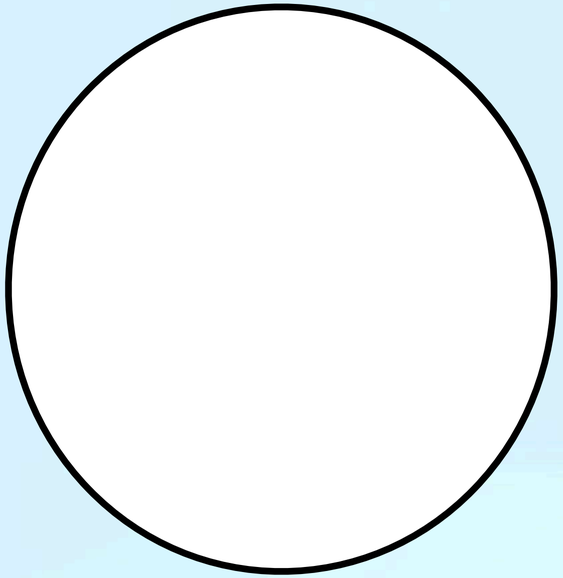
最小代理合约的原理和应用

33357 2023年10月27日

为什么讲最小代理合约

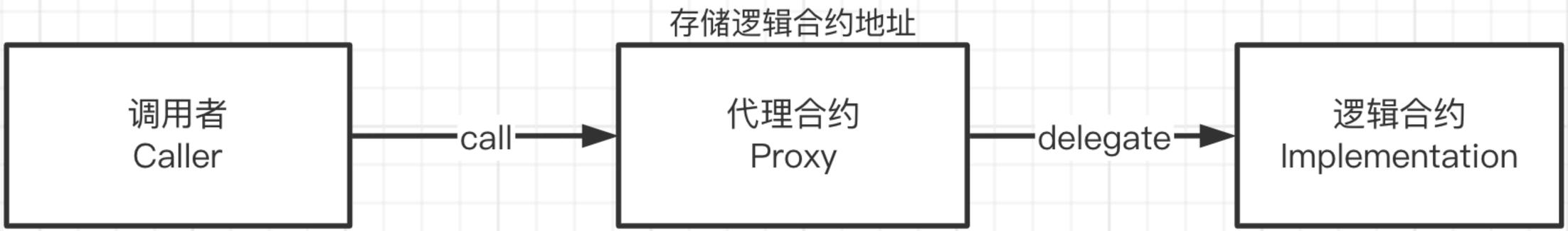
- 节约部署代理合约的 GAS。
- EIP 1167，0xAA 提出了 EIP7511。
- 作为学习 opcode 的案例。



- 我讲的东西就是一个锅
- 光有锅，没有水和面做不出饼
- 能做什么饼，要看个人手艺

什么是代理合约

- 代理合约（Proxy）
- 逻辑合约（Implementation）



代理合约将数据和逻辑分开

最小代理合约

- 部署 GAS 成本低
- 执行 GAS 成本低

- 不能修改逻辑合约地址
- 没有管理员

解析EIP7511

正文

EIP7511 优化了 EIP1167，实现了更加节省 GAS 的代理合约。提案地址：https://eips.ethereum.org/EIPS/eip-7511

想要从字节码的层面理解智能合约，就必须先了解 `堆栈` 、 `内存` 、 `存储` 、 `OPCODE` 等概念，这里推荐 https://www.evm.codes/ 和 https://www.wtf.academy/en/evm-opcodes-101 了解。

最小代理合约字节码

字节码

```
365f5f375f5f365f73bebebebebebebebebebebebebebebebebebebebebebebebebebebebe5af43d5f5f3e5f3d91602a57fd5bf3
```

复制CALLDATA到内存

分析

- 复制 CALLDATA 到内存

执行位置	字节码	操作名	堆栈	内存	说明
00	36	CALLDATASIZE	cds		将 calldatasize 计为 cds，并压入堆栈
01	5f	PUSH0	0 cds		将 0 压入堆栈
02	5f	PUSH0	0 0 cds		将 0 压入堆栈
03	37	CALLDATACOPY		calldata	将 0 - cds 的 calldata 复制到从 0 开始的内存空间

执行 DELEFATECALL

- DELEGATECALL

执行位置	字节码	操作名	堆栈	内存	说明
04	5f	PUSH0	0	calldata	将 0 压入堆栈
05	5f	PUSH0	0 0	calldata	将 0 压入堆栈
06	36	CALLDATASIZE	cds 0 0	calldata	将 calldatasize 计为 cds，并压入堆栈
07	5f	PUSH0	0 cds 0 0	calldata	将 0 压入堆栈
08	73bebe.	PUSH20 0xbebe.	0xbebe. 0 cds 0 0	calldata	将 20 个字节的数据压入堆栈
1d	5a	GAS	gas 0xbebe. 0 cds 0 0	calldata	将 gas 压入堆栈
1e	f4	DELEGATECALL	suc	calldata	将 0 - cds 的内存数据作为参数执行 0xbebe. 地址的代码，将执行是否成功计为 suc，并压入堆栈

复制 RETURNDATA 到内存

- 复制 RETURNDATA 到内存

执行位置	字节码	操作名	堆栈	内存	说明
1f	3d	RETURNDATASIZE	rds suc	calldata	将 returndatasize 计为 rds，并压入堆栈
20	5f	PUSH0	0 rds suc	calldata	将 0 压入堆栈
21	5f	PUSH0	0 0 rds suc	calldata	将 0 压入堆栈
22	3e	RETURNDATACOPY	suc	returndata	将 0 - rds 的 returndata 复制到从 0 开始的内存空间

返回数据或者回滚交易

- 返回数据或拒绝交易

执行位置	字节码	操作名	堆栈	内存	说明
23	5f	PUSH0	0 suc	returndata	将 0 压入堆栈
24	3d	RETURNDATASIZE	rds 0 suc	returndata	将 returndatasize 计为 rds，并压入堆栈
25	91	SWAP2	suc 0 rds	returndata	将堆栈第一个元素和第三个元素互换
26	602a	PUSH1 0x2a	0x2a suc 0 rds	returndata	将 1 个字节的数据压入堆栈
27	57	JUMPI	0 rds	returndata	如果 suc 不为 0 则执行位置跳转到 2a
29	fd	REVERT			返回 0 - rds 的内存数据并回滚状态
2a	5b	JUMPDEST	0 rds	returndata	跳转标记 |
2b	f3	RETURN			返回 0 - rds 的内存数据

部署最小代理合约字节码

部署最小代理合约字节码

部署合约字节码需要了解 `initcode` 和 `runtimecode` 的区别。

部署合约时会先执行 `initcode`， `initcode` 执行后返回的结果就是 `runtimecode`，真正被部署在合约上的字节码。

字节码

```
602c8060095f395ff3365f5f375f5f365f73bebebebebebebebebebebebebebebebebebebebebebebebebebebebe5af43d5f5f3e5f3d91602a57fd5bf3
```

部署代码

分析

- 部署代码

执行位置	字节码	操作名	堆栈	内存	说明
00	602c	PUSH1 2c	2c		将 1 个字节的数据压入堆栈
02	80	DUP1	2c 2c		复制堆栈第一个元素并压入堆栈
03	6009	PUSH1 09	09 2c 2c		将 1 个字节的数据压入堆栈
05	5f	PUSH0	0 09 2c 2c		将 0 压入堆栈
06	39	CODECOPY		365f5f375f5f365f73be...be5af43d5f5f3e5f3d91602a57fd5bf3	将 09 - 2c 的 code 复制到从 0 开始的内存空间
07	5f	PUSH0	0 2c	365f5f375f5f365f73be...be5af43d5f5f3e5f3d91602a57fd5bf3	将 0 压入堆栈
08	f3	RETURN		365f5f375f5f365f73be...be5af43d5f5f3e5f3d91602a57fd5bf3	将 0 - 2c 的内存数据返回

GAS 成本

- 部署：8828 + 32000

- 执行：192+

最小代理合约的应用

- 挖 TOKEN
- 抢 NFT
- 子合约

挖 TOKEN

```
260         */
261     function claimRank(uint256 term) external {
262         uint256 termSec = term * SECONDS_IN_DAY;
263         require(termSec > MIN_TERM, "CRank: Term less than min");
264         require(termSec < _calculateMaxTerm() + 1, "CRank: Term more than current max term");
265         require(userMints[_msgSender()].rank == 0, "CRank: Mint already in progress");
266 
267         // create and store new MintInfo
268         MintInfo memory mintInfo = MintInfo({
269             user: _msgSender(),
270             term: term,
271             maturityTs: block.timestamp + termSec,
272             rank: globalRank,
273             amplifier: _calculateRewardAmplifier(),
274             eaaRate: _calculateEAARate()
275         });
276         userMints[_msgSender()] = mintInfo;
277         activeMinters++;
278         emit RankClaimed(_msgSender(), term, globalRank++);
279     }
280 
281     /**
282      * @dev ends minting upon maturity (and within permitted Withdrawal Time Window), gets minted XEN
283      */
284     function claimMintReward() external {
285         MintInfo memory mintInfo = userMints[_msgSender()];
286         require(mintInfo.rank > 0, "CRank: No mint exists");
287         require(block.timestamp > mintInfo.maturityTs, "CRank: Mint maturity not reached");
288 
289         // calculate reward and mint tokens
290         uint256 rewardAmount = _calculateMintReward(
291             mintInfo.rank,
292             mintInfo.term,
293             mintInfo.maturityTs,
294             mintInfo.amplifier,
295             mintInfo.eaaRate
296         ) * 1 ether;
297         _mint(_msgSender(), rewardAmount);
298 
299         _cleanUpUserMint();
300         emit MintClaimed(_msgSender(), rewardAmount);
301     }
302 
```

抢 NFT

```
 */
function purchase(uint256 quantity)
    external
    payable
    nonReentrant
    canMintTokens(quantity)
    onlyPublicSaleActive
    returns (uint256)
{
    uint256 salePrice = salesConfig.publicSalePrice;

    if (msg.value != (salePrice + ZORA_MINT_FEE) * quantity) {
        revert Purchase_WrongPrice((salePrice + ZORA_MINT_FEE) * quantity);
    }

    // If max purchase per address == 0 there is no limit.
    // Any other number, the per address mint limit is that.
    if (
        salesConfig.maxSalePurchasePerAddress != 0 &&
        _numberMinted(_msgSender()) +
            quantity -
            presaleMintsByAddress[_msgSender()] >
        salesConfig.maxSalePurchasePerAddress
    ) {
        revert Purchase_TooManyForAddress();
    }

    _mintNFTs(_msgSender(), quantity);
    uint256 firstMintedTokenId = _lastMintedTokenId() - quantity;

    _payoutZoraFee(quantity);

    emit IERC721Drop.Sale({
        to: _msgSender(),
        quantity: quantity,
        pricePerToken: salePrice,
        firstPurchasedTokenId: firstMintedTokenId
    });
    return firstMintedTokenId;
}
```

子合约

- 钱包
- 批量发行TOKEN
- 批量发行NFT

能否比最小代理合约更小

- 不要CALLDATA
- 不要RETURNDATA
- 不要指定逻辑合约地址
- 不要回滚交易

- 通用性变差

祝大家都能使用这口锅