lab4

实验目的

本实验的目的是通过编写和部署一个基于以太坊区块链的委托投票智能合约，来了解智能合约的基本概念、Solidity编程语言的使用以及如何在区块链上进行去中心化投票。

实验工具

- Solidity编程语言
- 在线平台

实验原理

委托投票（Delegated Voting）是一种投票机制，允许投票人将他们的投票权委托给他人，进而由被委托人代为投票。此机制可以用于多种去中心化治理场景中，提高投票的参与度和效率。

实验步骤

一、部署和调用合约，实现以太坊上智能合约部署的基本流程

```
user@5b071359a189:~/workspace$ truffle compile

Compiling your contracts...
===========================
✓ Fetching solc version list from solc-bin. Attempt #1
✓ Downloading compiler. Attempt #1.
> Compiling ./contracts/hello.sol
> Artifacts written to /home/user/workspace/build/contracts
> Compiled successfully using:
   - solc: 0.8.20+commit.a1b79de6.Emscripten.clang
```

```
Private Keys:
(0) 0c4318ad58b3152cb613c143b59ded6af2c6c19d3fb8582349923a9bc3264d88
(1) bc196dff8792a620a9993f9a956b401074abee5b9b650a8458748aca7dc4356a
(2) 3954a1dc9be16c73b569ffe8d6e409d4969505e84512477b40ec897456c91e8b
(3) 693b3f9418666c00f47e09192558eecf5b12b386928978b3920bc8dba7ec7bca
(4) 35005706f8675bfbfc78d79870883b294d5d0e89b9dc078150d0f1a7f1690a03
(5) fde3f0b618abb53a6ab6d160fd4c5a46706d33d3c097e27b8f30e1ed2f87d3cf
(6) 29abe9f556ea071df146d8244e19e5e5d163270370d27a51f6ce54b099ba2cde
(7) a407f348afea9d8bddf76957a4eeb566524e3781a64ca00fdc0c1c71d3ab9e23
(8) 4307a8425ef7517b476c73816f65d5a02cec5732b28a57205925183975037c71
(9) f4c275076ad28842f49dab675e81da1a27ed0fbac31c740d8c8e94398959d5b3

Mnemonic: gift picture egg guilt census spawn slot title artwork before rack rib

⚠ Important ⚠ : This mnemonic was created for you by Truffle. It is not secure.
Ensure you do not use it on production blockchains, or else you risk losing funds.
```

二、

1. 编写智能合约

首先，编写一个名为`Ballot`的智能合约，代码如下：

```
// SPDX-License-Identifier: UNLICENSED
pragma solidity >=0.4.22;

contract BallotSystem {

    struct Voter {
        bool hasVoted;
        uint weight;
        uint8 selectedProposal;
        address delegateAddress;
    }

    struct Proposal {
        string title;
        string description;
        uint voteCount;
        bool isActive;
    }

    address public administrator;

    Proposal[] public proposalsList;

    uint public currentProposalCount;

    uint public constant MAX_UINT8 = (2**8) - 1;

    mapping(address => Voter) public voterRegistry;

    constructor(uint initialWeight) {
        administrator = msg.sender;
        voterRegistry[administrator].weight = initialWeight;

        for (uint i = 0; i <= MAX_UINT8; i++) {
            proposalsList.push(Proposal("", "", 0, false));
        }
    }

    function addProposal(string memory proposalTitle, string memory
proposalDescription) public {
        require(currentProposalCount < MAX_UINT8, "Maximum proposal limit
reached");

        currentProposalCount++;
        proposalsList[currentProposalCount].title = proposalTitle;
        proposalsList[currentProposalCount].description = proposalDescription;
        proposalsList[currentProposalCount].isActive = true;
    }

    modifier onlyAdministrator {
```

```
        require(msg.sender == administrator, "Action restricted to
administrator");
        _;
    }

    modifier hasNotVoted(address voter) {
        require(!voterRegistry[voter].hasVoted, "Voter has already voted");
        _;
    }

    function authorizeVoter(address voter, uint weight) public onlyAdministrator
hasNotVoted(voter) {
        require(weight >= 0 && weight <= 100, "Invalid voting weight");
        require(voterRegistry[voter].weight == 0, "Voter already authorized");
        voterRegistry[voter].weight = weight;
    }

    function delegateVote(address to) public hasNotVoted(msg.sender) {
        require(to != msg.sender, "Cannot delegate to self");

        while (voterRegistry[to].delegateAddress != address(0) &&
voterRegistry[to].delegateAddress != msg.sender) {
            to = voterRegistry[to].delegateAddress;
        }

        require(voterRegistry[to].delegateAddress != msg.sender, "Circular
delegation not allowed");

        voterRegistry[msg.sender].delegateAddress = to;

        if (voterRegistry[to].hasVoted) {
            proposalsList[voterRegistry[to].selectedProposal].voteCount +=
voterRegistry[msg.sender].weight;
            voterRegistry[msg.sender].selectedProposal =
voterRegistry[to].selectedProposal;
        } else {
            voterRegistry[to].weight += voterRegistry[msg.sender].weight;
        }

        voterRegistry[msg.sender].hasVoted = true;
    }

    function castVote(uint8 proposalID) public hasNotVoted(msg.sender) {
        require(proposalID >= 0 && proposalID <= MAX_UINT8, "Proposal ID out of
range");
        require(proposalsList[proposalID].isActive, "Proposal is not active");

        proposalsList[proposalID].voteCount += voterRegistry[msg.sender].weight;
        voterRegistry[msg.sender].selectedProposal = proposalID;
        voterRegistry[msg.sender].hasVoted = true;
    }

    function winningProposal() public view returns (uint winningProposal_) {
        uint winningVoteCount = 0;
        for (uint p = 0; p < proposalsList.length; p++) {
            if (proposalsList[p].voteCount > winningVoteCount) {
                winningVoteCount = proposalsList[p].voteCount;
                winningProposal_ = p;
```

```
            }
        }
    }
}
```

2. 测试合约功能

1. 赋予投票权

   调用`giveRightToVote`函数，将投票权赋予某个地址。确保只有主席（部署合约的地址）可以调用该函数。

2. 委托投票

   使用不同的账户调用`delegate`函数，将投票权委托给其他账户。确保被委托的账户最终将代替委托人投票。

3. 进行投票

   调用`vote`函数，选择一个提案进行投票。确保投票后的账户不能再次投票。

4. 计算并查看获胜提案

   调用`winningProposal`和`winnerName`函数，查看得票最多的提案及其名称。

Myballot.test.js

```
const Myballot = artifacts.require("Myballot");
const { expect } = require('chai');
const { BN, expectRevert } = require('@openzeppelin/test-helpers');

contract("Myballot", (accounts) => {
    let instance;
    const chairperson = accounts[0];
    const voter1 = accounts[1];
    const voter2 = accounts[2];

    beforeEach(async () => {
        instance = await Myballot.new(new BN(100), { from: chairperson });
    });

    it("should initialize the chairperson with the correct weight", async () =>
{
        const chairpersonVoter = await instance.voters(chairperson);
        expect(chairpersonVoter.weight).to.be.bignumber.equal(new BN(100));
    });

    it("should allow the chairperson to authorize a voter", async () => {
        await instance.authorizedVote(voter1, new BN(50), { from: chairperson
});
        const voter1Details = await instance.voters(voter1);
        expect(voter1Details.weight).to.be.bignumber.equal(new BN(50));
    });

    it("should not allow non-chairperson to authorize a voter", async () => {
        await expectRevert(
            instance.authorizedVote(voter1, new BN(50), { from: voter2 }),
            "Only the chairman can operate"
        );
    });
```

```
    it("should allow a voter to apply a proposal", async () => {
        await instance.applyProposal("Proposal 1", "Proposal Content 1", { from:
chairperson });
        const proposal = await instance.proposals(new BN(1));
        expect(proposal.name).to.equal("Proposal 1");
        expect(proposal.content).to.equal("Proposal Content 1");
        expect(proposal.enable).to.be.true;
    });

    it("should allow a voter to vote on a proposal", async () => {
        await instance.applyProposal("Proposal 1", "Proposal Content 1", { from:
chairperson });
        await instance.authorizedVote(voter1, new BN(50), { from: chairperson
});
        await instance.vote(new BN(1), { from: voter1 });
        const proposal = await instance.proposals(new BN(1));
        expect(proposal.voteCount).to.be.bignumber.equal(new BN(50));
    });

    it("should not allow double voting", async () => {
        await instance.applyProposal("Proposal 1", "Proposal Content 1", { from:
chairperson });
        await instance.authorizedVote(voter1, new BN(50), { from: chairperson
});
        await instance.vote(new BN(1), { from: voter1 });
        await expectRevert(
            instance.vote(new BN(1), { from: voter1 }),
            "Voting weight range exceeded"
        );
    });

    it("should allow delegation of voting rights", async () => {
        await instance.applyProposal("Proposal 1", "Proposal Content 1", { from:
chairperson });
        await instance.authorizedVote(voter1, new BN(50), { from: chairperson
});
        await instance.authorizedVote(voter2, new BN(30), { from: chairperson
});
        await instance.delegateVoting(voter2, { from: voter1 });
        await instance.vote(new BN(1), { from: voter2 });
        const proposal = await instance.proposals(new BN(1));
        expect(proposal.voteCount).to.be.bignumber.equal(new BN(80));
    });
});
```

初始化：

- `beforeEach` 函数在每个测试之前都会运行，它负责实例化 `Myballot` 合约，并将初始权重分配给 `chairperson`（主席）账户。这里用的是 OpenZeppelin 的 `new` 函数来创建一个新的合约实例，并传入一个初始权重值作为参数。

授权投票权：

- 这个测试用例验证了主席可以授权其他地址的投票权，并设置其权重。使用 `authorizedVote` 函数来进行授权，然后检查被授权地址的权重是否正确设置。

提出提案：

- 这个测试用例验证了主席可以提出提案，并设置提案的名称、内容，并启用提案。使用 `applyProposal` 函数来创建一个提案，然后检查提案的属性是否正确设置。

投票：

- 这个测试用例验证了已授权的投票者可以对提案进行投票。首先创建一个提案，然后授权一个投票者，接着该投票者对提案进行投票，并检查提案的投票数量是否增加。

防止双重投票：

- 这个测试用例验证了系统防止同一投票者重复投票。首先创建一个提案并授权一个投票者，然后该投票者对提案进行了一次投票，接着尝试再次投票，预期会触发异常并验证异常信息是否正确。

授权投票权代理：

- 这个测试用例验证了投票者可以将其投票权委托给另一个地址。首先创建一个提案，并授权两个投票者，然后一个投票者将其投票权委托给另一个地址，然后委托地址进行了投票，最后验证提案的投票数量是否正确。

自测结果

```
Contract: Myballot
  ✓ should initialize the chairperson with the correct weight
  ✓ should allow the chairperson to authorize a voter (39ms)
  ✓ should not allow non-chairperson to authorize a voter (264ms)
  ✓ should allow a voter to apply a proposal (61ms)
  ✓ should allow a voter to vote on a proposal (110ms)
  ✓ should not allow double voting (143ms)
  ✓ should allow delegation of voting rights (166ms)
```