Ethereum调用机制总结
call/delegatecall/callcode/send/transfer

1、Message vs. Transaction

- **Transaction：**
  - 被EVM外部角色签名的数据，在EVM内部表现为触发Message或智能合约操作。Transaction会被写入区块链区块中。
  - A piece of data, signed by an External Actor. It represents either a Message or a new Autonomous Object. Transactions are recorded into each block of the blockchain.
- **Message：**
  - 在两个账户之间传递的Data(以字节流形式)或Value(以ether为单位)。Message或者由智能合约的确定性操作触发，或者是外部操作者加密签名的Transaction触发
  - Data (as a set of bytes) and Value (specified as Ether) that is passed between two Accounts, either through the deterministic operation of an Autonomous Object or the cryptographically secure signature of the Transaction.
- **参考资料**
  - ethereum黄皮书：https://ethereum.github.io/yellowpaper/paper.pdf
  - https://ethereum.stackexchange.com/questions/12065/what-is-the-difference-between-a-call-message-call-and-a-message
  - https://ethereum.stackexchange.com/questions/24031/how-ethereum-contracts-transfer-ether-without-a-blockchain-confirmation

2、EVM外部调用

- **EVM外部调用EVM为transaction，分为call、transaction调用**
  - **call**： 读操作，只在本地节点模拟transaction操作，调用智能合约或触发Message操作，执行往后不保存任何对区块链的改变。结果不会广播到整个区块链，不改变区块链数据。对应JSON RPC的eth_call或web3.eth.call
  - **transaction**：读写操作，会调用EVM的智能合约或触发Message，结果会广播整个区块链，改变区块链数据。对应JSON RPC的eth_sendTransaction或web3.eth.sendTransaction
  - **transaction vs. call**
    - https://ethereum.stackexchange.com/questions/765/what-is-the-difference-between-a-transaction-and-a-call

3、EVM内部调用

- **EVM内部调用为message call**
- **call/send/transfer**
  - **参考资料：**

        - https://consensys.github.io/smart-contract-best-practices/recommendations/#be-aware-of-the-tradeoffs-between-send-transfer-and-callvalue
        - https://blog.ethereum.org/2016/06/10/smart-contract-security/
    - **someAddress.transfer(amount)**
        - 建议使用，transfer方法相对send/call，更安全、健壮
        - x.transfer(y)等价于require(x.send(y))，如果失败，会自动回滚（revert）
        - transfer操作的接收方 EVM 只提供了2300 gas limit
    - **someAddress.send(amount)**
        - deprecated，不再建议使用
        - 原因：
            - send 操作的接收方 EVM 只提供了2300 gas limit，由于接收合约的fallback可能会进行其他处理，超过gas limit不够，导致send操作失败
            - send过程中如果出现异常，send只返回true/false，不抛出对应异常
    - **someAddress.call.value(ethAmount).gas(gasAmount)()**
        - transfer/send的底层调用方法，相对灵活，可以指定gas值
    - **call/send/transfer的推荐使用模式：**
        - transfer/send用于push类业务场景合约中
        - call用于pull类场景的合约中
- **call/callcode/delegatecall**
    - **call调用的陷阱**
        - 在call时候只能用长变量类型，不能用昵称 。例如用uint256，不能用uint；用int256，不能用int
            - 使用someAddress.call(bytes4(keccak256("setAction(**uint256**)")),value)，而不是call(bytes4(keccak256("setAction(**uint**)")),value)
        - 通过外部接口（JSON RPC）传入地址address类型，调用call/callcode/delegatecall时候，必须使用双引号包上 （不能用单引号，JSON RPC只能用双引号""）。
    - **delegatecall/callcode调用陷阱**
        - 调用者和被调用者的变量顺序要保持一致（结构一致），否则会设置成对应顺序位变量。solidity在assembly时候确定了通过delegatecall设置变量在stack的位置，顺序不对，会设置错误变量的值
        - 调用library，如果library函数需要使用存储，则推荐使用proxy library实现模式 https://blog.zeppelin.solutions/proxy-libraries-in-solidity-79fbe4b970fd
        - callcode 没正确传递msg.sender是DAO事件的罪魁祸首，已经被delegatecall取代，尽量别再使用
    - **contract A 调用contract B 方法需要返回值的解决办法**
        - call/delegatecall/callcode只能返回true、false，不能返回调用函数的值
        - 方案一： 使用contract实例
            - 方法1：B b=B(addressB)； result=b.myMethod();
            - 方法2：B b=new B(); result=b.myMethod();

- 方案二：使用event
  - 使用delegatecall/callcode时候，由于是改变调用者的存储空间，因此可以在调用者和被调用者定义相同的event，在被调用者中函数中触发event，在调用者中可以触发对应event，在truffle等客户端监听event
  - 使用call时候，web3 0.x版本采用web3.eth.filter过滤监听，web3 1.x版本采用web3.eth.subscribe

- **使用delegatecall 来实现upgrade smart contract的设计模式**
  - https://vomtom.at/upgrade-smart-contracts-on-chain/
  - https://blog.colony.io/writing-upgradeable-contracts-in-solidity-6743f0eecc88
  - https://medium.com/level-k/flexible-upgradability-for-smart-contracts-9778d80d1638
  - https://medium.com/rocket-pool/upgradable-solidity-contract-design-54789205276d
  - https://blog.zeppelin.solutions/proxy-libraries-in-solidity-79fbe4b970fd