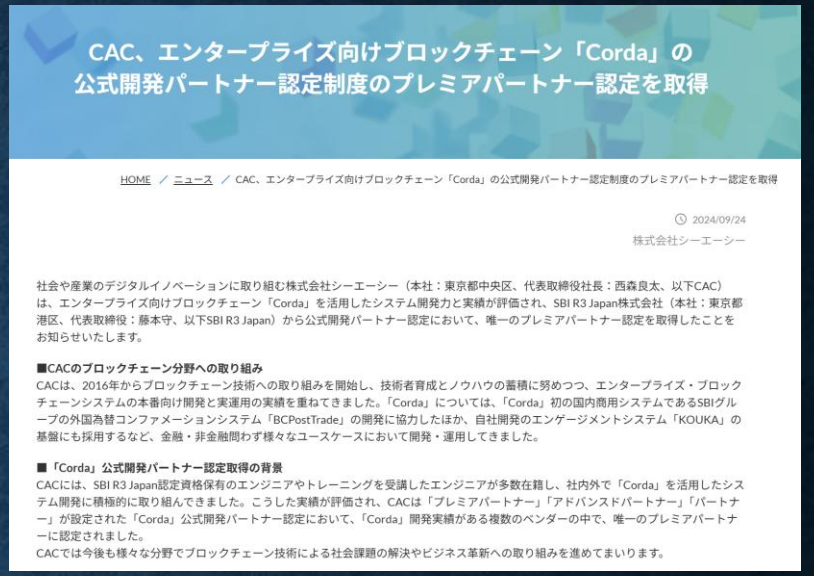
Sui Campus Workshop in 千葉工業大学 Session #3

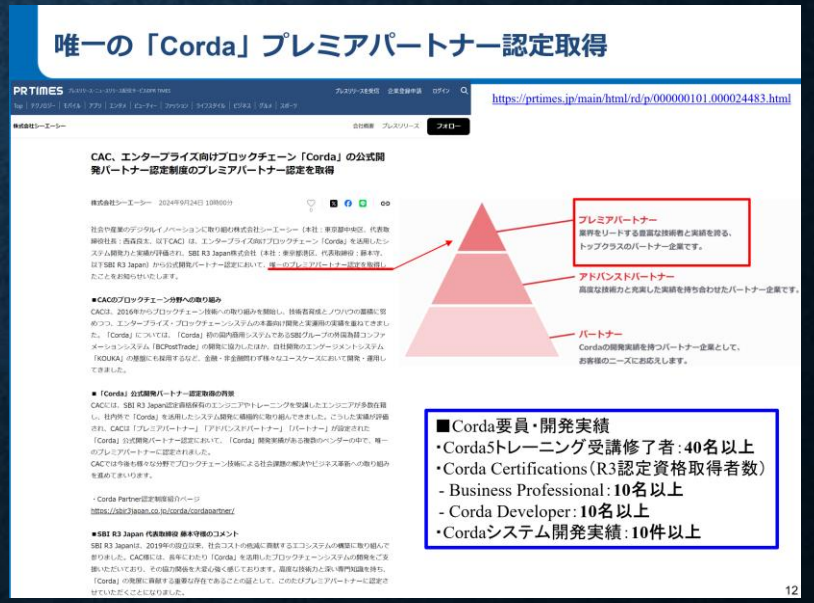
自己紹介

2021年にCryptoGamesに入社し、ブロックチェーンエンジニアとして活動。AstarGames（現MOCHIRON社）、SuperTeam Japanを経て、現在は株式会社シーエーシーのブロックチェーン推進グループに所属。

シーエーシーは2017年からのブロックチェーン事業を開始し、**Corda**を利用したサービスを手がけ、**Avalanche, Solana, Suiの3基盤の技術選定**を行なっています。



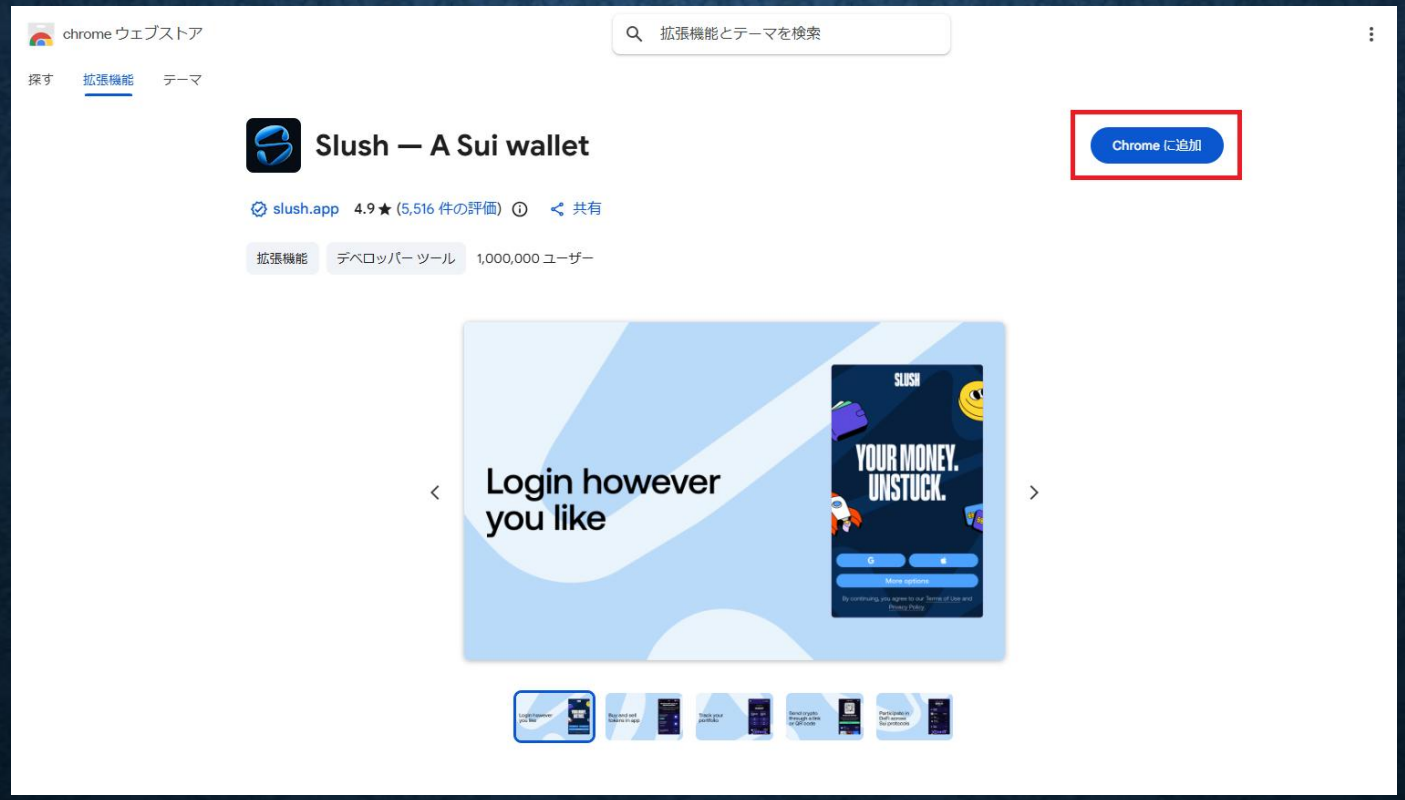
https://www.cac.co.jp/news/topics_240924/



事前準備

1　Suiウォレットの追加

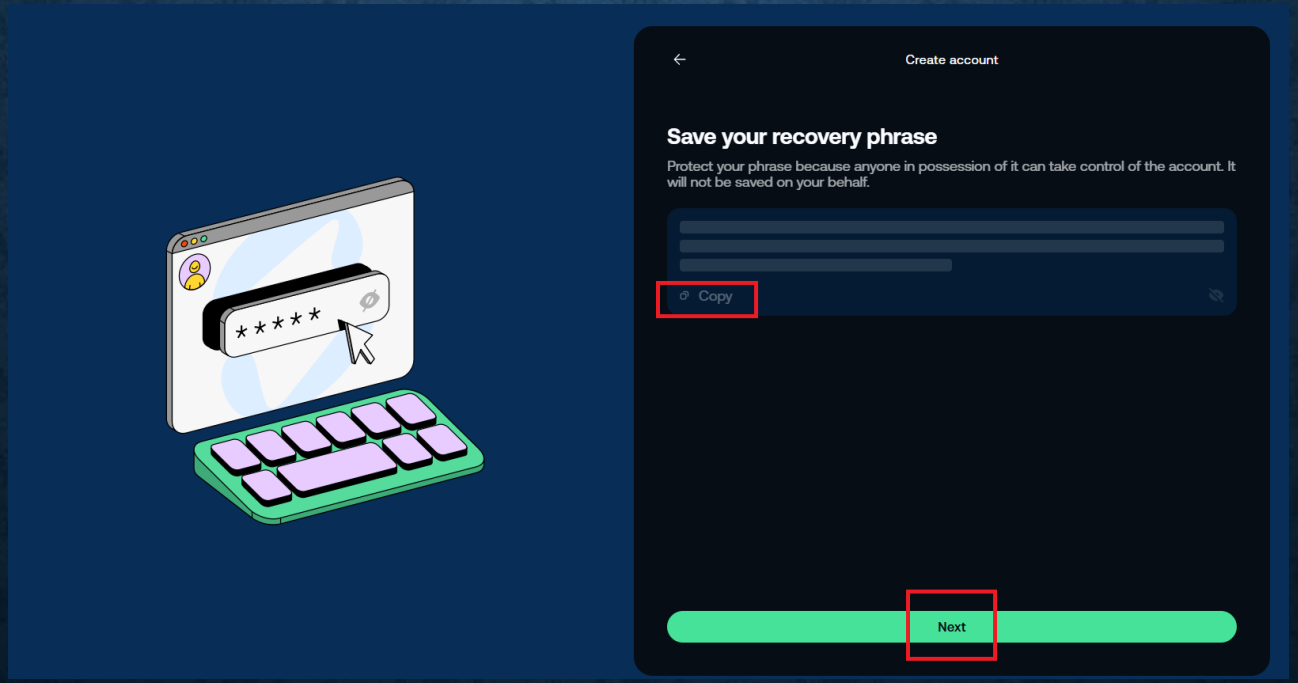
Chromeに追加します。



https://chromewebstore.google.com/detail/slush-%E2%80%94-a-sui-wallet/opcgpfmipidbgpenhmajoajpbobppdil?hl=ja&pli=1

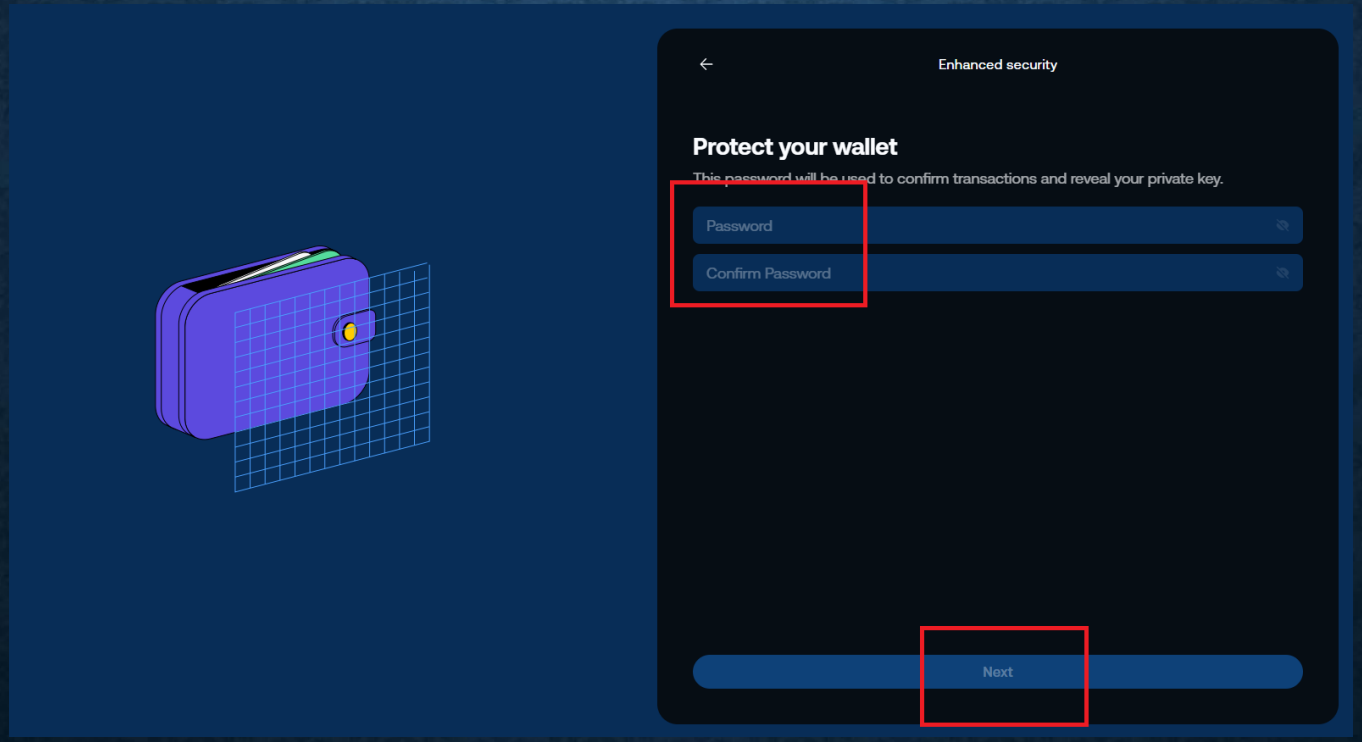
1　Suiウォレットの追加

私や他のメンター含め、絶対に他の人には見せず、厳重に保管してください。



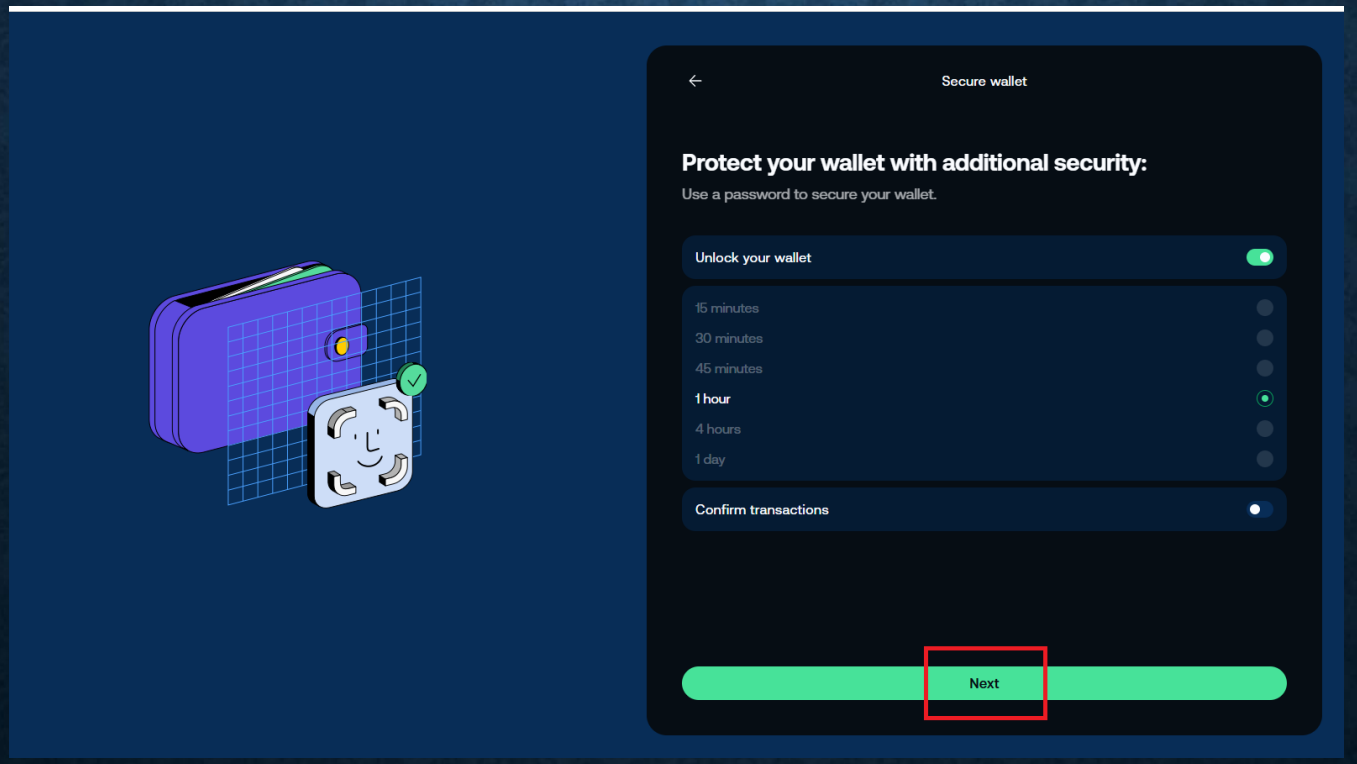
1　Suiウォレットの追加

パスワードを設定して、次へ



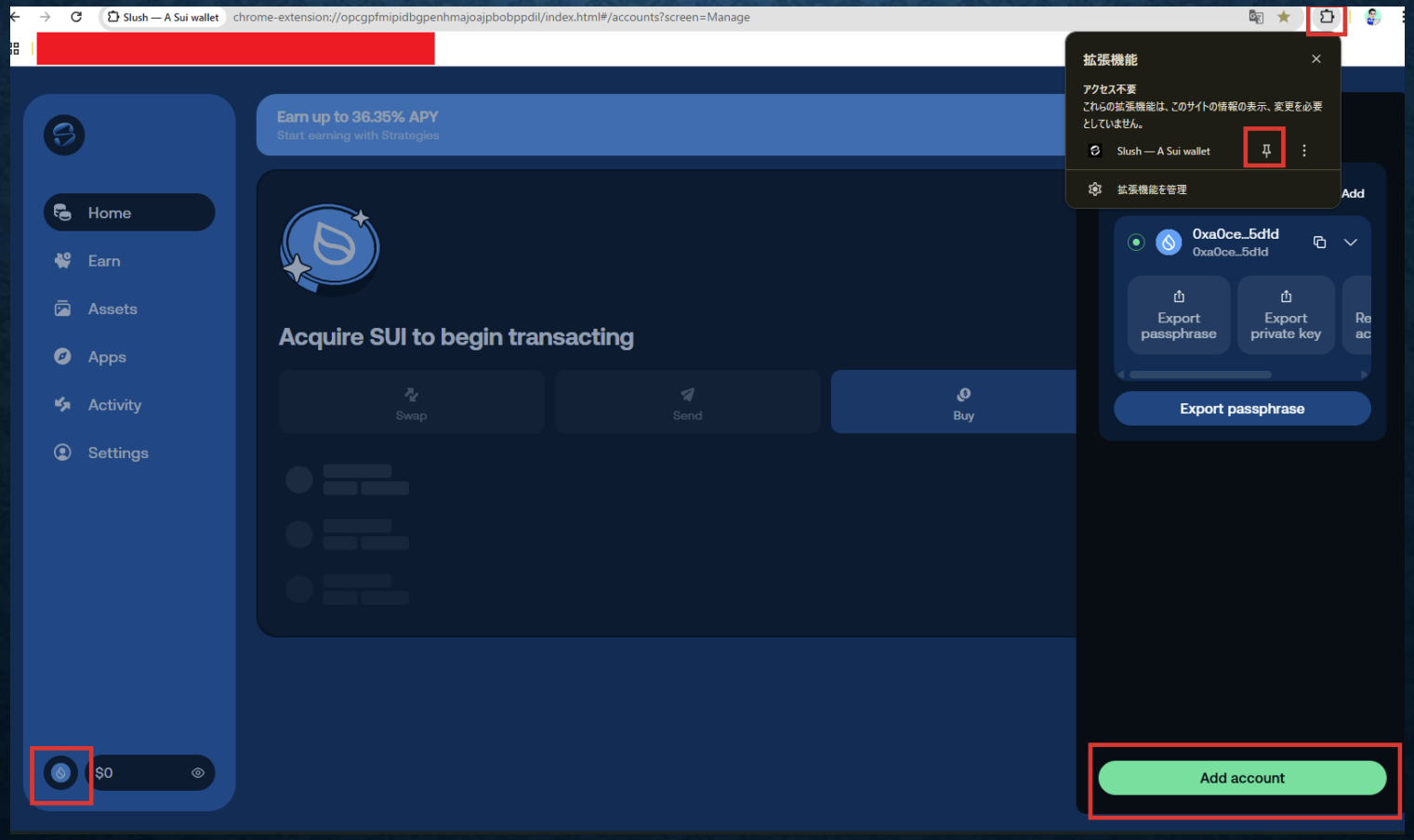
1　Suiウォレットの追加

アンロックの時間を設定して次へ



2　Suiアカウントの作成

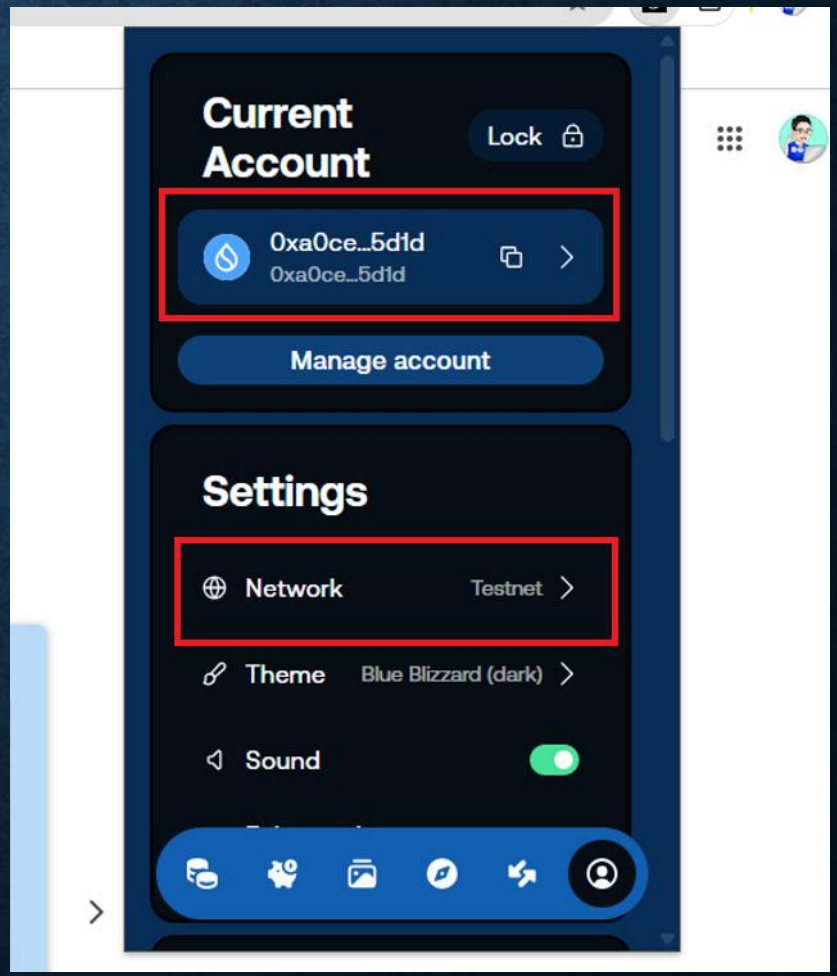
アカウントを作りましょう。
また、ピン止めをしておくと便利です。



２　Suiアカウントの作成

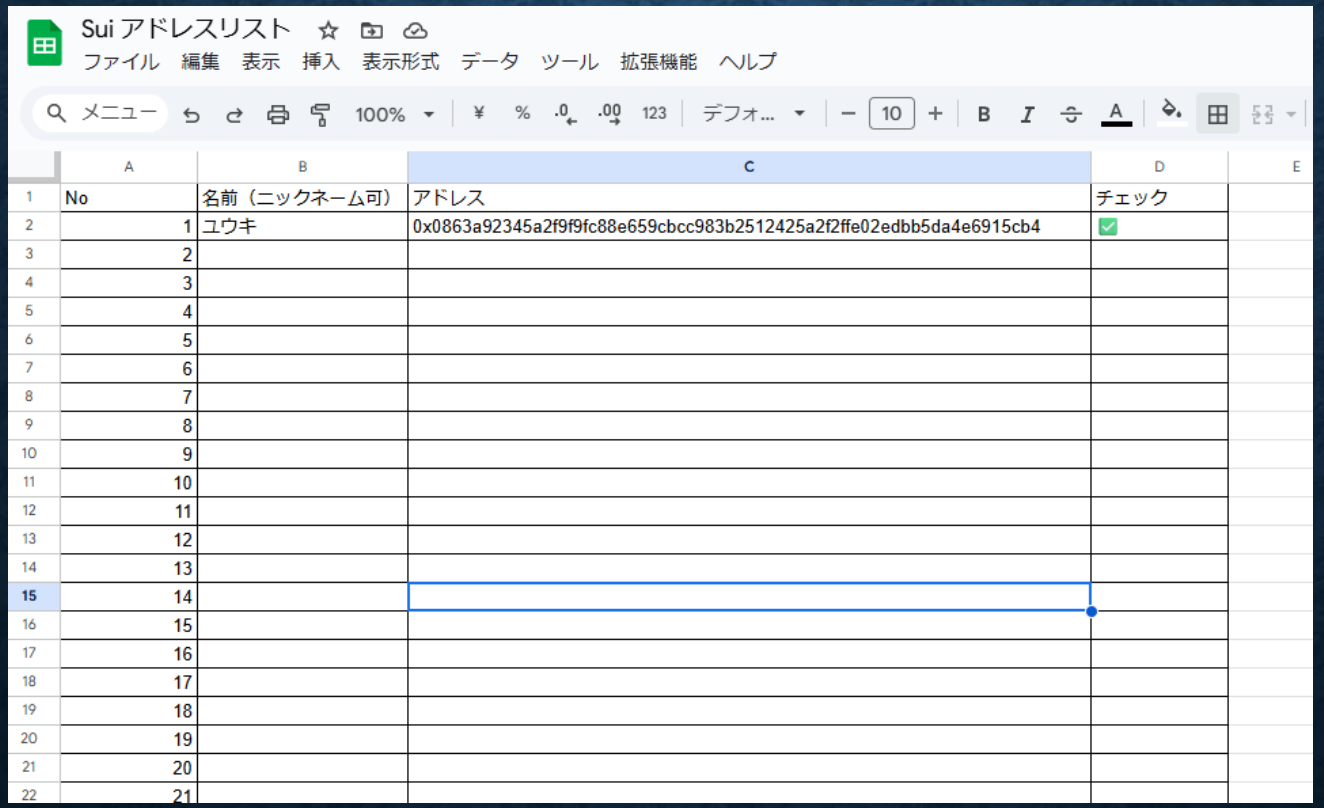
アカウントができました。

Networkはテストネットにしましょう。



2　Suiアカウントの作成

今回初めて作った方はアドレスリストに入れてください。
テストトークンを送付いたします。



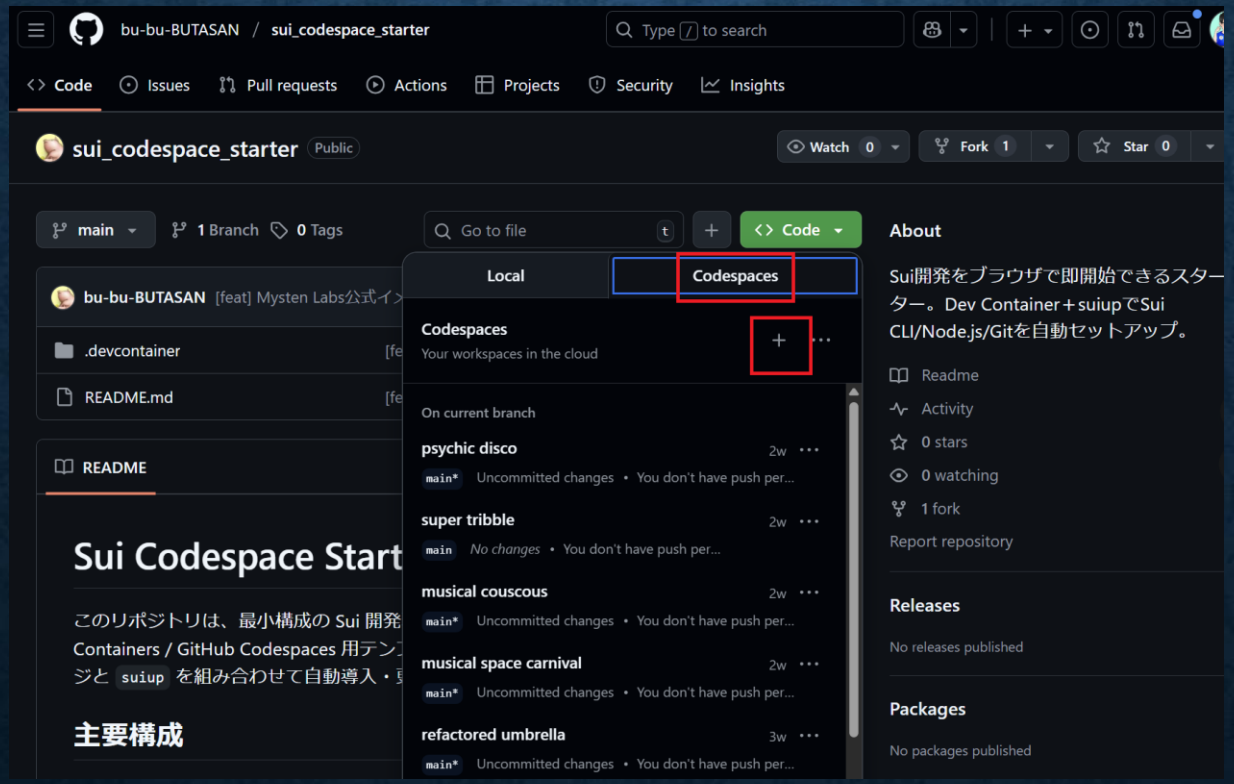
	A	B	C	D	E
1	No	名前（ニックネーム可）	アドレス	チェック	
2	1	ユウキ	0x0863a92345a2f9f9fc88e659cbcc983b2512425a2f2ffe02edbb5da4e6915cb4	☑	
3	2				
4	3				
5	4				
6	5				
7	6				
8	7				
9	8				
10	9				
11	10				
12	11				
13	12				
14	13				
15	14				
16	15				
17	16				
18	17				
19	18				
20	19				
21	20				
22	21				

https://docs.google.com/spreadsheets/d/1snDFZe0U_drcWWBgaltIkfaFSgImM5ZLLKxnQA2OTlw/edit?gid=1254926714#gid=1254926714

開発体験

1　Github Codespacesからの立ち上げ

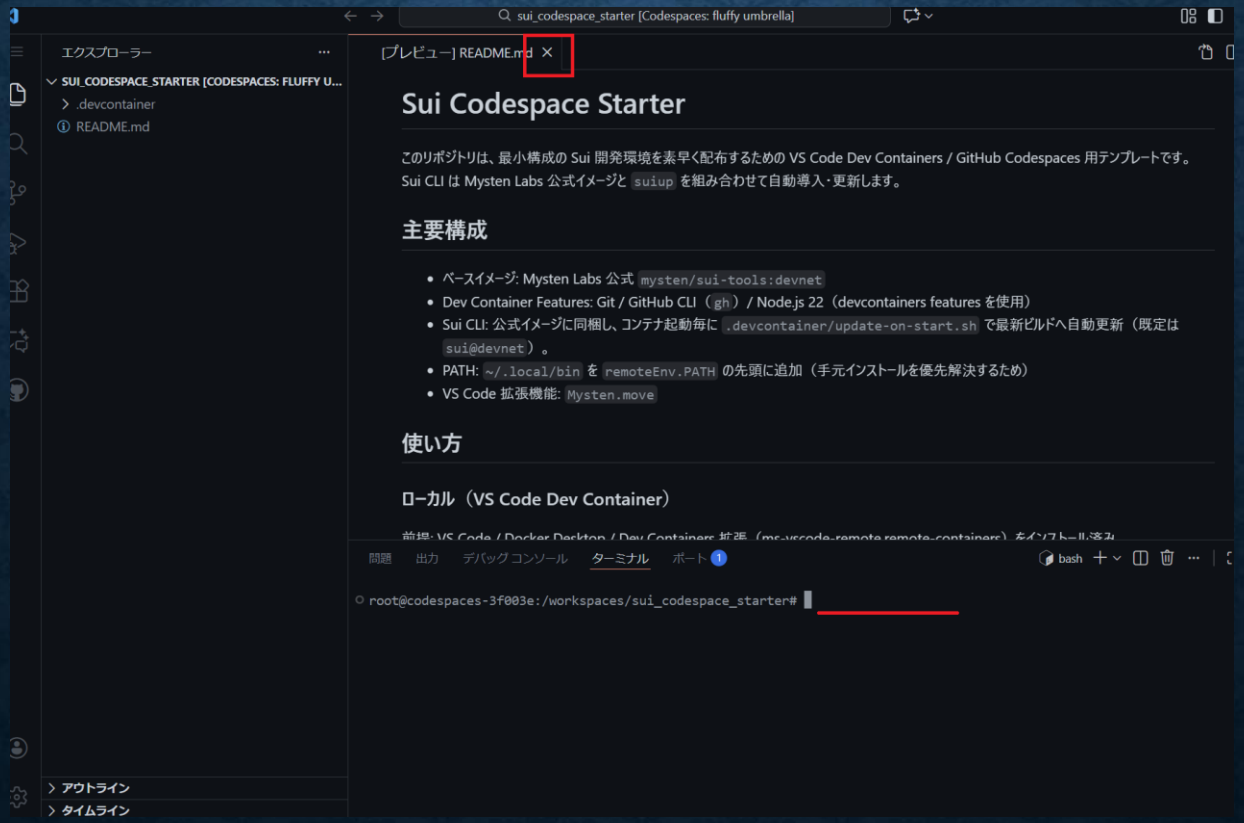
以下のようにして、Githubから立ち上げます。
（Githubアカウントが必要です。）



https://github.com/bu-bu-BUTASAN/sui_codespace_starter

1　Github Codespacesからの立ち上げ

立ち上げることで、コードを入力することができるようになります。
「README.md」は使用しないため消しましょう。



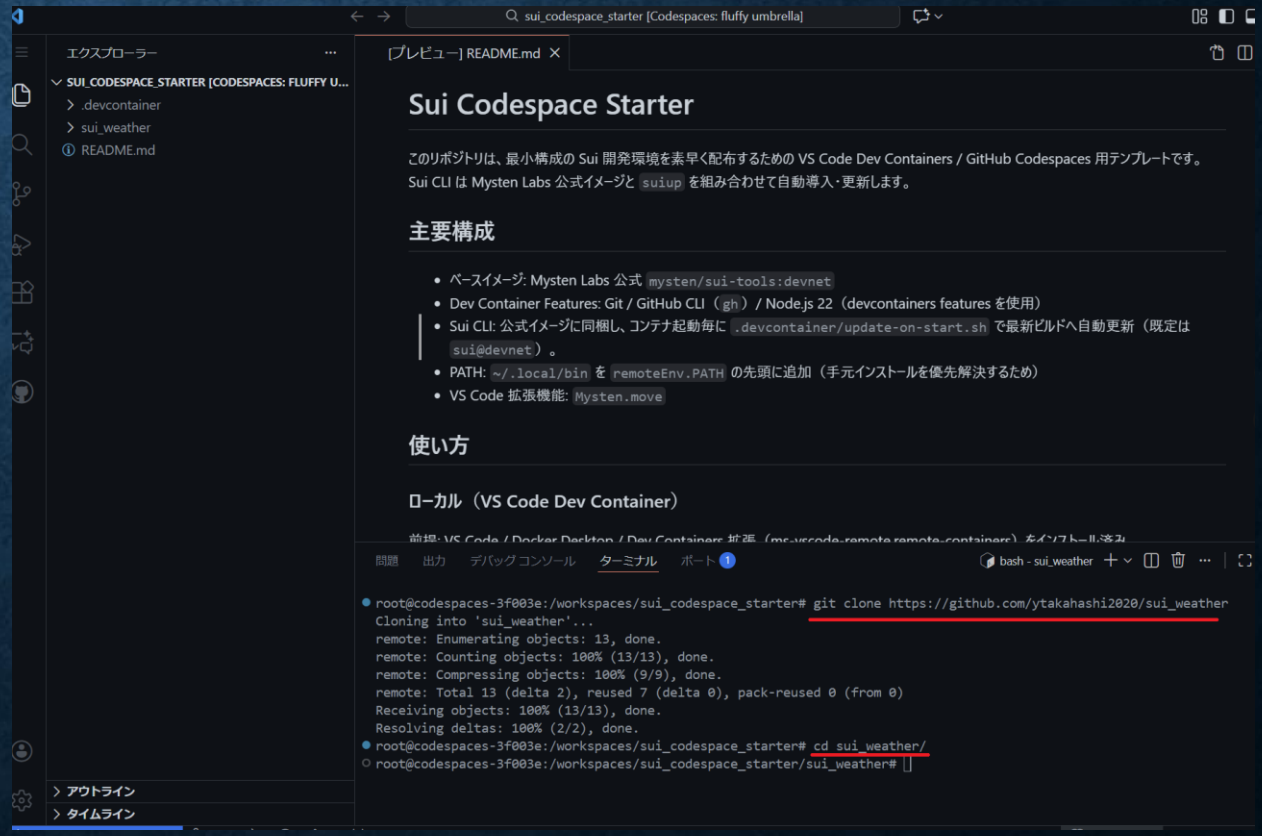
2　Git Cloneの実行

// Git clone

git clone https://github.com/ytakahashi2020/sui_weather

// ディレクトリの移動

cd sui_weather/



2　Git Cloneの実行

大まかな構成を確認しましょう。

sui_weather / sources / weather.move

ytakahashi2020 Update weather.move　bf38028 · 3 weeks ago　History

Code　Blame　26 lines (22 loc) · 685 Bytes　Raw

```
1  module weather::weather;
2
3  /// 東京の気温だけを保持するシンプルなオブジェクト
4  public struct Weather has key, store {
5      id: UID,
6      tokyo: u64,
7  }
8
9  /// 初期化関数（モジュール発行後に自分で呼ぶ）
10 public fun create(ctx: &mut TxContext) {
11     let weather = Weather {
12         id: object::new(ctx),
13         tokyo: 20, // 初期値
14     };
15     transfer::share_object(weather);
16 }
17
18 /// 🍡 setter（気温を更新する）
19 public fun set_temperature(weather: &mut Weather, temperature: u64) {
20     weather.tokyo = temperature;
21 }
22
23 /// 👀 getter（気温を取得する）
24 public fun get_temperature(weather: &Weather): u64 {
25     weather.tokyo
26 }
```

構造体（IDと東京の気温を持つ）

Create関数（オブジェクトを作る）

セッター（値（気温）をセットする）

ゲッター（値（気温）をゲットする）

3　Suiのウォレットの作成

// キーペアの作成

sui client –y

また、アクティブアドレスを表示するコマンドを探しましょう。

```
Resolving deltas: 100% (2/2), done.
root@codespaces-3f003e:/workspaces/sui_codespace_starter# cd sui_weather/
root@codespaces-3f003e:/workspaces/sui_codespace_starter/sui_weather# sui client -y
Creating config file ["/root/.sui/sui_config/client.yaml"] with default (devnet) Full node server and ed25519 k
.Generated new keypair and alias for address with scheme "ed25519" [hardcore-carnelian: 0x11ba0ff5c75d73bbfdc4e
b6d8d2fa9137c9c8d01d7ad7ed82fe1cb1]
Secret Recovery Phrase : [vacant orchard fruit beyond usage eternal spike dune talent gossip horn acoustic]
Client for interacting with the Sui network

Usage: sui client [OPTIONS] [COMMAND]

Commands:
  active-address             Default address used for commands when none specified
  active-env                 Default environment used for commands when none specified
  addresses                  Obtain the Addresses managed by the client
  balance                    List the coin balance of an address
  call                       Call Move function
  chain-identifier           Query the chain identifier from the rpc endpoint
  dynamic-field              Query a dynamic field by its address
  envs                       List all Sui environments
  execute-signed-tx          Execute a Signed Transaction. This is useful when the user prefers to sign elsewh
                                 and use this command to execute
  execute-combined-signed-tx Execute a combined serialized SenderSignedData string
  faucet                     Request gas coin from faucet. By default, it will use the active address and the
                                 active network
```

３　Suiのウォレットの作成

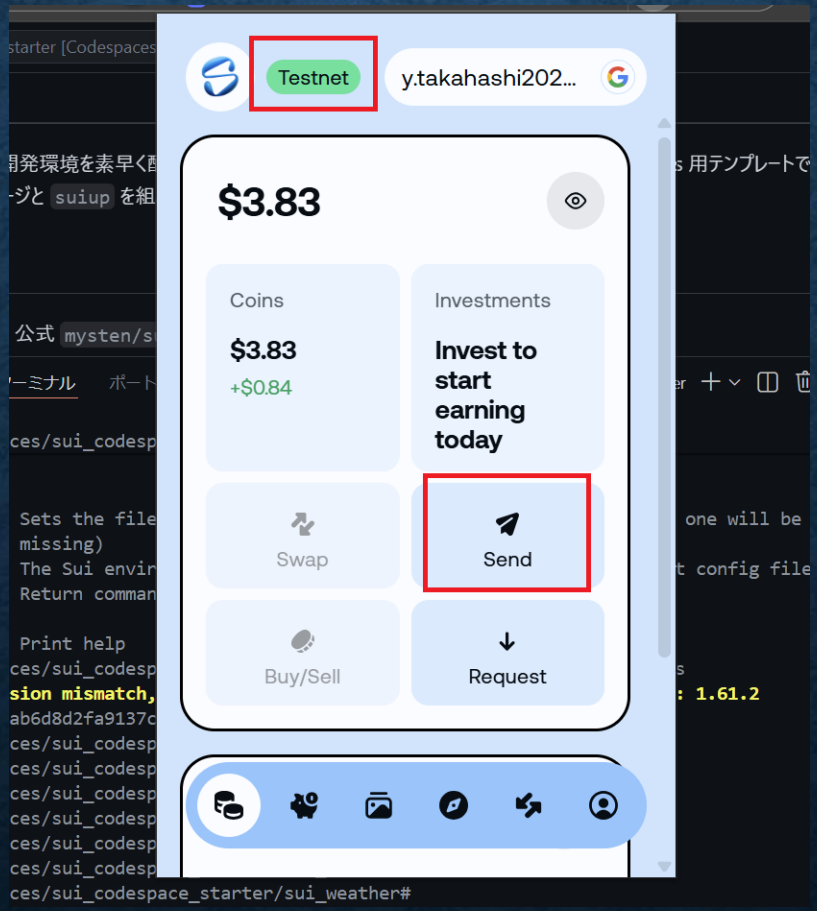
// アクティブアドレスの表示

sui client active-address

```
Options:
      --client.config <CONFIG>  Sets the file storing the state of our user accounts (an empty one will be
                                missing)
      --client.env <ENV>        The Sui environment to use. This must be present in the current config fi
      --json                    Return command outputs in json format
  -y, --yes
  -h, --help                    Print help
root@codespaces-3f003e:/workspaces/sui_codespace_starter/sui_weather# sui client active-address
[warning] Client/Server api version mismatch, client api version : 1.62.0, server api version : 1.61.2
0x11ba0ff5c75d73bbfdc4eb60be1dcab6d8d2fa9137c9c8d01d7ad7ed82fe1cb1
root@codespaces-3f003e:/workspaces/sui_codespace_starter/sui_weather#
root@codespaces-3f003e:/workspaces/sui_codespace_starter/sui_weather#
root@codespaces-3f003e:/workspaces/sui_codespace_starter/sui_weather#
root@codespaces-3f003e:/workspaces/sui_codespace_starter/sui_weather#
root@codespaces-3f003e:/workspaces/sui_codespace_starter/sui_weather#
root@codespaces-3f003e:/workspaces/sui_codespace_starter/sui_weather#
root@codespaces-3f003e:/workspaces/sui_codespace_starter/sui_weather#
root@codespaces-3f003e:/workspaces/sui_codespace_starter/sui_weather#
root@codespaces-3f003e:/workspaces/sui_codespace_starter/sui_weather#
```

4　ガス代の確認

アクティブアドレスに向けて、少量のガス代を送りましょう。



4　ガス代の確認

// 残高の確認

sui client gas

```
No gas coins are owned by this address
root@codespaces-3f003e:/workspaces/sui_codespace_starter/sui_weather# sui client gas
[warning] Client/Server api version mismatch, client api version : 1.62.0, server api version : 1.61.2
```

gasCoinId	mistBalance (MIST)	suiBalance (SUI)
0x61a2cc113e1a97cd6872485c420345451bf59ef8b86753570bb5c822479813ae	100000000	0.10

```
root@codespaces-3f003e:/workspaces/sui_codespace_starter/sui_weather#
root@codespaces-3f003e:/workspaces/sui_codespace_starter/sui_weather#
root@codespaces-3f003e:/workspaces/sui_codespace_starter/sui_weather#
root@codespaces-3f003e:/workspaces/sui_codespace_starter/sui_weather#
root@codespaces-3f003e:/workspaces/sui_codespace_starter/sui_weather#
root@codespaces-3f003e:/workspaces/sui_codespace_starter/sui_weather#
root@codespaces-3f003e:/workspaces/sui_codespace_starter/sui_weather#
```

5　デプロイの実行

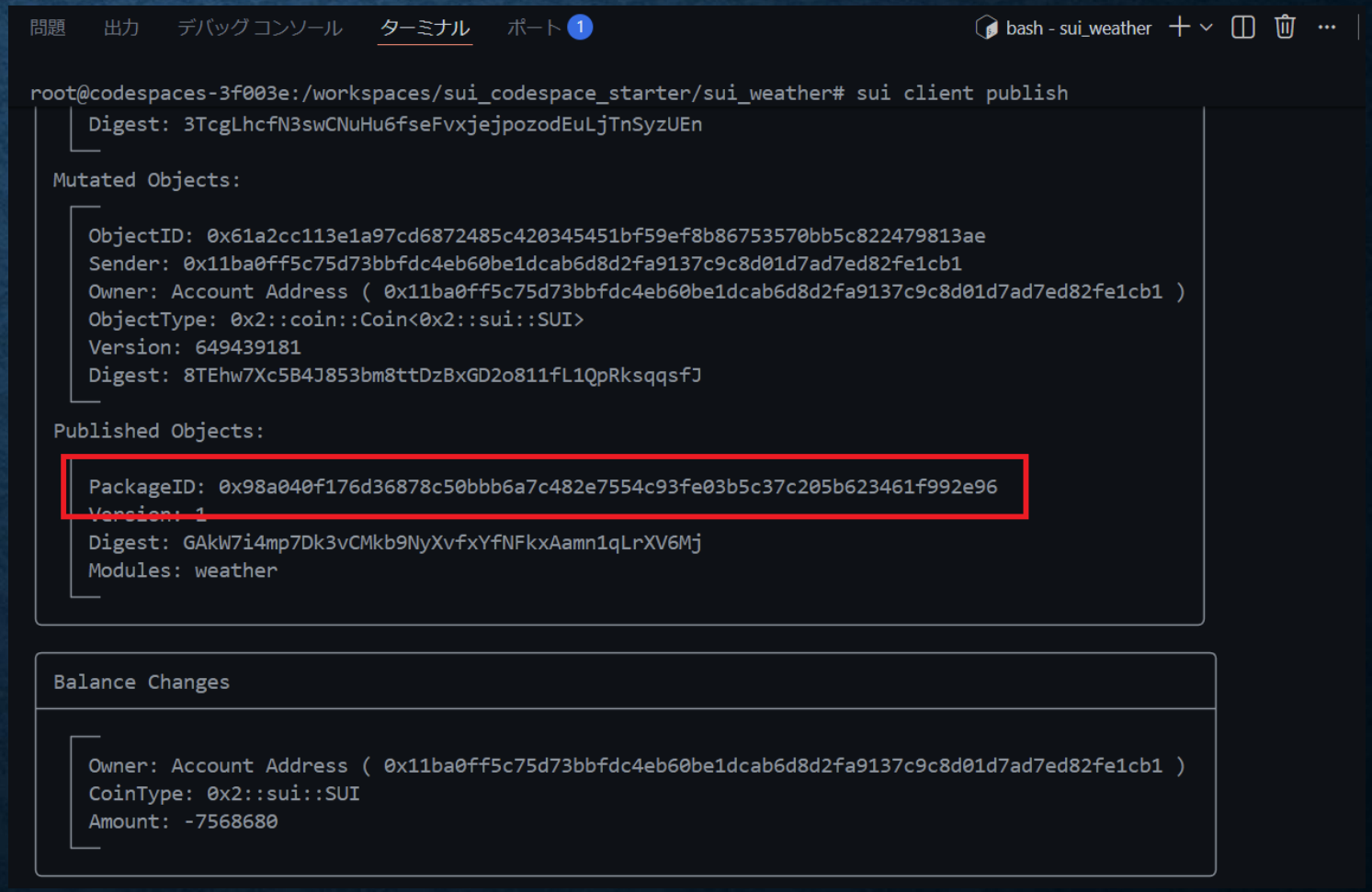
// デプロイの実行

sui client publish

```
root@codespaces-3f003e:/workspaces/sui_codespace_starter/sui_weather#
root@codespaces-3f003e:/workspaces/sui_codespace_starter/sui_weather# sui client publish
[warning] Client/Server api version mismatch, client api version : 1.62.0, server api version : 1.61.2
[Note]: Dependency sources are no longer verified automatically during publication and upgrade. You can pass the
fy-deps` option if you would like to verify them as part of publication or upgrade.
FETCHING GIT DEPENDENCY https://github.com/MystenLabs/sui.git
Cloning into '/root/.move/https___github_com_MystenLabs_sui_git_c1f1ae650fb9f9248b39a569400b4420820868db'...
remote: Enumerating objects: 34419, done.
remote: Counting objects: 100% (16/16), done.
remote: Compressing objects: 100% (16/16), done.
remote: Total 34419 (delta 0), reused 0 (delta 0), pack-reused 34403 (from 2)
Receiving objects: 100% (34419/34419), 18.34 MiB | 15.53 MiB/s, done.
Resolving deltas: 100% (3502/3502), done.
FETCHING GIT DEPENDENCY https://github.com/MystenLabs/sui.git
Cloning into '/root/.move/https___github_com_MystenLabs_sui_git_f63c9fc78e2171fa174dc43e757ded416c204558'...
remote: Enumerating objects: 34419, done.
remote: Counting objects: 100% (16/16), done.
remote: Compressing objects: 100% (16/16), done.
remote: Total 34419 (delta 0), reused 0 (delta 0), pack-reused 34403 (from 2)
Receiving objects: 100% (34419/34419), 18.34 MiB | 12.01 MiB/s, done.
Resolving deltas: 100% (3502/3502), done.
INCLUDING DEPENDENCY Bridge
INCLUDING DEPENDENCY SuiSystem
INCLUDING DEPENDENCY Sui
INCLUDING DEPENDENCY MoveStdlib
BUILDING weather
Skipping dependency verification
Transaction Digest: 7wpDGyRGQhNgU1mNhm8v7Ly4XG7KMikQwZ3BUVaT3HJr
```

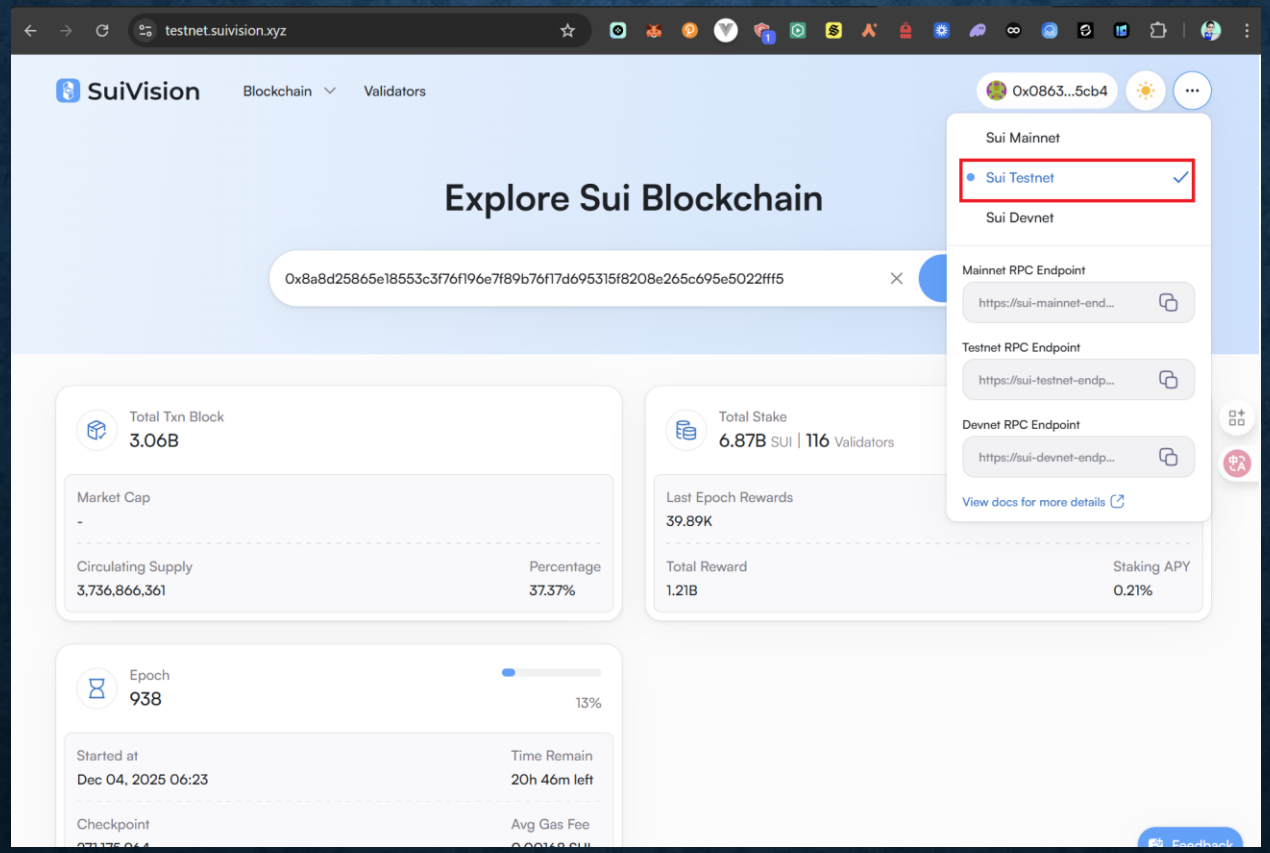
6　パッケージの確認

パッケージIDはこちらです。



6　パッケージの確認

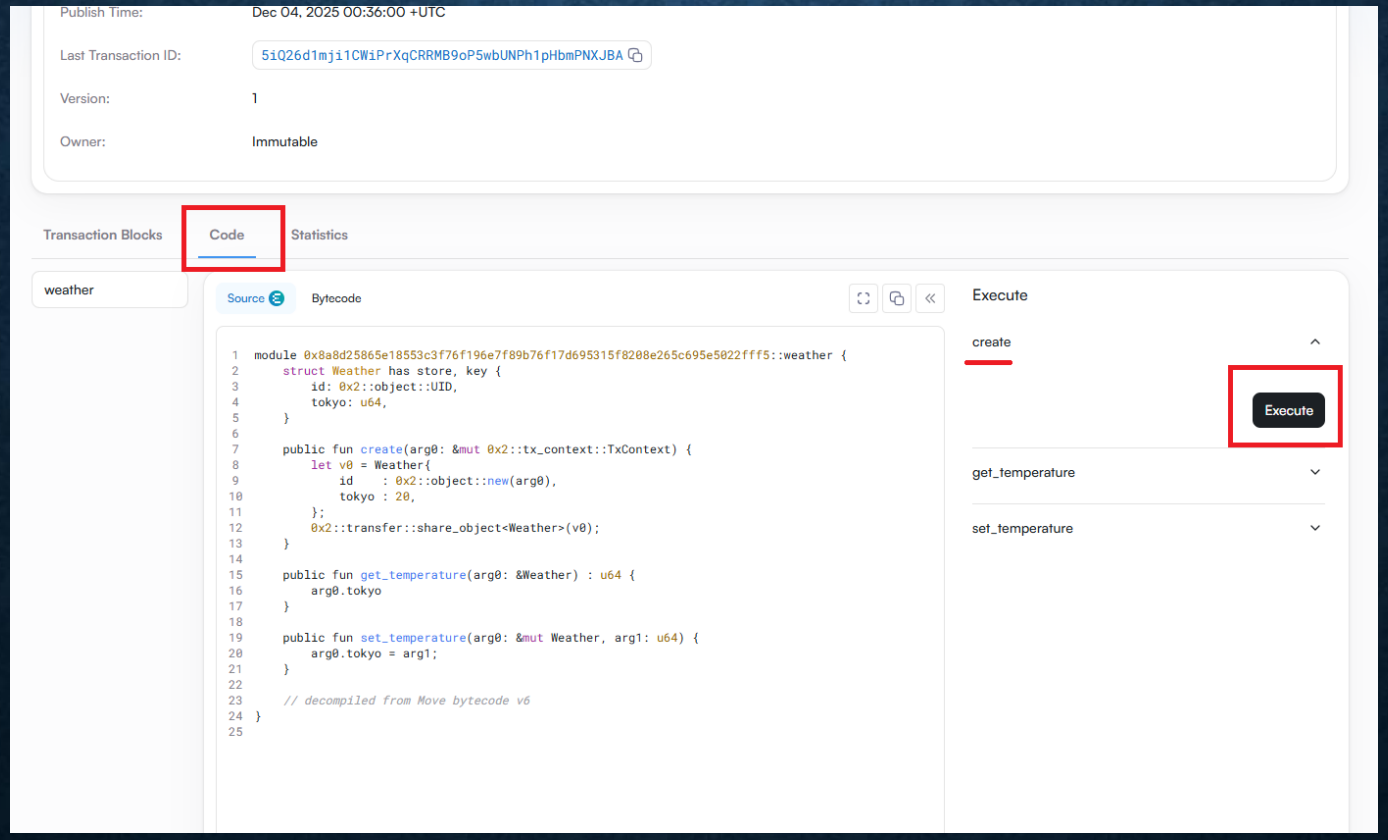
エクスプローラ（Sui Vision）でも確認してみましょう。



https://testnet.suivision.xyz/

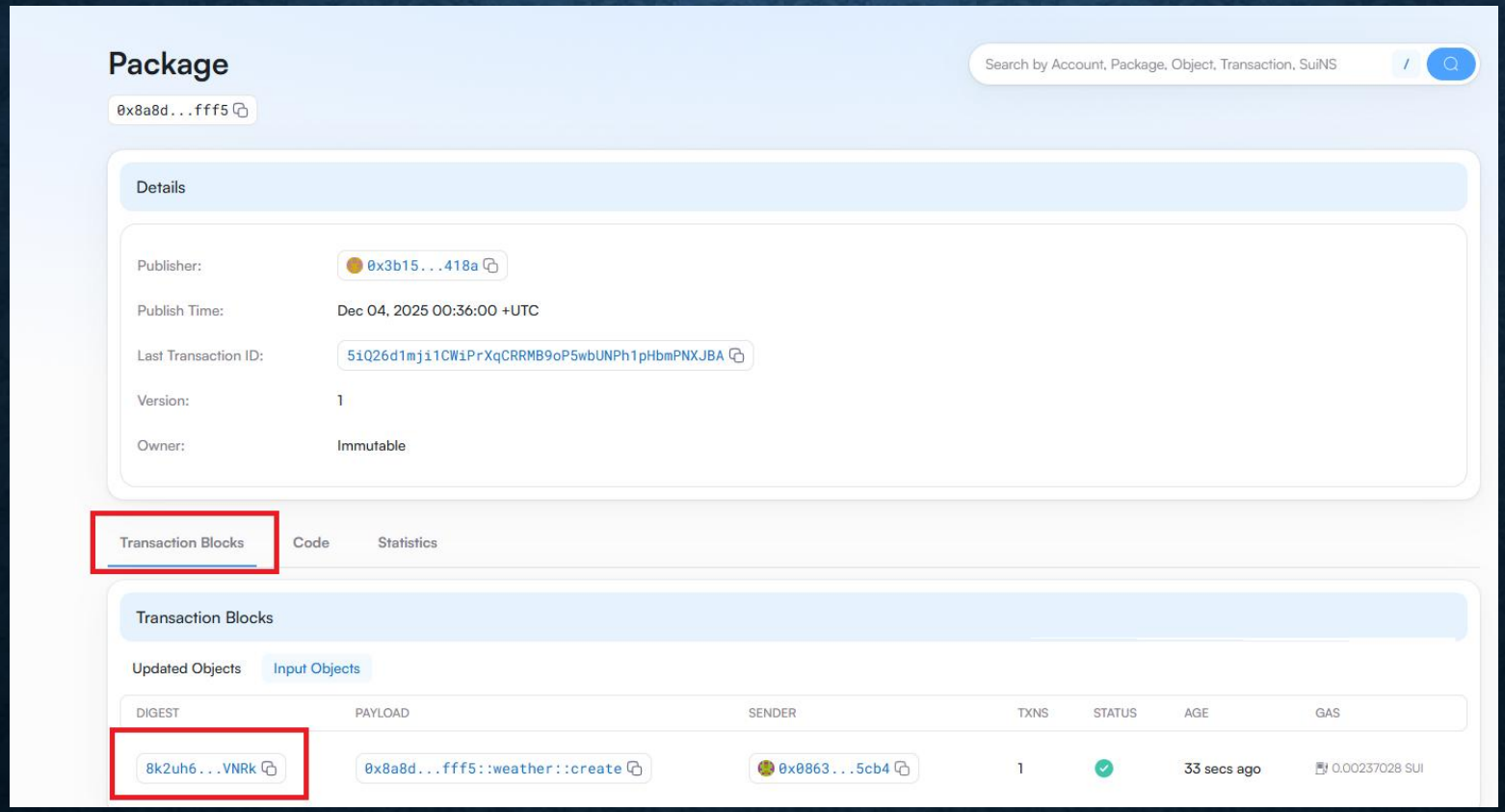
7　関数の実行

「Create」関数からオブジェクトを作ってみましょう。



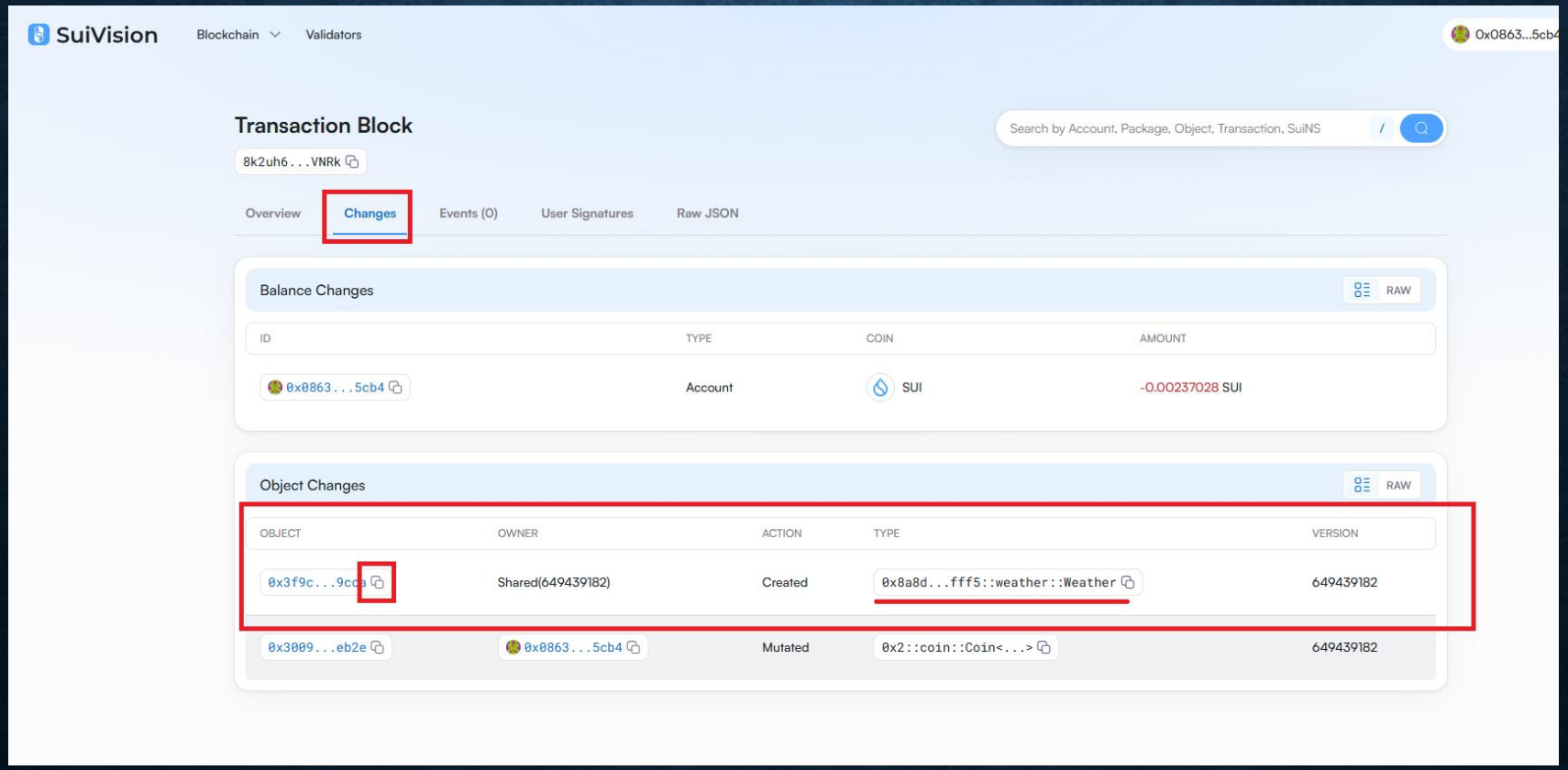
7　関数の実行

処理（トランザクション）が実施されていることが確認できます。



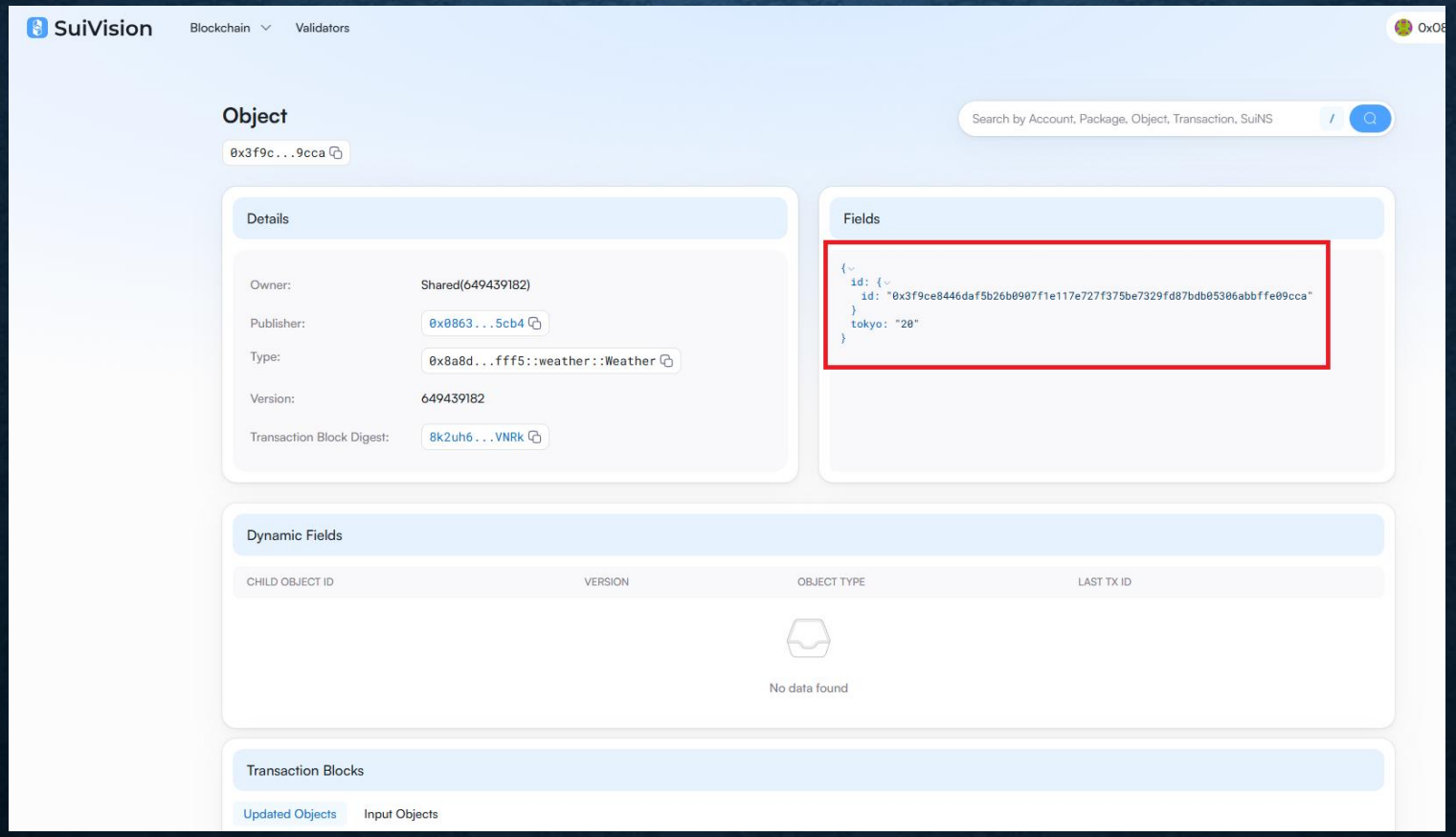
7　関数の実行

新しく、「オブジェクト」が作成されていることが確認できました。



7　関数の実行

オブジェクトを確認すると、東京の気温が確認できました。



7　関数の実行

別の気温に変えてみましょう。
コードを確認すると、二つの引数が必要なようです。

sui_weather / sources / weather.move

ytakahashi2020 Update weather.move　bf38028 · 3 weeks ago　History

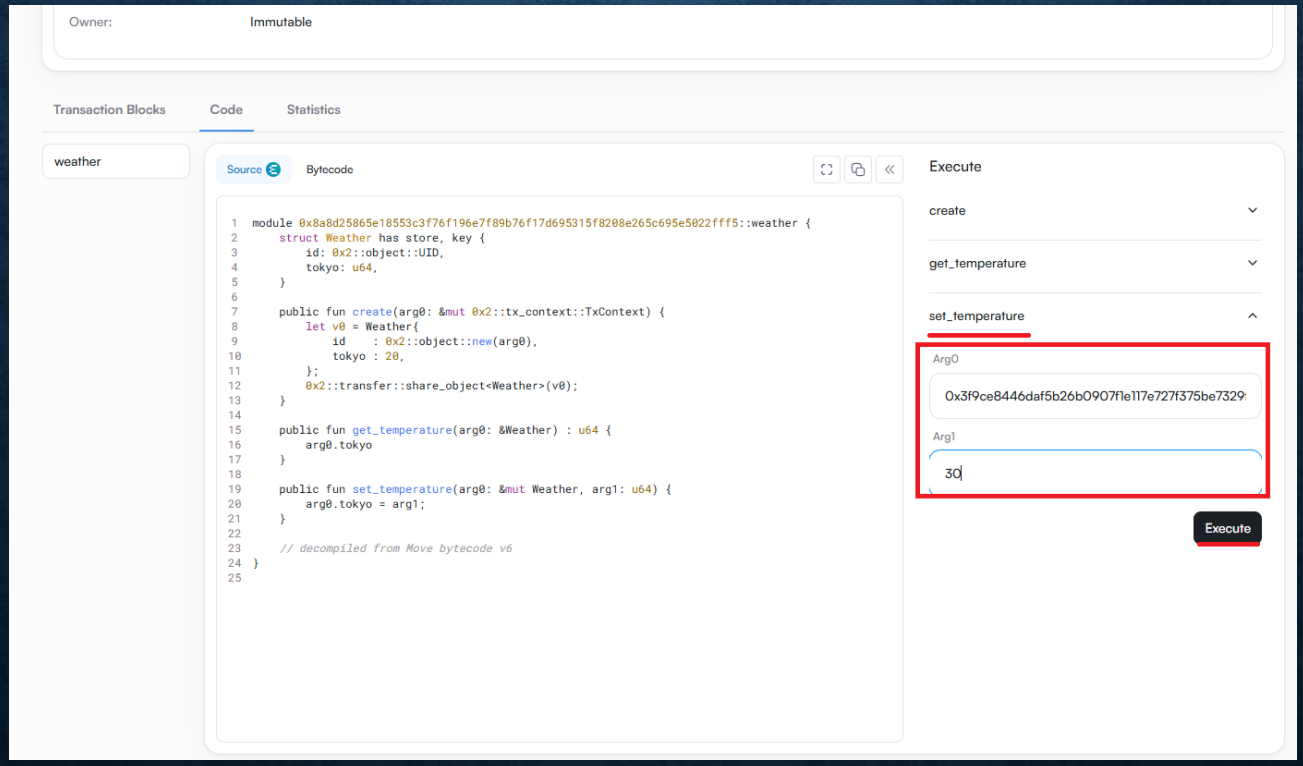
Code　Blame　26 lines (22 loc) · 685 Bytes　Raw

```move
module weather::weather;

/// 東京の気温だけを保持するシンプルなオブジェクト
public struct Weather has key, store {
    id: UID,
    tokyo: u64,
}

/// 初期化関数（モジュール発行後に自分で呼ぶ）
public fun create(ctx: &mut TxContext) {
    let weather = Weather {
        id: object::new(ctx),
        tokyo: 20, // 初期値
    };
    transfer::share_object(weather);
}

/// 🍡 setter（気温を更新する）
public fun set_temperature(weather: &mut Weather, temperature: u64) {
    weather.tokyo = temperature;
}

/// 👀 getter（気温を取得する）
public fun get_temperature(weather: &Weather): u64 {
    weather.tokyo
}
```

set_temperature関数には２つの引数（関数に渡すもの）があります。
そして、型（どういう形か、例えば、数字か文字かなど）は：の後ろに書きます。

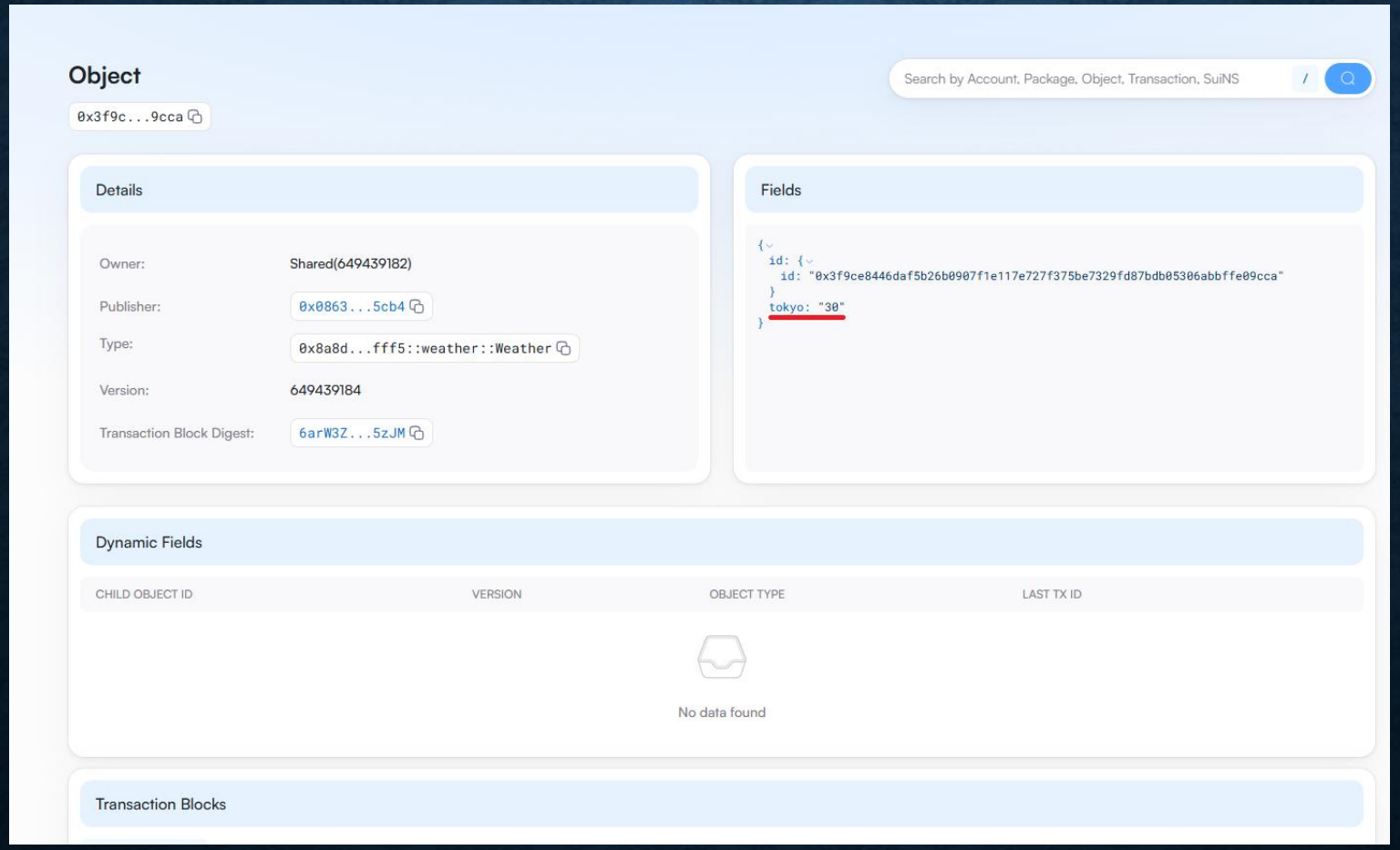
7　関数の実行

set_temperature関数に二つの引数を渡して実行してみましょう。



7　関数の実行

このように気温が変更できました



実装体験

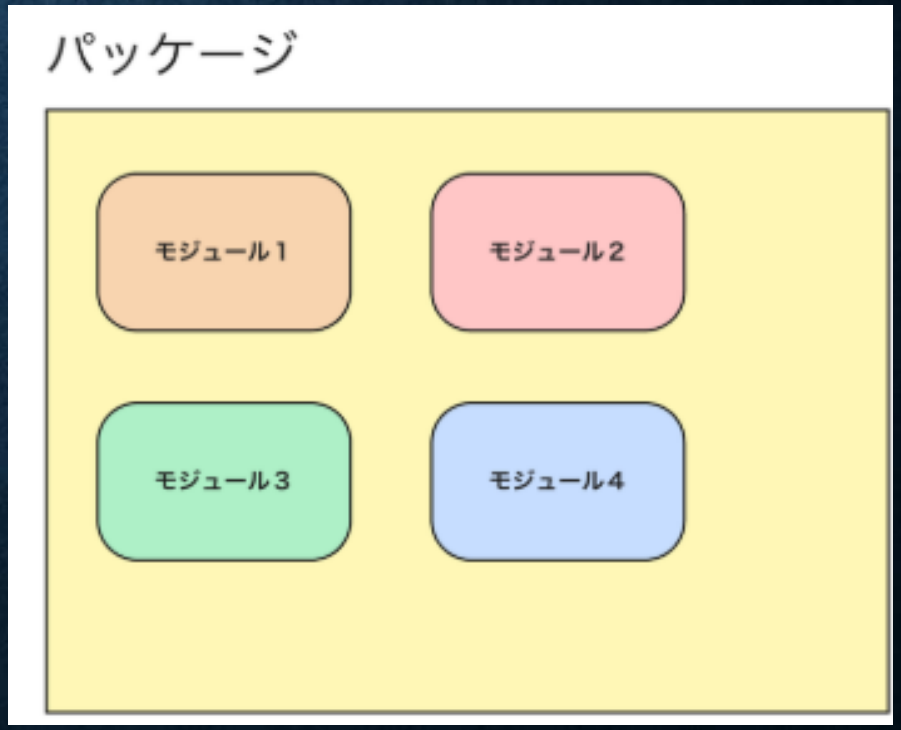
1 パッケージの作成

// パッケージの作成
sui move new <パッケージ名>
// ディレクトリの移動
cd <パッケージ名>

2 パッケージとモジュール

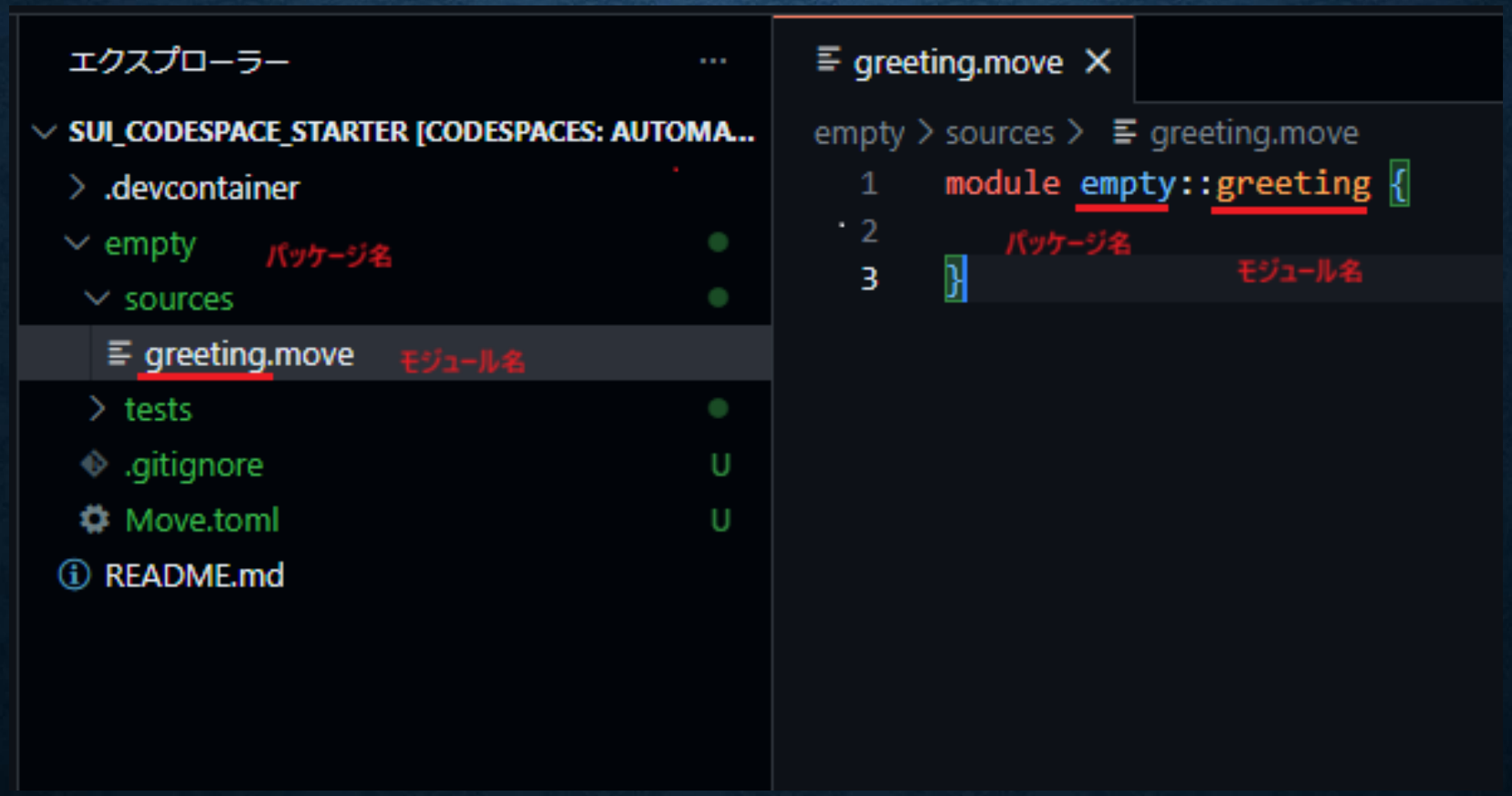
Suiではパッケージの中に複数のモジュールが配置されます。
（とはいえ、多くの場合はモジュールは一つです。）

このモジュールがいわゆるスマートコントラクトです。



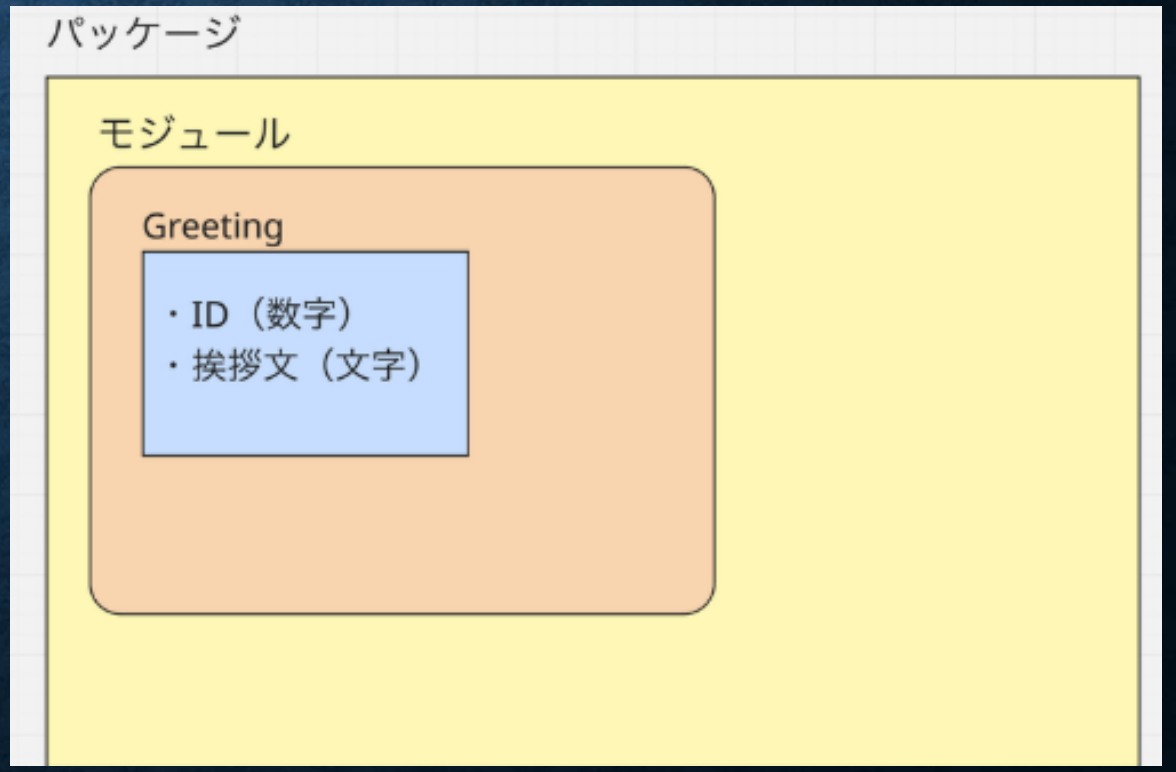
2 パッケージとモジュール

まずはモジュールを作成しましょう。



3 構造体

いくつかの変数（値を入れる入れ物）をまとめたものを構造体と言います。
今回はあいさつ用の構造体を作ろうと思います。



3 構造体

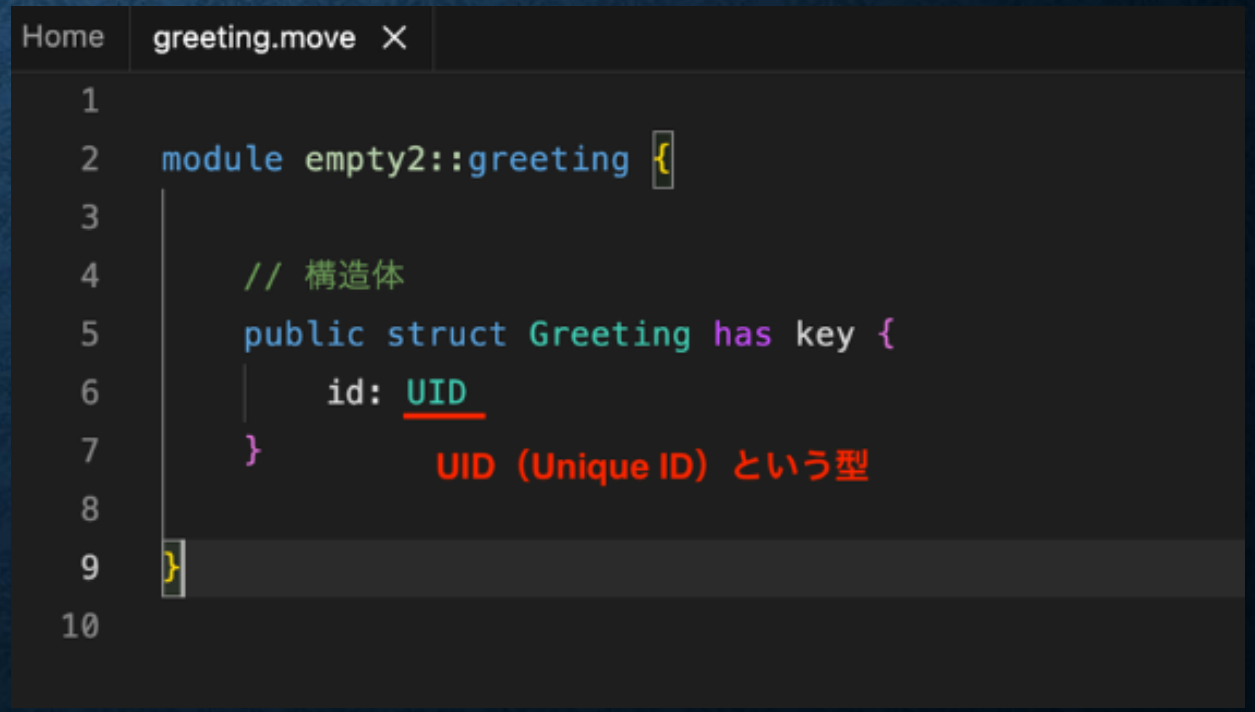
構造体の外側を書いていきましょう。
今回は「Greeting」という名前の構造体を作ります。

4 Keyの設定

この構造体には4つのアビリティの一つである「key」を設定します。
これでこの構造体が世界で唯一のものとなります。

4 Keyの設定

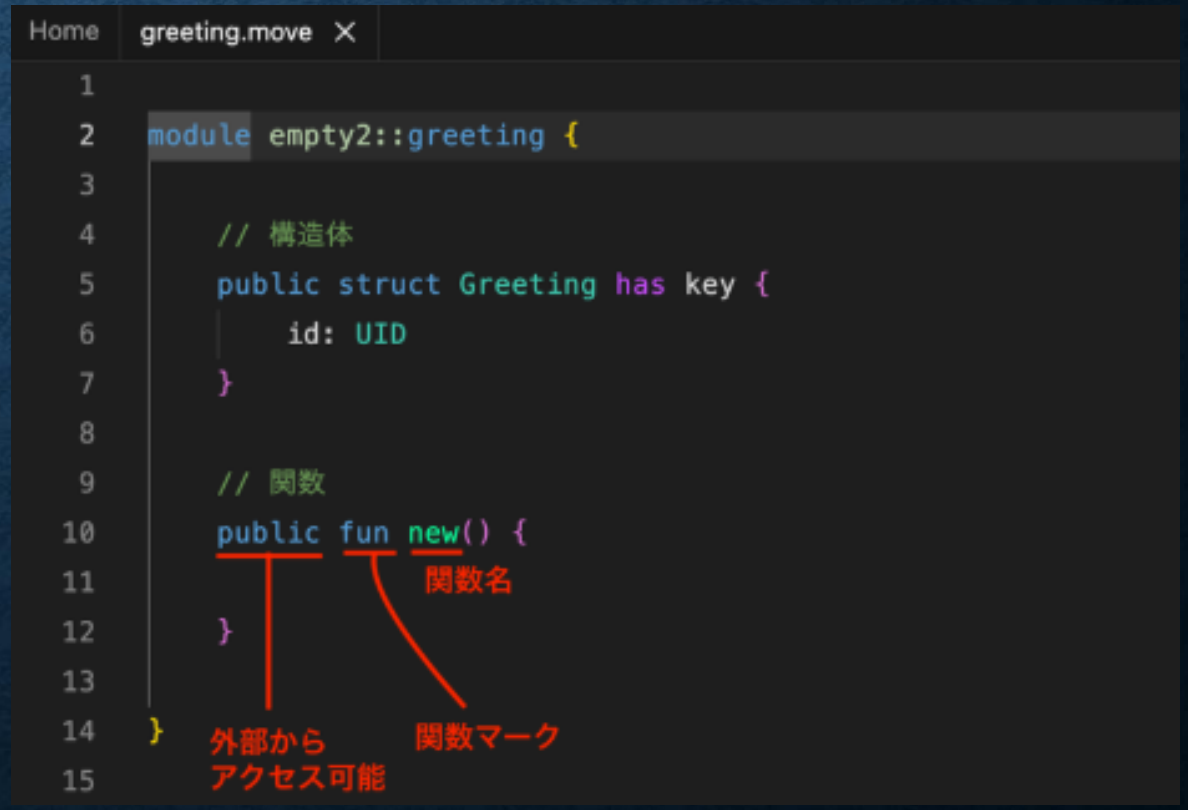
Suiでは、変数に型を設定します。
その変数に数字が入るのか、文字が入るのかなどを明確にするためです。
idにはUIDという特殊な型を入れます。



５ 関数の作成

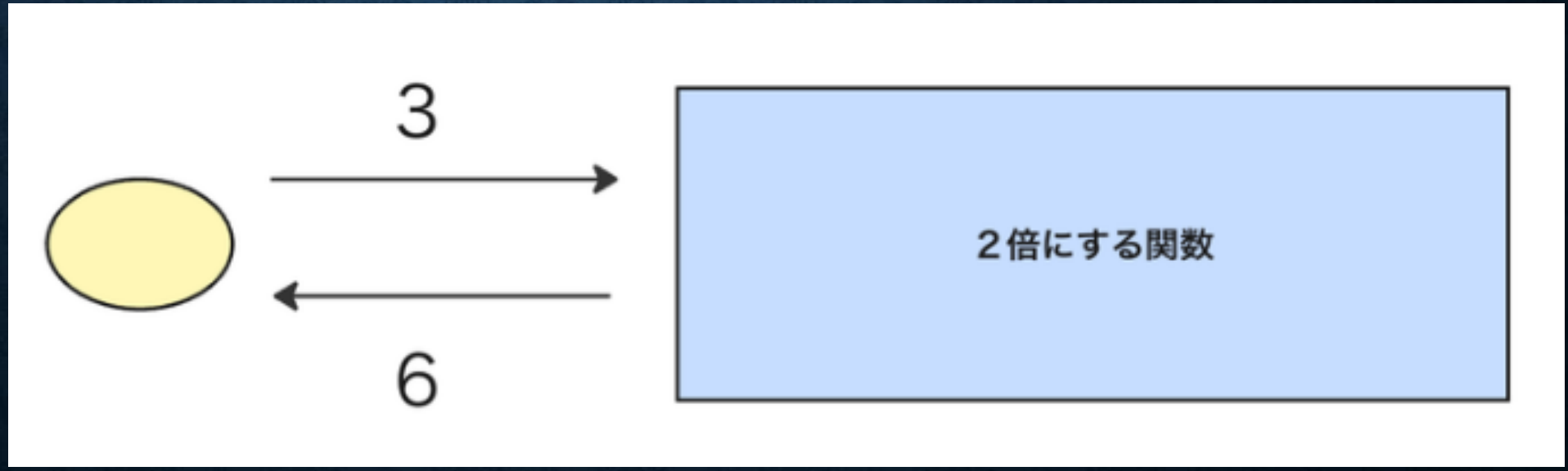
特定の処理のことを関数と言います。

ここでは、新しく構造体用のオブジェクト（もの）を作るための関数を作ります。



５ 関数の作成

関数に渡す値のことを引数と言います。
例えば、下の場合、3が引数です。



５ 関数の作成

new関数には「TxContext」という型のctxという引数を渡します。

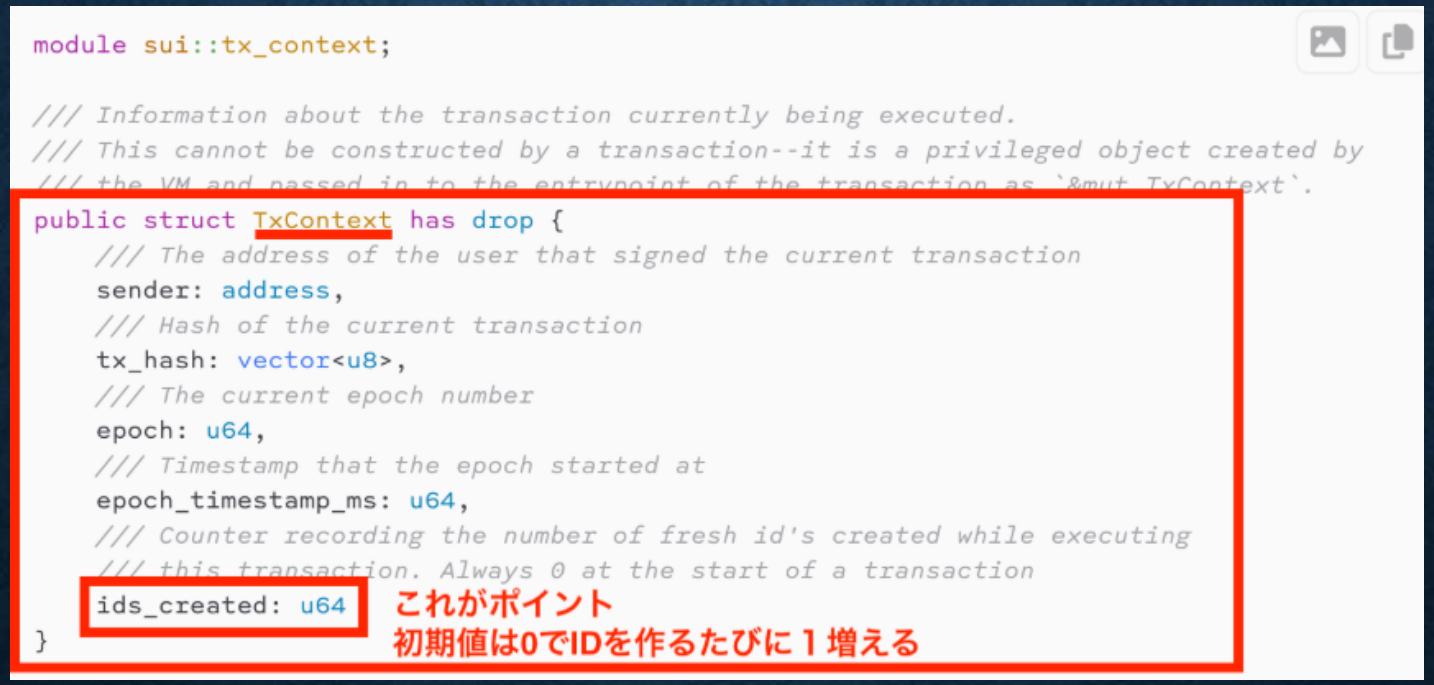
ctxはトランザクションを渡すときの変数のセットです。

新しくオブジェクトを作るときには「&mut TxContext」、そうでないときには「&TxContext」として渡します。

```
4     // 構造体
5     public struct Greeting has key {
6         id: UID
7     }
8
9     // 関数
10    public fun new(ctx: &mut TxContext) {
11
12    }
13
14 }
15
```

５ 関数の作成

参考に、TxContextはこのような型です。



https://move-book.com/programmability/transaction-context

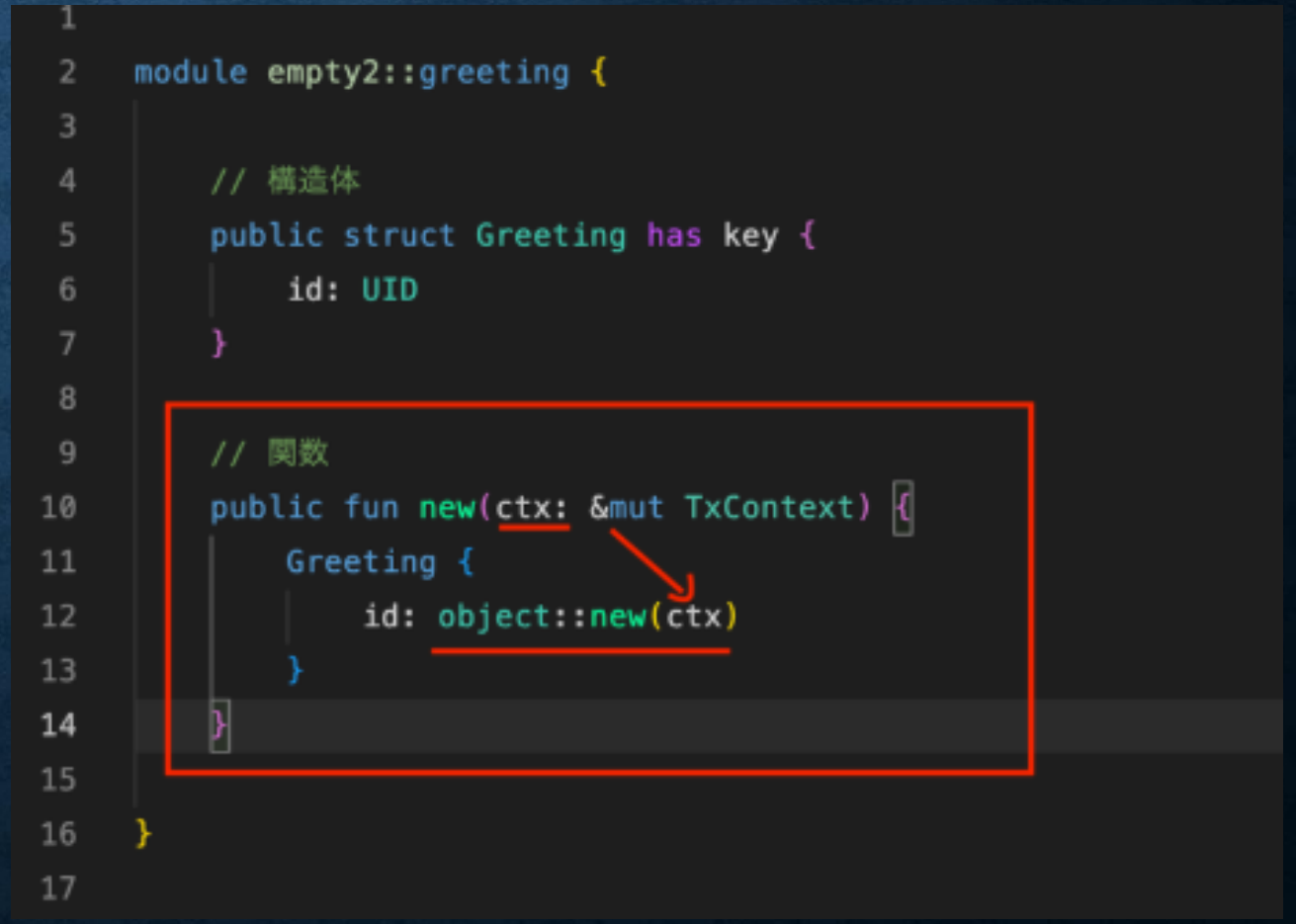
５ 関数の作成

先ほど作成した構造体を作成します。
idは適当に設定しましたが、これでは唯一ではなさそうですね。

```
2  module empty2::greeting {
3
4      // 構造体
5      public struct Greeting has key {
6          id: UID
7      }
8
9      // 関数
10     public fun new(ctx: &mut TxContext) {
11         Greeting {
12             id: 12345
13         }
14     }
15
16 }
17
```

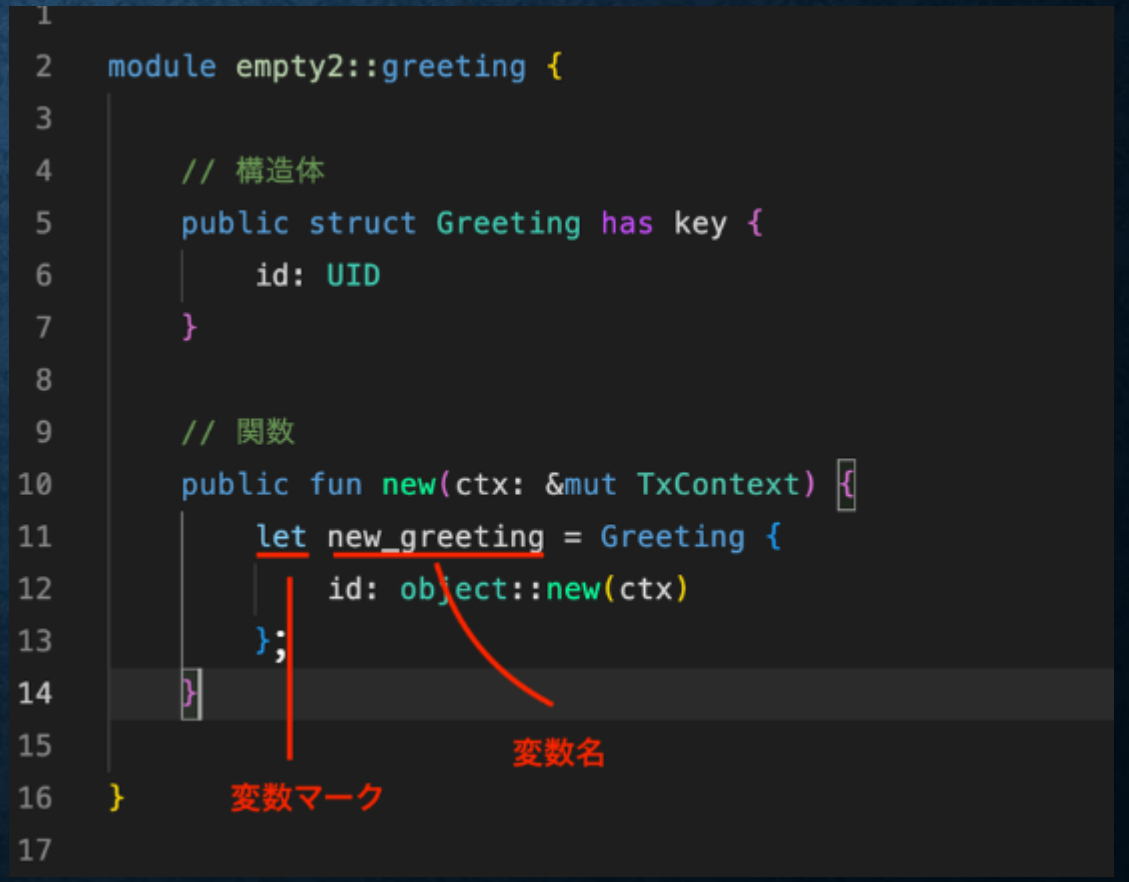
５ 関数の作成

実際には引数のctxを使って、このように作成します。



５ 関数の作成

作った構造体を変数に格納します。
これによって操作がしやすくなります。



５ 関数の作成

最後に、この構造体の所有者を設定します。
ここでは、共有所有とします。

Home　greeting.move ×

```
1
2   module empty2::greeting {
3
4       // 構造体
5       public struct Greeting has key {
6           id: UID
7       }
8
9       // 関数
10      public fun new(ctx: &mut TxContext) {
11          let new_greeting = Greeting {
12              id: object::new(ctx)
13          };
14          transfer::share_object(new_greeting);
15      }
16
17  }
```

５ 関数の作成

最後に、この構造体の所有者を設定します。
ここでは、共有所有とします。

Home　greeting.move ×

```
1
2   module empty2::greeting {
3
4       // 構造体
5       public struct Greeting has key {
6           id: UID
7       }
8
9       // 関数
10      public fun new(ctx: &mut TxContext) {
11          let new_greeting = Greeting {
12              id: object::new(ctx)
13          };
14          transfer::share_object(new_greeting);
15      }
16
17  }
```

6 文字列の設定

文字列を扱うためにuse文を使います。
これを使うことで、文字列であるStringを使う事ができるようになります。

Home greeting.move

```
1
2  module empty2::greeting {
3      use std::string::String;
4
5      // 構造体
6      public struct Greeting has key {
7          id: UID,
8          text: String,
9      }
10
11     // 関数
12     public fun new(ctx: &mut TxContext) {
13         let new_greeting = Greeting {
14             id: object::new(ctx),
15             text: b"this is test".to_string()
16         };
17         transfer::share_object(new_greeting);
18     }
19
20 }
21
```

6 文字列の設定

構造体、関数に文字列を設定します。

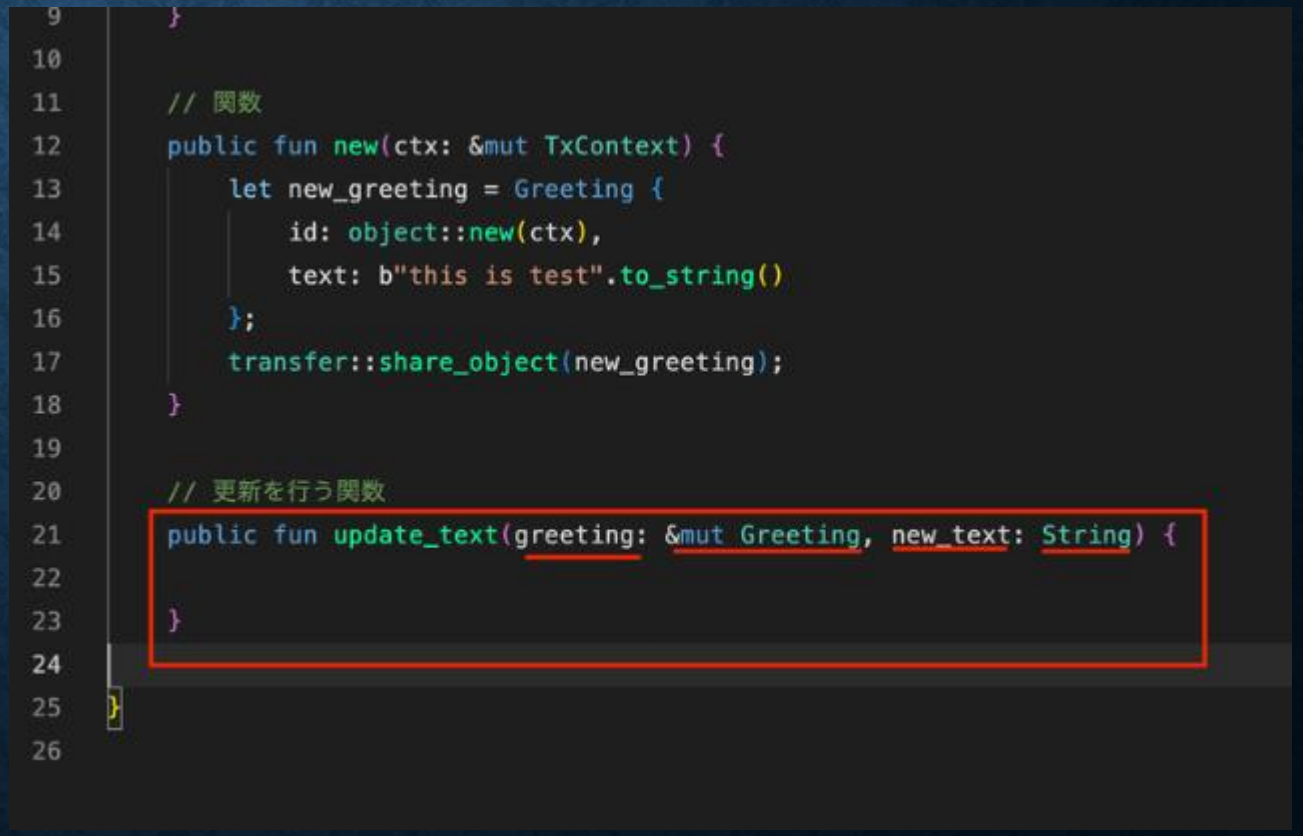
b""”という形により、バイト列という形であり、to_string関数で文字列にしています。

Home greeting.move ×

```
1
2  module empty2::greeting {
3      use std::string::String;
4
5      // 構造体
6      public struct Greeting has key {
7          id: UID,
8          text: String,
9      }
10
11     // 関数
12     public fun new(ctx: &mut TxContext) {
13         let new_greeting = Greeting {
14             id: object::new(ctx),
15             text: b"this is test".to_string()
16         };
17         transfer::share_object(new_greeting);
18     }
19
20 }
21
```

7 変更を行う関数の作成

構造体の文字列を変える関数を作成します。
引数に構造体（オブジェクト）と変更後の文字列を渡します。



7 変更を行う関数の作成

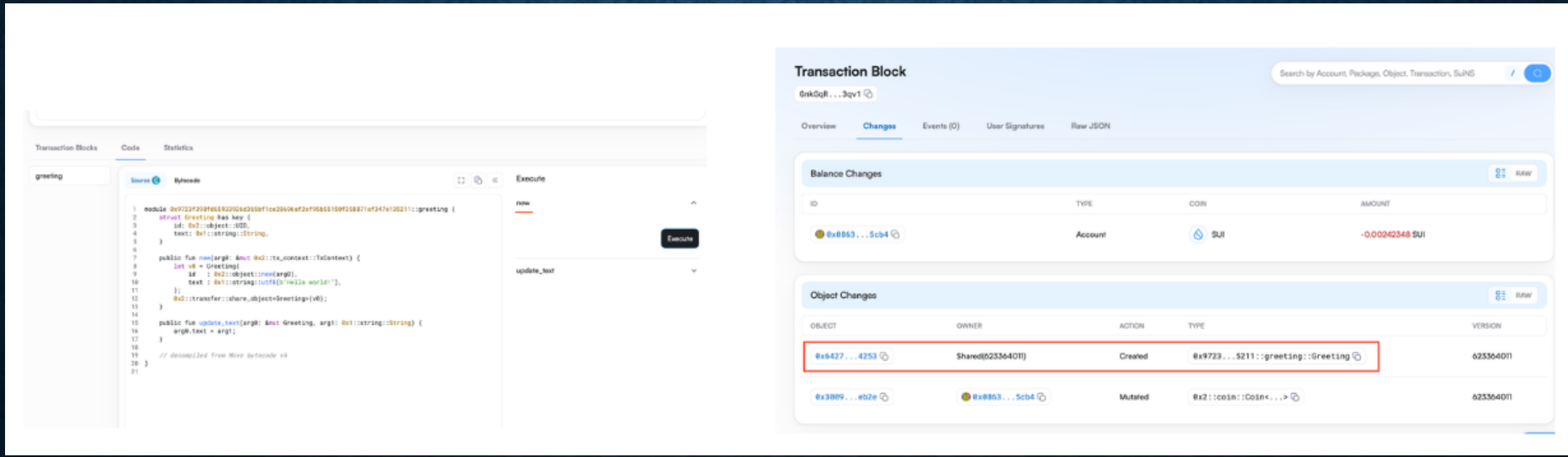
引数のnew_textをgreetingの「text」に代入します。
「=」は代入を表します。

```
4
5       // 構造体
6       public struct Greeting has key {
7           id: UID,
8           text: String,
9       }
10
11      // 関数
12      public fun new(ctx: &mut TxContext) {
13          let new_greeting = Greeting {
14              id: object::new(ctx),
15              text: b"this is test".to_string()
16          };
17          transfer::share_object(new_greeting);
18      }
19
20      // 更新を行う関数
21      public fun update_text(greeting: &mut Greeting, new_text: String) {
22          greeting.text = new_text;
23      }
24
25  }
26
```

8 SuiVisionでの実行

SuiVisionで作成しましょう。

new関数で実行するとオブジェクトが作成され、実行できたObjectにIDがついていることも確認しましょう。

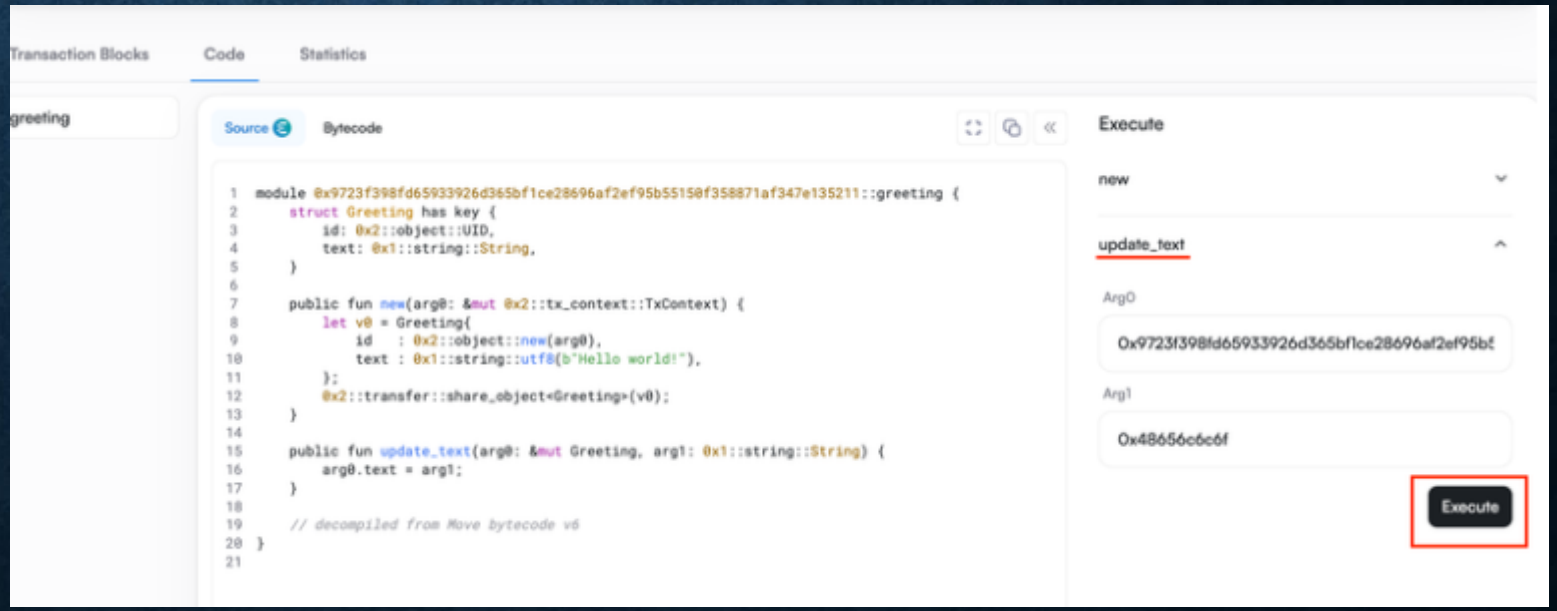


9 共有オブジェクトの確認

update_text関数も動くことも確認しましょう。

また、SuiVisionでは実際の文字列を入れられないので0xaaaaなどを入れます。

16進数であることを示しています。



補足　実行者への所有者の設定

共有ではなく、自分を所有者にしてみましょう。
ctx.sender()が実行者を表します。

```
module empty::greeting {
    use std::string;

    public struct Greeting has key {
    id: UID,
    text: string::String,
    }

    public fun new(ctx: &mut TxContext) {
        let new_greeting = Greeting {
            id: object::new(ctx),
            text: b"Hello world!".to_string()
        };
        transfer::public_transfer(new_greeting, ctx.sender());
    }
    public fun update_text(greeting: &mut Greeting, new_text: string::String) {
        greeting.text = new_text;
    }
}
```

ワークショップ

オリジナルのコントラクトを作りましょう。

チームごとにスマートコントラクトを実際に作ってみましょう。
またAIを使い、ここまでで説明していない内容を盛り込むことも大歓迎です。

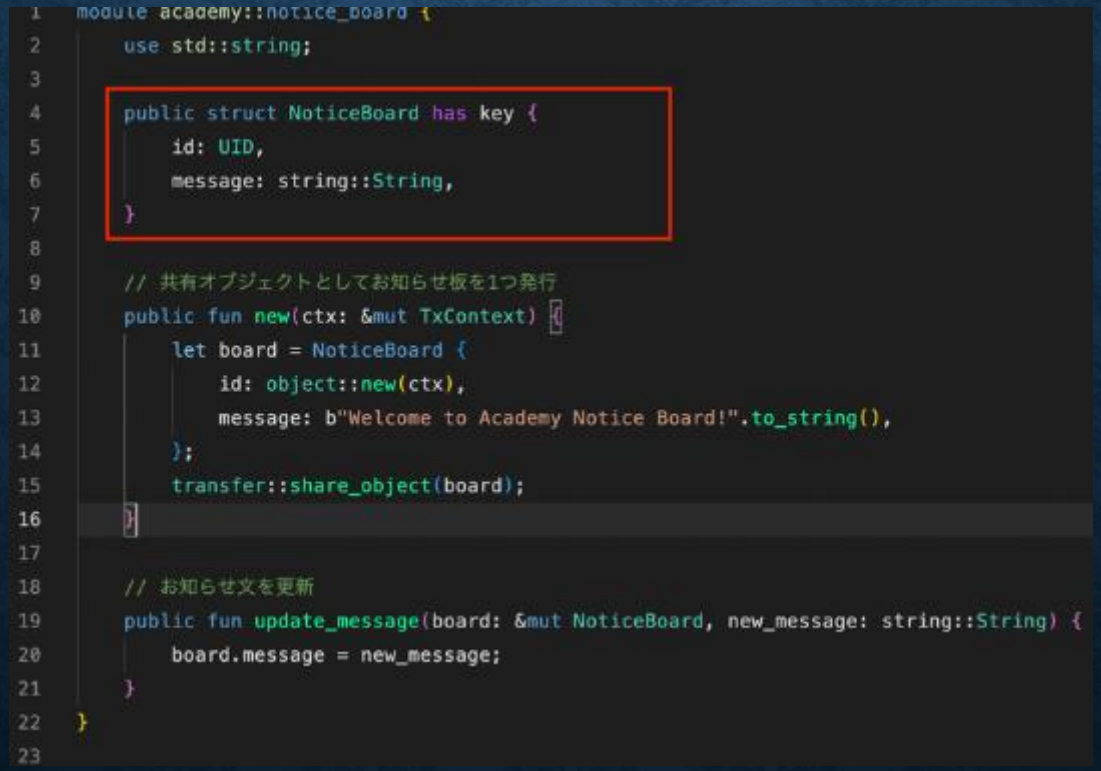
ワーク時間　30分
発表時間　各３～５分

発表内容

- どのようなコントラクトなのかの発表
- コードの説明
- 現在の課題の説明

サンプル①　大学の公式掲示板

常に最新のお知らせのみが表示される大学の公式掲示板です。
これはそのまま利用できるでしょうか。



```
module academy::notice_board {
    use std::string;

    public struct NoticeBoard has key {
        id: UID,
        message: string::String,
    }

    // 共有オブジェクトとしてお知らせ板を1つ発行
    public fun new(ctx: &mut TxContext) {
        let board = NoticeBoard {
            id: object::new(ctx),
            message: b"Welcome to Academy Notice Board!".to_string(),
        };
        transfer::share_object(board);
    }

    // お知らせ文を更新
    public fun update_message(board: &mut NoticeBoard, new_message: string::String) {
        board.message = new_message;
    }
}
```

サンプル②　大学の公式掲示板（＋更新回数）

数字を使う場合は、このように、u64という型を使います。
変化させたい場合は、数式を使う事ができます。

```
module academy::notice_board {
    use std::string;

    public struct NoticeBoard has key {
        id: UID,
        message: string::String,
        update_count: u64,
    }

    public fun new(ctx: &mut TxContext) {
        let board = NoticeBoard {
            id: object::new(ctx),
            message: b"Welcome to our project!".to_string(),
            update_count: 0,
        };
        transfer::share_object(board);
    }

    public fun update_message(board: &mut NoticeBoard, new_message: string::String)
    {
        board.message = new_message;
        board.update_count = board.update_count + 1;
    }
}
```

サンプル③　投票コントラクト

数字を使って、投票を行うためのコントラクトを作ってみました。何か問題はありますでしょうか。

```
module poll::simple_poll {
    use std::string;

    public struct Poll has key {
        id: UID,
        question: string::String,
        yes_count: u64,
        no_count: u64,
    }

    public fun new(ctx: &mut TxContext) {
        let poll = Poll {
            id: object::new(ctx),
            question: b"Should we adopt this new policy?".to_string(),
            yes_count: 0,
            no_count: 0,
        };
        transfer::share_object(poll);
    }

    public fun vote_yes(poll: &mut Poll) {
        poll.yes_count = poll.yes_count + 1;
    }

    public fun vote_no(poll: &mut Poll) {
        poll.no_count = poll.no_count + 1;
    }
}
```

参考 追加実装

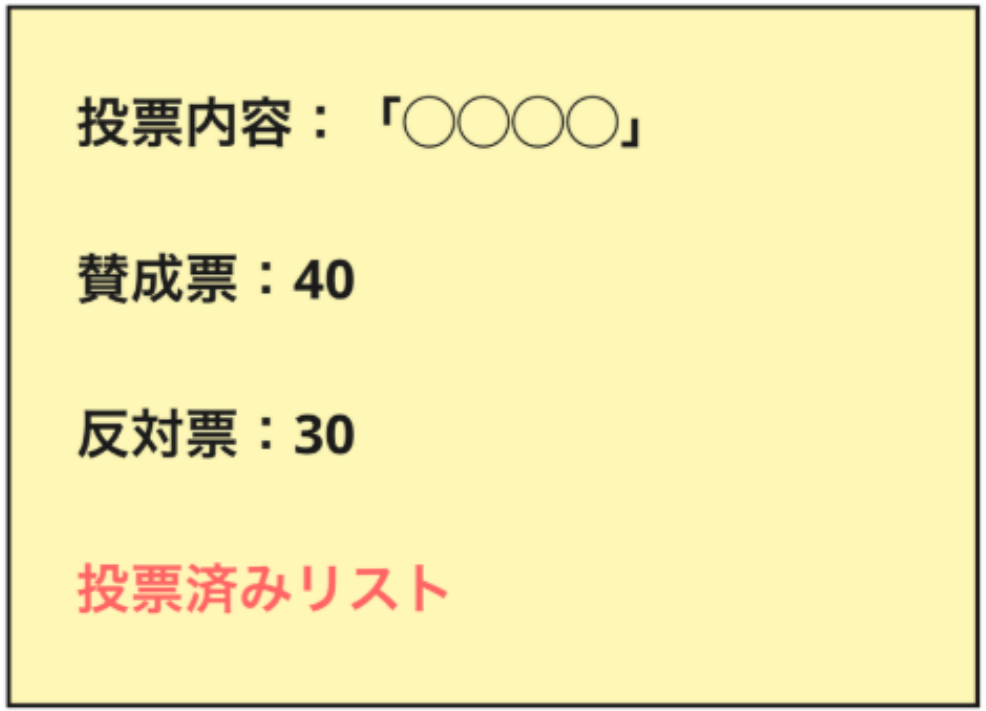
対策①　管理者だけが処理できる設定にしたい

assert!関数を使ってみると条件をつける事ができます。

```
2   use std::string;
3
4   public struct NoticeBoard has key {
5       id: UID,
6       admin: address,
7       message: string::String,
8   }
9
10  // 作成者 = admin
11  public fun new(ctx: &mut TxContext) {
12      let board = NoticeBoard {
13          id: object::new(ctx),
14          admin: ctx.sender(),
15          message: string::utf8(b"Welcome to our project!"),
16      };
17      transfer::share_object(board);
18  }
19
20  // admin だけ更新可能
21  public fun update_message(
22      board: &mut NoticeBoard,
23      new_message: string::String,
24      ctx: &TxContext,
25  ) {
26      assert!(ctx.sender() == board.admin, 1);    assert!(条件, エラーコード)
27      board.message = new_message;
28  }
29
```

対策②　１人一回の処理にしたい

下のように、投票済みリストを作れば良い。



①投票済みリストに登録済みか確認

⇩

②確認処理

１）登録済みの場合：エラー

２）未登録の場合：登録

⇩

③未登録であった場合は投票

対策②　１人一回の処理にしたい

下のように「Table」を使うことで達成する事ができます。

```
module poll::simple_poll {
    use std::string;
    use sui::table::{Self, Table};

    public struct Poll has key {
        id: UID,
        question: string::String,
        yes_count: u64,
        no_count: u64,
        voters: Table<address, bool>, // ここに「投票済みアドレス」を記録
    }

    public fun new(ctx: &mut TxContext) {
        let poll = Poll {
            id: object::new(ctx),
            question: string::utf8(b"Should we adopt this new policy?"),
            yes_count: 0,
            no_count: 0,
            voters: table::new(ctx),
        };
        transfer::share_object(poll);
    }

    public fun vote_yes(poll: &mut Poll, ctx: &TxContext) {
        let sender = ctx.sender();
        assert!(!table::contains(&poll.voters, sender), 1);
        table::add(&mut poll.voters, sender, true);
        poll.yes_count = poll.yes_count + 1;
    }
```

table::contains(リスト、チェックしたい値)
→これを否定したい時は先頭に「!」

table::add(リスト、追加したい値、YesかNo)
→YesかNoは「true」か「false」で表します。

登録済みリスト	内容
Aアドレス	Yes
Bアドレス	Yes
Cアドレス	Yes
Dアドレス	No
Eアドレス	Yes

YesかNoをわざわざ登録する理由はわかりにくいかもしれません。

ただ、これにより、あえてNoにしているDアドレスと、登録がそもそもされていないFアドレスを区別する事ができます。

対策③　許可された者のみの実行としたい

ホワイトリストを作成することで、実行者を絞る事ができます。
逆に、ブラックリストを作ることで明示的に実行を拒否する事ができます。

```
public fun new(ctx: &mut TxContext) {
    let poll = Poll {
        id: object::new(ctx),
        admin: ctx.sender(),
        question: string::utf8(b"Should we adopt this new policy?"),
        yes_count: 0,
        no_count: 0,
        allow: table::new(ctx),
    };
    transfer::share_object(poll);
}
```

管理者のみがホワイトリストを追加できます。

```
/// 管理者だけが許可アドレスを追加
public fun enroll(poll: &mut Poll, who: address, ctx: &TxContext) {
    assert!(ctx.sender() == poll.admin, E_NOT_ADMIN);
    if (!table::contains(&poll.allow, who)) {
        table::add(&mut poll.allow, who, true);
    }
}

public fun vote_yes(poll: &mut Poll, ctx: &TxContext) {
    let sender = ctx.sender();
    assert!(table::contains(&poll.allow, sender), E_NOT_ALLOWED);
    poll.yes_count = poll.yes_count + 1;
}

public fun vote_no(poll: &mut Poll, ctx: &TxContext) {
```

実行前にホワイトリストに格納されていることを確認します

対策④　管理者を別の者に設定したい

@をつけることで直接アドレスを指定する事ができます。

```
module poll3::simple_poll {
    use std::string;
    use sui::table::{Self, Table};

    const E_NOT_ADMIN: u64 = 1;
    const E_NOT_ALLOWED: u64 = 2;
    const WHITELIST_MANAGER: address = @0x0863a92345a2f9f9fc88e659cbcc983b2512425a2f2ffe02edbb5da4e6915cb4;

    public struct Poll has key {
        id: UID,
        admin: address,                 // 管理者（作成者）
        question: string::String,
        yes_count: u64,
        no_count: u64,
        allow: Table<address, bool>,    // 投票を許可するアドレス集合
    }

    public fun new(ctx: &mut TxContext) {
        let poll = Poll {
            id: object::new(ctx),
            admin: WHITELIST_MANAGER,
            question: string::utf8(b"Should we adopt this new policy?"),
            yes_count: 0,
            no_count: 0,
            allow: table::new(ctx),
        };
```

対策⑤　時間を設定したい

時間はタイムスタンプを使用します。
現在時刻はclock::timestamp_ms(clock)を使用します。

```move
module poll4::simple_poll {
    use std::string;
    use sui::clock::{Self, Clock};

    // 2025-12-31 23:59:59.999 JST = 2025-12-31T14:59:59.999Z = 1767193199999 ms
    const DEADLINE_MS: u64 = 1767193199999;

    public struct Poll has key {
        id: UID,
        question: string::String,
        yes_count: u64,
        no_count: u64,
        end_ms: u64,  // 期限 (UTCエポックms)
    }

    // 期限固定で作成
    public fun new_until_2025_12_31_jst(ctx: &mut TxContext) {
        let poll = Poll {
            id: object::new(ctx),
            question: string::utf8(b"Should we adopt this new policy?"),
            yes_count: 0,
            no_count: 0,
            end_ms: DEADLINE_MS,
        };
        transfer::share_object(poll);
    }

    public fun vote_yes(poll: &mut Poll, clock: &Clock) {
        let now = clock::timestamp_ms(clock);
        assert!(now <= poll.end_ms, 1);
        poll.yes_count = poll.yes_count + 1;
    }
```