

東大寄付講座Solidity講義2024_2実装

DAOの簡易実装


株式会社 Ecdysis CEO / Solidity House 運営

落合 渉悟

- 事前準備: 開発環境セットアップ
- 開発演習①: シンプルな提案と投票による状態遷移
 - 状態遷移提案（可決で数値インクリメントのみ）
 - 単投票（多数決）
 - 単集計
 - 状態遷移実行

開発環境セットアップ

- 推奨エディタ
 - [Cursor](#)
 - [Visual Studio Code](#) (VSCode)
- スマートコントラクト開発環境
 - [Foundry](#)
- スマートコントラクト開発フレームワーク
 - [Meta Contract](#)
 - GPTs: MC GPT / Schema



開発演習①

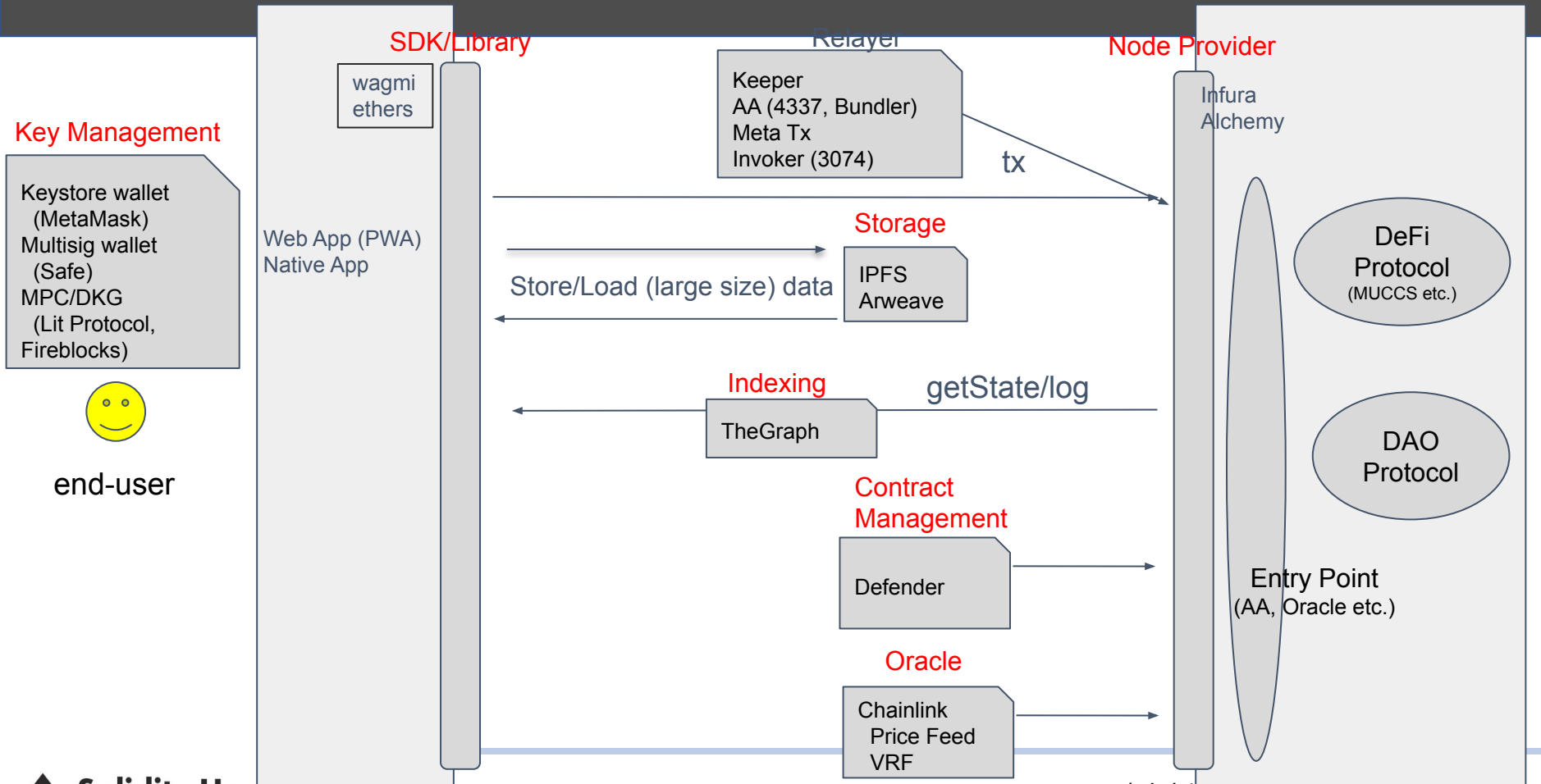
シンプルな提案と投票を通じて状態遷移するコントラクト

作っていくもの

シンプルな提案と投票を通じて状態遷移する（DAO）コントラクト
(e.g., MolochDAO, NounsDAO, etc...)

完成例：

<https://github.com/ecdysisxyz/simple-dao>



Key Management

- Keystore wallet (MetaMask)
- Multisig wallet (Safe)
- MPC/DKG (Lit Protocol, Fireblocks)



end-user

Web App (PWA)
Native App

SDK/Library

wagmi
ethers

Relayer

Keeper
AA (4337, Bundler)
Meta Tx
Invoker (3074)

Node Provider

Infura
Alchemy

Storage

IPFS
Arweave

Store/Load (large size) data

Indexing

TheGraph

getState/log

Contract Management

Defender

Oracle

Chainlink
Price Feed
VRF

DeFi
Protocol
(MUCCS etc.)

DAO
Protocol

Entry Point
(AA, Oracle etc.)

tx

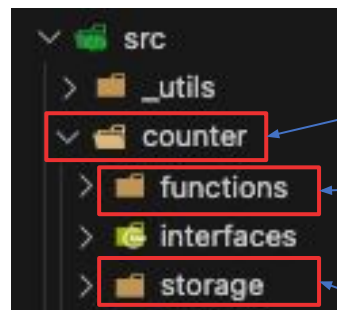
プロジェクトの立ち上げ

```
forge init <project name> -t metacontract/template
```

例)

```
forge init simple-dao -t metacontract/template
```

プロジェクトの構成を確認する



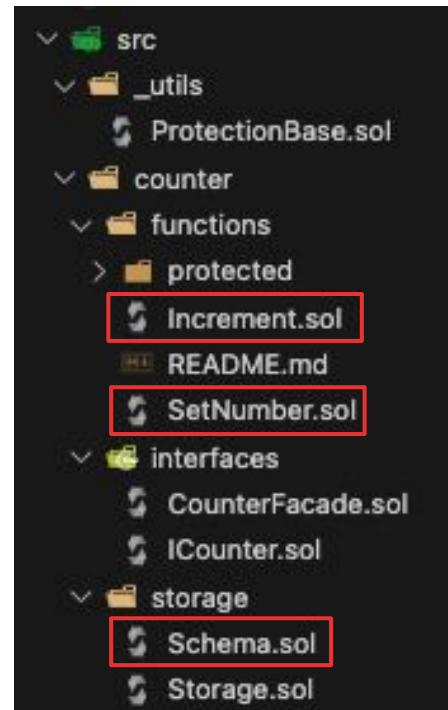
Bundle

Functions

Bundleの機能(振る舞い)

Storage

Bundleのデータ構造



Meta ContractのフレームワークではBundleという
単位でコントラクトを管理/開発します

Q. なぜそのような概念のもと開発するの？

A. ソフトウェアを複雑にしていくのにも技術が必要で、それをツールの側でやっておいてあげたいから

=> おさらい

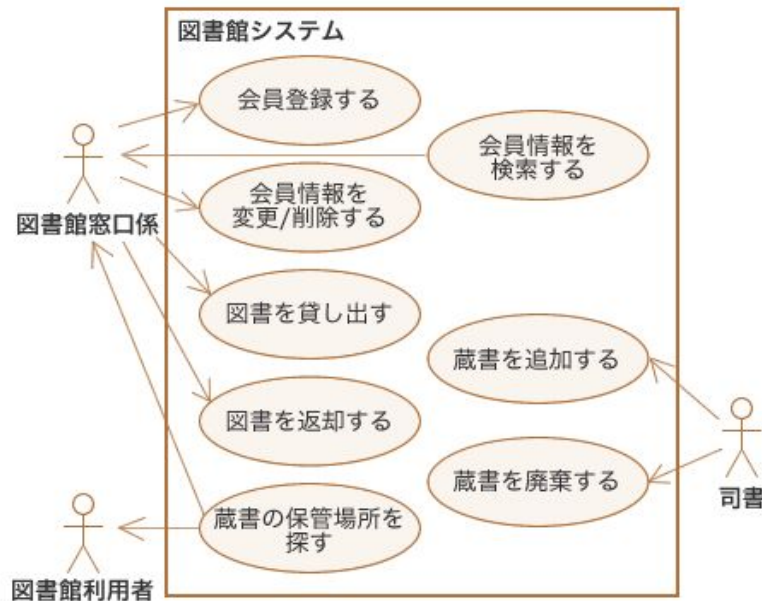
Upgradeability, Clonability, Contract Size Limit, Test Execution Time

コントラクト (Bundle) の構造を単純化して理解する

- どのような機能が必要か？
 - Meta ContractにおけるFunctions
- どのようなデータ構造（モデル）が必要か？
 - Meta ContractにおけるSchema.sol

設計の道へ - 必要な機能の洗い出し

ユースケース図が便利



https://www.mamezou.com/sites/default/files/2020-10/modewaza05_sp_002_005_f01.gif

設計の道へ - 製図ツールの紹介

- [Mermaid.js](#) / [Mermaid Live Editor](#)

専用の記法を使ってテキストで書くと図を描画してくれる

GitHubが標準で描画対応していたり、LLMの支援を受けやすいが清書できないので複雑な図にはあまり向いていない

- [draw.io](#)

Google Drive等と連携させて共同編集も出来るクラウドサービス

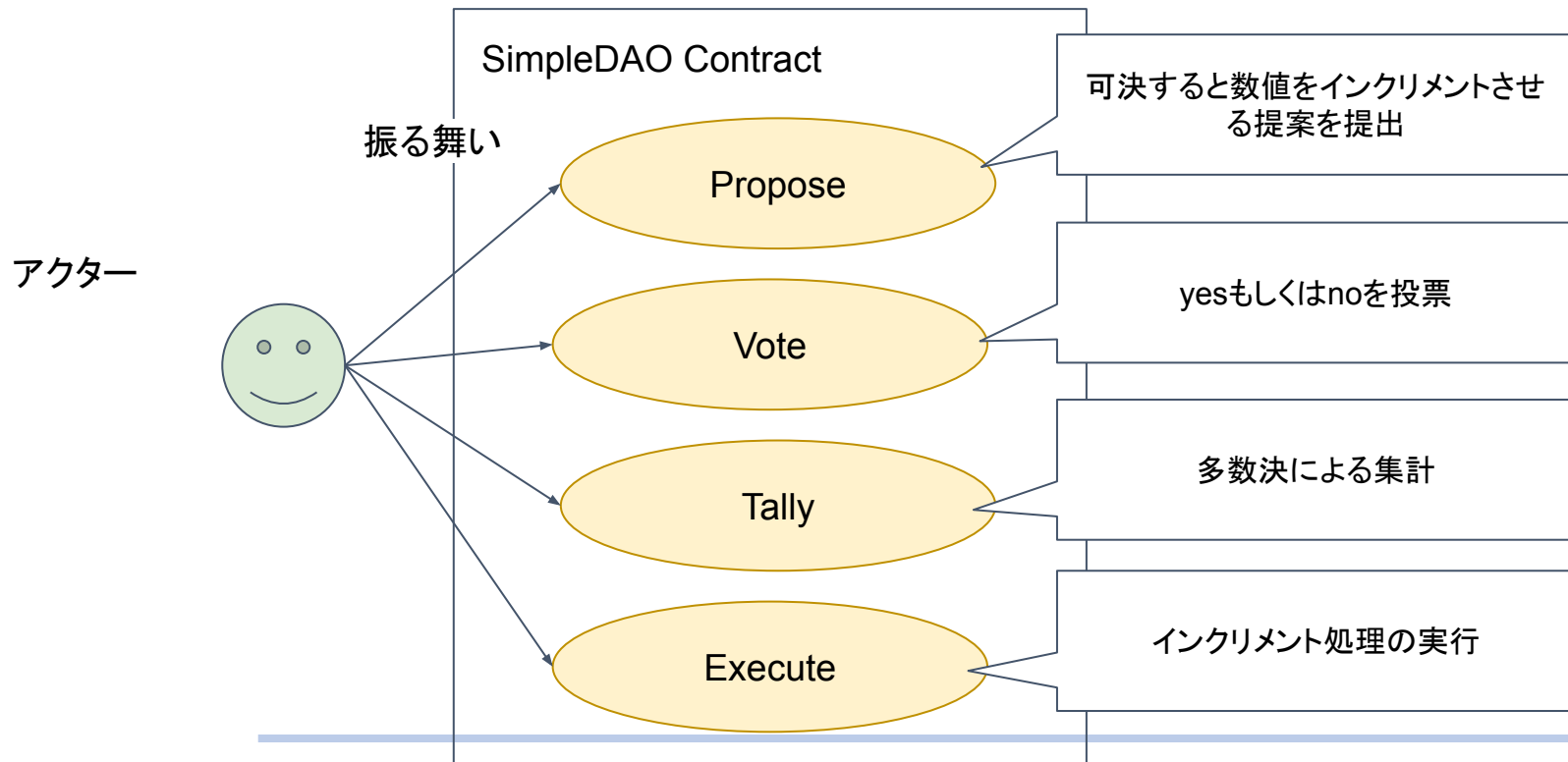
- [eraser.io](#)

AIと対話してシームレスにMermaid記法の各種ダイアグラムを作成する。

微修正がきき、複数種ダイアグラムを並行して作成できる。

Pro PlanならAPIが使用できるが、まだダイアグラム画像の生成のみ

設計の道へ - アクターに必要な機能の洗い出し



ER図（エンティティ・リレーション・ダイアグラム）

プロンプト

以下のようなデータが必要である。ER図を描け。

グローバルステート

- ・ （提案と投票によって増加する）数値
- ・ 提案
 - 提案者
 - 投票権保有者
 - 提案に対する投票状況があり
 - 提案に対する投票結果がある
 - 提案開催時刻
- ・ 提案 1 つあたりの開催期間
- ・ Tallyの実行頻度（秒）

| Vote | |
|-------------|------------|
| id | string pk |
| proposal_id | uint fk |
| voter_id | address fk |
| decision | bool |

| Proposal | |
|------------------|-----------|
| id | string pk |
| proposer | address |
| start_time | uint |
| duration_seconds | uint |
| votes: | Vote[] |

| GlobalState | |
|-------------------------|------------|
| id | string pk |
| value | uint |
| tally_frequency_seconds | uint |
| proposals: | Proposal[] |
| members: | address[] |
| proposals | |

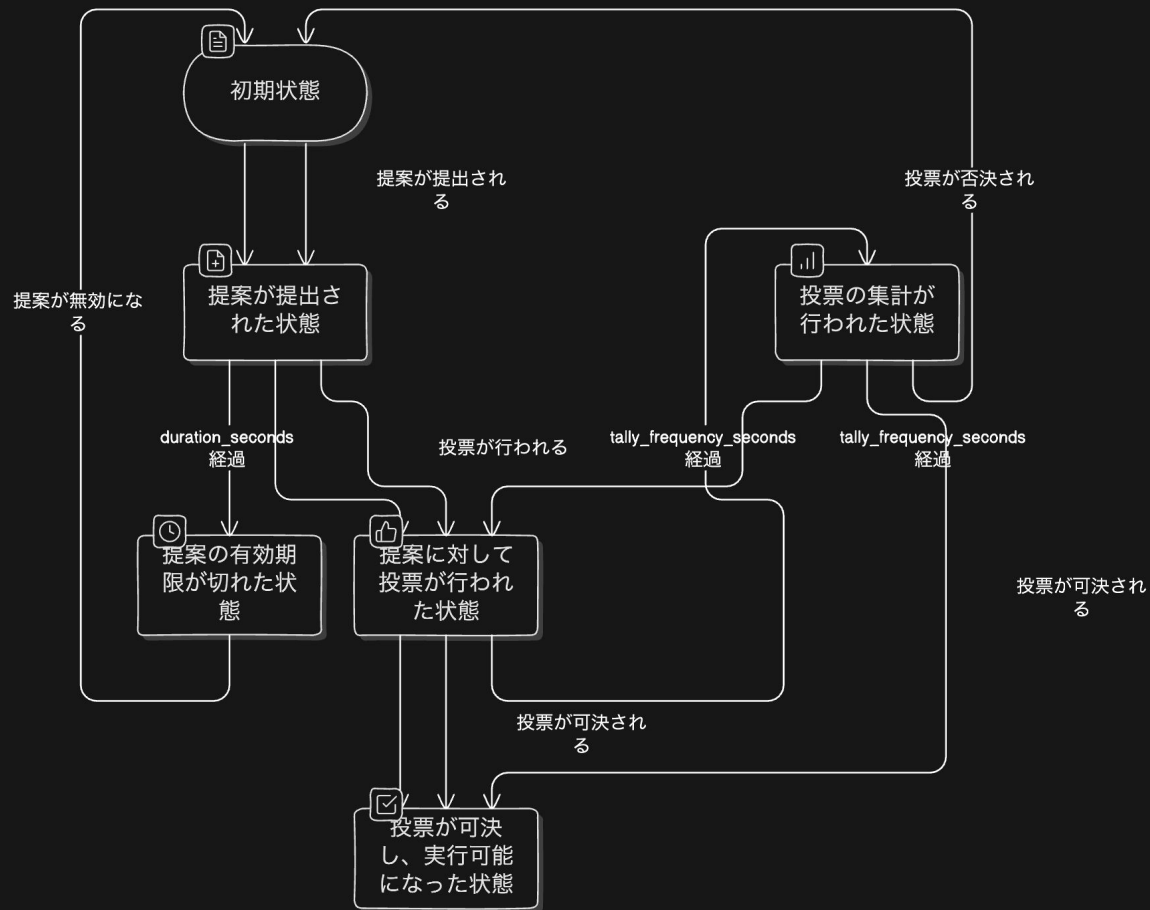
Global State and Proposals Voting System

状態遷移図（ステートトランジションダイアグラム）

プロンプト

- 以下の状態に遷移するものとする
 - 初期状態（何も提案されていない状態）
 - 提案が提出された状態
 - 提案に対して投票が行われた状態
 - 投票が可決し、実行可能になった状態
- ER図の提案の `duration_seconds` とグローバルステートの `tally_frequency_seconds` は、状態遷移にとって重要な情報です。`tally_frequency_seconds` は `duration_seconds` よりもはるかに短いため、提案が終了するまでに複数回のラウンドが行われることになります。

状態遷移図



アクティビティ図

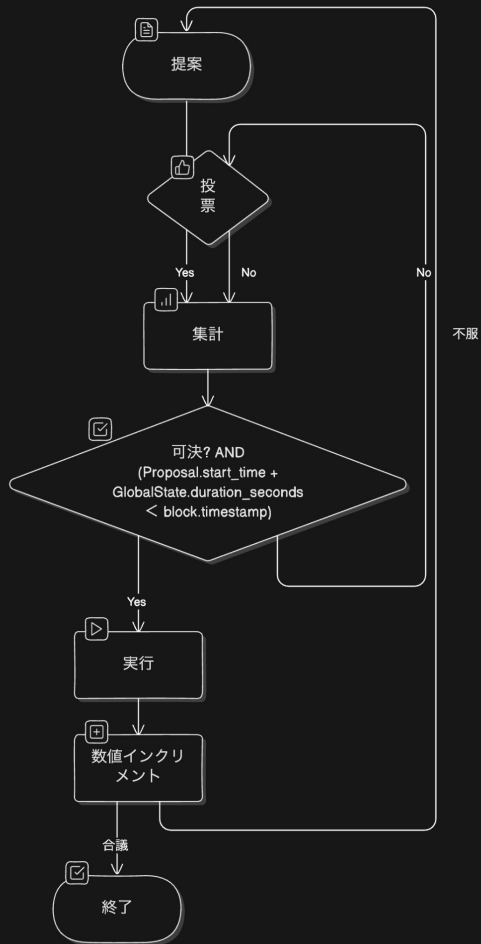
以下の関数を有するものとする

- 提案（可決すると数値インクリメントする）
- 投票（yes/noで）
- 集計（多数決）
- 実行（数値インクリメントの状態遷移を実行）

また、Globa

ER図と状態遷移図を参考にしてアクティビティ図を作成してください。

提案と投票のフロー



今日だけ公開: MC BOT

「このER図と状態遷移図とアクティビティ図のMermaid DSLを参考に、meta contractのSchema.sol を生成してください。」

<https://chatgpt.com/g/g-LpVtGHVKn-mc-gpt>

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.24;

library Schema {

    /// @custom:storage-location erc7201:simpleDAO.counter.GlobalState
    struct GlobalState {
        string id; // pk
        uint256 value;
        uint256 tally_frequency_seconds;
        Proposal[] proposals;
        address[] members;
        address owner;
        bool initialized;
    }

    struct Proposal {
        string id; // pk
        address proposer;
        uint256 start_time;
        uint256 duration_seconds;
        Vote[] votes;
    }

    struct Vote {
        string id; // pk
        uint256 proposal_id; // fk to Proposal.id
        address voter_id; // fk to GlobalState.members
        bool decision;
    }
}
```

Schemaファイルの作成

1. Schema.solファイルを作成する

src/simple-dao/storage/Schema.sol

2. 割と不要な情報もあるので削っていく
3. たまにER図の方をお直しする

関数を作ろう: プロンプト例

- `src/functions/MemberAdd.sol` をメタコントラクトとして作ってください。
- メタコントラクトとして`onlyOwner`を`MemberAdd.sol`に追加してください。
- `Propose.sol` をメタコントラクトとして作ってください。
- `Vote.sol`をメタコントラクトとして作ってください。
- `Tally.sol` をメタコントラクトとして作ってください。
- `Execute.sol`をメタコントラクトとして作ってください。あるIDを持つ`Propose`が可決されていると判断できる時に処理を実行します。（足し算）
- `Increment.sol`をメタコントラクトとして作ってください。
- `Increment.sol`に`onlyApproved` modifierを実装してください。
- `Propose.sol`が可決時に例えば`Increment.sol`を実行できるようにコントラクトアドレスを引数に追加してください。
- `Execute.sol`が`Propose`の引数のコントラクトアドレスを参照するようにしてください。

関数からデータへアクセスする (Storage.sol)

1. データモデルに従ってSchema.solファイルを実装する
2. FunctionsからSchemaを利用するためのStorage.solを実装する
3. 状態遷移図やユースケース図を元に各Functionを実装する
(必要に応じてそれらの図も更新する)
 - a. Propose.sol
 - b. Vote.sol
 - c. Tally.sol
 - d. Execute.sol

Storageファイルの作成

1. Storage.solファイルを作成する
src/simple-dao/storage/Storage.sol
2. ERC7201のスポットを決める
sh calc-storage-location.sh simpleDAO.counter.GlobalState

=> 0xdaf58ca28ede8a45c64a220d53624f87f5a5273327b1f02abfb8bc2b898b8800

動作確認とその自動化 - 単体テスト (a.k.a. 自動テスト)

ソフトウェアテストには粒度の概念がある

1. ユニット（単体）テスト

→ Function毎のState Fuzzing Test

```
import {MCTest} from "@mc/devkit/Flattened.sol";
```

2. インテグレーション（統合）テスト

→ デプロイスクリプトやMockを利用する

テストを通した設計（テスト駆動開発・TDD）

ソフトウェアテストは仕様としての役割もある

- どのような前提条件で、
- どのような操作を行なったとき、
- どのような結果になるか

1. 正常系テスト
2. 異常系テスト

テストとリリースプロセス（安全なデプロイのために）

作成したコントラクトを実際にネットワークにデプロイ（展開して利用可能な状態に）する流れの一例：

1. ローカルネットワーク環境での自動テスト
2. テストネット環境にデプロイし、[システム全体](#)を統合したテスト
3. 既にリリース済みの場合や、外部サービスとの連携がある場合などにはFork環境でのテスト
4. ローンチ前テスト
5. システムモニタリング

ネットワークにデプロイするためのスクリプトを用意する

- **script/DeployLib.sol**

デプロイするコントラクトの内容を記述するファイル

- **script/Deploy.s.sol**

デプロイするためのネットワーク情報やデプロイヤー情報を記述するファイル

Meta Contract DevKitの使い方

基本的な操作

- `mc.init()`

新しいBundleを用意する

- `mc.use()`

BundleにFunctionを登録する

- `mc.useFacade()`

BundleにFacadeを登録する

- `mc.deploy()`

Bundle情報を元にコントラクトをデプロイする

より高度なDAOやDeFiの研修は合宿にて！

佐賀SolidityHouse 4泊5日 30万円(税抜)
(先着20名まで10万円引)

応募フォームはこちら

t.ly/9JG75