华南理工大学

课程报告

课程名称：	企业软件项目实训
学生姓名：	翁焕滨
学生学号：	**201630665861**
学生专业：	软件工程专业
开课学期：	**2018-2019 第二学期**

软件学院

2019 年 7 月

目录

1.实训目的

（1） 掌握区块链基础知识。

（2） 在掌握区块链基础知识的前提下合作开发一个基于智能合约的区块链。

2.实训环境

（1） 软件：vs code（前端） IDEA（后端）REMIX（智能合约）

（2） 硬件：ubantu 系统

3.关键知识

（1）区块链简介

“区块链”技术最初是由一位化名中本聪的人为比特币（一种数字货币）而设计出的一种特殊的数据库技术，它基于密码学中的椭圆曲线数字签名算法（ECDSA）来实现去中心化的 P2P 系统设计。但区块链的作用不仅仅局限在比特币上。现在，人们在使用“区块链”这个词时，有的时候是指数据结构，有时是指数据库，有时则是指数据库技术，但无论是哪种含义，都和比特币没有必然的联系。

从数据的角度来看：区块链是一种分布式数据库（或称为分布式共享总账，DistributedShared Ledger），这里的“分布式”不仅体现为数据的分布式存储，也体现为数据的分布式记录（即由系统参与者来集体维护）。简单的说，区块链能实现全球数据信息的分布式记录（可以由系统参与者集体记录，而非由一个中心化的机构集中记录）与分布式存储（可以存储在所有参与记录数据的节点中，而非集中存储于中心化的机构节点中）。

从效果的角度来看：区块链可以生成一套记录时间先后的、不可篡改的、可信任的数据库，这套数据库是去中心化存储且数据安全能够得到有效保证的。

结论：区块链是一种把区块以链的方式组合在一起的数据结构，它适合存储简单的、有先后关系的、能在系统内验证的数据，用密码学保证了数据的不可篡改和不可伪造。它能够使参与者对全网交易记录的事件顺序和当前

状态建立共识。

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。

（2）区块链技术原理：

实际上，区块链是分布式数据存储、点对点传输、共识机制、加密算法等计算机技术的新型应用模式。

从架构模型来看，区块链系统由数据层、网络层、共识层、激励层、合约层和应用层组成 。其中，数据层封装了底层数据区块以及相关的数据加密和时间戳等基础数据和基本算法；网络层则包括分布式组网机制、数据传播机制和数据验证机制等；共识层主要封装网络节点的各类共识算法；激励层将经济因素集成到区块链技术体系中来，主要包括经济激励的发行机制和分配机制等；合约层主要封装各类脚本、算法和智能合约，是区块链可编程特性的基础；应用层则封装了区块链的各种应用场景和案例。

从技术角度来看，区块链系统主要运用了一下几项技术：

1） 分布式账本技术：相较于传统的中心化数据库存储所有数据，分布式账本是分布在多个节点或多台计算设备上的数据库，每个节点都可以复制并保存一个账本，且每个节点可以独立更新其内容。分布式账本是区块链实现去中心化这一特点的重要部分。

2） 共识机制：有了分布式账本，就产生了另一个问题：如何确定分布

式账本上新的一条记录是否真实可信呢？这个时候就需要共识机制了。共识机制就是所有记账节点达成共识，认定一个记录的有效性的一种机制。共识机制具有“少数服从多数”，“人人平等”的特点，其中“少数服从多数”不一定指节点数，也可以是计算能力、股权数或其他可以比较的特征量；“人人平等”是当节点满足条件时，所有节点都有权优先提出共识结果、直接被其他节点认同后并最后有可能成为最终共识结果。这样的两个特点使得伪造一条记录的代价是至少控制链上 51%的节点，而当链上的节点足够多的时候，这样的条件是基本不可能达成的。

3） 非对称加密：非对称加密是指加密和解密所使用的密钥是不同的，即使用公钥加密的内容，只有使用私钥加密才能看到。由于存储在区块链上的交易信息是公开的，而账户的身份信息是高度加密的，这一技术使得区块链上只需存储用户的公钥即可唯一标识一个用户的身份，而用户可以用私钥去操作自己的账户、进行交易，这就保证了区块链的安全性。

4） 智能合约：智能合约是基于区块链上不可篡改的数据这一特性，预先定义好的一些可以自动化执行的规则与条款。通过设计智能合约，我们可以按照想要的方式对区块链上的数据进行存储与查找，实现与传统的运用数据库作为存储工具的中心化系统有着相近功能的去中心化系统。

（3）联盟链和公有链的区别：

私有链是对单独的个人或实体进行开放的区块链系统。系统内的每个节点的权限都需要组织来分配，对每个节点开放的数据量要视情况由组织来决定。虽然对各个节点都进行了限制，但私有链仍然是区块链的多节点的框架。

私有链有一些自己的特点。首先就是它的交易速度会很快，私有链的交易速度是其他公有链和联盟连所不能比的，主要是因为不需要每个节点来验证一个交易，少量的节点就可以完成验证。其次，更好的隐私保护。由于读取数据的权限受限，任何节点参与者很难获得数据链上面的数据。再次，节

点连接方便。私有链中的节点连接是很方便的。最后，交易成本很是便宜。对于每一笔交易，只需要算力比较好的由信任度高的几个节点验证就可以了。这样就大大降低了交易所花费的成本。

公有区块链不受第三方机构控制，世界上所有的人都可读取链上的数据记录、参与交易以及竞争新区块的记账权等。程序开发者无权干涉用户，各参与者（即节点）可自由加入以及退出网络，并按照意愿进行相关操作。

联盟链则是介于公有链以及私有链之间的区块链，可实现“部分去中心化”。链上各个节点通常有与之相对应的实体机构或者组织；参与者通过授权加入网络并组成利益相关联盟，共同维护区块链运行。从某种程度上来说，联盟链也属于私有链的范畴，只是私有化程度有所不同而已。为此其同样具有成本较低、效率较高的特点，适用于不同实体间的交易、结算等 B2B 交易。

（4）链式存储和 MPT 存储：

链式存储结构，又叫链接存储结构。在计算机中用一组任意的存储单元存储线性表的数据元素(这组存储单元可以是连续的,也可以是不连续的)。它不要求逻辑上相邻的元素在物理位置上也相邻.因此它没有顺序存储结构所具有的弱点,但也同时失去了顺序表可随机存取的优点。链式存储有以下六大特点：1、比顺序存储结构的存储密度小(链式存储结构中每个结点都由数据域与指针域两部分组成，相比顺序存储结构增加了存储空间)。2、逻辑上相邻的节点物理上不必相邻。3、插入、删除灵活 (不必移动节点，只要改变节点中的指针)。4、查找节点时链式存储要比顺序存储慢。5、每个节点是由数据域和指针域组成。6、由于簇是随机分配的，这也使数据删除后覆盖几率降低，恢复可能提高。

MPT 存储：

Trie 树，又称前缀树或字典树，是一种有序树，用于保存关联数组，其中的键通常是字符串。与二叉查找树不同，键不是直接保存在节点中，而是由节点在树中的位置决定。一个节点的所有子孙都有相同的前缀，也就是这个节点对应的字符串，而根节点对应空字符串。一般情况下，不是所有的节点都有对应的值，只有叶子节点和部分内部节点所对应的键才有相关的值。

Patricia 树，或称 Patricia trie，或 crit bit tree，压缩前缀树，是一种更节省空间的 Trie。对于基数树的每个节点，如果该节点是唯一的儿子的话，就和父节点合并。

Merkle Tree，通常也被称作 Hash Tree，顾名思义，就是存储 hash 值的一棵树。Merkle 树的叶子是数据块(例如，文件或者文件的集合)的 hash 值。非叶节点是其对应子节点串联字符串的 hash。在点对点网络中作数据传输的时候，会同时从多个机器上下载数据，而且很多机器可以认为是不稳定或者不可信的。为了校验数据的完整性，更好的办法是把大的文件分割成小的数据块（例如，把分割成 2K 为单位的数据块）。这样的好处是，如果小块数据在传输过程中损坏了，那么只要重新下载这一快数据就行了，不用重新下载整个文件。怎么确定小的数据块没有损坏哪？只需要为每个数据块做 Hash。BT 下载的时候，在下载到真正数据之前，我们会先下载一个 Hash 列表。那么问题又来了，怎么确定这个 Hash 列表本事是正确的哪？答案是把每个小块数据的 Hash 值拼到一起，然后对这个长字符串在作一次 Hash 运算，这样就得到 Hash 列表的根 Hash(Top Hash or Root Hash)。下载数据的时候，首先从可信的数据源得到正确的根 Hash，就可以用它来校验 Hash 列表了，然后通过校验后的 Hash 列表校验数据块。在最底层，和哈希列表一样，我们把数据分成小的数据块，有相应地哈希和它对应。但是往上走，并不是直接去运算根哈希，而是把相邻的两个哈希合并成一个字符串，然后运算这个字符串的哈希，这样每两个哈希就结婚生子，得到了一个”子哈希“。如果最底层的哈希总数是单数，那到最后必然出现一个单身哈希，这种情况就直接对它进行哈希运算，所以也能得到它的子哈希。于是往上推，依然是一样的方式，可以得到数目更少的新一级哈希，最终必然形成一棵倒挂的树，到了树根的这个位置，这一代就剩下一个根哈希了，我们把它叫做 Merkle Root。在 p2p 网络下载网络之前，先从可信的源获得文件的 Merkle Tree 树根。一旦获得了树根，就可以从其他从不可信的源获取 Merkle tree。通过可信的树根来检查接受到的 MerkleTree。如果 Merkle Tree 是损坏的或者虚假的，就从其他源获得另一个 Merkle Tree，直到获得一个与可信树根匹配的 MerkleTree。

MPT（Merkle Patricia Tree）就是 Patricia 树与 Merkle 树混合后的产物。MPT 树中的节点包括空节点、叶子节点、扩展节点和分支节点:空节点，简

单的表示空，在代码中是一个空串；叶子节点（leaf），表示为[key,value]的一个键值对，其中 key 是 key 的一种特殊十六进制编码，value 是 value 的 RLP 编码；扩展节点（extension），也是[key，value]的一个键值对，但是这里的 value 是其他节点的 hash 值，这个 hash 可以被用来查询数据库中的节点。也就是说通过 hash 链接到其他节点；分支节点（branch），因为 MPT 树中的 key 被编码成一种特殊的 16 进制的表示，再加上最后的 value，所以分支节点是一个长度为 17 的 list，前 16 个元素对应着 key 中的 16 个可能的十六进制字符，如果有一个[key,value]对在这个分支节点终止，最后一个元素代表一个值，即分支节点既可以搜索路径的终止也可以是路径的中间节点。

MPT 树中另外一个重要的概念是一个特殊的十六进制前缀(hex-prefix, HP)编码，用来对 key 进行编码。因为字母表是 16 进制的，所以每个节点可能有 16 个孩子。因为有两种[key,value]节点(叶节点和扩展节点)，引进一种特殊的终止符标识来标识 key 所对应的是值是真实的值，还是其他节点的 hash。如果终止符标记被打开，那么 key 对应的是叶节点，对应的值是真实的 value。如果终止符标记被关闭，那么值就是用于在数据块中查询对应的节点的 hash。无论 key 奇数长度还是偶数长度，HP 都可以对其进行编码。最后我们注意到一个单独的 hex 字符或者 4bit 二进制数字，即一个 nibble。

HP 编码很简单。一个 nibble 被加到 key 前（下图中的 prefix），对终止符的状态和奇偶性进行编码。最低位表示奇偶性，第二低位编码终止符状态。如果 key 是偶数长度，那么加上另外一个 nibble，值为 0 来保持整体的偶特性。

（5）Gas 在智能合约中的作用：

在以太坊网络当中有个比较重要的概念——费用，具体来说，在以太坊网络上的交易而产生的每一次计算，都会产生费用——这个费用是以称之为”gas”的单位来支付。gas 就是用来衡量在一个具体计算中要求的费用单位。gas price 就是你愿意在每个 gas 上花费 Ether 的数量，以“gwei”进行衡量。“Wei”是 Ether 的最小单位，1Ether 表示 10^18Wei. 1gwei 是 1,000,000,000 Wei。对每个交易，发送者设置 gas limit 和 gas price。gas limit 和 gas price 就

代表着发送者愿意为执行交易支付的 Wei 的最大值。

gas limit 代表用户愿意花费在 gas 上的钱的最大值。如果在他们的账户余额中有足够的 Ether 来支付这个最大值费用，那么就没问题。在交易结束时任何未使用的 gas 在发送者没有提供足够的 gas 来执行交易，那么交易执行就会出现“gas 不足”然后被认为是无效的。在这种情况下，交易处理就会被终止以及所有已改变的状态将会被恢复，最后我们就又回到了交易之前的状态—完完全全的之前状态就像这笔交易从来没有发生。因为网络在耗尽 gas 之前还是为计算做出了努力，所以理论上，将不会有任何的 gas 被返回给发送者。都会被返回给发送者，以原始费率兑换。

那么这些 gas 的钱到底去了哪里？发送者在 gas 上花费的所有钱都发送给了“受益人”地址，通常情况下就是矿工的地址。因为矿工为了计算和验证交易做出了努力，所以矿工接收 gas 的费用作为奖励。

通常，发送者愿意支付更高的 gas price，矿工从这笔交易总就能获得更多的价值。因此，矿工也就更加愿意选择这笔交易。这样的话，矿工可以自由的选择一笔交易自己愿意验证或忽略。为了引导发送者应该设置 gas price 为多少，矿工可以选择建议一个最小的 gas 值他们愿意执行一个交易。

gas 不仅仅是用来支付计算这一步的费用，而且也用来支付存储的费用。存储的总费用与所使用的 32 位字节的最小倍数成比例。

存储费用有一些比较细微的方面。比如，由于增加了的存储增加了所有节点上的以太坊状态数据库的大小，所以激励保持数据存储量小。为了这个原因，如果一个交易的执行有一步是清除一个存储实体，那么为执行这个操作的费用就会被放弃，并且由于释放存储空间的退款就会被返回给发送者。

以太坊可以运作的一个重要方面就是每个网络执行的操作同时也被全节点所影响。然而，计算的操作在以太坊虚拟机上是非常昂贵的。因此，以太坊智能合约最好是用来执行最简单的任务，比如运行一个简单的业务逻辑或者验证签名和其他密码对象，而不是用于复杂的操作，比如文件存储，电子邮件，或机器学习，这些会给网络造成压力。施加费用防止用户使网络超负荷。

以太坊是一个图灵完备语言（短而言之，图灵机器就是一个可以模拟任何电脑算法的机器。对于图灵机器不太熟悉的人可以看看这个 和这个 ）。这

就允许有循环，并使以太坊受到停机问题 的影响，这个问题让你无法确定程序是否无限制的运行。如果没有费用的话，恶意的执行者通过执行一个包含无限循环的交易就可以很容易的让网络瘫痪而不会产生任何反响。因此，费用保护网络不受蓄意攻击。

（6）EVM 中的数据存储结构:

在 Solidity 中，有两个地方可以存储变量 ：存储（storage）以及内存（memory）。Storage 变量是指永久存储在区块链中的变量。Memory 变量则是临时的，当外部函数对某合约调用完成时，内存型变量即被移除。

内存(memory)位置还包含 2 种类型的存储数据位置，一种是 calldata，一种是栈（stack）。calldata 是一块只读的，且不会永久存储的位置，用来存储函数参数。 外部函数的参数（非返回参数）的数据位置被强制指定为 calldata ，效果跟 memory 差不多。EVM 是一个基于栈的语言，栈实际是在内存(memory)的一个数据结构，每个栈元素占为 256 位，栈最大长度为 1024。 值类型的局部变量是存储在栈上。

这四种不同的存储方式的开销是不同的：

storage 会永久保存合约状态变量，开销最大；

memory 仅保存临时变量，函数调用之后释放，开销很小；

stack 保存很小的局部变量，免费使用，但有数量限制(16 个变量)；

calldata 的数据包含消息体的数据，其计算需要增加 n*68 的 GAS 费用；

storage 和 memory 还有；两个不同点：

storage 存储结构是在合约创建的时候就确定好了的，它取决于合约所声明状态变量。但是内容可以被（交易）调用改变。Solidity 称这个为状态改变，这也是合约级变量称为状态变量的原因。也可以更好的理解为什么状态变量都是 storage 存储。

memory 只能用于函数内部，memory 声明用来告知 EVM 在运行时创建一块（固定大小）内存区域给变量使用。

storage 在区块链中是用 key/value 的形式存储，而 memory 则表现为字节数组。

（7）分布式存储有什么优势：

分布式存储是一种数据存储技术，通过网络使用每台机器上的磁盘空间，并将这些分散的存储资源构成一个虚拟的存储设备，数据分散的存储在网络中的各个角落。所以，分布式存储技术并不是每台电脑都存放完整的数据，而是把数据切割后存放在不同的电脑里。就像存放 100 份数据，不是放在同一个服务器里，而是分开放在不同的地方，加起来的总和是 100 份。对于比特币来说，它的交易记录必须要有地方存放，不然没人知道今天有哪些人做了交易，同时根据去中心化的思想，这些交易记录不能够只存在一台电脑里面，那么就只能存放在世界上所有的电脑里面。这样做的好处是：虽然每个人的电脑硬盘容量有限，但是所有人的电脑硬盘加起来容量几乎是无限的，而且就算你通过黑客手段修改了自己计算机里面的交易记录，但是你没法修改全世界每台电脑的交易记录。从表面上理解，上面说的这种存储方式很粗暴——每台电脑都存放世界上所有人的交易数据。但其实对于比特币来说，只有一些节点才会存放世界上所有人的交易记录，这些节点往往是那些挖矿的矿工，只有他们的电脑才能完整的记录下世界上所有的交易记录，大家不用担心矿工修改记录，因为世界上的矿工有很多，而且几乎相互都不认识。同时他们修改记录需要付出的代价非常大，没有人能承担这个成本。把亿万用户已有的亿万设备中闲置的空间变成别人数据的储存空间，这会降低世界对服务器的需求。

随着存储技术的发展，存储设备的成本越来越小，中心化云服务的成本主要来自于员工工资、法律成本、数据中心租金等，这些固定成本是不变的或逐渐增加，使中心化云服务的价格较高。而去中心化

存储成本只有中心化存储的 1/100-1/10，如果去中心化存储系统是完全自动化的，云存储价格最终会降到接近 0，中心化云服务的规模优势将败给了去中心化云服务。

4.实训具体实践

在这次实训的大作业中，我主要负责了需求分析、数据模型设计、智能合约开发和系统物理部署图的绘制这几项工作。

需求分析阶段，我与几位组员共同商讨了本系统——宠物商店的定位、包含的功能、系统所包含的角色、每个角色可执行的操作以及系统的功能需求与以及非功能需求。最终，我们确定了用户、管理员这两个使用者，订单、宠物两个被操作的系统对象以及购买宠物、出售宠物、退货等一系列需要实现的功能。

数据模型设计阶段，我根据用户、管理员、订单和宠物这几个对象，结合系统所需要实现的功能以及组员编写的用例文档，设计出了本系统中所需的所有数据的模型，包括数据的名称、意义、类型、字段名、所属角色等信息，为前端、后端与合约的后续开发在数据方面统一了命名标准以及帮助前后端在已知字段名的情况下独立开发，为后续对接节省沟通字段名的时间。

智能合约开发阶段，我先按照前一阶段设计的数据模型，将合约中的对象分为用户、订单、宠物这三者，然后就开始对这三个对象在合约中实现。一开始，我将用户、订单、宠物以结构体的方式声明，每个用户结构体有一个宠物结构体数组，用于存放用户个人的宠物，后来在实现上架宠物时发现，如果用户要看到市场上所有在售的宠物，那必须维护一个市场在售宠物的列表，这将占用区块链上不少空间，而且在其中一只宠物被购买之后，需要将该宠物后面的所有宠物向前移一位，这将会消耗大量的 gas，综合实现的难度与区块链上宝贵的空间，我选择了在全局维护一个所有人的宠物列表，将拥有者的地址存为一个属性，用以标识宠物的所属，上架状态也以宠物的一个属性实现，这样在展示市场在售宠物的时候只需要将状态为上架的宠物显示出来就行，上架、下架、购买宠物引起的宠物上架状态的更改也只需要更改一个属性，大大减少了需要的存储空间，也将原本繁琐的代码变得简单易

懂，大大提高了代码的可读性与可维护性。对于订单对象，我也做了于宠物对象相似的实现，只是其中的属性有所不同。对于用户对象，我在经过仔细考察之后认为，用户对象不需要像订单和宠物对象那样需要遍历所有用户的需要，而且用户的属性一般使用用户地址来找到其对应值，这与 solidity 编程中的 mapping 非常相似，我便将用户对象的属性拆分为多个映射，将用户地址映射到用户的余额、是否创建宠物等属性上，这样也使得在通过用户地址获取其余额的这类操作的情况下不需要使用 for 循环结构去遍历所有用户的地址，而可以简单地使用一个映射去得到所需的数据，使得代码更加简练，也提高了效率。

宠物、订单结构体及用户映射：

```
//pet结构体
struct Pet {
    string petName;
    string petId;
    string petType;
    uint16 petPrice;
    uint8 petStatus;
    string petImg;
//映射：
//判断是否已创建过宠物
mapping (address => uint8) createdPe }

//地址与余额映射
mapping (address => uint) Balance;

//判断是否老用户
mapping (address => uint8) oldUser;

//宠物id到用户地址的映射
mapping (string => address) petIdToOwner;

//用户地址与身份的映射,未注册用户是0,普通用户是1,管理员是2
mapping(address => uint8) userIden;
```

```
//订单结构体
struct Order {
    string orderId;
    address orderBuyer;
    address orderSeller;
    string orderTime;
    string petId;
    uint16 petPrice;
    uint8 orderStatus;
    string returnReason;
```

完成对象的实现后，就开始实现相关函数了。根据系统需要实现的功能，我先将对应的后端所需调用的外部调用函数实现，再去实现这些函数需要的内部函数，如支付、改变宠物持有者，生成订单等。在完成了大部分功能，尝试着部署的时候，我发现 remix 上的部署工具报出了 gas 不足的错误，无法成功部署，根据网上的资料以及几次测试发现，这是合约中函数过多引起的，于是将原先的合约的订单部分独立出来，写成一个合约，并把市场合约需要用到的订单合约的函数写成一个接口函数，成功解决无法部署的问题。

在实现返回用户宠物、返回市场在售宠物、返回所有订单这类需要返回结构体数组的功能中，我发现 solidity 编程语言不支持返回结构体，结构体数组以及字符串数组，这给我实现这类功能带来了困难。经组员的启发，我尝试将需要返回的内容组装成一个很长的字符串，每一项数据之间用逗号隔开，让后端接收并分解，组装成 json 发送给前端，然而，需求中订单内容需要返回买家和卖家的地址，而我无法实现将地址转化为字符串，只得将这一方法放弃。最后，我改变了实现的方法，将需要返回的订单在数组中的下标作为一个整型数组返回给后端，后端再利用这个下标数组循环向合约取单个订单的内容，而合约中函数用多值返回将订单中的信息一并返回给后端，再由后端打包数据，传给前端。

将合约的功能全部实现之后，我们对合约进行了控制台部署并测试其功能能否正常运行，然后利用 postman 测试后端能否正常调用合约中的各个函数以及函数能否正常使用，最后将前端也连接起来，对整个系统进行了系统测试。在系统测试中，我们不但进行了正常操作流程的测试，也做了一些容易造成系统异常或逻辑错误的操作测试，并发现了一些本不应出现的逻辑错误：用户 A 将自己的宠物以 200 元的价格上架，在用户 B 查看到了市场在售宠物中用户 A 上架的宠物以及其价格 200 元后，用户 A 将宠物价格更改为 5000 元，由于我们的系统无法实时更新，会导致该宠物在后台的价格已经更改为 5000 元，而用户 B 的界面上显示的仍然是 200 元，在这种情况下，用户 B 下单之后将会在不知情的情况下支付 5000 元，这就要求合约实现在宠物上架的时候不可更改宠物价格；用户 B 向用户 A 购买了宠物，之后选择退货，在管理员处理该笔退货请求之前，用户 B 将宠物卖给了用户 C，若

此后管理员同意了用户 B 发起的退货请求，宠物将会从用户 C 手上回到用户 A 手上，钱会从用户 A 手里回到用户 B 手里，这是不应出现的严重错误，这要求系统在用户上架宠物之前检查该宠物是否正处于被退货的情况；与前一个情况相近，用户在上架了一个宠物之后也不应被允许将这只宠物退还给卖家；若一个用户购买了同一宠物两次，那他的订单列表中将有两个订单可以选择退还该宠物，而在预设的逻辑中，只有一个退货请求可以被允许，因为一次退货之后，宠物将归还原主，第二次退货请求不应成功，这要求系统在完成退货之前先检测该宠物是否仍在买家手上，确认后再进行宠物以及钱的退还。

最后，我还进行了系统物理部署图的绘制。我们的系统由两台服务器组成——前端服务器与后端服务器，前端由 vue 实现，负责展示合约里存储的用户、订单、宠物、市场信息，与用户进行图形界面的交互以及将用户输入的信息传给后端这三项功能。后端分为两部分，一部分是 spring boot 部分，用于接收前端发送过来的用户操作信息，经过处理之后，调用合约中相关的函数，将合约需要的数据传进去，起承接作用；另一部分是智能合约部分，用于接收后端发送来的数据，对其进行处理并存储在区块链中。spring boot 部分与合约部分通过 Web3SDK 进行交互。

5.实训个人心得体会

在这次实训之前，我对区块链的认识仅仅停留在“区块链”“比特币”这两个名词上，而通过这次实训，我首次接触了区块链技术，并对区块链技术的原理以及应用有了初步的认识与了解。在前半部分的学习中，我在老师的指导下搭建了自己的区块链，学习了 solidity 编程语言，编写了第一个合约并将其部署在链上。在后半部分的实践中，我与组员合作开发了一个基于区块链的宠物商店，让我学到了很多。在项目的初期，由于对 solidity 编程语言的不了解，合约的开发进度非常慢，但得益于合理的项目管理的产物——Api 文档，前端和后端的部分可以不受合约开发进度的制约，独立进行开发。在合约开发的过程中，为了实现功能需求，合约中数据的存储方式、函数的实现方式也经历了一次又一次的改变，花费了大量的时间，这也让我意识到如

果在合约开发之前对合约有一个大致的实现规划可以节省不少开发的时间成本。在合约开发的后期，通过组员的细致检查与周密的测试，我们发现了几个严重的逻辑错误，甚至可能造成用户的经济损失，这让我意识到要设计出一个可靠的、不容易出错的系统需要在设计阶段就把这些异常情况做出详细分析，这样才能在开发阶段尽早地将其解决。通过这个项目的开发，我也切身感受到了规范的项目管理带来的好处——在项目进行的各个阶段可以更好地追踪项目进展，控制项目规模，让项目按部就班地进行，并在规定期限之前将项目完成。

6.参考文献

[1] 区块链技术原理——https://blog.csdn.net/zhangcanyan/article/details/51933424

[2] 百度百科区块链——https://baike.baidu.com/item/%E5%8C%BA%E5%9D%97%E9%93%BE/13465666?fr=aladdin#reference-[11]-13112042-wrap

[3] 以太坊中MPT详解—— https://blog.csdn.net/zhangcanyan/article/details/51933424

[4] 以太坊中费用概念详解——https://3kemao.com/archives/53044

[5] 以太坊存储类型(memory,storage)及变量存储详解——https://www.jianshu.com/p/f98836ced529

[6] 区块链分布式存储的优势——https://www.jianshu.com/p/9dba1af3734d