点。

比特币和以太坊这两者采取的都是公有链。因为没有限制，任何人都可以参加。

**公有链特点：**

· 访问门槛低

只要你有计算机，只要计算机能联网，都能够访问。

· 数据公开透明且无法篡改

公有链是高度去中心化的分布式账本，每个人在任何节点的行为都是可以查看的，而且基于庞大的用户体系，想要篡改交易数据，几乎不可能实现。

·匿名性

由于节点之间无需彼此信任，所有的操作都可以匿名进行，很好地保护使用者的隐私。

·免受开发者影响

公有链数据的读写是不受任何人控制和篡改的，就连程序开发者也无权干涉用户。所以，极大程度上保护了用户免于程序开发者的影响。

**联盟链：**

联盟链是指由多个机构共同参与管理的区块链，每个组织或机构管理一个或多个节点，其数据只允许系统内不同的机构进行读写和发送。

联盟链的各个节点通常有与之对应的实体机构组织，通过授权后才能加入与退出网络。各机构组织组成利益相关的联盟，共同维护区块链的健康运转。

联盟链是指由若干个机构共同参与管理的区块链，每个机构都运行着一个或多个节点，其中的数据只允许系统内不同的机构进行读写和发送交易，并且共同来记录交易数据。

私有链和联盟链之间的设计隐私权限会有不同，联盟链中的权限设计要求往往会更加复杂。

联盟链适合组织机构间的交易和结算，目前国内比较有影响力的联盟链有中国分布式总账基础协议联盟（ChinaLedger）、中国区块链研究联盟（CBRA）和微众银行等。

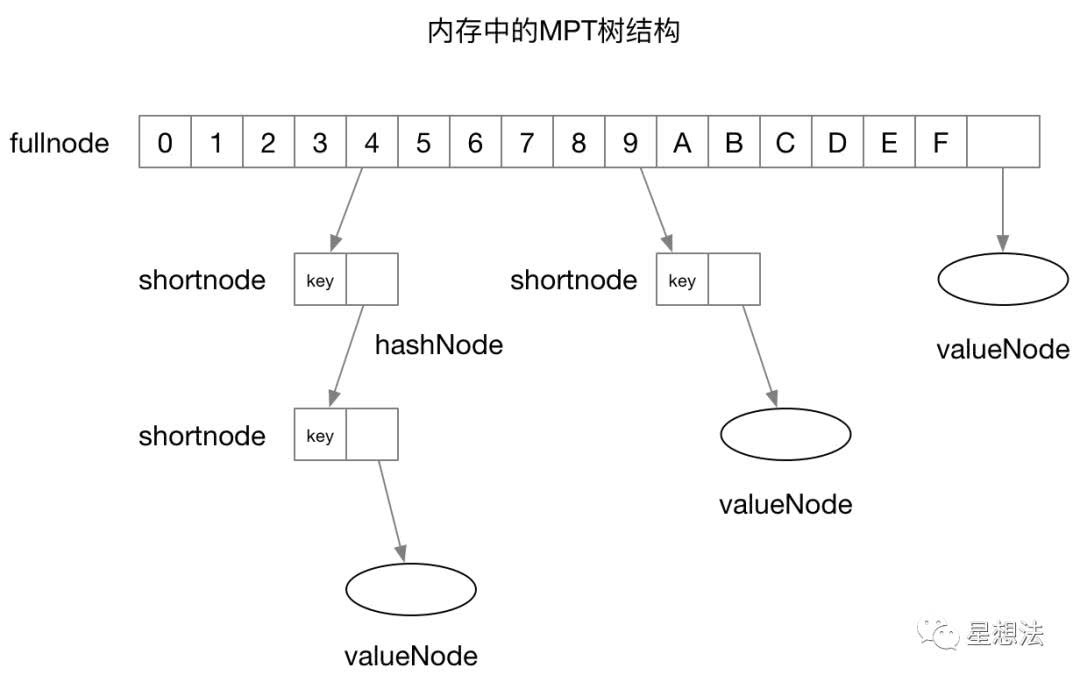
**联盟链特点**

从某种程度上来说，其实联盟链也属于私有链，但它私有的程度不同，其权限设计要求更复杂，可信度更高。

1. **链式储存和MPT存储**

****1）MPT树的内存结构****

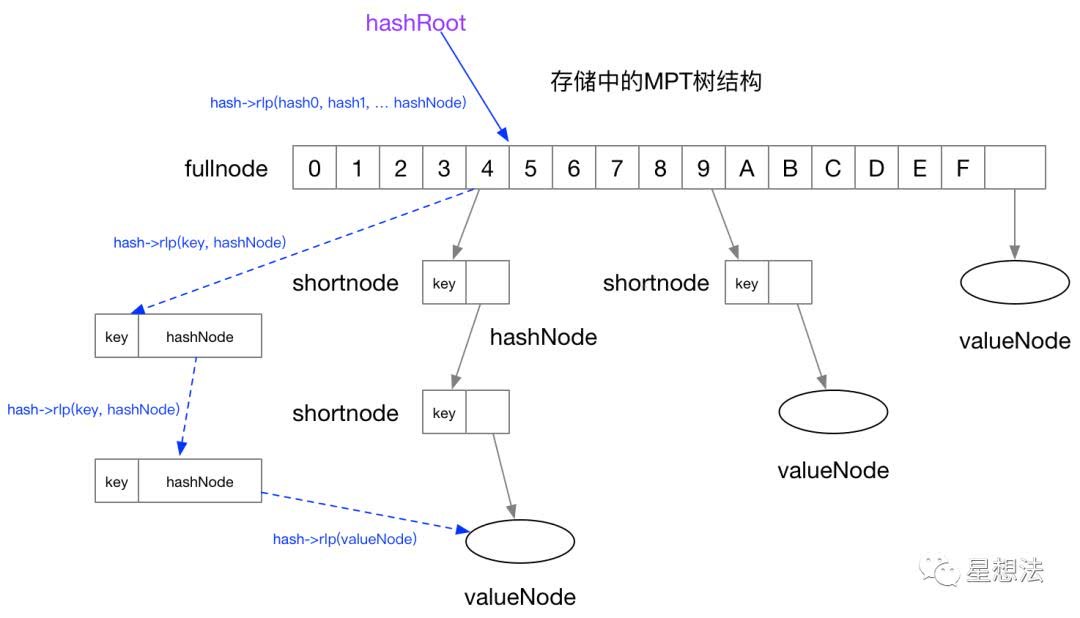
MPT树的内存结构，如下图。一颗MPT树由4种节点连接组成：fullnode是分支节点，shortnode代表扩展节点或者叶子节点（取决于value是hashNode还是valueNode）。



需要指出的是：MPT的叶子节点（账户信息）是按照账户的地址（Address）的字典排序形成。也就是说，MPT树上的一个叶子节点路径上的所有的key组成的是账户地址。

****2）MPT树物理存储****

在内存中，节点和节点的连接关系可以通过“指针”完成。为了将连接关系持久化，需要将节点内容生成对应的，唯一的“地址”。其他引用节点内容的节点，只需要记录引用节点的地址即可，示意如下图中的蓝色部分。



以太坊中，节点内容的地址就是节点内容的hash。比如一个由“key”以及一个“hashNode”组成的节点，节点内容是rlp（key，hashNode），对应的地址是内容的hash。在存储中，存储的是hash到rlp的KV对。其他引用节点的“连接”用hash值代替，从而隐性的实现“连接”。

再举fullnode的例子，在存储中，fullnode的内容是rlp（hashNode，hashNode ... hashNode, valueNode)组成。

****3）账户修改****

从上述可见，一个叶子节点的改变，导致节点内容的改变，节点的“地址”也会发生变化。也就是说，父亲节点和该节点的“连接”地址发生变化。因为父亲节点的内容包括“连接”地址，所以父亲节点的内容也发生变化，父亲节点的“地址”也随着变化。随即，父亲的父亲也发生变化，一直变化到MPT的树根。

总的来说，一个账户的改变，导致MPT树，从叶子到树根，整条路径上的所有节点的KV对都发生变化，需要更新到存储中。

以太坊中的节点间的“连接”关系用hash来代替，主要原因，hash结果唯一。还有个原因，有关账户中的Storage数据的MPT树。在Storage的数据构建的MPT树，节点排序是通过Storage数据内容的hash，这样保证MPT树的最大高度，避免攻击者通过构造Storage创建深度很高的MPT树。

MPT树中的分支节点的分支个数设置为16，而不是更大的原因，可能是怕分支节点内容变的太大，在节点内容大小和树高之间的一个平衡。

1. **Gas在智能合约中的作用**

**1）.**

“gas”是以太坊使用的特殊单位的名称。它衡量一个动作或一系列动作需要执行多少“工作”：例如，计算一个Keccak256密码散列，每计算一次散列需要30个气体，每256位 数据被哈希。 Ethereum平台上的一项交易或合同可以执行的每项操作都会花费一定数量的天然气，其运营所需的计算资源比计算资源要求较少的运算需要更多的天然气。

gas的重要性在于它有助于确保提交给网络的交易支付适当的费用。 通过要求交易支付每个操作的执行（或导致合同执行），我们确保网络不会因为执行大量对任何人无价值的密集工作而陷入困境。 这与比特币交易费用不同，它仅基于交易的千字节大小。 由于以太坊允许运行任意复杂的计算机代码，所以短的代码实际上可能导致大量计算工作的完成。 所以衡量直接完成的工作非常重要，而不是仅仅根据交易或合同的长度选择费用。

所以，如果gas基本上是交易费用，那么你如何支付？ 这是一个棘手的地方。 虽然gas是一个可以测量物质的单位，但gas并没有任何实际的标志。 也就是说，你不能拥有1000gas。 相反，gas只存在于以太坊虚拟机内部，作为正在执行多少工作的计数。 在实际支付gas时，交易费用是ether的一定数量，以太坊网络上的内置令牌和矿工奖励生产块的令牌。

起初这可能看起来很奇怪。 为什么不直接用ether衡量成本？ 答案是，就像比特币一样，以太网的市场价格可能会迅速变化！ 但是计算的代价并不是因为以太的价格变化而上升或下降的。 所以将计算价格与以太币的价格区分开来是很有用的，这样每次市场走势就不需要改变操作成本。

这里的术语有点混乱。 EVM中的操作具有gas成本，但gas本身也具有以ether的gas价格。 每笔交易都规定了每个gas单位愿意支付的gas价格，从而使市场能够决定gas价格和计算成本（以天然气计量）之间的关系。 这是两者的总和，即所用gas总量乘以gas price，得到交易支付的全部费用。

尽管这很棘手，但了解这个区别是很重要的，因为这会导致以太坊交易对最初的学习者来说最混乱的一件事情：您的交易没有用完，交易也没有足够高费用。 如果我在我的交易中设定的gas price太低，那么没有人会在第一时间去管理我的交易。 它不会被矿工包括在区块链中。但如果我提供一个可以接受的天然气价格，那么我的交易就会产生如此多的计算工作，以至于合并后的天然气成本超过了我所附加的费用数额，那么这个天然气就会被计算为“花费”，我不会收回。 矿工将停止处理交易，恢复所做的任何更改，但仍将其作为“失败的交易”包含在区块链中，收取费用。 这看起来可能很苛刻，但是当你意识到矿工真正的工作是在执行计算的时候，你可以看到他们永远也不会获得这些资源。 所以，即使你设计糟糕的交易用完了，你付给他们的工作也是公平的。

提供太多的费用也不同于提供太多的ether。 如果你设置了一个非常高的gas price，那么你只需要付出很少的代价，就像在比特币中设置超高的交易费用一样。 你肯定会被排在最前面，但你的钱已经没有了。 但是，如果您提供了正常的gas price，并且只需要支付比您购买gas所需的更多的ether，那么超额部分将退还给您。 矿工只收取你实际工作的费用。 你可以把煤气价格看作矿工的小时工资，把煤气成本看作是工作时间表。

gas还有许多其他的微妙之处，但这应该给你基本的东西！ gas是使以太坊中的复杂计算“安全”的关键机制，因为任何失控的程序只会在请求运行的人提供的资金的情况下持续下去。 当资金停止时，矿工们就停止工作。 而你在程序中犯的错误只会影响付费使用它的人 - 网络的其他部分不会因为你的错误而遭受性能问题。 当性能问题消耗掉所有的ether时，他们只会得到一个大的薪水！ 如果没有这个关键技术，通用区块链的想法将是完全不可能的。

总结:

·gas是费用的计算方式

不过，这些费用仍然以ether支付，不同于gas

·gas成本就是劳动时间之类的工作量，而gas price就像你为完成工作而支付的小时工资。 两者的组合决定了您的总交易费用。

·如果您的天然气价格太低，没有人会处理您的交易

·如果你的天然气价格没问题，但是你的交易的天然气成本超过了预算，交易就会失败，但是仍然会进入区块链，你不会为劳动者的工作收回这笔钱。

·这可以确保没有任何东西可以永远运行，人们会仔细考虑它们运行的代码。 它保持矿工和用户的安全不受恶劣的代码影响！

**2）**

以太坊在区块链上实施了一个名为以太坊虚拟机（EVM）的执行环境。 当您运行分散式应用程序（dApp）时，每个指令都会在网络的每个节点上执行。 这有一个代价：对于脚本可以执行的每个操作，都有一个指定的成本，用gas单位数表示，您可以在EVM规范中看到。

天然气价格由矿主决定，目前约为5〜21 GWei（1 GWei为10 ^ 9 Wei或10 ^ -9 Ether）。 以太坊使用以太币作为其内部货币/标记。 您的帐户持有以ether表示。 当您部署合同或执行交易时，gas将从您的账户余额中提取。 你可以自由指定一个gas price，或保留建议。

从理论上的PoV来看，每个采矿节点应该选择一个最大化其利润的天然气价格。 由于消耗更多天然气的块体在网络中传播速度较慢，因此它将成为一名叔叔的可能性较高，只能减少奖励。 接受的最低天然气价格应足够高，以应付这种增加的风险。 到目前为止，在真实网络中观察到的情况是，矿业集团接受交易，降低天然气价格，这在经济上是合理的，有助于减少网络拥塞并提高整体网络/代币价值。

**3）**

gas是以太坊“世界电脑”使用的计量单位。 作为比喻，电力按千瓦时计量。 在以太坊使用更多的计算和存储意味着使用更多的气体。 计量的一个根本原因是，它激励人们（矿工）操作世界计算机。 这些矿工获得处理交易的费用，这是由计量方案确定的：gas。

EVM中的每个操作都会消耗gas。 例如，乘法（MUL）消耗5个gas，加法（ADD）消耗3个gas。 以下是以太坊的运行和耗气量的电子表格。 （也可以把它们看作是天然气的成本，但是这可能使得解释更加难以跟随成本，费用，价格飞涨。）

计量是不同的费用和天然气是不同的乙醚。 为了澄清这个问题，把气体看成是燃料的代名词。 交易必须提供足够的燃料或启动燃料来覆盖EVM计算和存储设施的整个使用。 所有剩余的天然气将退还给交易的发起者：发起交易的用户。 耗尽气体的交易已经恢复，但仍然包含在一个区块中，并且相关费用支付给矿工。

从燃料的角度来看，我们来看一下收费的概况。 尽管EVM中的每个操作都消耗了预定义的固定gas量（例如，MUL操作总是消耗5个气体），但是用户可以在每次交易中指定gas price。 目前的天然气价格是0.02μEther，或0.00000002ETH。 发起人支付给矿工的费用是交易的（开始天然气 - 剩余天然气）× gas price。

**4）**

**gas和ether**

gas应该是网络资源/利用的不变成本。您希望发送交易的实际成本总是相同的，所以您不能真正期望gas发行，货币一般是不稳定的。

因此，我们发行价值应该有所不同的以太网，而且以ether方式实施一个gas价格。如果ether的价格上涨，以ether计算的gas价格应该下降，以保持gas的实际成本相同。

gas有多个相关的条款：Gas Prices，Gas Cost,Gas Limit 和Gas Fees。gas的原理是为以太坊网络的交易或计算成本有一个稳定的价值。

·Gas Cost是一个静态值，表示气体的计算成本是多少，目的是gas的实际价值永远不会改变，所以这个成本应该始终保持稳定。

·Gas Prices是多少天然气成本在另一个货币或像Ether的象征。为了稳定天然气的价值，gas价格是一个浮动价值，如果令牌或货币的成本波动，gas价格变化保持相同的实际价值。gas价格由多少用户愿意花费的均衡价格以及多少处理节点愿意接受来设定。

·Gas Limit是指每个区块可以使用的最大的gas数量，它被认为是区块的最大计算负荷，交易量或区块大小，矿工可以随着时间慢慢地改变这个值。

·Gas Fees实际上是运行特定交易或计划（称为合同）所需支付的gas量。块的gas费用可以用来暗示块的计算负荷，交易量或大小。gas费用支付给矿工（或PoS保税承包商）。

**5）**

gas基本上是执行交易或合同的内部定价。

每笔交易或合同的天然气价格是为了处理以太坊及其EVM（以太坊虚拟机代码）

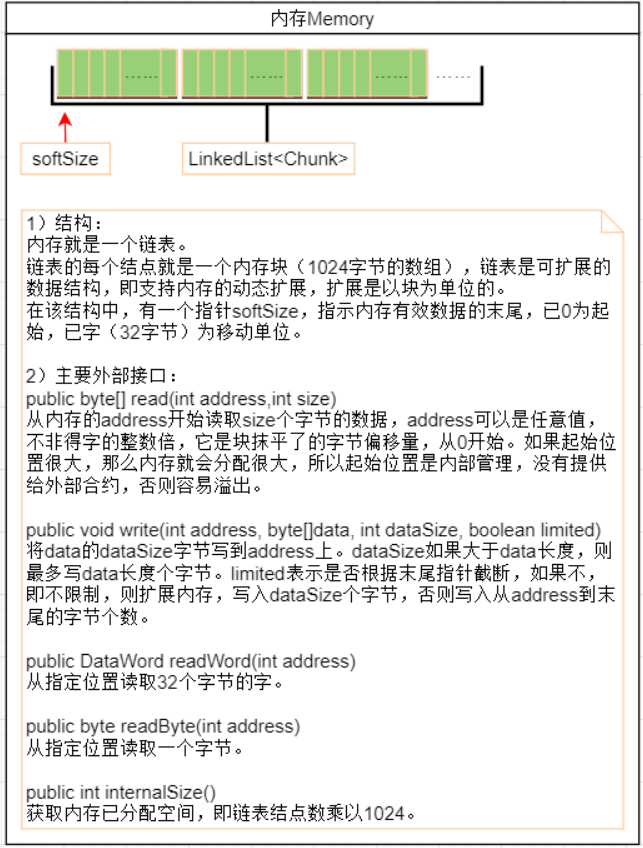
所以，交易或者操作越复杂，花费的gas就越多。

**5.EVM中的数据存储结构**

1. EVM的存储结构

1.1. VM内存

内存结构像堆栈一样，也提供了数据缓存的功能，但更作用的是提供了合约调用合约等过程中，子合约数据的临时存储。



1）实现类

public class Memory implements ProgramListenerAware

2）结构

private List<byte[]> chunks = new LinkedList<>();

内存就是一个链表。

链表的每个结点对应一个内存块，一个内存块1024字节，链块内存是可扩展的，扩展是以块为单位的。

private int softSize;

在内存中，有一个指针softSize，用于指示存储数据的末尾。

3）外部接口

public byte[] read(int address, int size) {

从内存的address（块抹平了的字节偏移量，从0开始，如1025代表第1块第1个位置）开始读取size个字节的数据。

address可以超过存储数据的末尾。

读取时，如果被读数据超出了末尾指针的范围，则会以块为单位增加块数，也就是说读到的新增数据是0。而末尾指针则以字，即32字节为单位移动，最终移动到被读数据的末尾。

public void write(int address, byte[] data, int dataSize, boolean limited)

将data的dataSize字节写到address上。

dataSize如果大于data长度，则最多写data长度个字节。

limited表示是否根据末尾指针截断，如果不，即不限制，则扩展内存，写入dataSize个字节，否则写入从address到末尾的字节个数。

public void extendAndWrite(int address, int allocSize, byte[] data) {

扩展然后写入数据。

public void extend(int address, int size) {

扩展内存，扩展了后末尾指针就变化。

public DataWord readWord(int address) {

从指定位置读取32个字节的字。

public byte readByte(int address) {

从指定位置读取一个字节。

public int size() {

获取末尾指针的位置。

public int internalSize() {

获取内存以分配空间，即链表结点数乘以1024。

public List<byte[]> getChunks() {

获取内存副本。

public String toString() {

以打印格式（含16进制和assii码形式）返回内存数据，softSize结尾。

4）程序监听

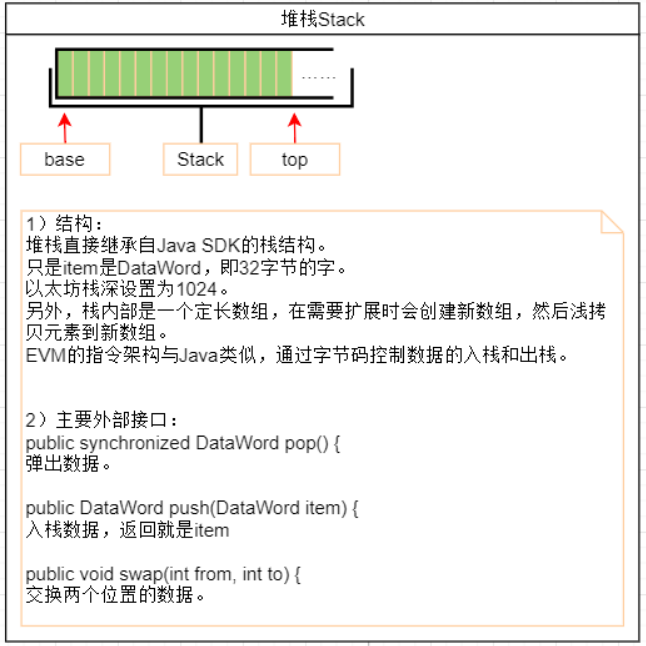
programListener.onMemoryExtend(toAllocate)

programListener.onMemoryWrite(address, data, dataSize)

在内存扩展（这里的扩展指应的字数）和写数据是会触发监听器。

1.2. VM堆栈

EVM的执行模型是基于栈结构的，像JVM一样，这里提供的堆栈就是用来存储字节码执行过程中的中间数据等。



1）实现类

public class Stack extends java.util.Stack<DataWord> implements ProgramListenerAware {

2）结构

直接继承自Java的Stack：

protected Object[] elementData;

只是item是DataWord，即32字节的字。

可以看到这是一个定长数组，在需要扩展时会创建新数组，然后浅拷贝元素到新数组。

以太坊栈深设置为1024。

3）外部接口

public synchronized DataWord pop() {

弹出数据。

public DataWord push(DataWord item)

入栈数据，返回就是item

public void swap(int from, int to) {

交换两个位置的数据。

4）程序监听

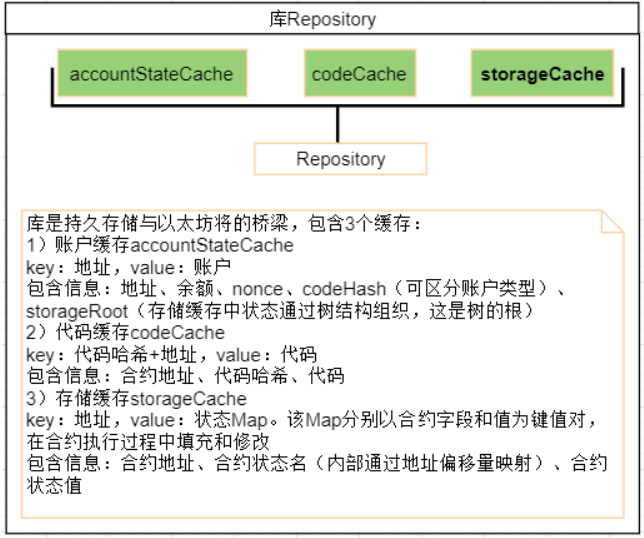
programListener.onStackPop();

programListener.onStackPush(item);

programListener.onStackSwap(from, to)

1.3. VM持久化存储

持久化存储主要是独立于区块存储之外，存储以太坊的账户、合约代码以及合约的状态。



1.3.1. 账户

org.ethereum.core.AccountState

可用来表示外部账户或合约账户。

含nonce、余额、状态根、代码哈希。

private final BigInteger nonce

对于外部账户，nonce表示从此账户发出的交易数量。

对于合约账户，nonce表示此合约内创建的合约数量。

设计nonce的目的是为了防止重放攻击，也就是防止一个交易被多次执行，因为每执行一个交易时，库中该交易发送者账户的nonce就会增加1。

默认值是从配置常量取出的0。

private final BigInteger balance

账户的余额，以Wei为单位。

默认为0。

private final byte[] stateRoot

用于存储合约内容Trie结构的哈希根。

默认是sha3(RLP.encodeElement(EMPTY\_BYTE\_ARRAY))。

private final byte[] codeHash;

合约代码的哈希值。

默认值是sha3(EMPTY\_BYTE\_ARRAY)。

提供了RLP编码和解码方法，以及编码结果缓存。

只要代码哈希或nonce不是默认的，则认为合约是存在的。

所谓空账户，是指代码哈希、nonce、余额都是默认的。空账户是需要被删除的。

1.3.2. 数据源结构体系

这里的Repository内部实现是比较复杂的，它封装了底层Key-Value数据库，并提供了批量提交和读、写缓存，并以此为基础提供了区块执行过程中的多级缓存、快照或者说事务提交。

**6.并行计算**

**1.什么是并行计算？**

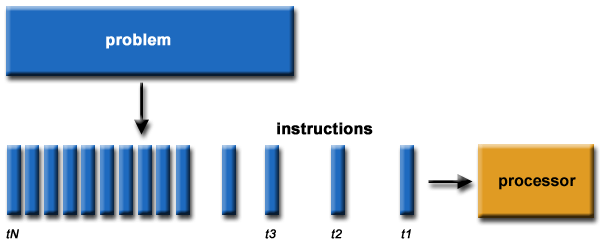
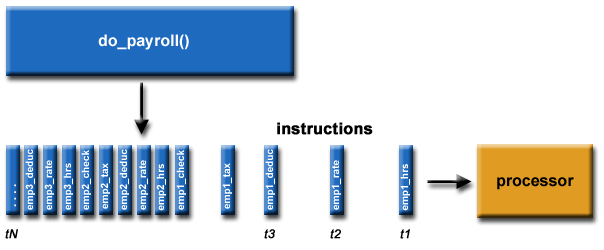
串行计算：

传统的软件通常被设计成为串行计算模式，具有如下特点：

一个问题被分解成为一系列离散的指令；

这些指令被顺次执行；

所有指令均在一个处理器上被执行；

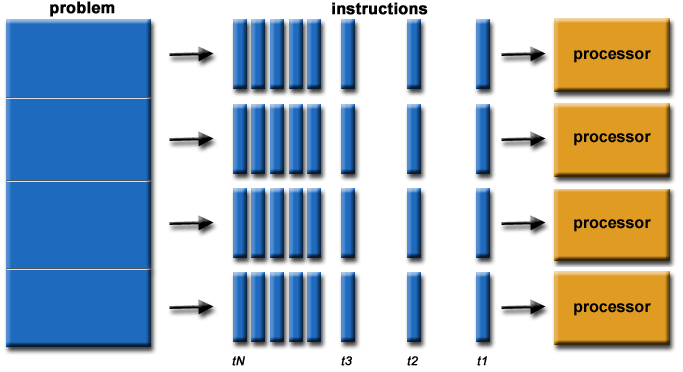
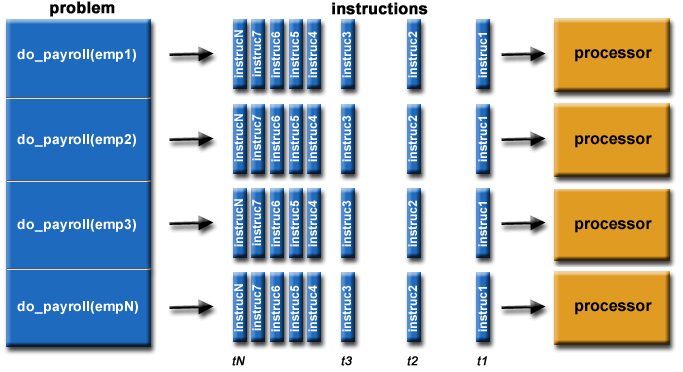
在任何时刻，最多只有一个指令能够被执行。   
   
例如，   


并行计算： 简单来讲，并行计算就是同时使用多个计算资源来解决一个计算问题：

一个问题被分解成为一系列可以并发执行的离散部分；

每个部分可以进一步被分解成为一系列离散指令；

来自每个部分的指令可以在不同的处理器上被同时执行；

需要一个总体的控制/协作机制来负责对不同部分的执行情况进行调度。   
   
例如，   


这里的 计算问题 需要具有如下特点：

能够被分解成为并发执行离散片段；

不同的离散片段能够被在任意时刻执行；

采用多个计算资源的花费时间要小于采用单个计算资源所花费的时间。

这里的 计算资源 通常包括：

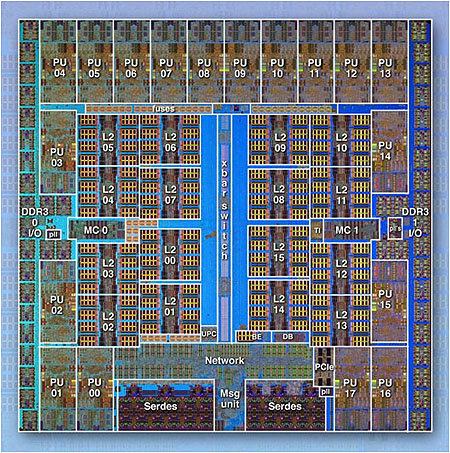
具有多处理器/多核(multiple processors/cores)的计算机；

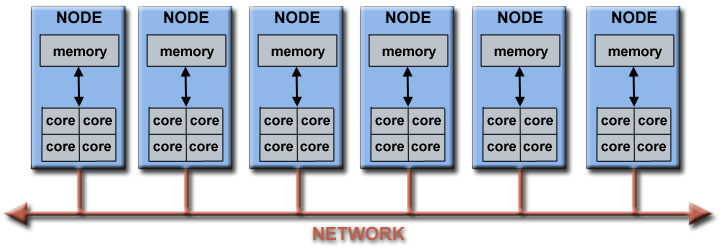
任意数量的被连接在一起的计算机。

并行计算机：   
通常来讲，从硬件的角度来讲，当前所有的单机都可以被认为是并行的：

多功能单元（L1缓存，L2缓存，分支，预取，解码，浮点数，图形处理器，整数等）

多执行单元/内核

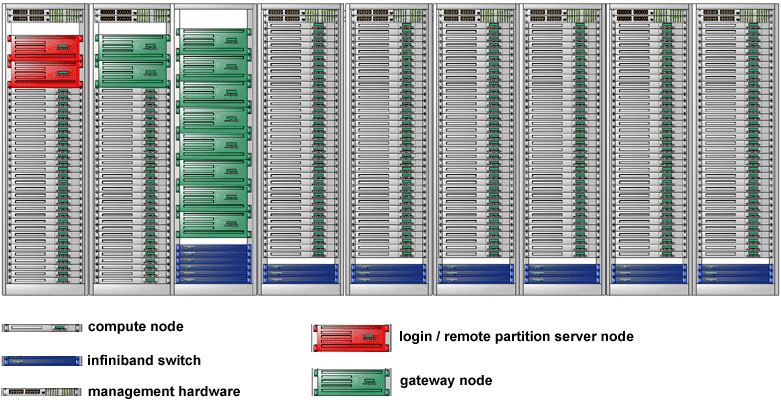
多硬件线程   
   
IBM BG/Q Compute Chip with 18 cores (PU) and 16 L2 Cache units (L2)

通过网络连接起来的多个单机也可以形成更大的并行计算机集群:   


例如，下面的图解就显示了一个典型的LLNL并行计算机集群：

每个计算结点就是一个多处理器的并行计算机；

多个计算结点用无限宽带网络连接起来；

某些特殊的结点（通常也是多处理器单机）被用来执行特定的任务。   


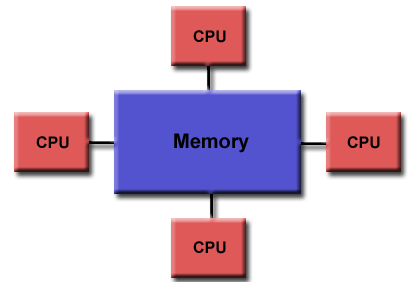
1. 并行计算机的内存架构

2.1 共享内存

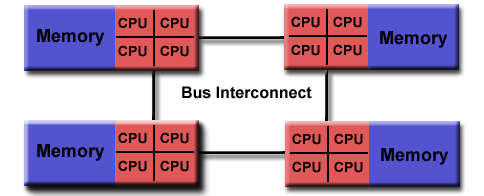
一般特征：

共享内存的并行计算机虽然也分很多种，但是通常而言，它们都可以让所有处理器以全局寻址的方式访问所有的内存空间。多个处理器可以独立地操作，但是它们共享同一片内存。一个处理器对内存地址的改变对其它处理器来说是可见的。根据内存访问时间，可以将已有的共享内存机器分为统一内存存取和非统一内存存取两种类型。

统一内存存取(Uniform Memory Access)： 目前更多地被称为对称多处理器机器(Symmetric Multiprocessor (SMP))，每个处理器都是相同的，并且其对内存的存取和存取之间都是无差别的。有时候也会被称为CC-UMA (Cache coherent - UMA)。缓存想干意味着如果一个处理器更新共享内存中的位置，则所有其它处理器都会了解该更新。缓存一致性是在硬件级别上实现的。



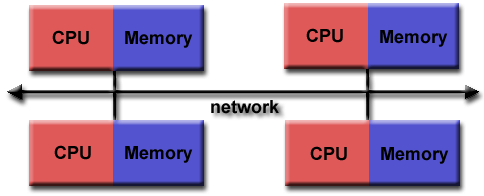
非统一内存存取(Non-Uniform Memory Access)： 通常由两个或者多个物理上相连的SMP。一个SMP可以存取其它SMP上的内存。不是所有处理器对所有内存都具有相同的存取或者存取时间。通过连接而进行内存存取速度会更慢一些。如果缓存相缓存想干的特性在这里仍然被保持，那么也可以被称为CC-NUMA。



优点：全局地址空间提供了一种用户友好的编程方式，并且由于内存与CPU的阶级程度，使得任务之间的数据共享既快速又统一。

缺点：最大的缺点是内存和CPU之间缺少较好的可扩展性。增加更多的CPU意味着更加共享内存和缓存想干系统上的存取流量，从而几何级别地增加缓存/内存管理的工作量。同时也增加了程序员的责任，因为他需要确保全局内存“正确”的访问以及同步。

2.2 分布式内存

一般概念： 分布式内存架构也可以分为很多种，但是它们仍然有一些共同特征。分布式内存结构需要通讯网络，将不同的内存连接起来。一般而言，处理器会有它们所对应的内存。一个处理器所对应的内存地址不会映射到其它处理器上，所以在这种分布式内存架构中，不存在各个处理器所共享的全局内存地址。   


由于每个处理器具有它所对应的局部内存，所以它们可以独立进行操作。一个本地内存上所发生的变化并不会被其它处理器所知晓。因此，缓存想干的概念在分布式内存架构中并不存在。

如果一个处理器需要对其它处理器上的数据进行存取，那么往往程序员需要明确地定义数据通讯的时间和方式，任务之间的同步因此就成为程序员的职责。尽管分布式内存架构中用于数据传输的网络结构可以像以太网一样简单，但在实践中它们的变化往往也很大。

优点：

1. 内存可以随着处理器的数量而扩展，增加处理器的数量的同时，内存的大小也在成比例地增加；
2. 每个处理器可以快速地访问自己的内存而不会受到干扰，并且没有维护全局告诉缓存一致性所带来的开销；

3）成本效益：可以使用现有的处理器和网络。

缺点：

1. 程序员需要负责处理器之间数据通讯相关的许多细节；
2. 将基于全局内存的现有数据结构映射到该分布式内存组织可能会存在困难；

3）非均匀的内存访问时间——驻留在远程结点上的数据比本地结点上的数据需要长的多的访问时间。

2.3 混合分布式-共享内存

一般概念： 目前世界上最大和最快的并行计算机往往同时具有分布式和共享式的内存架构。共享式内存架构可以是共线内存机器或者图形处理单元(GPU)。分布式内存组件可以是由多个共享内存/GPU连接而成的系统。每个结点只知道自己的内存，不知道网络上其它结点的内存。因此，需要在不同的机器上通过网络进行数据通讯。

从目前的趋势来看，这种混合式的内存架构将长期占有主导地位，并且成为高端计算在可见的未来中的最好选择。

优缺点：

1. 继承了共享式内存和分布式内存的优缺点；
2. 优点之一是可扩展性；

3）缺点之一是编程的复杂性。

1. **学习体会及总结**

通过本次实训课程的学习，我了解到了区块链是什么东西，并且认识到了区块链一路光明的发展前景：就从学术科研的角度来说，这种超前的规则和思想是我们应当学习和借鉴的，因为区块链不仅可以像深度学习一样创造领域突破，而且将会解决很多社会生活问题，例如社会固有的公平公正问题和大数据背景下的个人隐私问题。

同时，我还学习到了密码学的基本原理，如Hash函数。对称与非对称加密算法，群/环签名等；了解了共识算法的概念和几款常用共识算法（POW、POS、DPOS、PBFT等）；学习了Solidity语言等等。

同样的，我还了解到了FISCO BCOS平台，学会了如何用FISCO BCOS进行联盟链搭建，并编写智能合约，以及部署区块链应用的能力。

这些都将对我日后的学习工作生活起到很大的引导和帮助作用。

**参考文献**

1. 《一文读懂区块链技术逻辑和原理》——来源CSDN文章
2. 《中国区块链技术和应用发展白皮书》
3. 《区块链-以太坊MPT存储》——企鹅号-星想法
4. 《以太坊中智能合约调用中用的gas相关概念详解》——来源CSDN文章
5. 《以太坊：EVM的存储结构》——Firefly
6. 《并行计算简介》——来源CSDN文章