**区块链实验Ex5**

1. **实验设计**
2. 合约中的函数
3. lookup(address debtor, address creditor) public view returns

(uint32 ret):

该函数返回债务人欠债权人的金额.

1. add\_IOU(address creditor, uint32 amount, ...):

为调用者添加一个欠条,如果你已经欠钱，金额会增加。金额必须为正数。

1. verify\_and\_fix\_path(address start, address end, ...）:

用于验证路径是否有效，并在需要时修复路径。

1. add(uint32 a, uint32 b):

带有溢出检查的加法函数，用于确保加法操作不会导致溢出。

1. 客户端中的功能
2. getUsers():

返回一个地址列表，这个地址列表包含:“曾经发送或收到欠条的每个人”或“目前欠或被欠钱的每个人”。

1. getTotalOwed(user):

返回指定用户所欠的总金额。

1. getLastActive(user):

返回该用户上次记录活动的UNIX时间戳(自1970年1月1日以来的秒数)(发送欠条或被列为欠条上的“债权人”)。如果找不到活动，则返回null。.

1. add\_IOU(creditor, amount):

向合约提交一个欠条以及相应的债权人和所欠数量，不返回任何值，参见下面的循环解析说明。

1. getNeighbors(node):

返回node该欠债人所欠债的债权人，根据提示为方便调用doBFS函数及其他调用

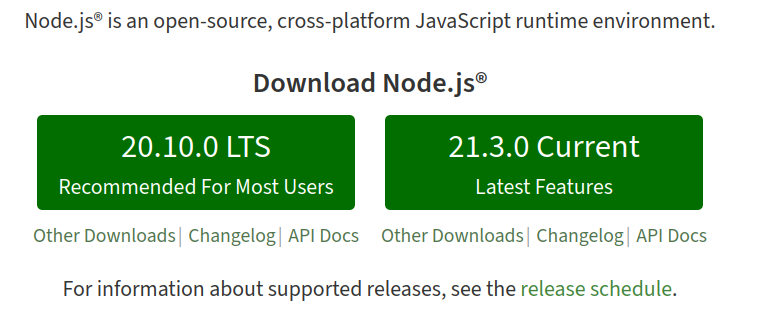
1. 辅助函数：
2. getAllFunctionCalls(addressOfContract, functionName)：

在块历史记录中搜索对“addressOfContract”合约上函数名为“functionName”的所有调用返回一个对象数组，每个调用一个对象数组，包含发送方（“from”）和参数（“args”）

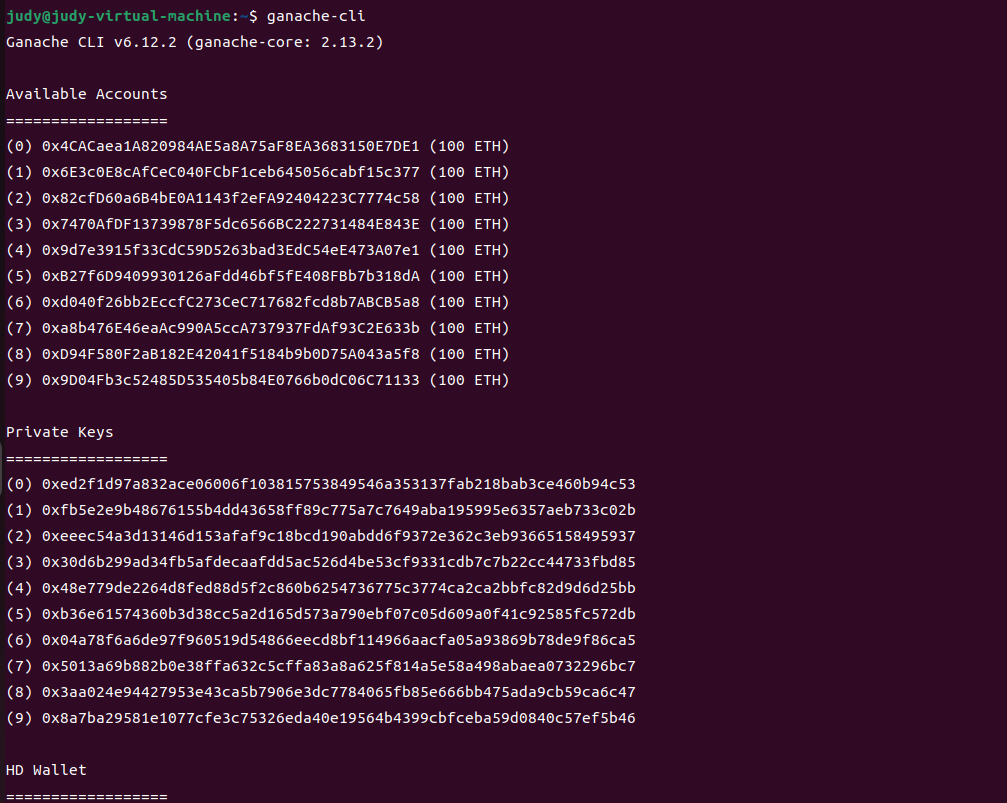
1. doBFS(start, end, getNeighbors)：

找到从头到尾的路径（如果不存在，则返回 null）

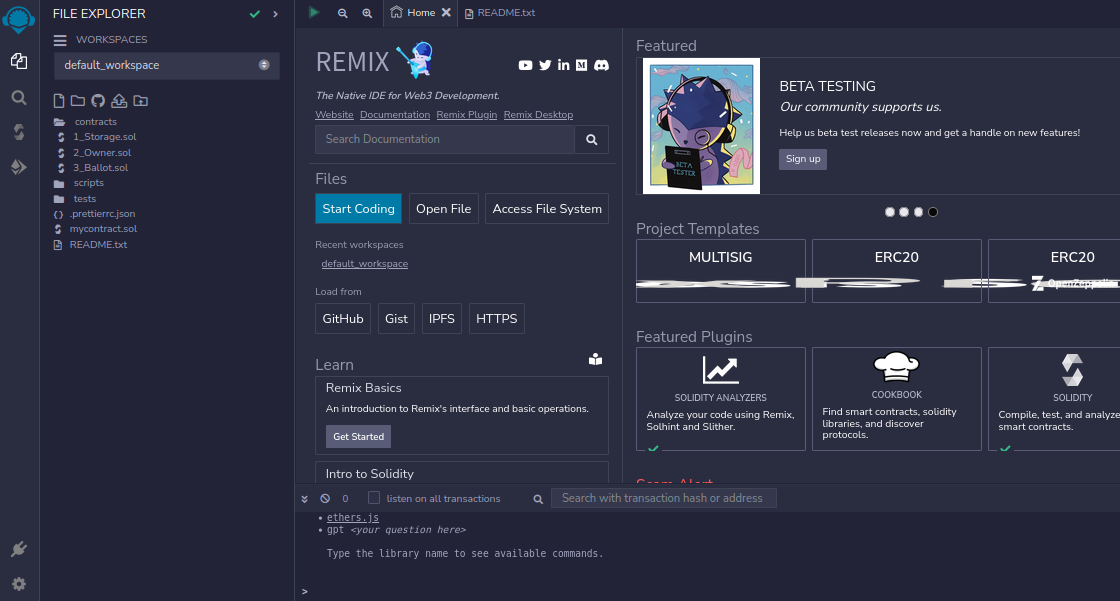
1. **实验过程**
2. 选择LTS版本Node.js下载安装

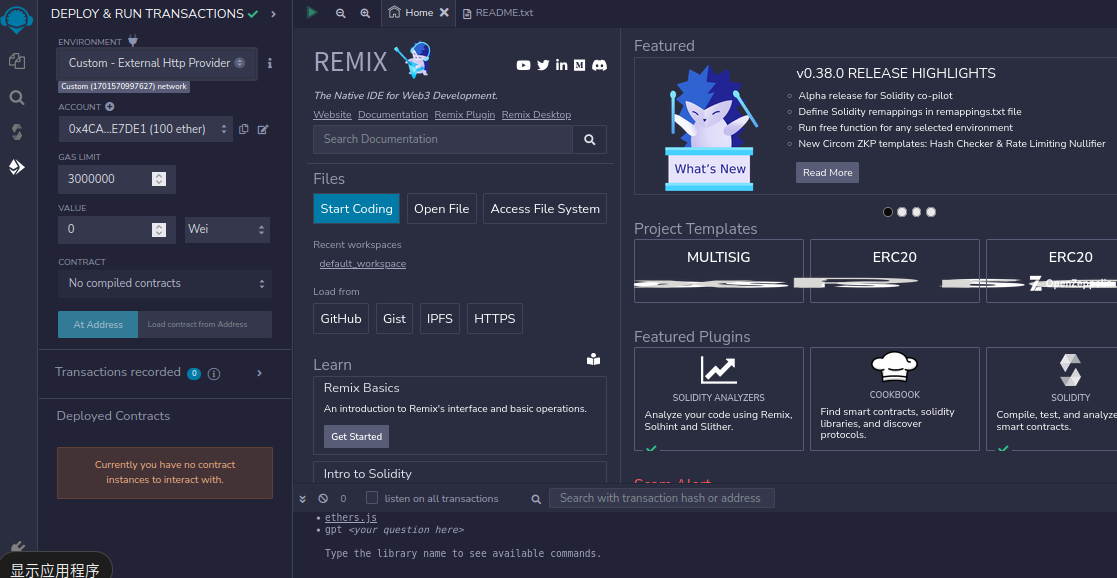
****

1. 运行npm install -g Ganache-CLI来安装Ganache CLI，终端ganache-cli运行节点，使用Ctrl-C停止节点。

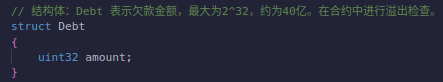


1. 在网页浏览器中打开https://remix.ethereum.org。Workspaces选择default-workspace，Environment选择Custom - External Http Provider。

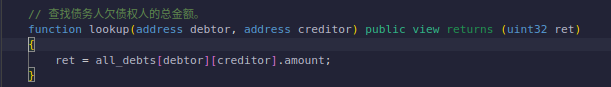




1. 编写Solidity智能合约，名为BlockchainSplitwise
2. 结构体Debt,表示欠款金额，使用 uint32 类型存储，最大值为2^32，约为40亿。在合约中使用溢出检查来确保安全操作。



1. all\_debts 是一个双重映射，从债务人到债权人的映射，记录了每个用户之间的欠款关系。例如，debts[Alice][Bob] = 10 表示Alice欠Bob 10。
2. lookup 函数用于查询债务人欠债权人的总金额，返回一个 uint32 类型的值。



1. add\_IOU 函数用于添加欠款记录，表示消息发送者欠债权人一定金额。先检查债务人和债权人是否相同，获取债务记录的引用。如果没有提供路径（即没有循环），则只需添加欠款并返回；如果提供了路径，确保路径存在并连接了债权人到债务人。验证提供的路径是否存在循环，并修复路径。添加新的欠款记录，确保不会创建循环（避免出现死循环或无限循环的情况）。路径可以是部分固定的，所以调用者负责撤消部分固定的路径。最简单的方法是使用require（），因为这将回滚事务。

参数：

creditor：债权人的地址。

amount：欠款金额。

path：一个地址数组，表示从债权人到消息发送者的已存在路径。

min\_on\_cycle：循环的最小金额，将从路径上的所有欠款中减去。



1. verify\_and\_fix\_path 函数用于验证路径是否有效，并在需要时修复路径。该函数验证路径的起点和终点是否正确，验证路径的长度是否合法。遍历路径，检查欠款是否存在且大于等于最小金额，如果不是，则表示路径无效；如果路径有效，继续修复路径。

参数：

start：路径起点。

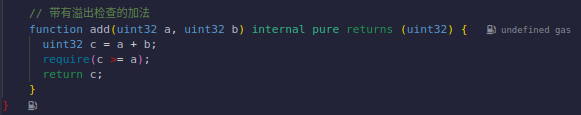
end：路径终点。

path：一个地址数组，表示路径。

min\_on\_cycle：循环的最小金额。



1. add 函数是一个带有溢出检查的加法函数，用于确保加法操作不会导致溢出。

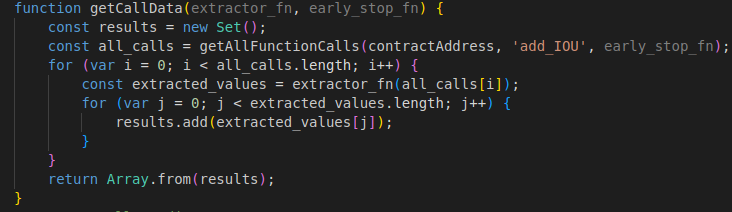


1. 在script.js中实现客户端

getCallData(extractor\_fn, early\_stop\_fn)函数从区块链上的所有调用中提取信息，并使用提供的提取器函数进行处理。

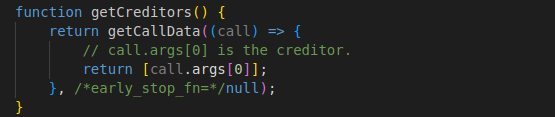
提取器函数将一个调用作为输入，并将其转换为多个值（列表），这些值将被收集并去重。通过遍历所有调用，应用提取器函数，并将提取的值添加到集合中，最后返回一个数组。

1. extractor\_fn：提取器函数，接受一个调用对象并返回一个包含提取的值的数组。
2. early\_stop\_fn：提前停止函数，传递给 getAllFunctionCalls 函数。

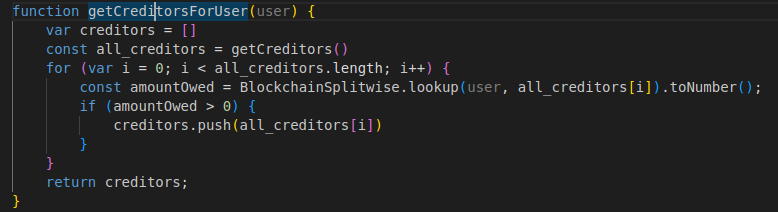


getCreditors()函数获取系统中所有的债权人（所有曾经发生过 IOU 的人）。

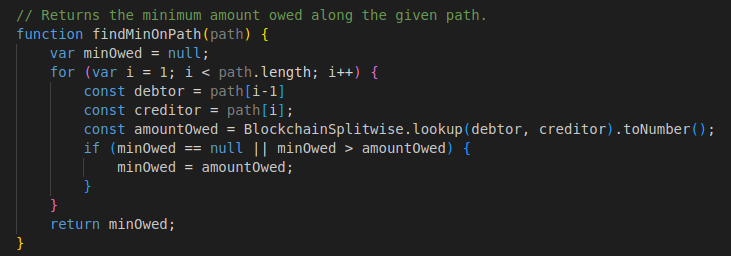
调用 getCallData 函数，提供一个提取器函数，该函数从每个调用中提取债权人地址。



1. getCreditorsForUser(user)函数用于获取给定用户欠款的所有债权人。遍历所有债权人，通过调用 BlockchainSplitwise.lookup 函数获取用户与每个债权人之间的欠款金额，如果金额大于0，则将债权人添加到结果数组中。

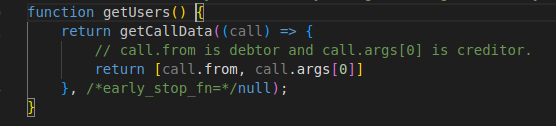


1. findMinOnPath(path)函数用于查找沿给定路径的最小欠款金额。遍历路径，通过调用 BlockchainSplitwise.lookup 函数获取路径上每一段欠款金额，找到最小值并返回。



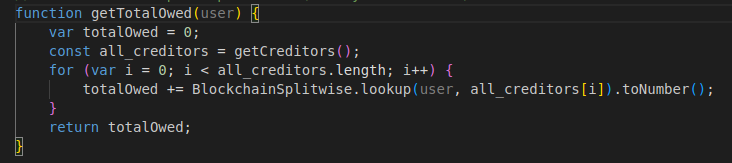
1. getUsers() 函数返回系统中所有用户（债务人或债权人）的列表。

利用 getCallData 函数从所有调用中提取信息。回调函数从每个调用中提取债务人（call.from）和债权人（call.args[0]）。返回一个包含债务人和债权人地址的数组，是所有曾经发送或接收过 IOU 的用户列表。



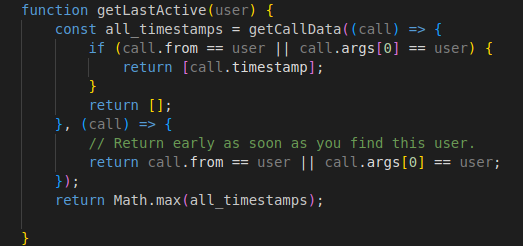
1. getTotalOwed(user) 函数获取指定用户所欠款的总金额。

假设查找函数是最新的，所有循环已被移除。遍历指定用户的所有债权人，使用 BlockchainSplitwise.lookup 获取用户欠每个债权人的金额。



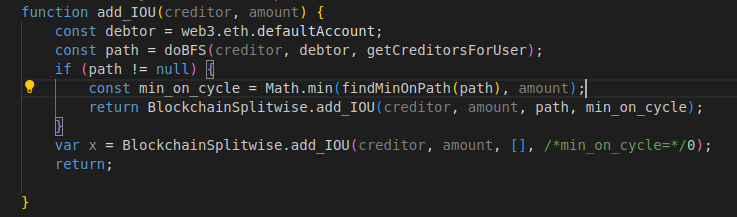
1. getLastActive(user) 函数获取用户最后一次发送或接收 IOU 的时间戳（自1970年1月1日以来的秒数）。

利用 getCallData 函数从所有调用中提取信息。回调函数检查每个调用的发送方（call.from）和第一个参数（call.args[0]），如果匹配到用户，则返回该调用的时间戳。一旦找到用户的活动，提前返回找到的所有时间戳中的最大值（即最后一次活动的时间戳）；如果找不到用户的任何活动，则返回 null。



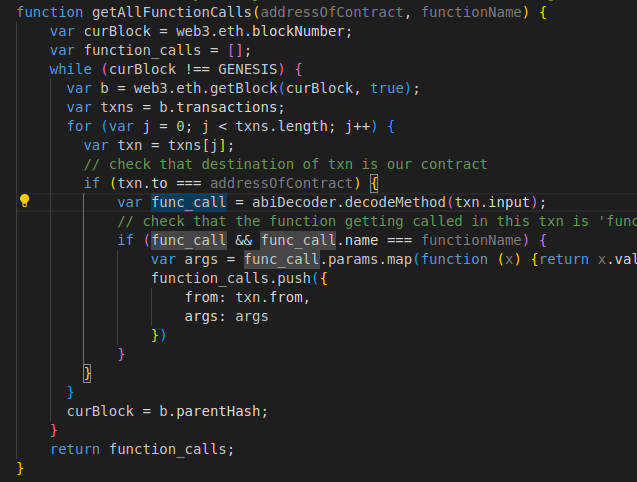
1. add\_IOU(creditor, amount) 函数用于向系统中添加 IOU（债务记录）。

假设债务人是发起交易的用户，使用 doBFS 函数查找从债权人到债务人的路径。如果存在路径，则计算路径上的最小金额（min\_on\_cycle），然后调用 BlockchainSplitwise.add\_IOU 函数，传递债权人、金额、路径和最小金额；如果不存在路径，直接调用 BlockchainSplitwise.add\_IOU 函数，传递债权人、金额、空路径和最小金额为 0。

g

1. getAllFunctionCalls(addressOfContract, functionName) 函数用于获取特定智能合约中特定函数的所有调用。

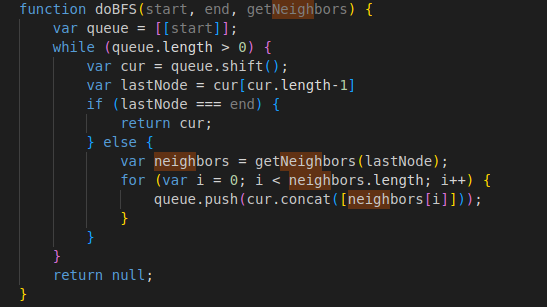
通过遍历区块链中的区块，从当前区块到创世区块（GENESIS），逐一检查每个区块的交易。对于每个交易，检查其目标地址是否是指定的智能合约地址。如果是，则使用 abiDecoder.decodeMethod 解码交易输入数据，以获取调用的函数名；如果函数名与指定的函数名相符，则提取调用的参数，并将调用信息（发送者地址和参数）添加到 function\_calls 数组中。最后，返回所有匹配的调用信息数组。



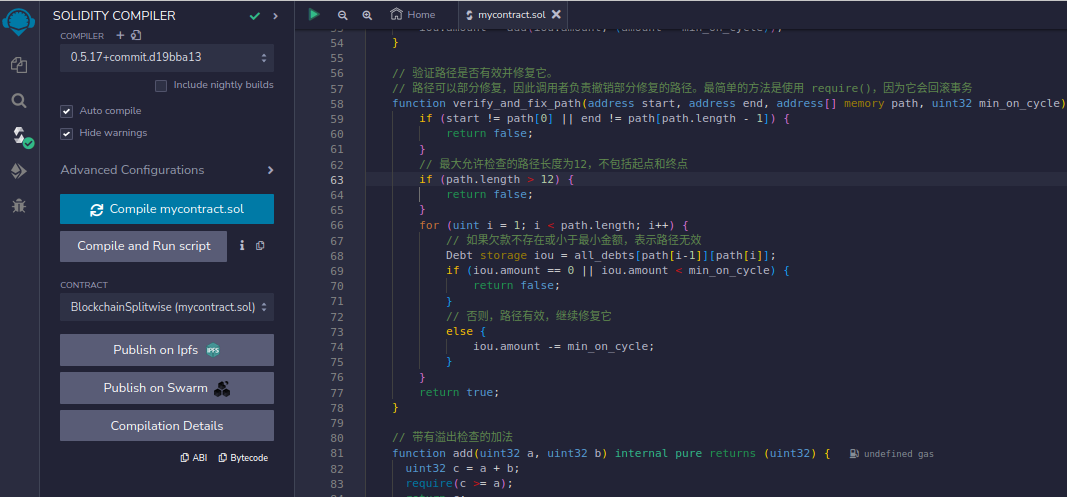
1. doBFS(start, end, getNeighbors) 函数执行广度优先搜索，寻找从起始节点到目标节点的路径。

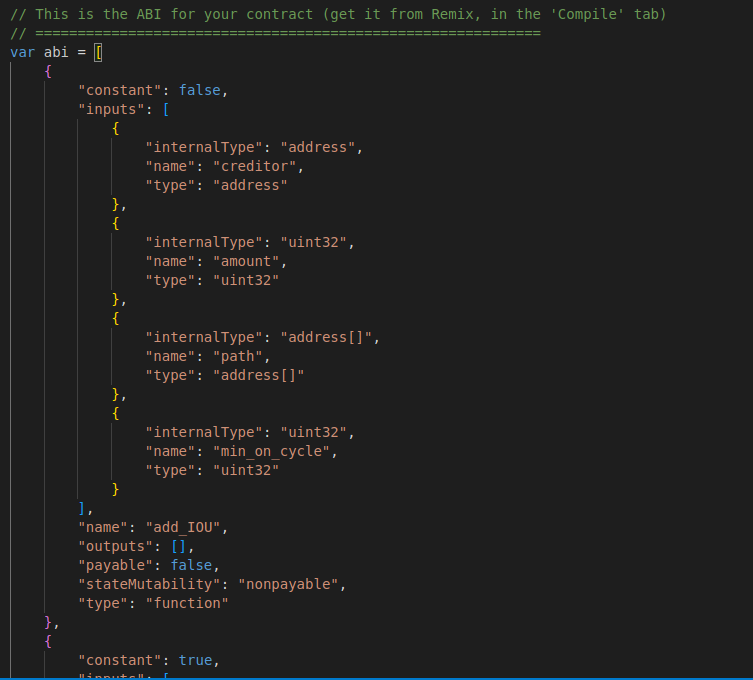
使用队列来实现广度优先搜索。从起始节点开始，将其添加到队列中。

在队列不为空的情况下，循环执行以下步骤：弹出队列的第一个元素，表示当前节点。如果当前节点是目标节点，则返回找到的路径；否则，获取当前节点的邻居节点列表（通过调用 getNeighbors 函数）。将邻居节点逐一添加到当前路径的末尾，并将新路径添加到队列中。如果循环结束时仍未找到路径，则返回 null，表示没有从起始节点到目标节点的路径。

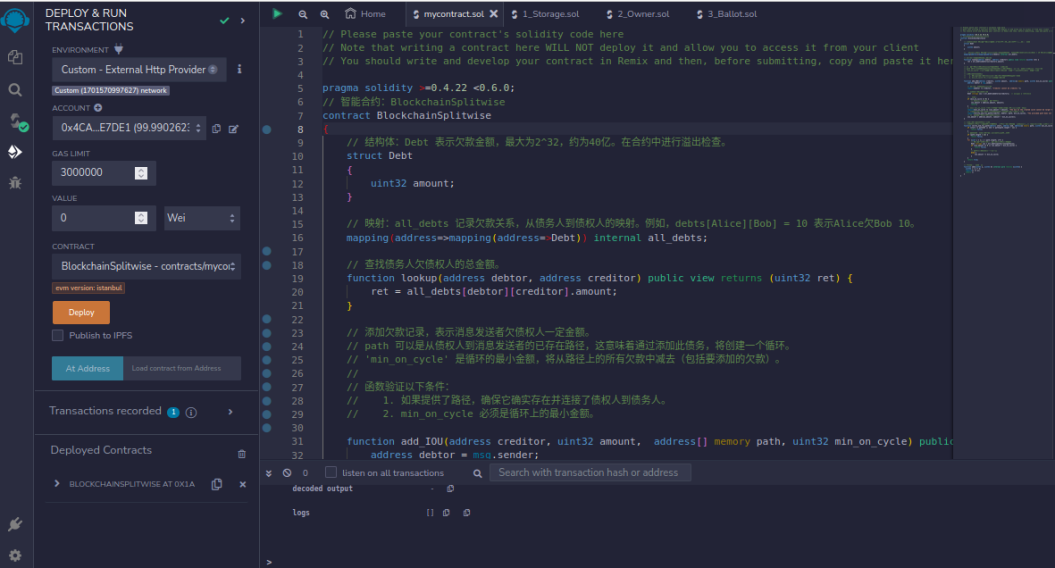


1. 编译运行solidity代码
2. 编译Solidity代码，并将ABI复制到script.js对应位置





1. 对合约进行部署，并复制合约地址至script.js的对应位置





1. 问题
2. 哪些数据应该存储在链上?哪些计算将由合同完成，哪些计算将在客户端完成?
3. 存储在链上的数据：
4. 欠款记录 (all\_debts 映射)：用于追踪用户之间的欠款关系。应存储在链上以确保合同状态的持久性。
5. 路径验证的最小金额 (min\_on\_cycle)：为了避免循环路径上的最小金额，这个值在链上存储是有意义的。
6. 合同完成的计算：
7. 添加欠款记录 (add\_IOU 函数)：根据路径的验证和最小金额的逻辑，合同在添加欠款记录时执行了一些计算。
8. 路径验证和修复 (verify\_and\_fix\_path 函数)：验证和修复路径的逻辑在合同内部进行，以确保添加的欠款记录不会导致循环。
9. 客户端完成的计算：
10. 获取用户列表 (getUsers 函数)：从调用数据中提取用户列表是在客户端完成的，因为这是用于用户界面展示的信息。
11. 获取总欠款金额 (getTotalOwed 函数) 和获取最后活动时间 (getLastActive 函数)：涉及在客户端查询所有合同的调用数据，然后根据需要计算总欠款和最后活动时间。
12. 添加欠款记录 (add\_IOU 函数中的路径查找)：在没有循环路径的情况下，直接添加欠款记录是在客户端完成的，而有循环路径时，执行路径查找和最小金额计算的逻辑是在客户端完成的。
13. 在仅仅两个人之间形成循环的情况下该怎么做？

在verify\_and\_fix\_path中已处理。