

Build on Sui
Sui Campus Workshop in
千葉工業大学 // Session #2

Powered by



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、「**KOUKA**」などの**Corda**を利用したサービスを手がけている。



A screenshot of the KOUKA website. The header includes the KOUKA logo and navigation links: KOUKAとは, 導入のメリット, 主な機能, 活用シーン, ユーザーの声, and ご利用開始までの流れ. The main heading is 'ブロックチェーン技術を活用した つながり可視化ツール KOUKA'. Below this is a paragraph explaining that KOUKA uses blockchain technology to build a token economy and visualize employee value, leading to improved motivation and expanded communication. There are two buttons: '資料ダウンロード' (Download Materials) and 'お問合せはこちら' (Contact Us Here). On the right, there is a large illustration showing a network of people connected by lines, with a central figure holding a gift box, symbolizing the interconnectedness and value visualization provided by the tool.

<https://www.cac.co.jp/product/finance-dx/blockchain/>

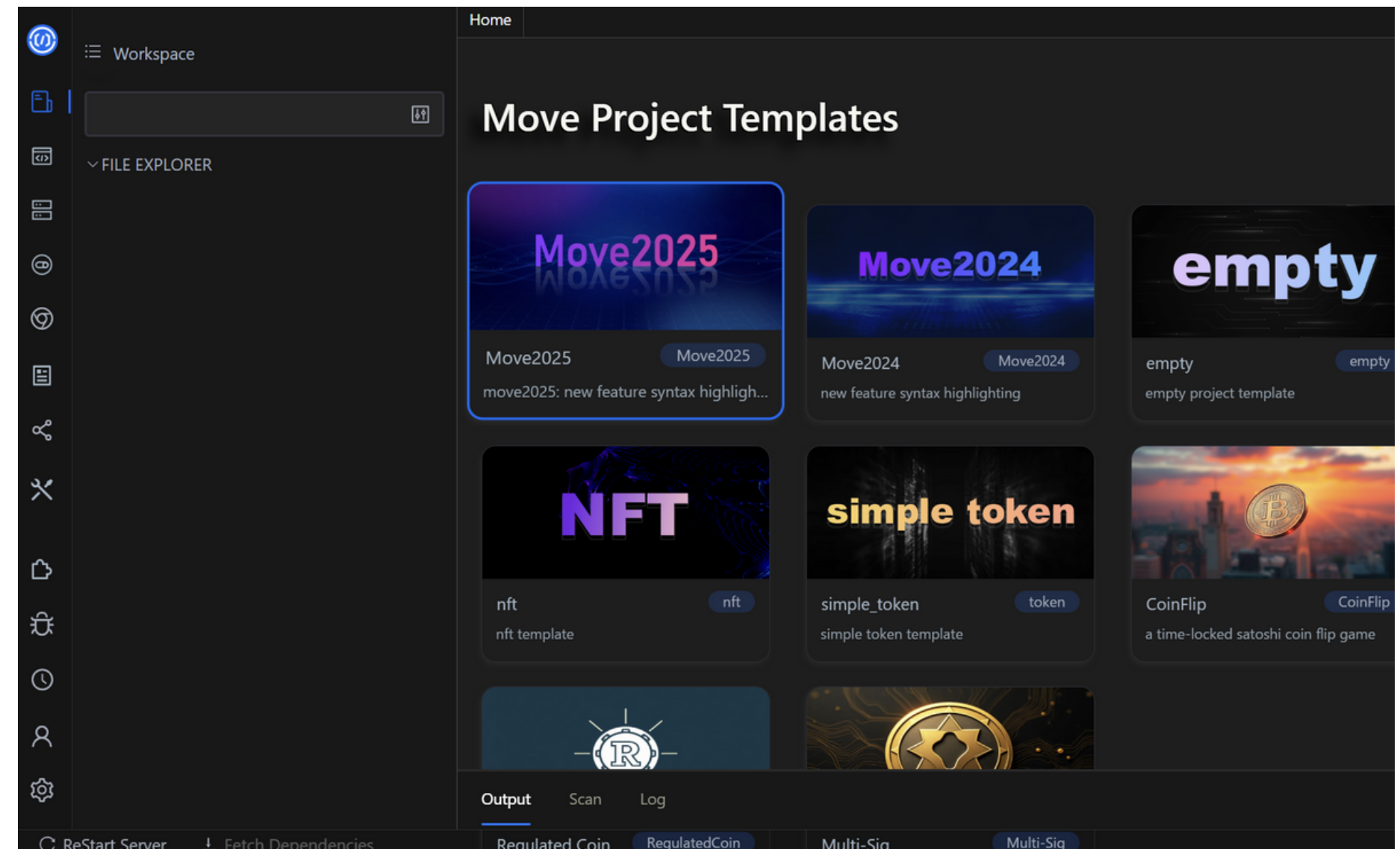
目次

1. 開発体験
2. 実装体験
3. ワークショップ

1 開発体験

1 Web IDEでの開発

Web IDE（Web Integrated Development Environment）を利用することでブラウザ上で開発をすることができます。
今回は BitsLabを利用します。

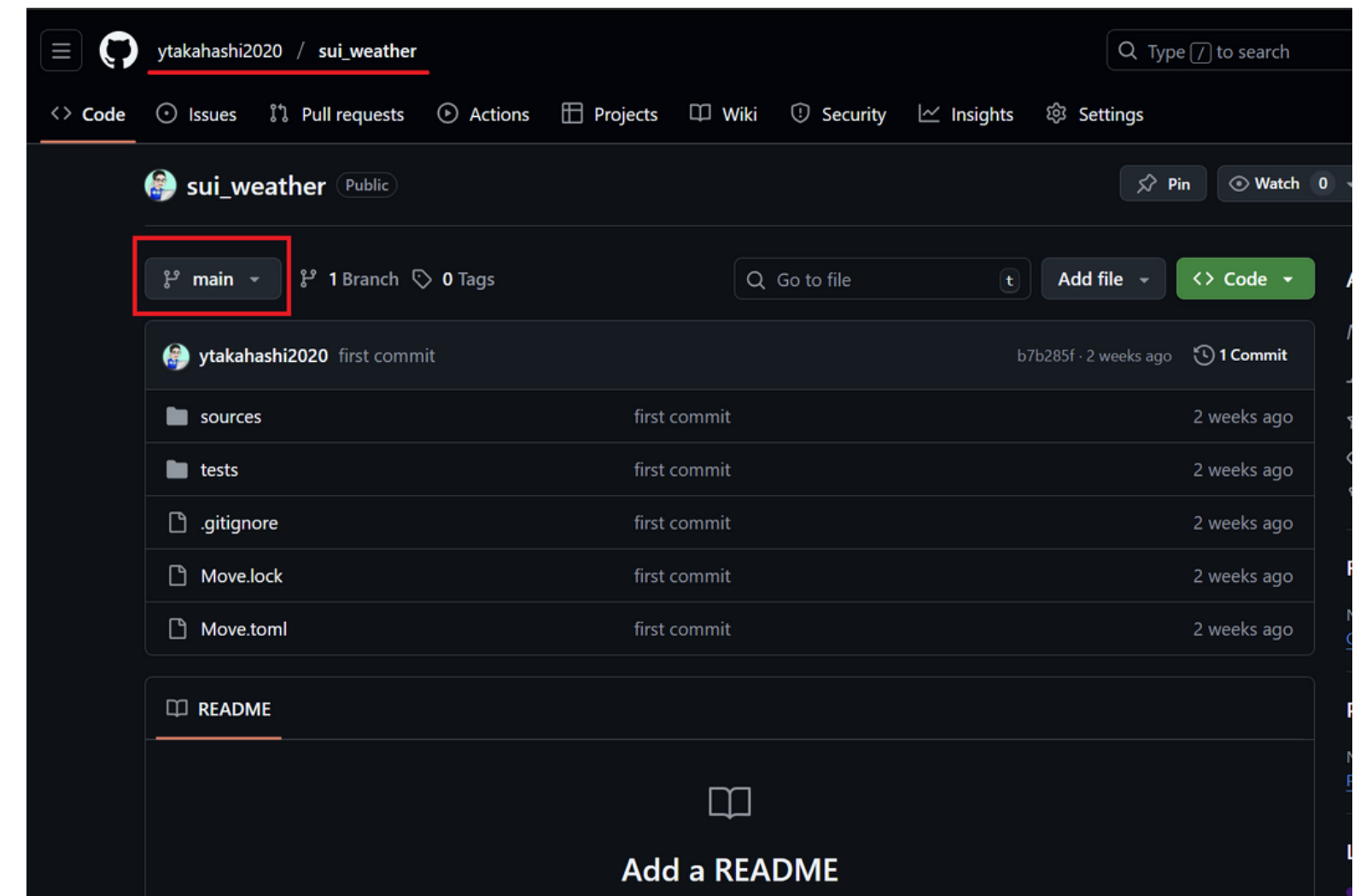


<https://ide.bitslab.xyz/>

2 Githubの利用

GitHubは、プログラムのコードを保存・共有し、複数の人で開発するためのサービスです。

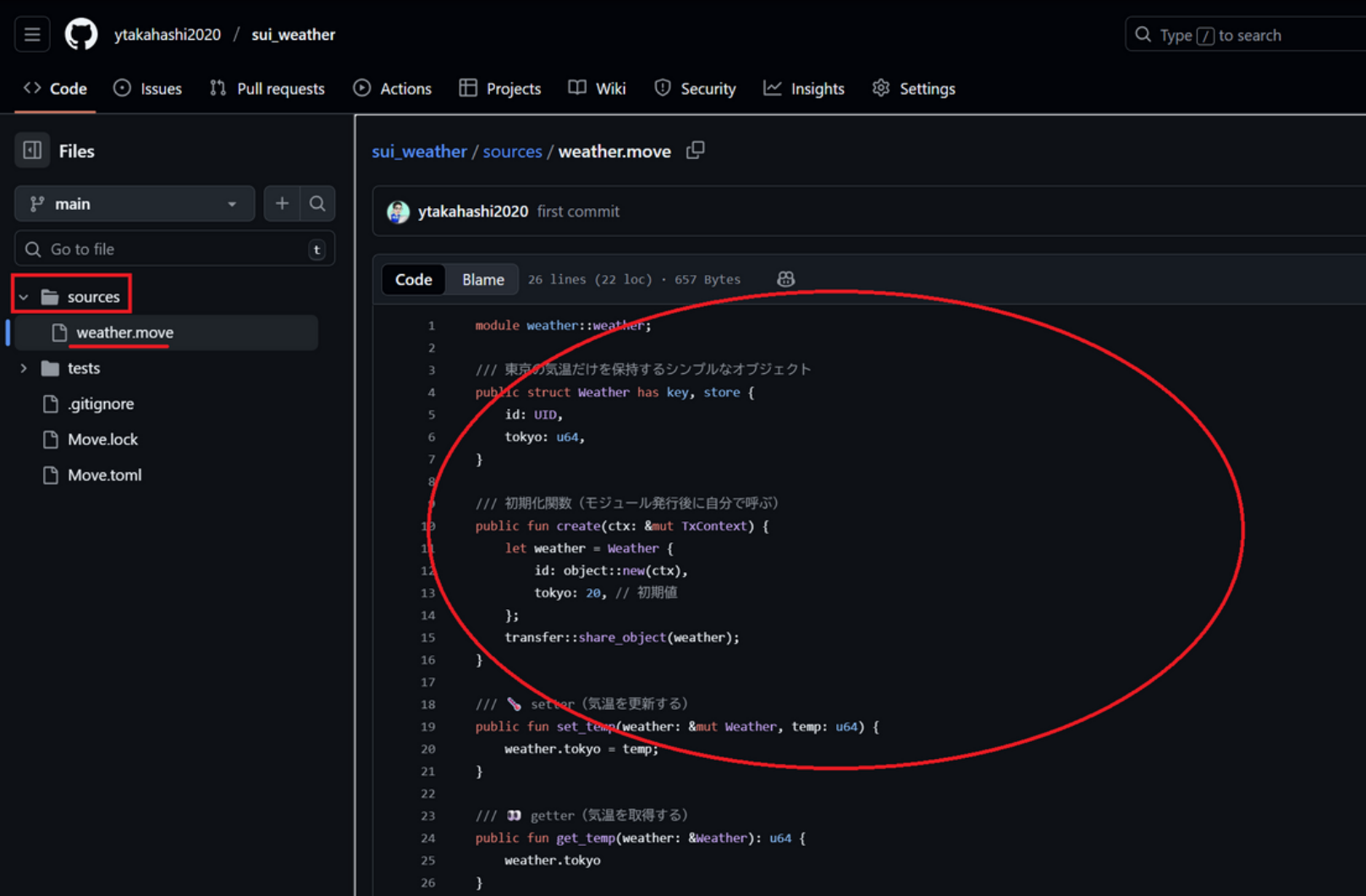
まずはこのコードを使ってみましょう。



https://github.com/ytakahashi2020/sui_weather

2 Githubの利用

ファイルを開いてみましょう。
こちらは、「Move」というプログラミング言語で書かれています。



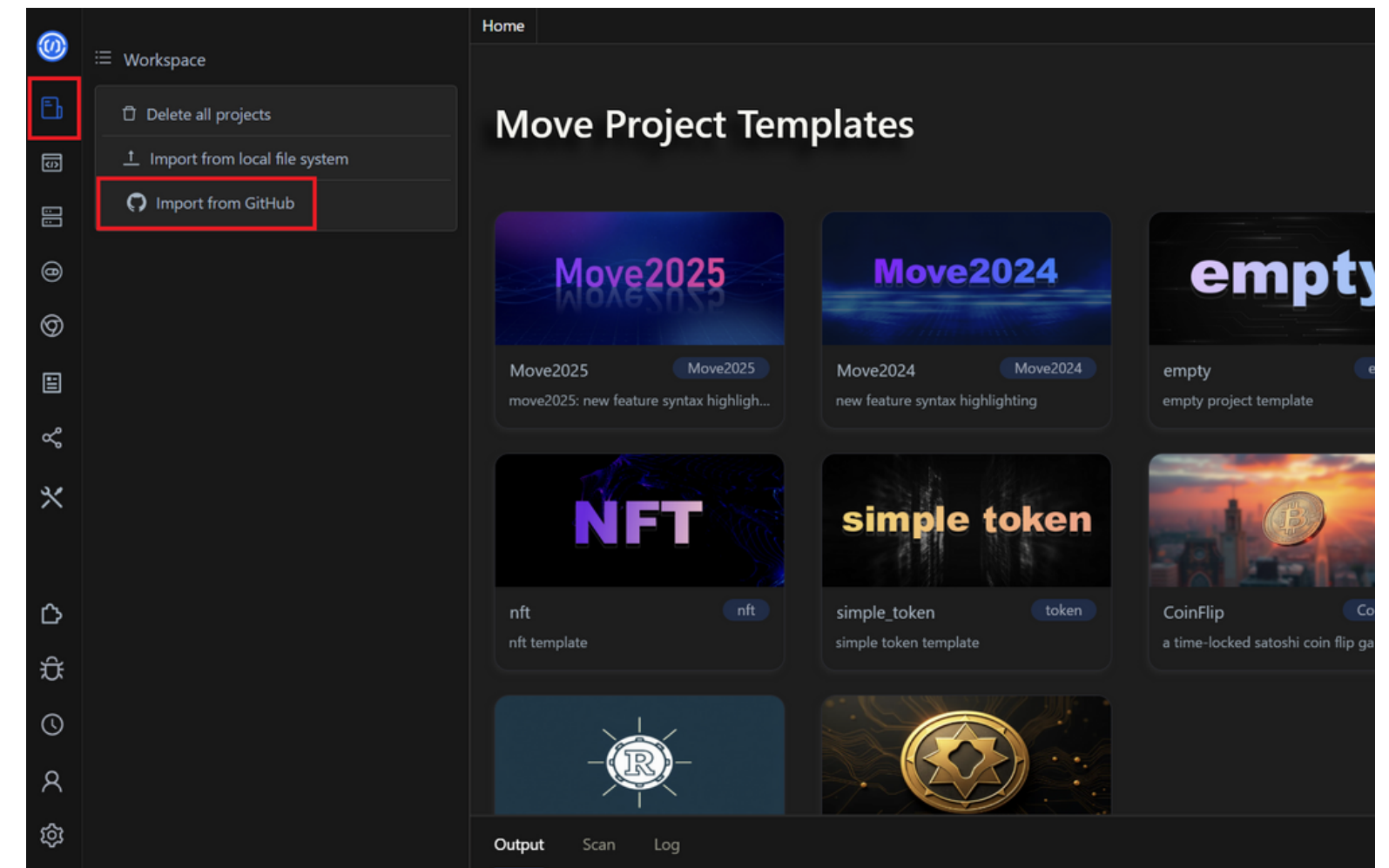
```
1 module weather::weather;
2
3 /// 東京の気温だけを保持するシンプルなオブジェクト
4 public struct Weather has key, store {
5   id: UID,
6   tokyo: u64,
7 }
8
9 /// 初期化関数 (モジュール発行後に自分で呼ぶ)
10 public fun create(ctx: &mut TxContext) {
11   let weather = Weather {
12     id: object::new(ctx),
13     tokyo: 20, // 初期値
14   };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

https://github.com/ytakahashi2020/sui_weather

3 Githubからのインポート

GitHubにあるコードやプロジェクトを、自分の開発環境などに取り込むことをインポートと言います。

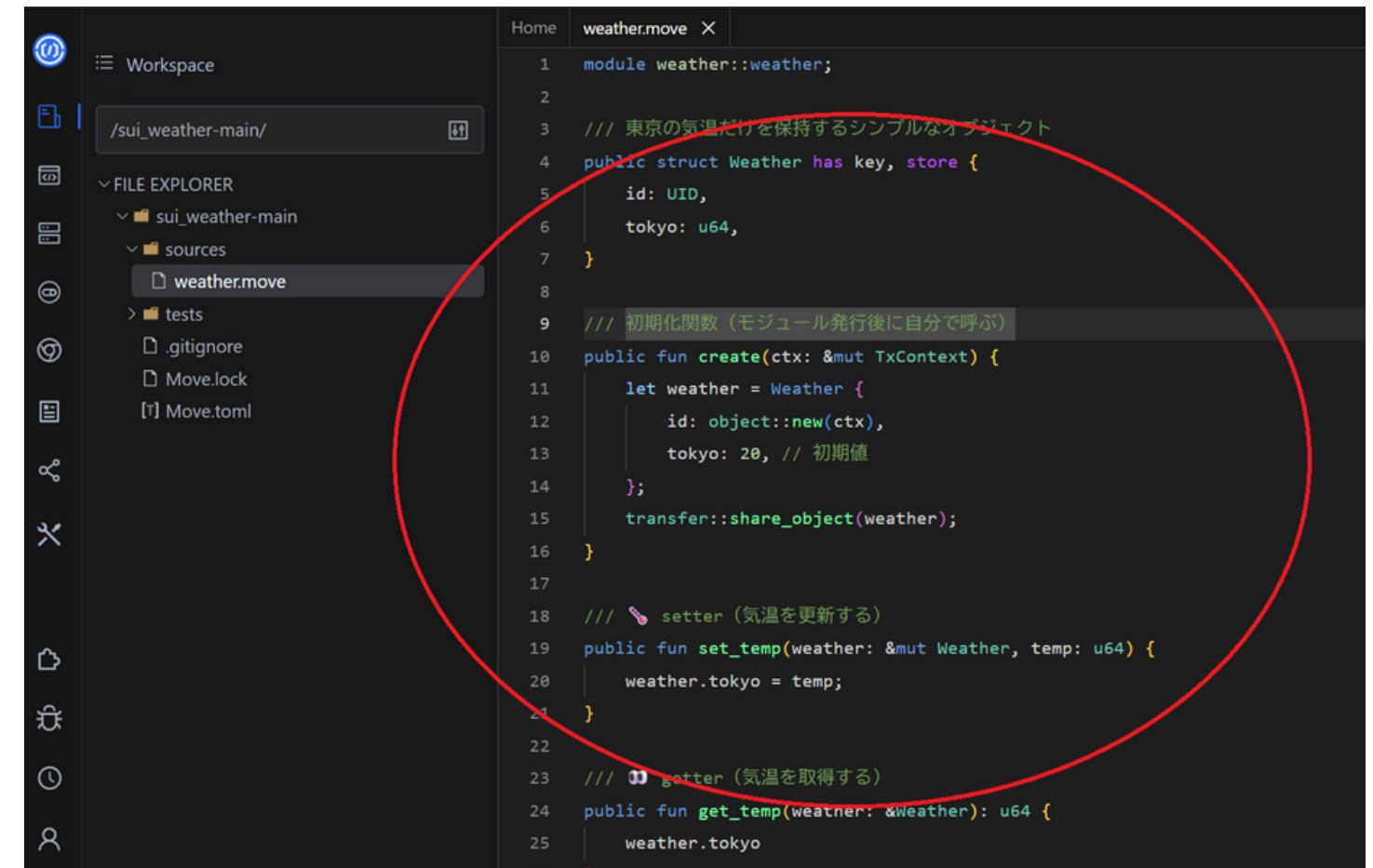
先ほどのコードをインポートしてみましょう。



<https://ide.bitslab.xyz/>

3 Githubからのインポート

「https://github.com/ytakahashi2020/sui_weather」を設定します。
下のように、インポートを行うことができました。

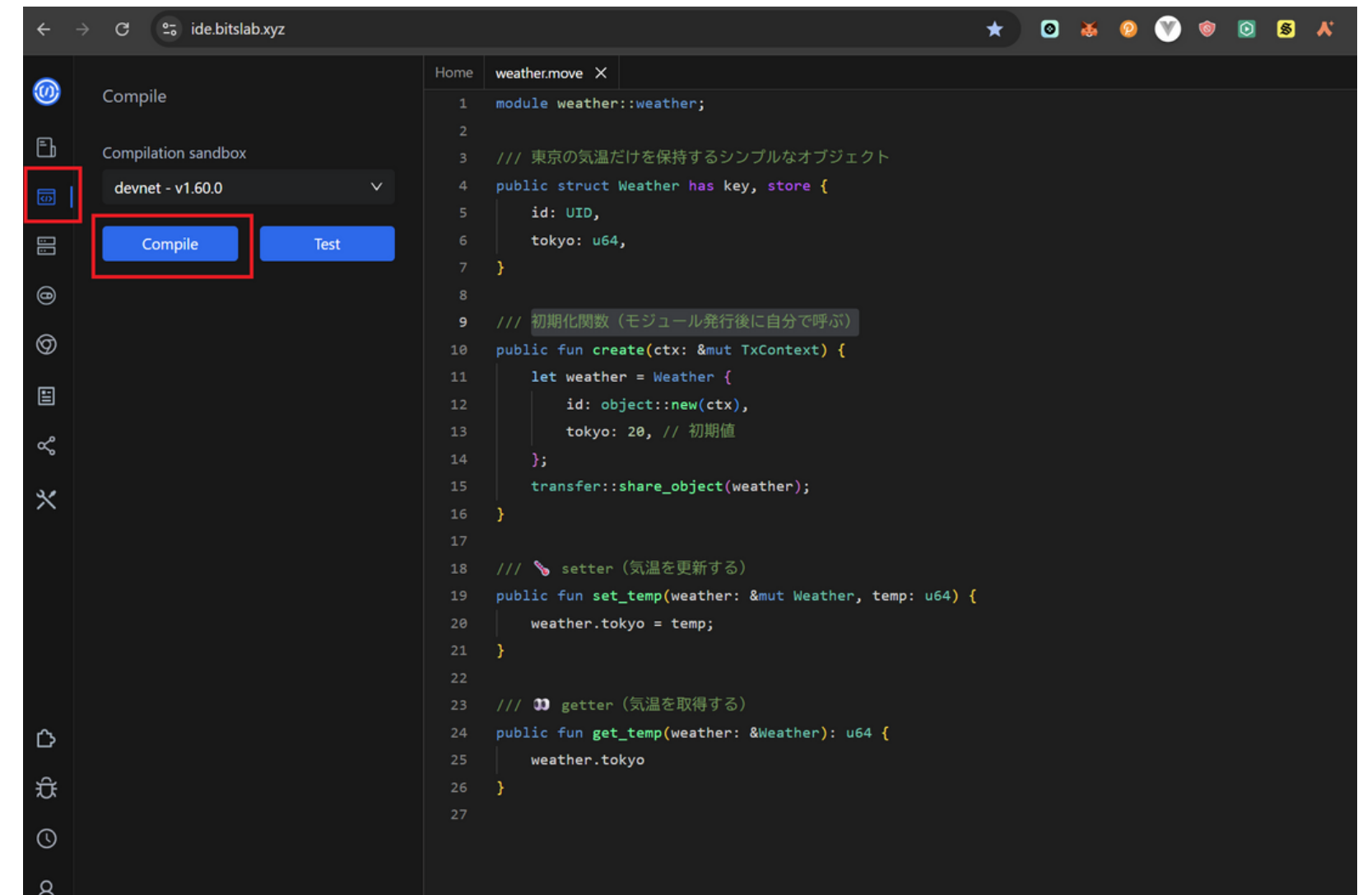


```
1 module weather::weather;
2
3 /// 東京の気温だけを保持するシンプルなオブジェクト
4 public struct Weather has key, store {
5   id: UID,
6   tokyo: u64,
7 }
8
9 /// 初期化関数 (モジュール発行後に自分で呼ぶ)
10 public fun create(ctx: &mut TxContext) {
11   let weather = Weather {
12     id: object::new(ctx),
13     tokyo: 20, // 初期値
14   };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

<https://ide.bitslab.xyz/>

4 コンパイルの実行

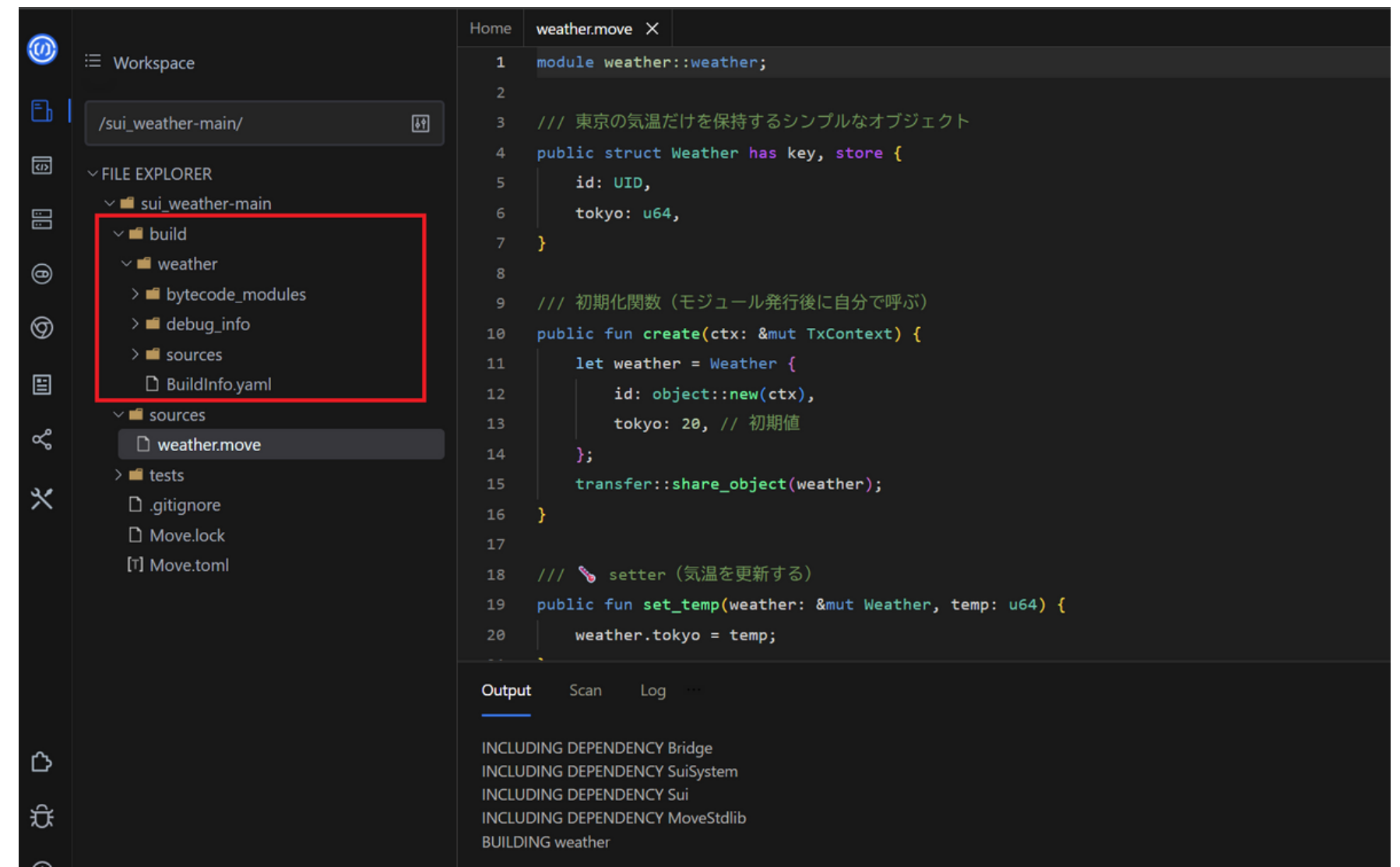
人間が読める形式から機械が読める形式への変換であるコンパイルを行います。



<https://ide.bitslab.xyz/>

4 コンパイルの実行

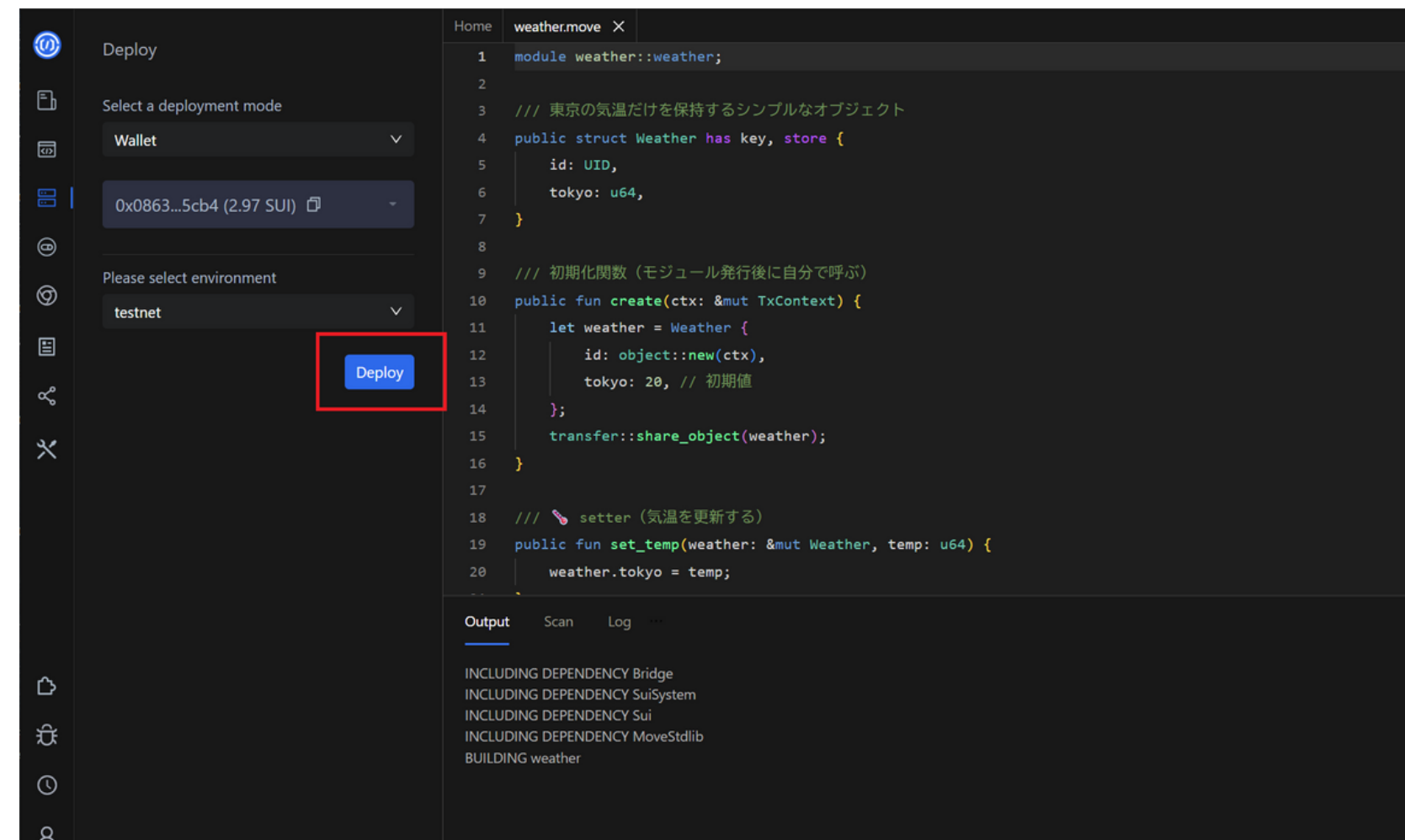
このように「build」というフォルダが作られました。



<https://ide.bitslab.xyz/>

5 デプロイの実行

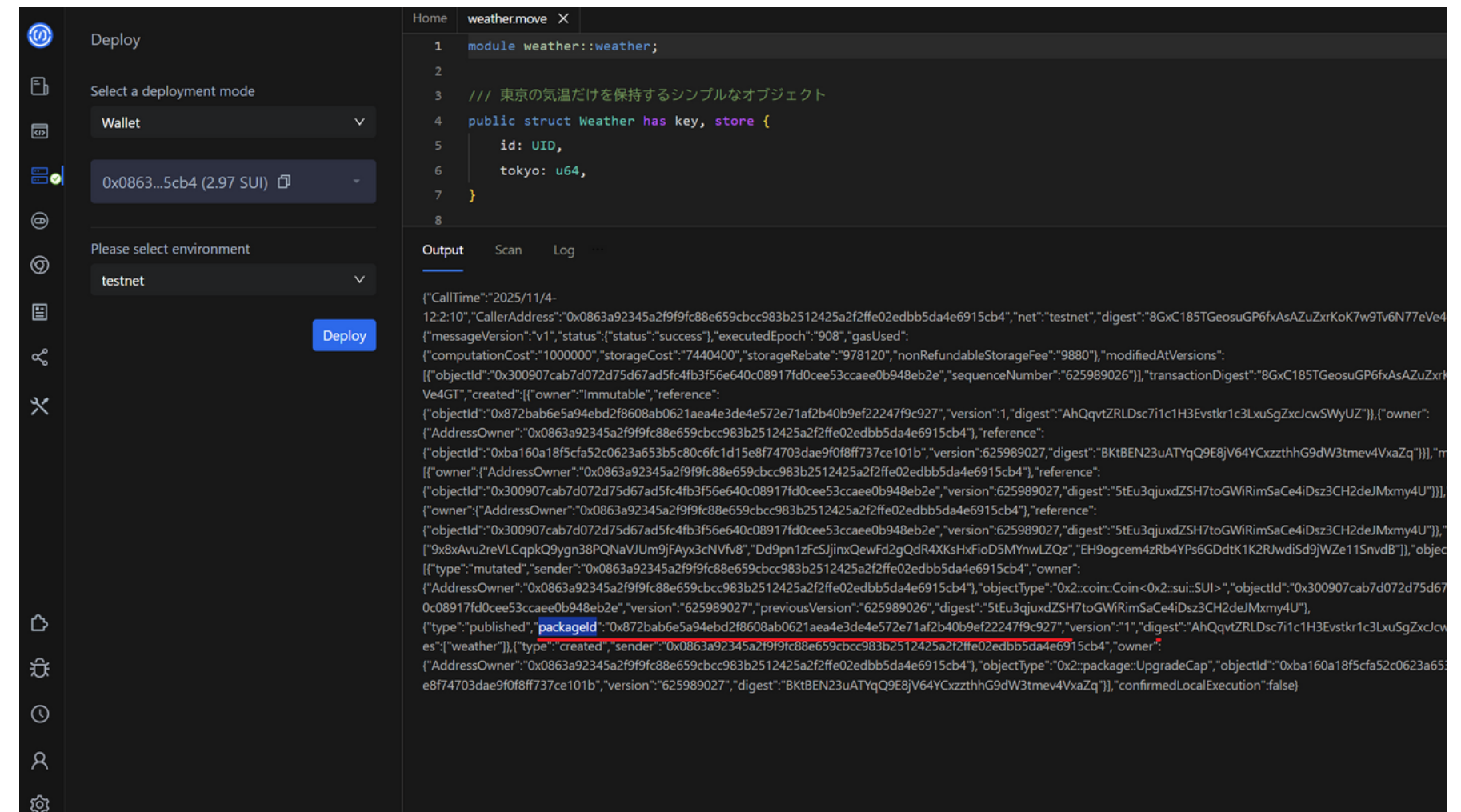
ブロックチェーン上に実装する処理である「デプロイ」を行います。



<https://ide.bitslab.xyz/>

6 パッケージIDの取得

Suiのスマートコントラクト（モジュール）を包括する単位であるパッケージのIDを取得します。

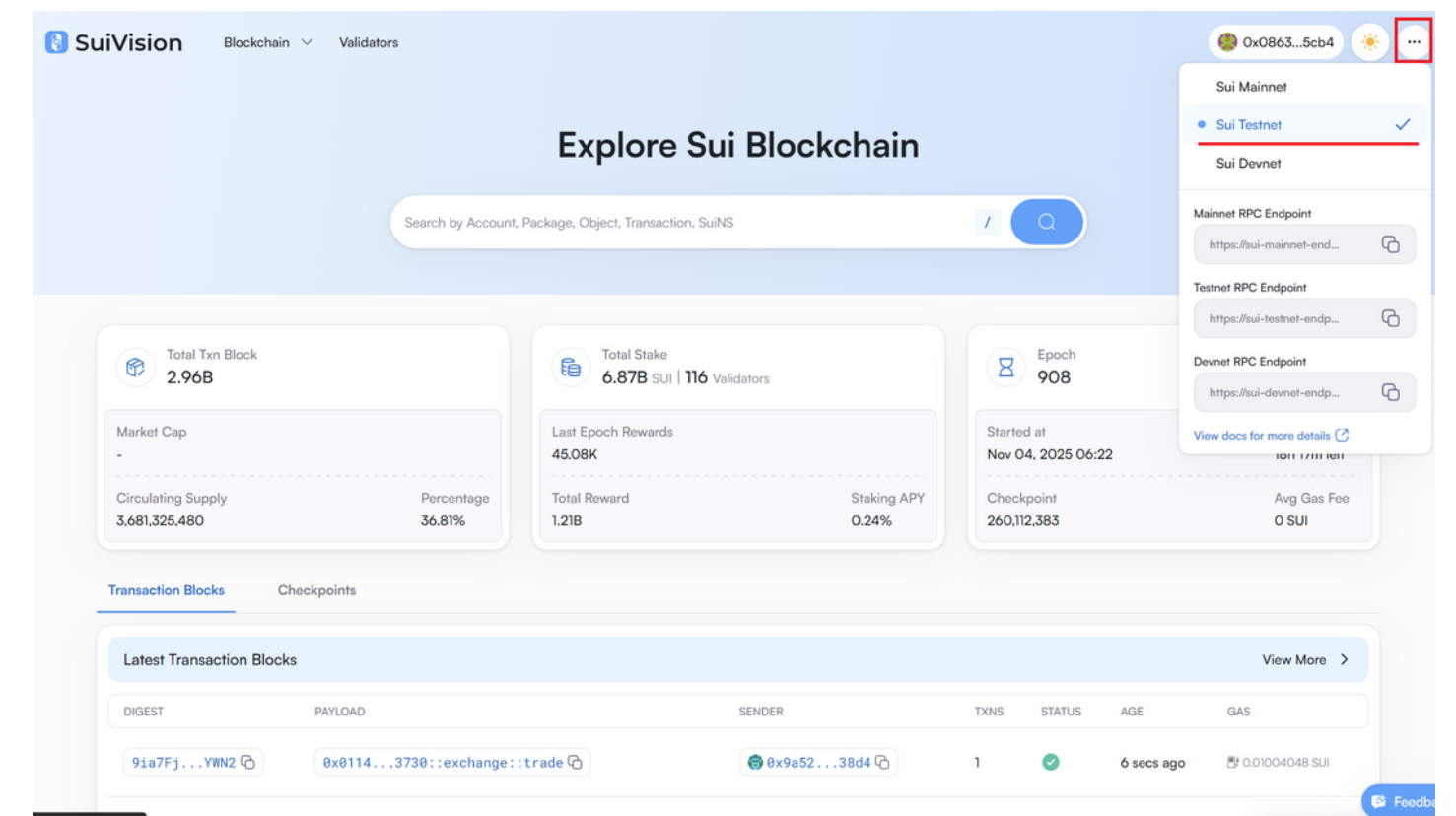


<https://ide.bitslab.xyz/>

7 エクスプローラの確認

ブロックチェーンの状況を把握することができる「エクスプローラ」を確認します。

今回は「SuiVision」というエクスプローラを使用します。

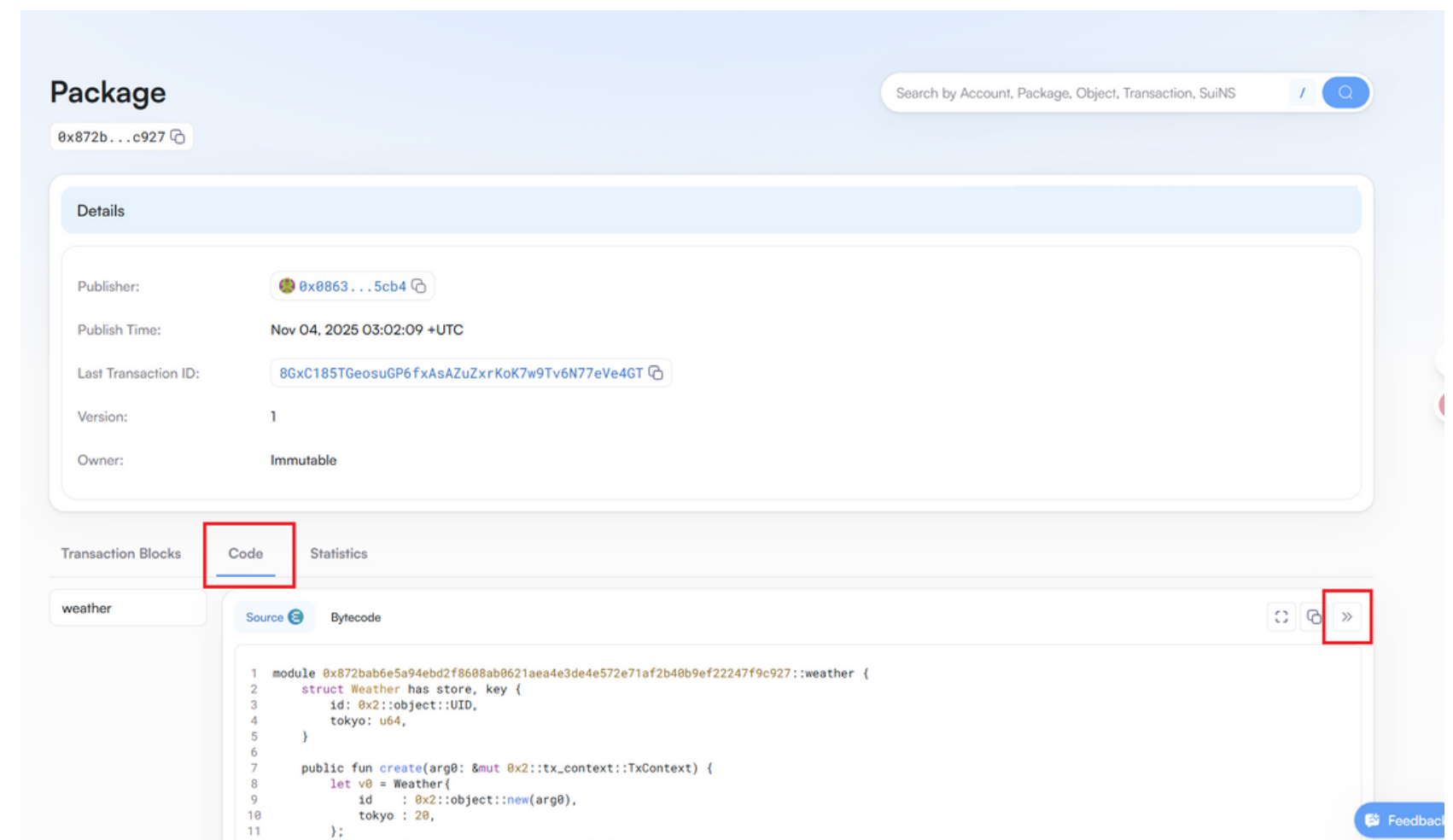


<https://testnet.suivision.xyz/>

7 エクスプローラの確認

検索すると、先ほどデプロイしたスマートコントラクトが確認できます。

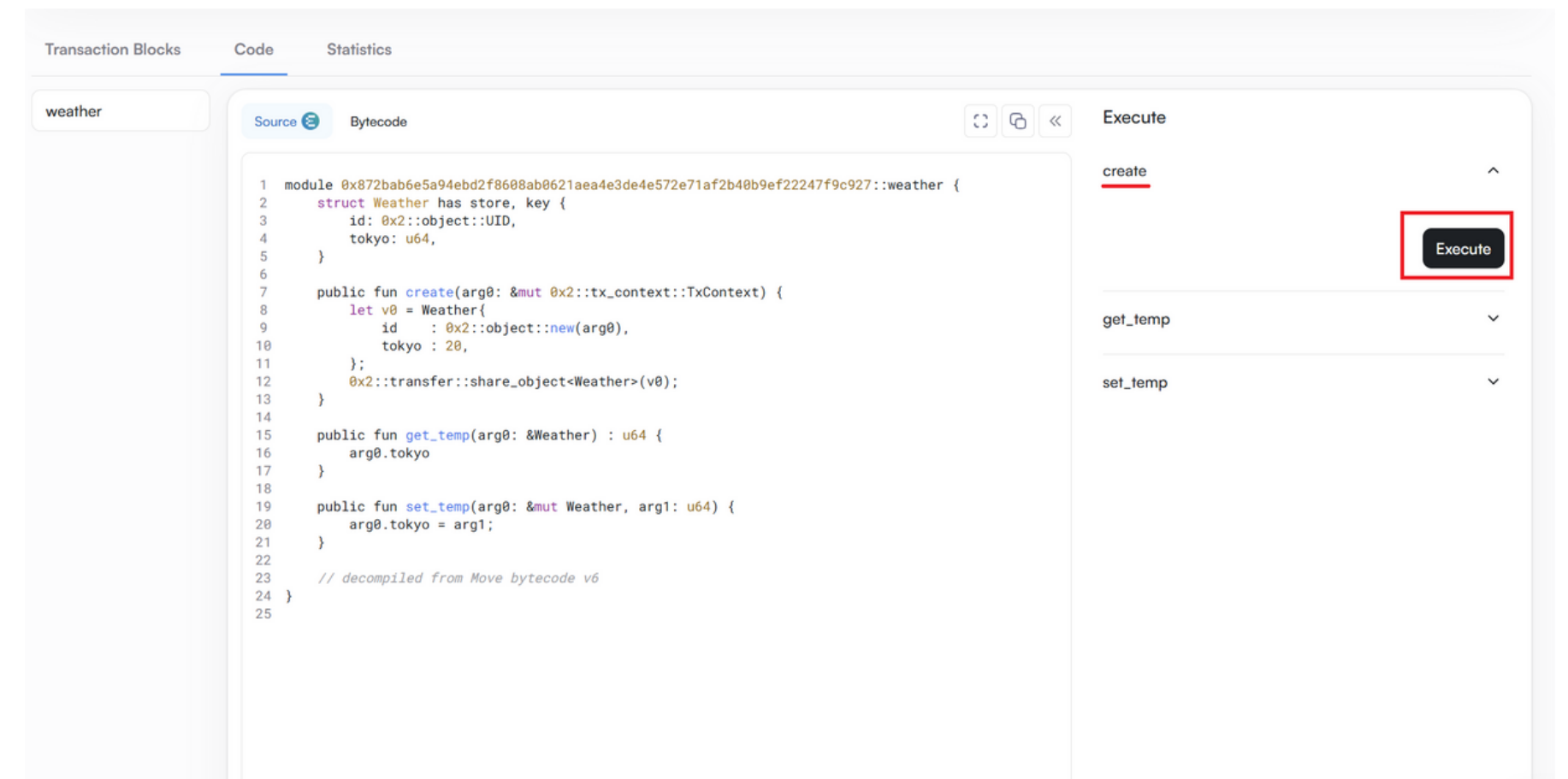
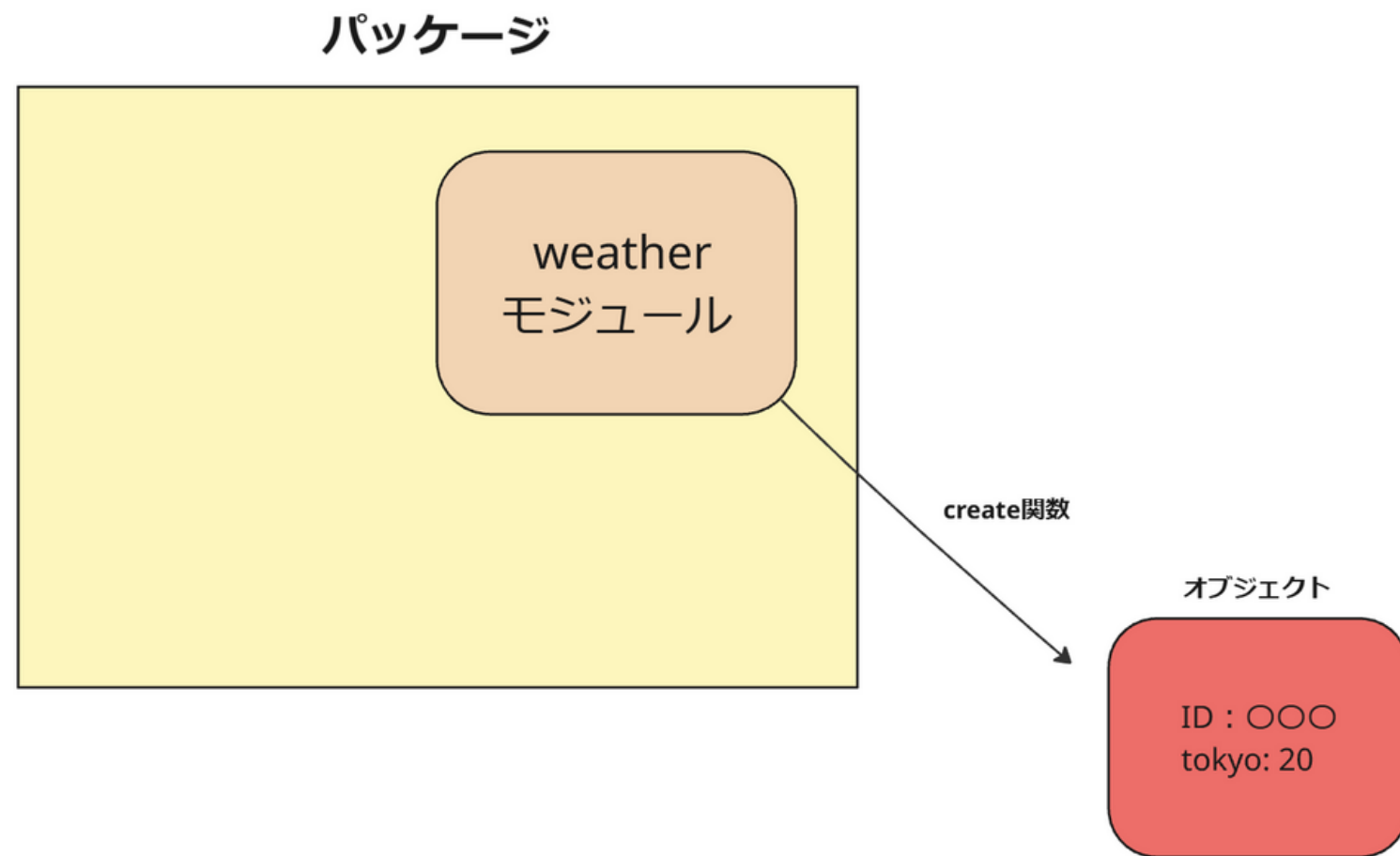
コードも確認ができます。



<https://testnet.suivision.xyz/>

8 関数の実行

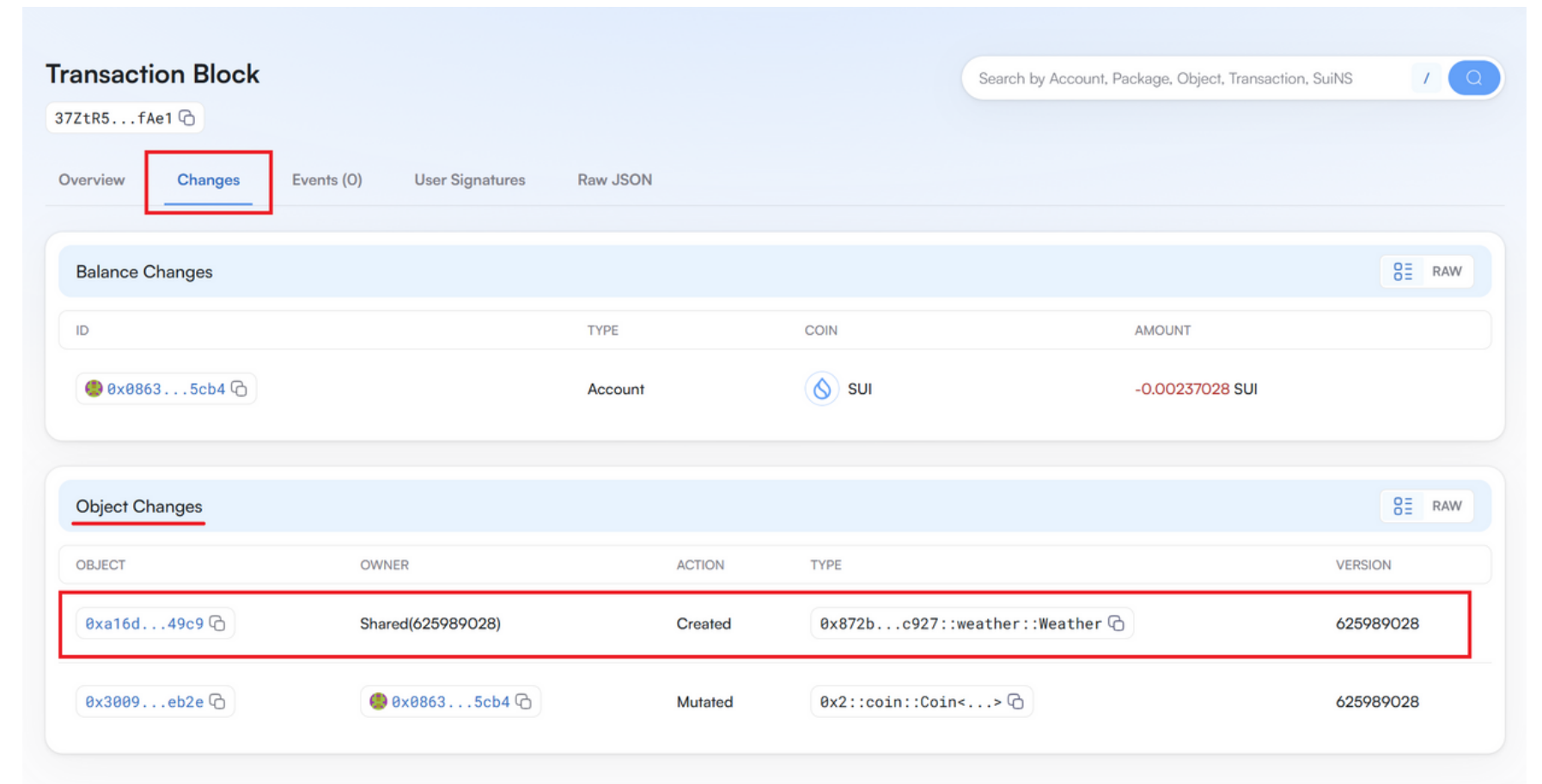
「Create」という関数を実行し、オブジェクトを作成しましょう。



<https://testnet.suivision.xyz/>

9 トランザクションの確認

「Changes」からオブジェクトの変更を確認できます。
今回は新しく作られた「Created」を確認します。



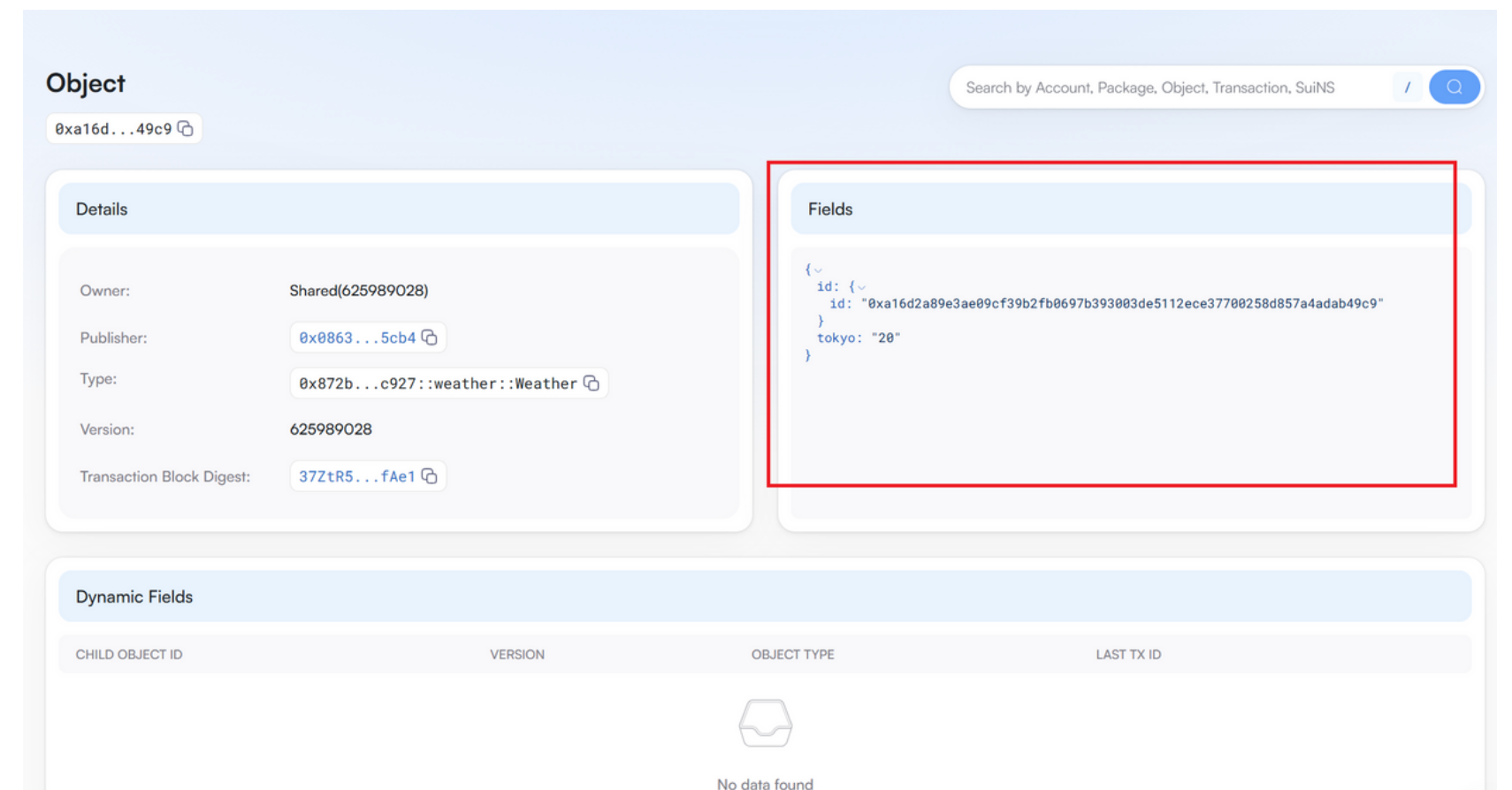
The screenshot shows the 'Transaction Block' page for transaction ID 37ZtR5...fAe1. The 'Changes' tab is selected and highlighted with a red box. Below the tabs, there are two sections: 'Balance Changes' and 'Object Changes'. The 'Object Changes' section contains a table with the following data:

OBJECT	OWNER	ACTION	TYPE	VERSION
0xa16d...49c9	Shared(625989028)	Created	0x872b...c927::weather::Weather	625989028
0x3009...eb2e	0x0863...5cb4	Mutated	0x2::coin::Coin<...>	625989028

<https://testnet.suivision.xyz/>

9 トランザクションの確認

下のように「Fields」を確認すると、IDとtokyoの値が格納されていることがわかります。



The screenshot displays the Suivision testnet interface. At the top, there's a search bar with the text "Search by Account, Package, Object, Transaction, SuiNS". Below this, the "Object" section is highlighted, showing the object ID "0xa16d...49c9". The "Details" tab is active, displaying the following information:

- Owner: Shared(625989028)
- Publisher: 0x0863...5cb4
- Type: 0x872b...c927::weather::Weather
- Version: 625989028
- Transaction Block Digest: 37ZtR5...fAe1

The "Fields" tab is also visible, showing the following JSON data:

```
{
  id: {
    id: "0xa16d2a89e3ae09cf39b2fb0697b393083de5112ece37708258d857a4adab49c9"
  },
  tokyo: "20"
}
```

Below the "Fields" tab, the "Dynamic Fields" section is visible, showing a table with columns: CHILD OBJECT ID, VERSION, OBJECT TYPE, and LAST TX ID. The table is currently empty, with a message "No data found" at the bottom.

<https://testnet.suivision.xyz/>

10 引数のある関数の実行

関数に渡す値を引数と言います。

set_temp関数で値を変更してみましょう。

第一引数がオブジェクト、第二引数が数値です。

The screenshot displays the Testnet Suivision interface, which is used for executing smart contracts on a blockchain network. The interface is divided into several sections:

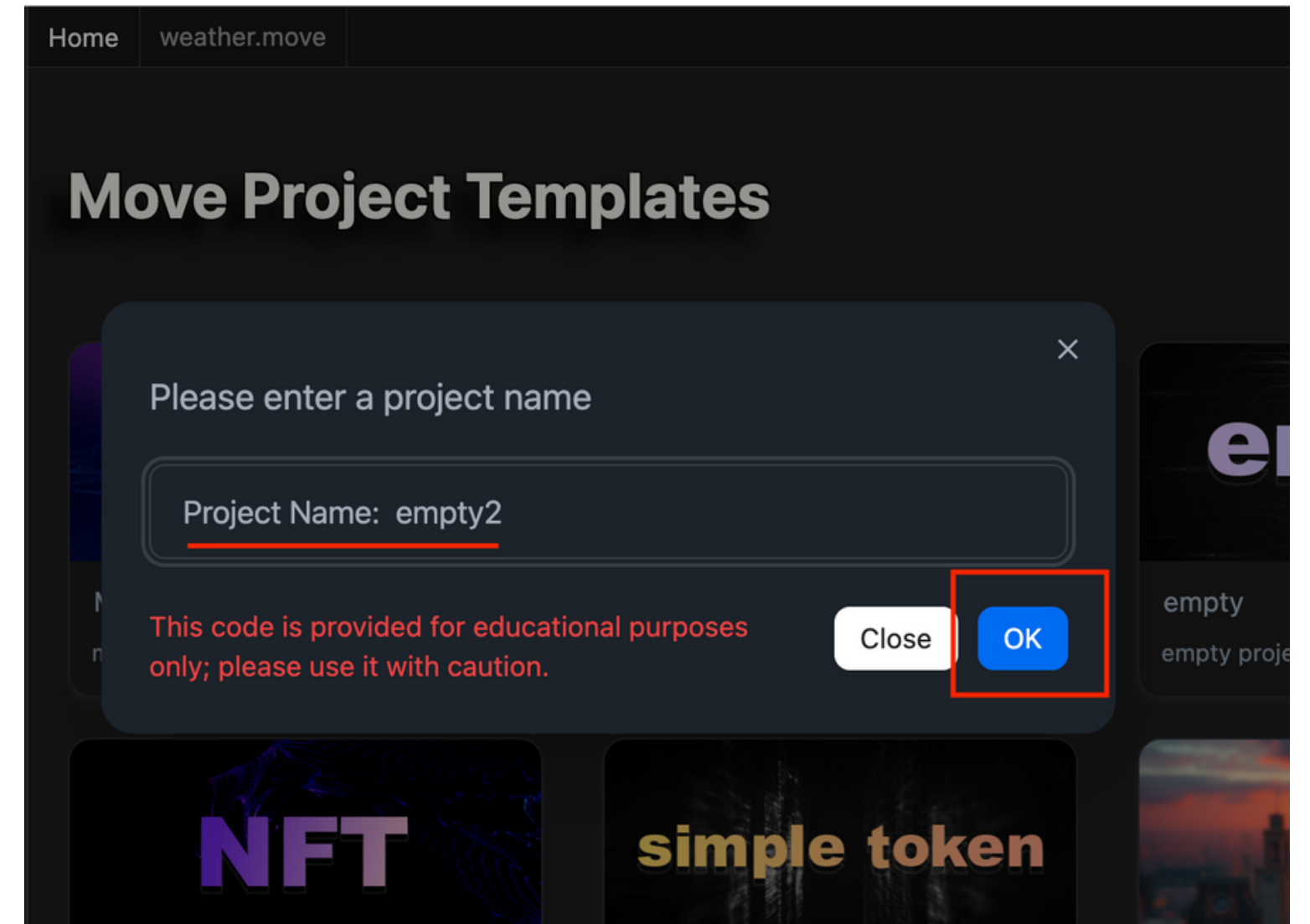
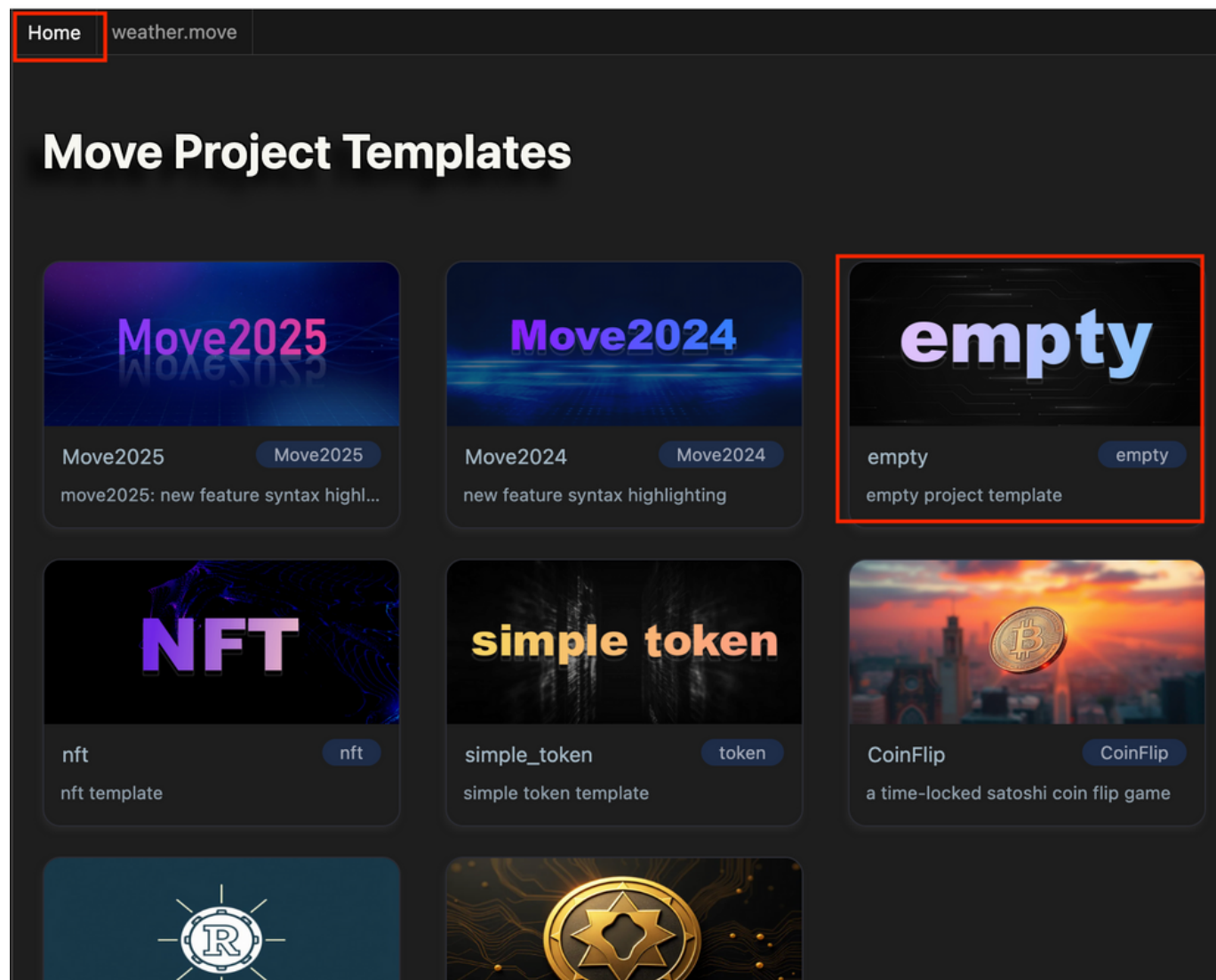
- Transaction Blocks:** A sidebar on the left showing a list of transactions, with 'weather' selected.
- Code:** The main area showing the source code of the 'weather' contract. The code is written in a language that resembles Solidity or a similar smart contract language. A red box highlights the `set_temp` function definition: `public fun set_temp(arg0: &mut Weather, arg1: u64) { arg0.tokyo = arg1; }`.
- Statistics:** A tab on the right side of the code editor.
- Execute:** A section on the right side of the interface where the function can be executed. It shows the function name 'set_temp' and its arguments: 'Arg0' (a hexadecimal address) and 'Arg1' (the value '15'). A red box highlights the 'Arg1' input field and the 'Execute' button.

<https://testnet.suivision.xyz/>

2 実装体験

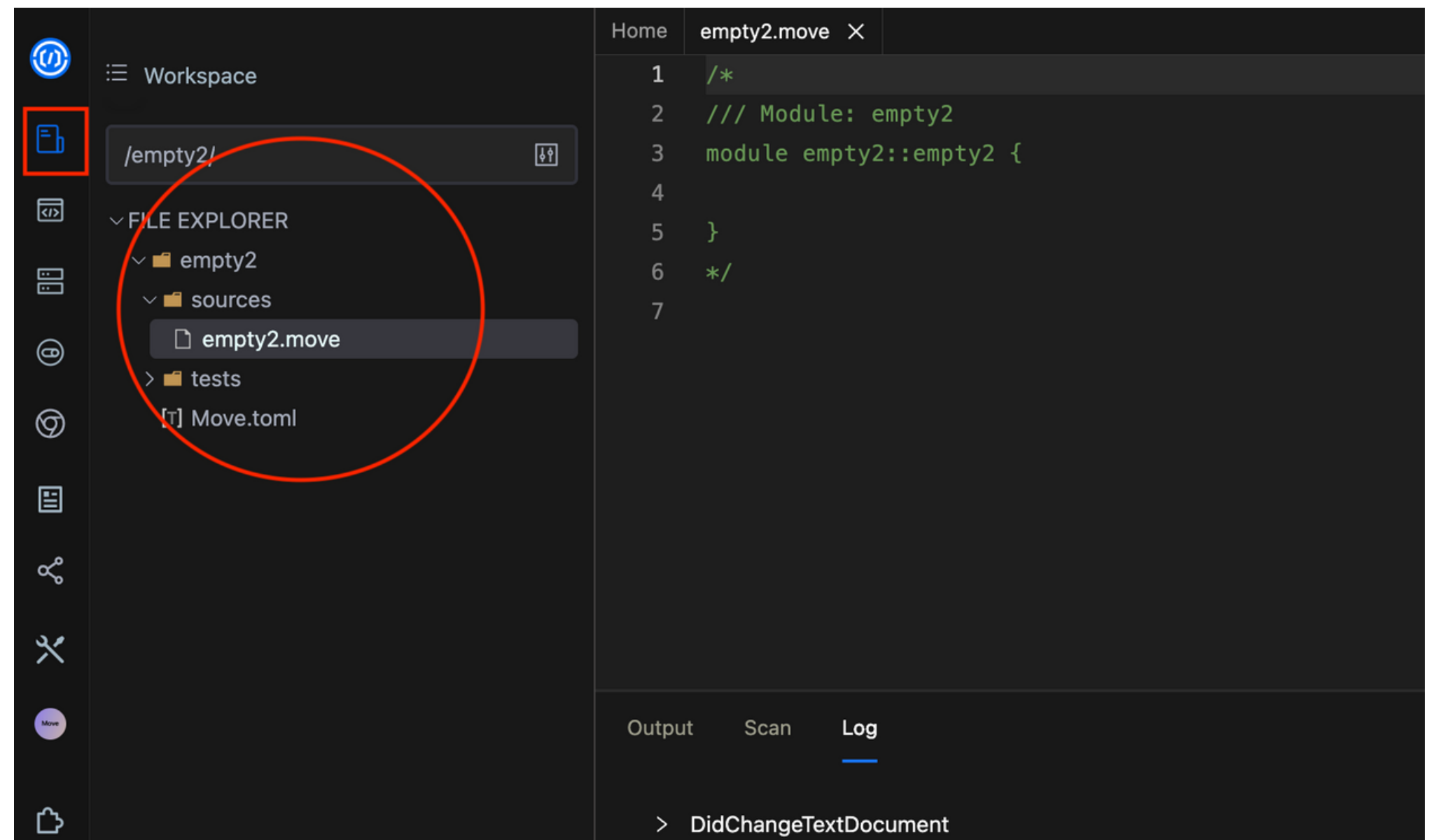
1 プロジェクトの作成

では、1からプロジェクトを作りましょう。
Homeから「empty」を選択します。



1 プロジェクトの作成

このように空のプロジェクトの雛形ができました。

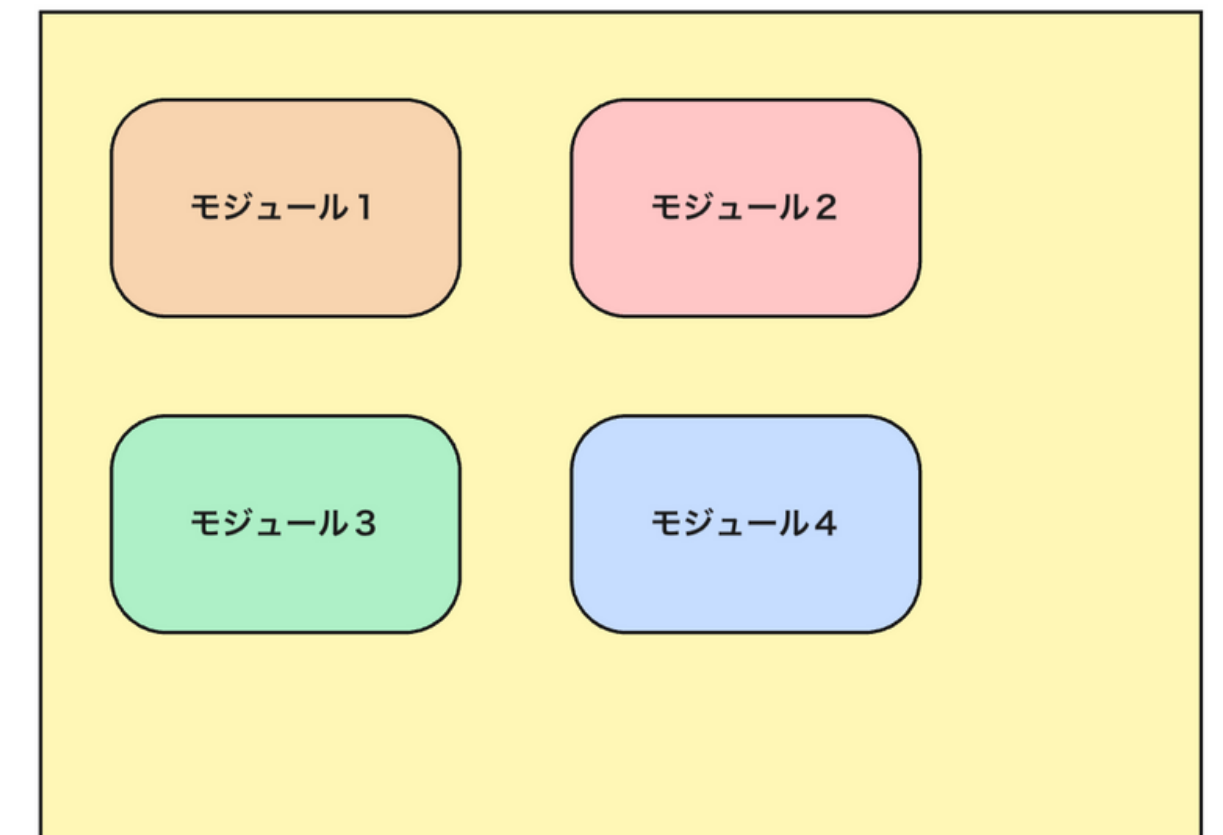


2 パッケージとモジュール

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。

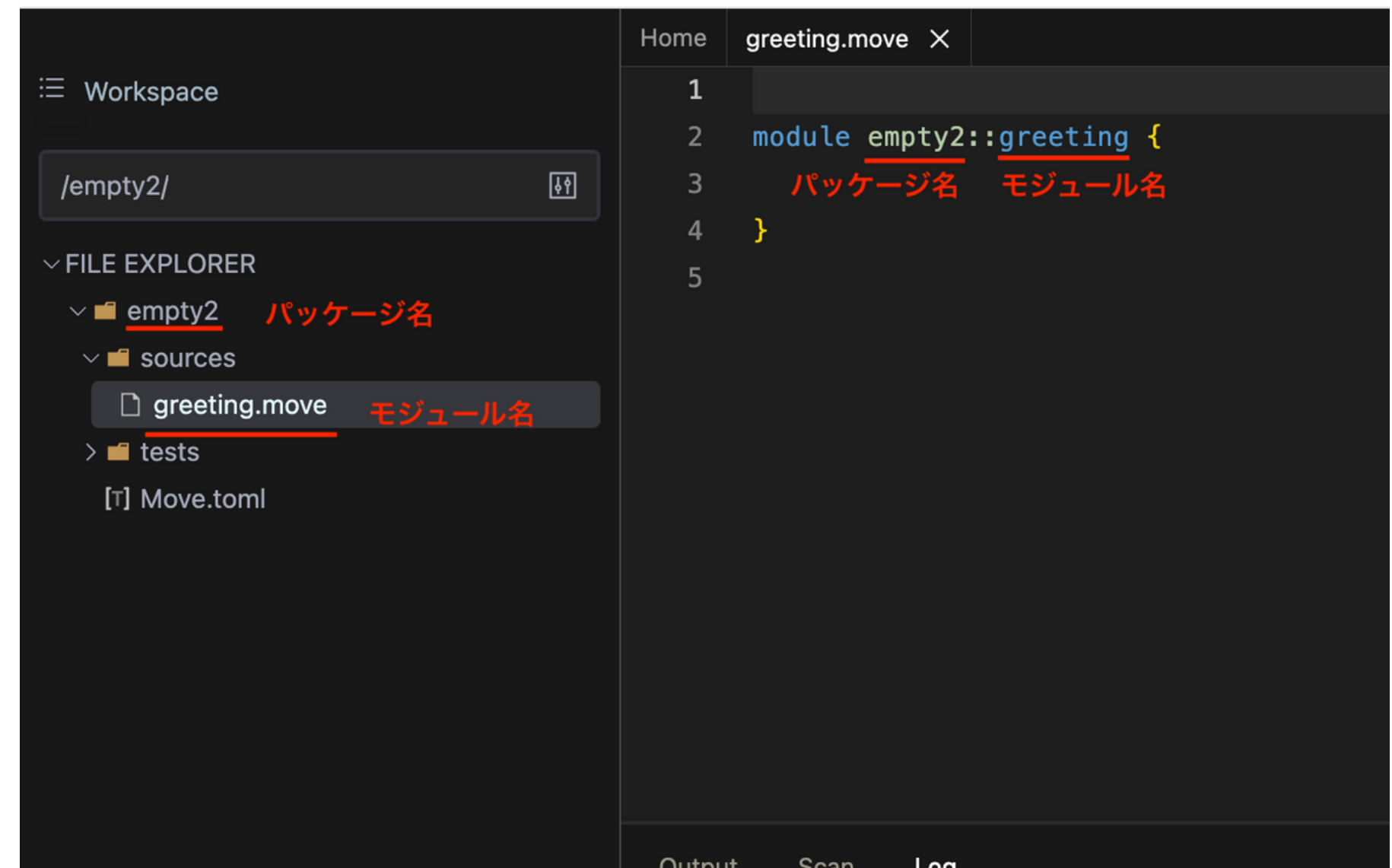
パッケージ



2 パッケージとモジュール

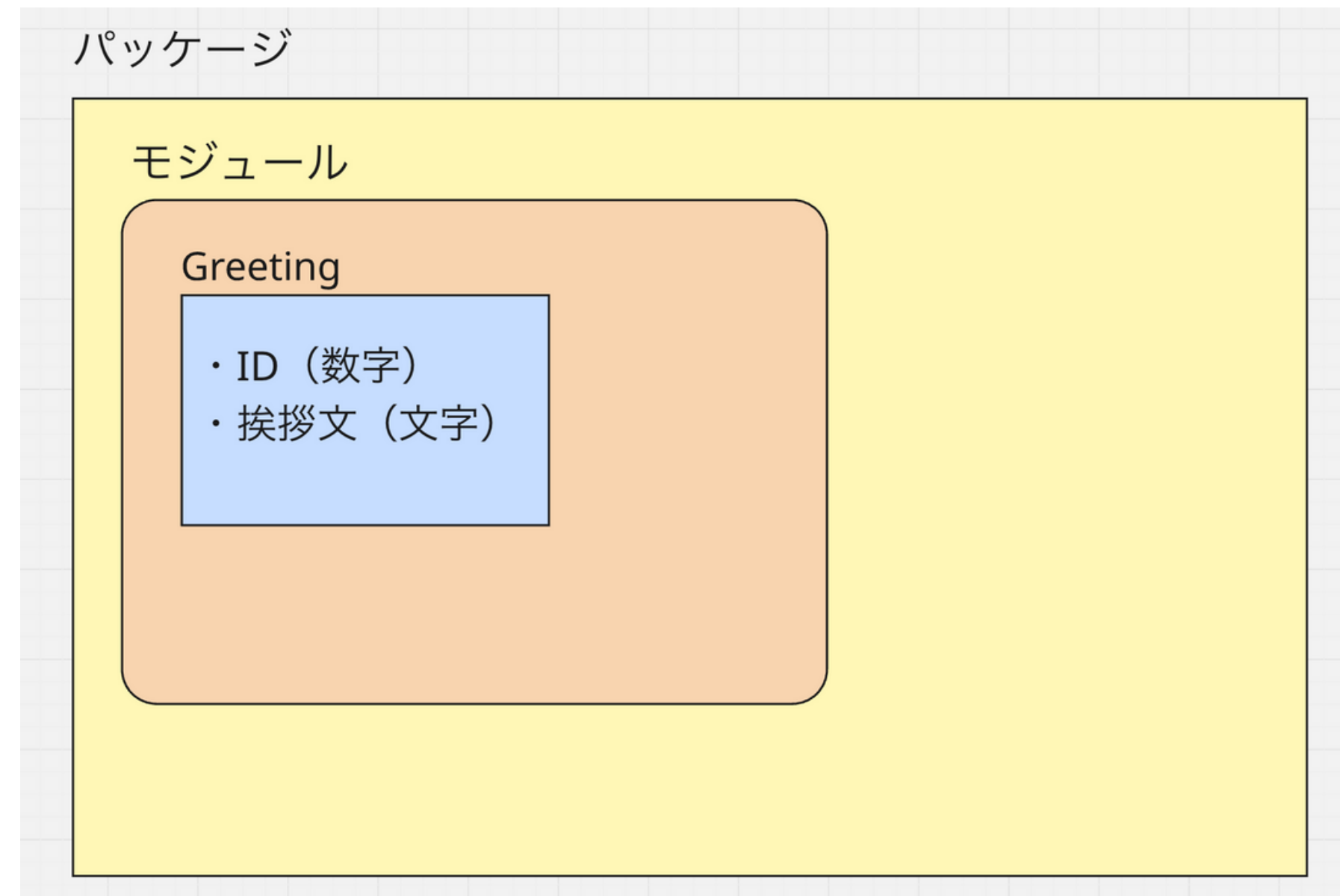
では、作ってみましょう。

パッケージ名とモジュール名を次のように設定します。



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。

```
1  
2 module empty2::greeting {  
3  
4     // 構造体  
5     public struct Greeting {  
6  
7     }  
8  
9 }  
10
```

外部から確認可能

構造体名

構造体のマーク

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。
これでこの構造体の世界で唯一のものとなります。

```
1
2  module empty2::greeting {
3
4      // 構造体
5      public struct Greeting has key {
6          id
7      }
8
9  }
10
```

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。

```
Home  greeting.move ×
1
2  module empty2::greeting {
3
4      // 構造体
5      public struct Greeting has key {
6          id: UID
7      }
8          UID (Unique ID) という型
9  }
10
```

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。

```
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() {
11    }
12
13
14 }
15
```

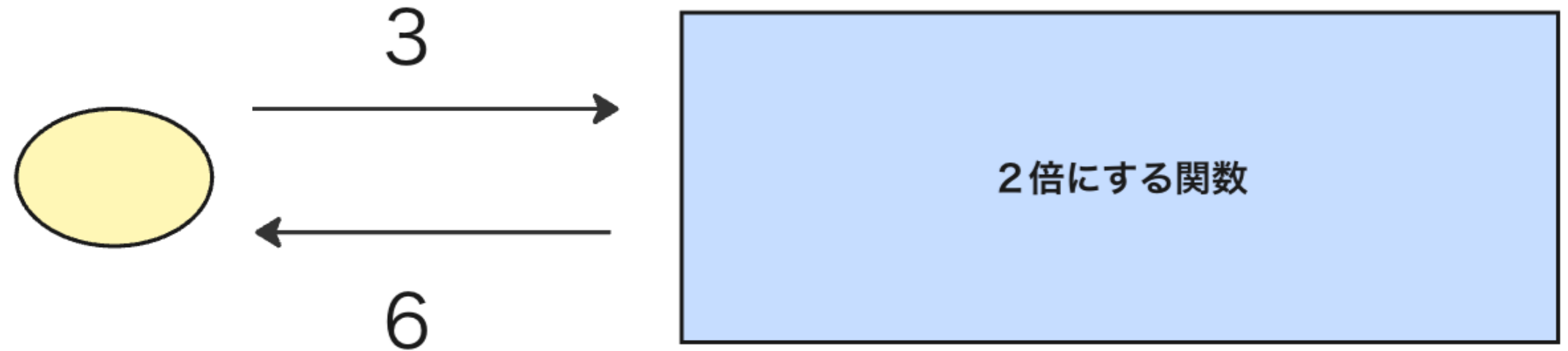
外部から
アクセス可能

関数名

関数マーク

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       |
10      // 関数
11      public fun new(ctx: &mut TxContext) {
12
13      }
14  }
15
```

5 関数の作成

参考に、TxContextはこのような型です。

```
module sui::tx_context;

/// Information about the transaction currently being executed.
/// This cannot be constructed by a transaction--it is a privileged object created by
/// the VM and passed in to the entrypoint of the transaction as `&mut TxContext`.

public struct TxContext has drop {
    /// The address of the user that signed the current transaction
    sender: address,
    /// Hash of the current transaction
    tx_hash: vector<u8>,
    /// The current epoch number
    epoch: u64,
    /// Timestamp that the epoch started at
    epoch_timestamp_ms: u64,
    /// Counter recording the number of fresh id's created while executing
    /// this transaction. Always 0 at the start of a transaction
    ids_created: u64
}
```

これがポイント
初期値は0でIDを作るたびに1増える

<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。

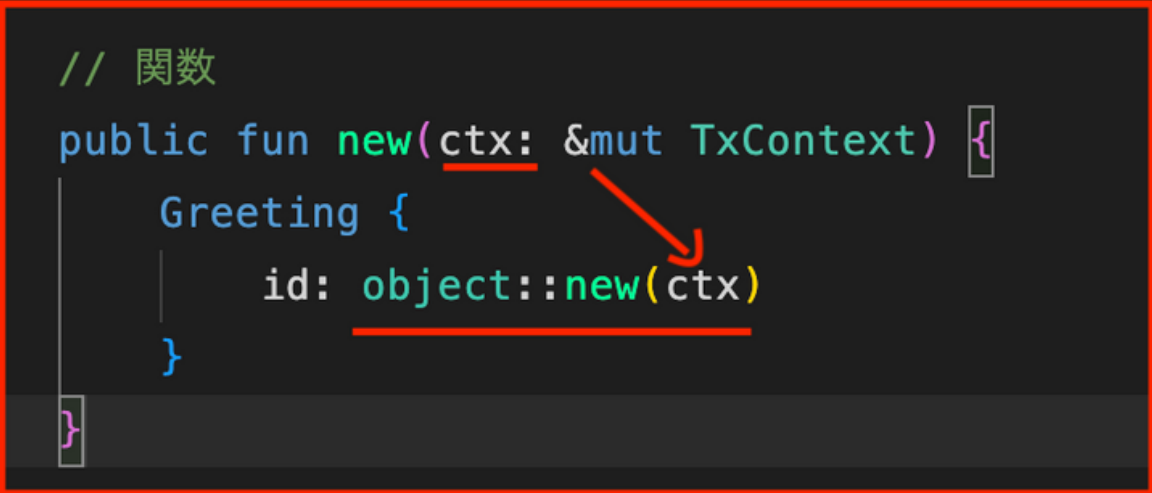
idは適当に設定しましたが、これでは唯一ではなさそうですね。

```
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         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。

```
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         Greeting {
12             id: object::new(ctx)
13         }
14     }
15
16 }
17
```



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。

```
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     }
15 }
16
17
```

変数マーク

変数名

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。

```
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     }
15 }
16
17
```

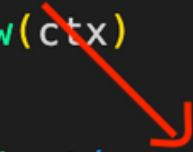
変数マーク

変数名

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 }
18
```



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 X
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。

```
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      }
23
24  }
25
26
```

7 変更を行う関数の作成

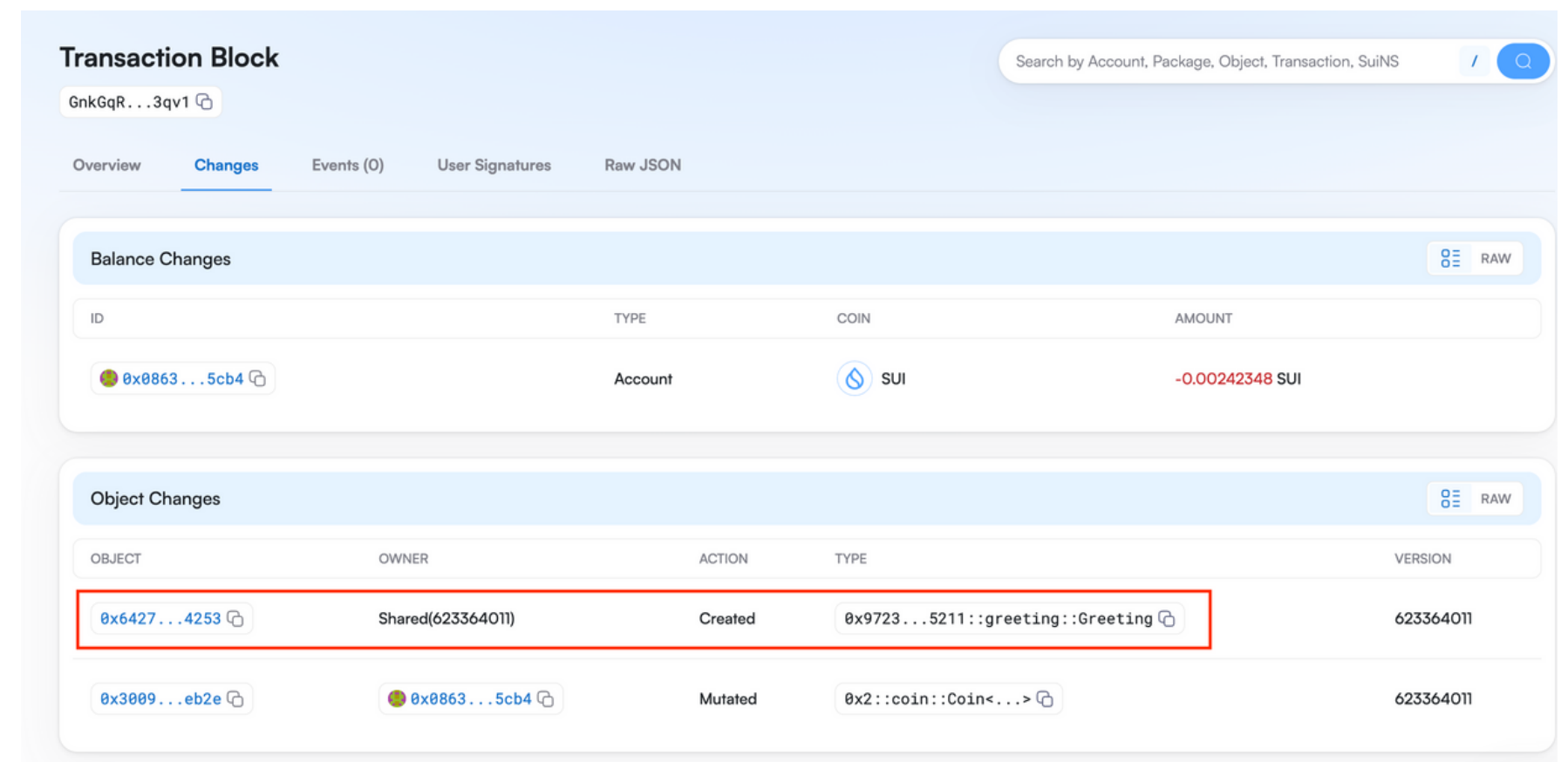
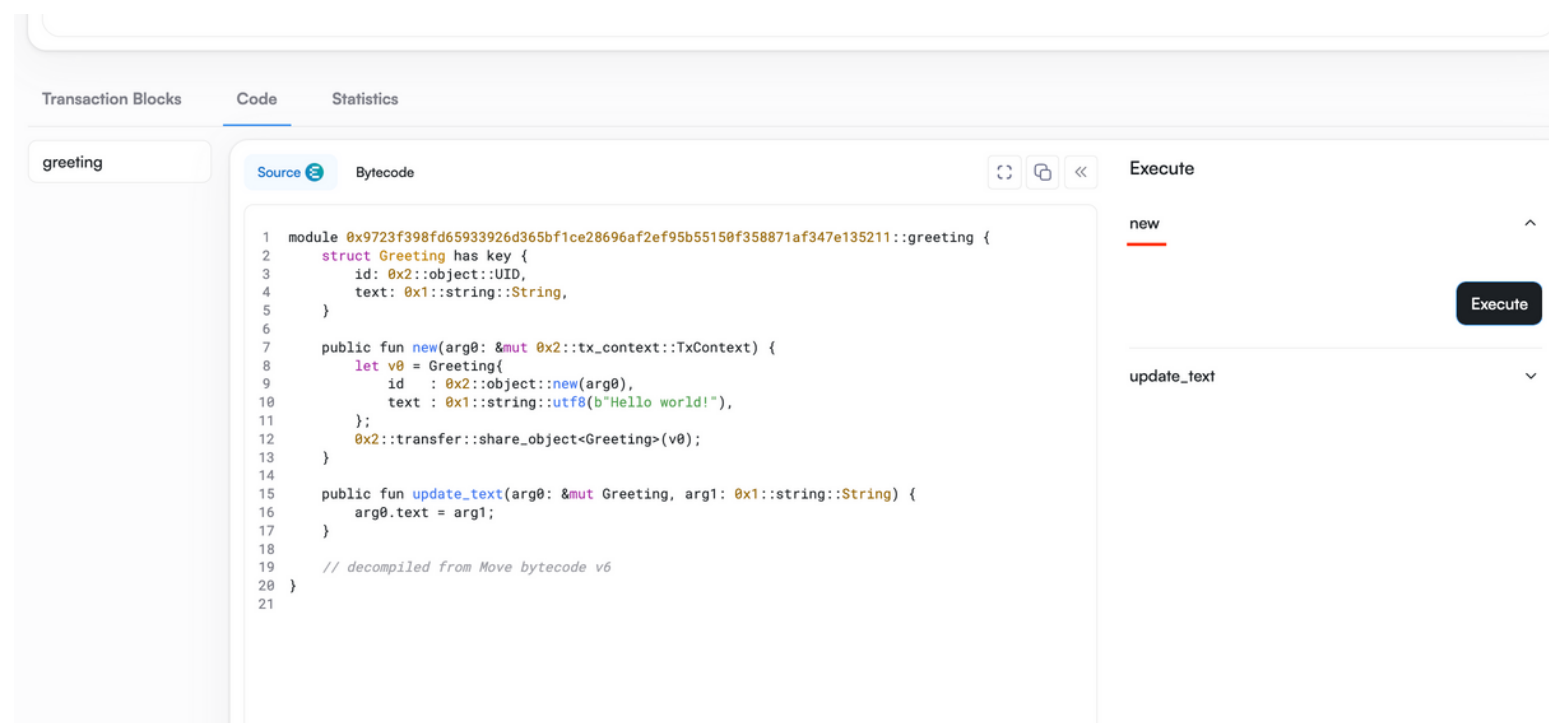
引数のnew_textをgreetingの「text」に代入します。
「=」は代入を表します。

```
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。

The screenshot displays the SuiVision interface for a transaction block named 'greeting'. The 'Code' tab is active, showing the decompiled Move code. The code defines a 'Greeting' struct and two functions: 'new' and 'update_text'. The 'update_text' function is highlighted in the right-hand 'Execute' panel. The 'Execute' panel shows the arguments for the 'update_text' function: Arg0 is '0x9723f398fd65933926d365bf1ce28696af2ef95b55150f358871af347e135211::greeting' and Arg1 is '0x48656c6c6f'. The 'Execute' button is highlighted with a red box.

```
1 module 0x9723f398fd65933926d365bf1ce28696af2ef95b55150f358871af347e135211::greeting {
2   struct Greeting has key {
3     id: 0x2::object::UID,
4     text: 0x1::string::String,
5   }
6
7   public fun new(arg0: &mut 0x2::tx_context::TxContext) {
8     let v0 = Greeting{
9       id : 0x2::object::new(arg0),
10      text : 0x1::string::utf8(b"Hello world!"),
11    };
12    0x2::transfer::share_object<Greeting>(v0);
13  }
14
15  public fun update_text(arg0: &mut Greeting, arg1: 0x1::string::String) {
16    arg0.text = arg1;
17  }
18
19  // decompiled from Move bytecode v6
20 }
21
```

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。

```
1  module empty::greeting {
2      use std::string;
3
4      public struct Greeting has key {
5          id: UID,
6          text: string::String,
7      }
8
9      public fun new(ctx: &mut TxContext) {
10         let new_greeting = Greeting {
11             id: object::new(ctx),
12             text: b"Hello world!".to_string()
13         };
14         transfer::public_transfer(new_greeting, ctx.sender());
15     }
16     public fun update_text(greeting: &mut Greeting, new_text: string::String) {
17         greeting.text = new_text;
18     }
19 }
```

3 ワークシヨ ヅップ

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、作っていく中で課題も見えてくると思います。

作ったスマートコントラクトにはどんな課題があるかも考え、発表しましょう。

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。

```
1 module academy::notice_board {
2     use std::string;
3
4     public struct NoticeBoard has key {
5         id: UID,
6         message: string::String,
7     }
8
9     // 共有オブジェクトとしてお知らせ板を1つ発行
10    public fun new(ctx: &mut TxContext) {
11        let board = NoticeBoard {
12            id: object::new(ctx),
13            message: b"Welcome to Academy Notice Board!".to_string(),
14        };
15        transfer::share_object(board);
16    }
17
18    // お知らせ文を更新
19    public fun update_message(board: &mut NoticeBoard, new_message: string::String) {
20        board.message = new_message;
21    }
22 }
23
```

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。

```
1 module academy::notice_board {
2     use std::string;
3
4     public struct NoticeBoard has key {
5         id: UID,
6         message: string::String,
7         update_count: u64,
8     }
9
10    public fun new(ctx: &mut TxContext) {}
11    | let board = NoticeBoard {
12        | id: object::new(ctx),
13        | message: b"Welcome to our project!".to_string(),
14        | update_count: 0,
15    };
16    transfer::share_object(board);
17 }
18
19 public fun update_message(board: &mut NoticeBoard, new_message: string::String)
20 {
21     board.message = new_message;
22     board.update_count = board.update_count + 1;
23 }
24 }
```

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。
何か問題がありますでしょうか。

```
1 module poll::simple_poll {  
2     use std::string;  
3  
4     public struct Poll has key {  
5         id: UID,  
6         question: string::String,  
7         yes_count: u64,  
8         no_count: u64,  
9     }  
10  
11     public fun new(ctx: &mut TxContext) {  
12         let poll = Poll {  
13             id: object::new(ctx),  
14             question: b"Should we adopt this new policy?".to_string(),  
15             yes_count: 0,  
16             no_count: 0,  
17         };  
18         transfer::share_object(poll);  
19     }  
20  
21     public fun vote_yes(poll: &mut Poll) {  
22         poll.yes_count = poll.yes_count + 1;  
23     }  
24  
25     public fun vote_no(poll: &mut Poll) {  
26         poll.no_count = poll.no_count + 1;  
27     }  
28 }  
29
```

4 参考 追加実装

対策① 誰でも処理できる設定に変えたい

assert!関数を使ってみると条件をつける事ができます。

```
1 module academy::notice_board {
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。

投票内容：「○○○○○」

賛成票：40

反対票：30

投票済みリスト

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。

```
1 module poll::simple_poll {
2   use std::string;
3   use sui::table::{Self, Table};
4
5   public struct Poll has key {
6     id: UID,
7     question: string::String,
8     yes_count: u64,
9     no_count: u64,
10    voters: Table<address, bool>, // ここに「投票済みアドレス」を記録
11  }
12
13  public fun new(ctx: &mut TxContext) {
14    let poll = Poll {
15      id: object::new(ctx),
16      question: string::utf8(b"Should we adopt this new policy?"),
17      yes_count: 0,
18      no_count: 0,
19      voters: table::new(ctx),
20    };
21    transfer::share_object(poll);
22  }
23
24  public fun vote_yes(poll: &mut Poll, ctx: &TxContext) {
25    let sender = ctx.sender();
26    assert!(!table::contains(&poll.voters, sender), 1);
27    table::add(&mut poll.voters, sender, true);
28    poll.yes_count = poll.yes_count + 1;
29  }
```

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 許可されたもの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) {
```

管理者のみがホワイトリストを追加できます。

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。

```
1 module poll3::simple_poll {
2     use std::string;
3     use sui::table::{Self, Table};
4
5     const E_NOT_ADMIN: u64 = 1;
6     const E_NOT_ALLOWED: u64 = 2;
7     const WHITELIST_MANAGER: address = @0x0863a92345a2f9f9fc88e659cbcc983b2512425a2f2ffe02edbb5da4e6915cb4;
8
9     public struct Poll has key {
10         id: UID,
11         admin: address,           // 管理者（作成者）
12         question: string::String,
13         yes_count: u64,
14         no_count: u64,
15         allow: Table<address, bool>, // 投票を許可するアドレス集合
16     }
17
18     public fun new(ctx: &mut TxContext) {
19         let poll = Poll {
20             id: object::new(ctx),
21             admin: WHITELIST_MANAGER,
22             question: string::utf8(b"Should we adopt this new policy?"),
23             yes_count: 0,
24             no_count: 0,
25             allow: table::new(ctx),
26         };
27     }
```

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。

```
1 module poll4::simple_poll {
2   use std::string;
3   use sui::clock::{Self, Clock};
4
5   // 2025-12-31 23:59:59.999 JST = 2025-12-31T14:59:59.999Z = 1767193199999 ms
6   const DEADLINE_MS: u64 = 1767193199999;
7
8   public struct Poll has key {
9     id: UID,
10    question: string::String,
11    yes_count: u64,
12    no_count: u64,
13    end_ms: u64, // 期限 (UTCエポックms)
14  }
15
16  // 期限固定で作成
17  public fun new_until_2025_12_31_jst(ctx: &mut TxContext) {
18    let poll = Poll {
19      id: object::new(ctx),
20      question: string::utf8(b"Should we adopt this new policy?"),
21      yes_count: 0,
22      no_count: 0,
23      end_ms: DEADLINE_MS,
24    };
25    transfer::share_object(poll);
26  }
27
28  public fun vote_yes(poll: &mut Poll, clock: &Clock) {
29    let now = clock::timestamp_ms(clock);
30    assert!(now <= poll.end_ms, 1);
31    poll.yes_count = poll.yes_count + 1;
32  }
33}
```

CodeStatistics

SourceBytecode

```
1 module 0x6fee75f286c448ec4405acb3c1ff61700d6ce753ae300f128ebca7e06c4:
2   struct Poll has key {
3     id: 0x2::object::UID,
4     question: 0x1::string::String,
5     yes_count: u64,
6     no_count: u64,
7     end_ms: u64,
8   }
9
10  public fun new_until_2025_12_31_jst(arg0: &mut 0x2::tx_context::TxContext) {
11    let v0 = Poll{
12      id      : 0x2::object::new(arg0),
13      question : 0x1::string::utf8(b"Should we adopt this new policy?"),
14      yes_count : 0,
15      no_count  : 0,
16      end_ms   : 1767193199999,
17    };
18    0x2::transfer::share_object<Poll>(v0);
19  }
20
21  public fun vote_no(arg0: &mut Poll, arg1: &0x2::clock::Clock) {
22    assert!(0x2::clock::timestamp_ms(arg1) <= arg0.end_ms, 1);
23    arg0.no_count = arg0.no_count + 1;
24  }
25
26  public fun vote_yes(arg0: &mut Poll, arg1: &0x2::clock::Clock) {
27    assert!(0x2::clock::timestamp_ms(arg1) <= arg0.end_ms, 1);
28    arg0.yes_count = arg0.yes_count + 1;
29  }
```

Execute

new_until_2025_12_31_jst

vote_no

vote_yes

Arg0

0x9db091364f7fbbeda18eec484ffd8e54cd226431

Arg1

0x6

clockオブジェクトは0x6固定です。

Execute

予備資料

開発11 public transferとstore

コンパイルしようとする、エラーになってしまいます。

storeがないためです。

どこかに入れるわけではないのになぜ？

```
Home greeting.move X
6   text: string::String,
7   }
8
9   public fun new(ctx: &mut TxContext) {
10      let new_greeting = Greeting {
11         id: object::new(ctx),
12         text: b"Hello world!".to_string()
13      };
14      transfer::public_transfer(new_greeting, ctx.sender());
15   }
16
17   public fun update_text(greeting: &mut Greeting, new_text: string::String) {
18      greeting.text = new_text;
19   }
```

Output Scan Log

```
10 | let new_greeting = Greeting {
11 | | id: object::new(ctx),
12 | | text: b"Hello world!".to_string()
13 | | };
14 | | transfer::public_transfer(new_greeting, ctx.sender());
   | ~~~~~ The type 'empty::greeting::Greeting' does not have the ability 'store'
   | ~~~~~ 'store' constraint not satisfied
```

/root/.move/https__github_com_MystenLabs_sui_git_framework__testnet/crates/sui-framework/packages/sui-framework/sources/transfer.move:63:37

```
63 | public fun public_transfer(obj: TxRecipient, address):
```

開発11 public transferとstore

ルール上、送れないようです。

私の仮説は右です。これが許されると、すでに何かに格納されているものには送られようとした場合、整合性が取れなくなるからではと考えています。

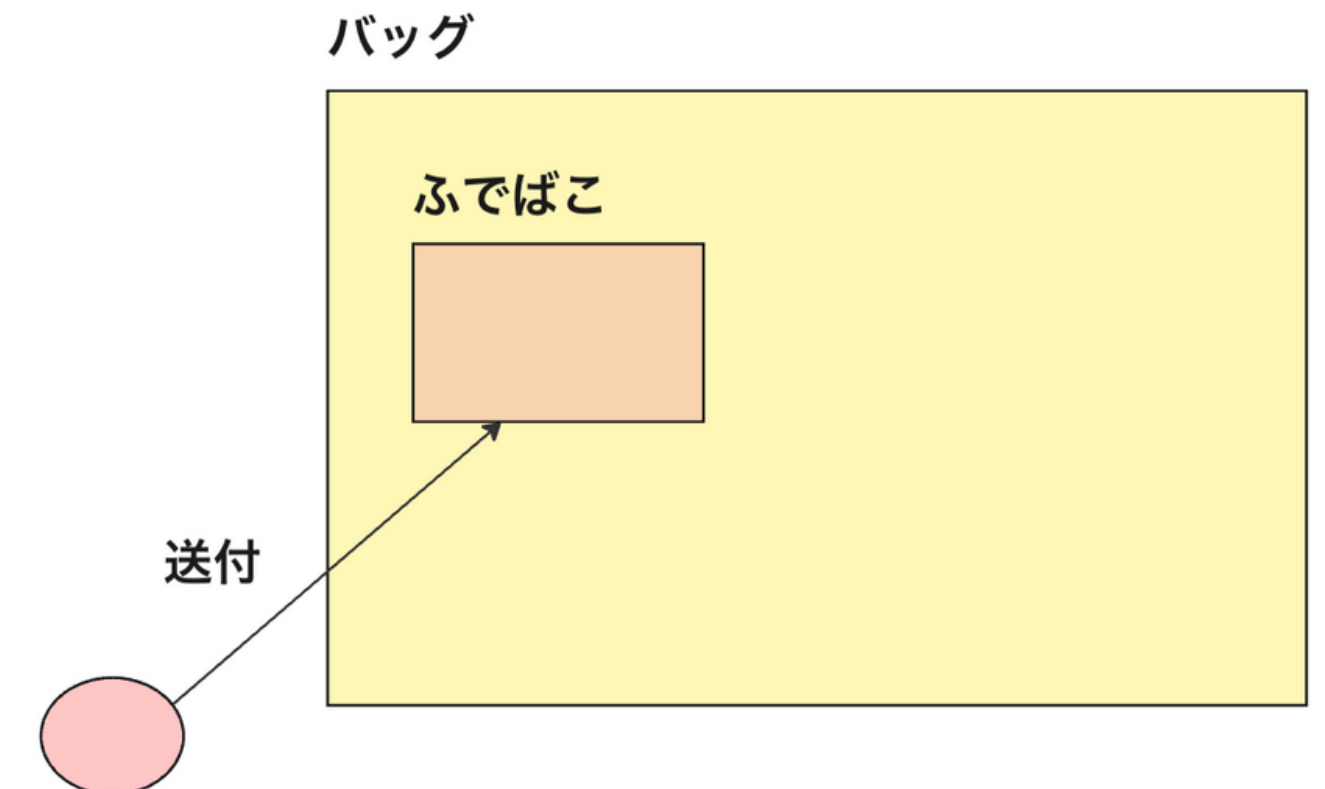
```
module a::my_module;

public struct A has key {
  id: sui::object::UID,
}

public struct B has key, store {
  id: sui::object::UID,
}
```

A can only be transferred using the `sui::transfer::transfer` inside of `a::my_module`, while B can be transferred anywhere using `sui::transfer::public_transfer`. These rules are enforced by a custom type system (bytecode verifier) rule in Sui.

Aは `a::my_module` 内の `sui::transfer::transfer` を使用してのみ転送できますが、B `sui::transfer::public_transfer` を使用してどこにでも転送できます。これらのルールは、Suiのカスタムタイプシステム(バイトコード検証ツール)ルールによって適用されます。



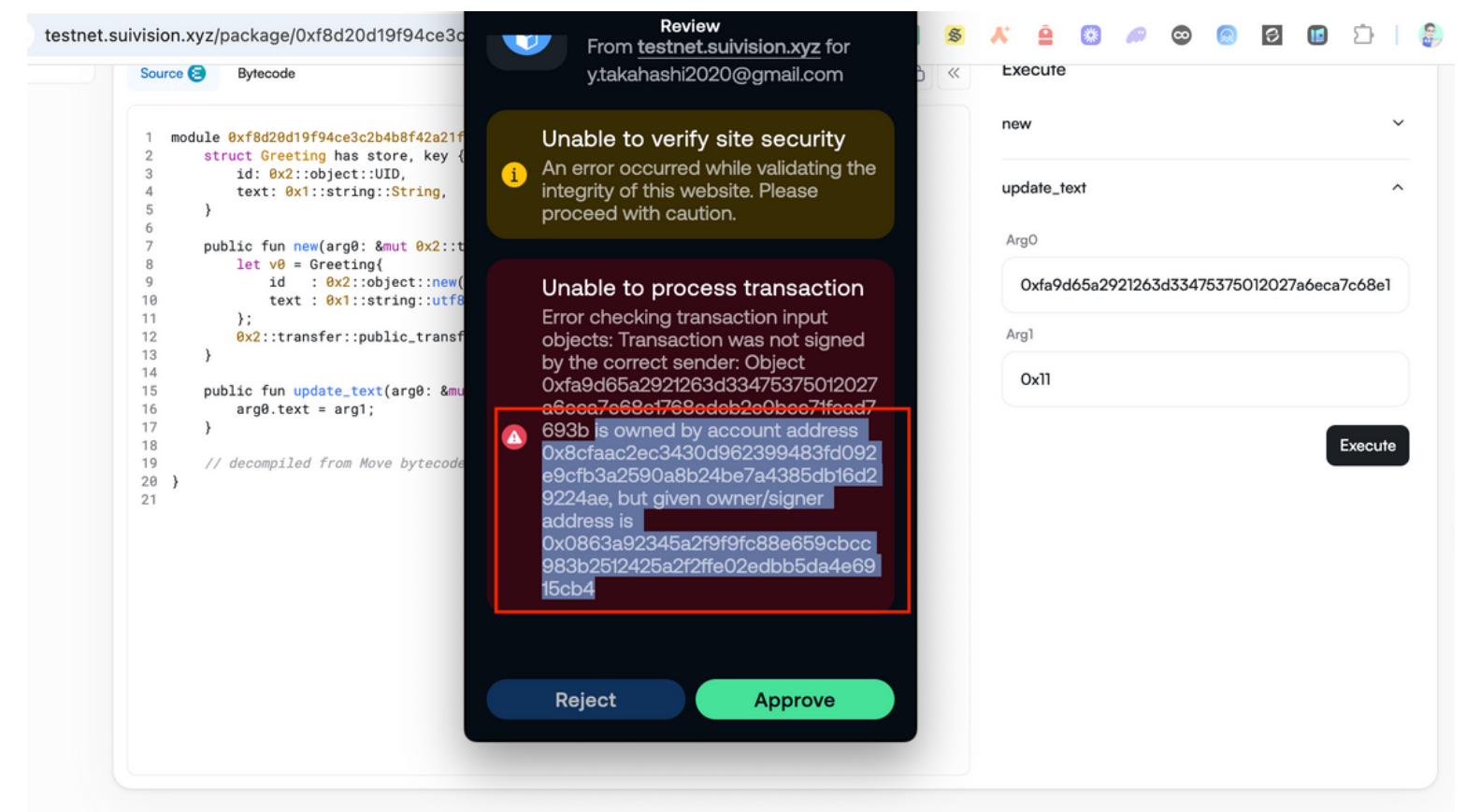
<https://move-book.com/reference/abilities/object>

開発12 所有者以外で実行できない確認

storeをつけて、デプロイしましょう。

その後、新しくオブジェクトを作成し、**作成者以外で実行するとエラー**になることが確認できます。

```
Home greeting.move X
1 module empty::greeting {
2   use std::string;
3
4   public struct Greeting has key, store {
5     id: UID,
6     text: string::String,
7   }
8
9   public fun new(ctx: &mut TxContext) {
10     let new_greeting = Greeting {
11       id: object::new(ctx),
12       text: b"Hello world!".to_string()
13     };
14     transfer::public_transfer(new_greeting, ctx.sender());
15   }
16   public fun update_text(greeting: &mut Greeting, new_text: string::String) {
17     greeting.text = new_text;
18   }
19 }
```



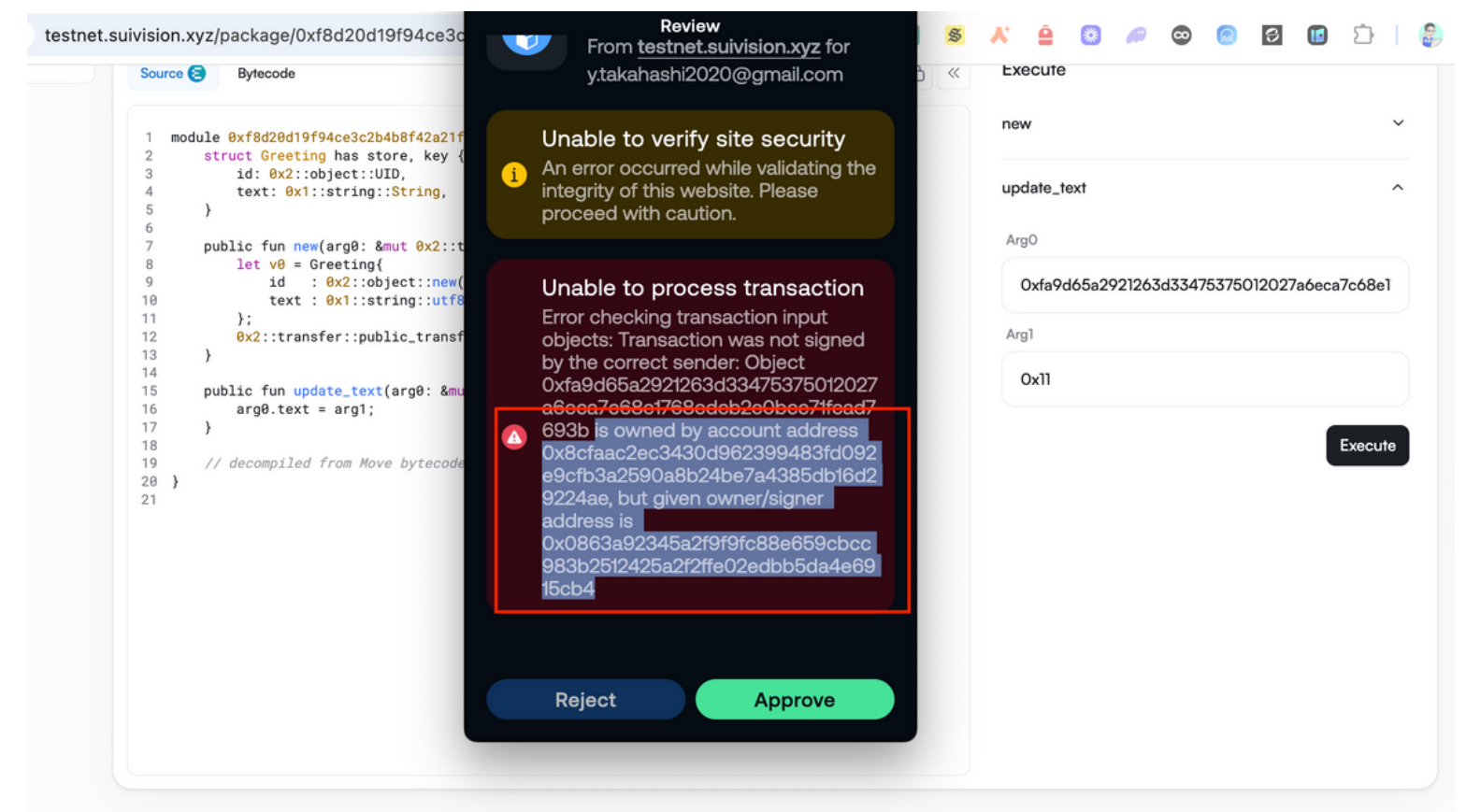
<https://move-book.com/reference/abilities/object>

開発12 所有者以外で実行できない確認

storeをつけて、デプロイしましょう。

その後、新しくオブジェクトを作成し、**作成者以外で実行するとエラー**になることが確認できます。

```
Home greeting.move X
1 module empty::greeting {
2   use std::string;
3
4   public struct Greeting has key, store {
5     id: UID,
6     text: string::String,
7   }
8
9   public fun new(ctx: &mut TxContext) {
10     let new_greeting = Greeting {
11       id: object::new(ctx),
12       text: b"Hello world!".to_string()
13     };
14     transfer::public_transfer(new_greeting, ctx.sender());
15   }
16   public fun update_text(greeting: &mut Greeting, new_text: string::String) {
17     greeting.text = new_text;
18   }
19 }
```



<https://move-book.com/reference/abilities/object>

開発13 コピーができないことの確認

Copy能力がついていないため、**コピーしようとするときエラーになる**ことを確認しましょう。

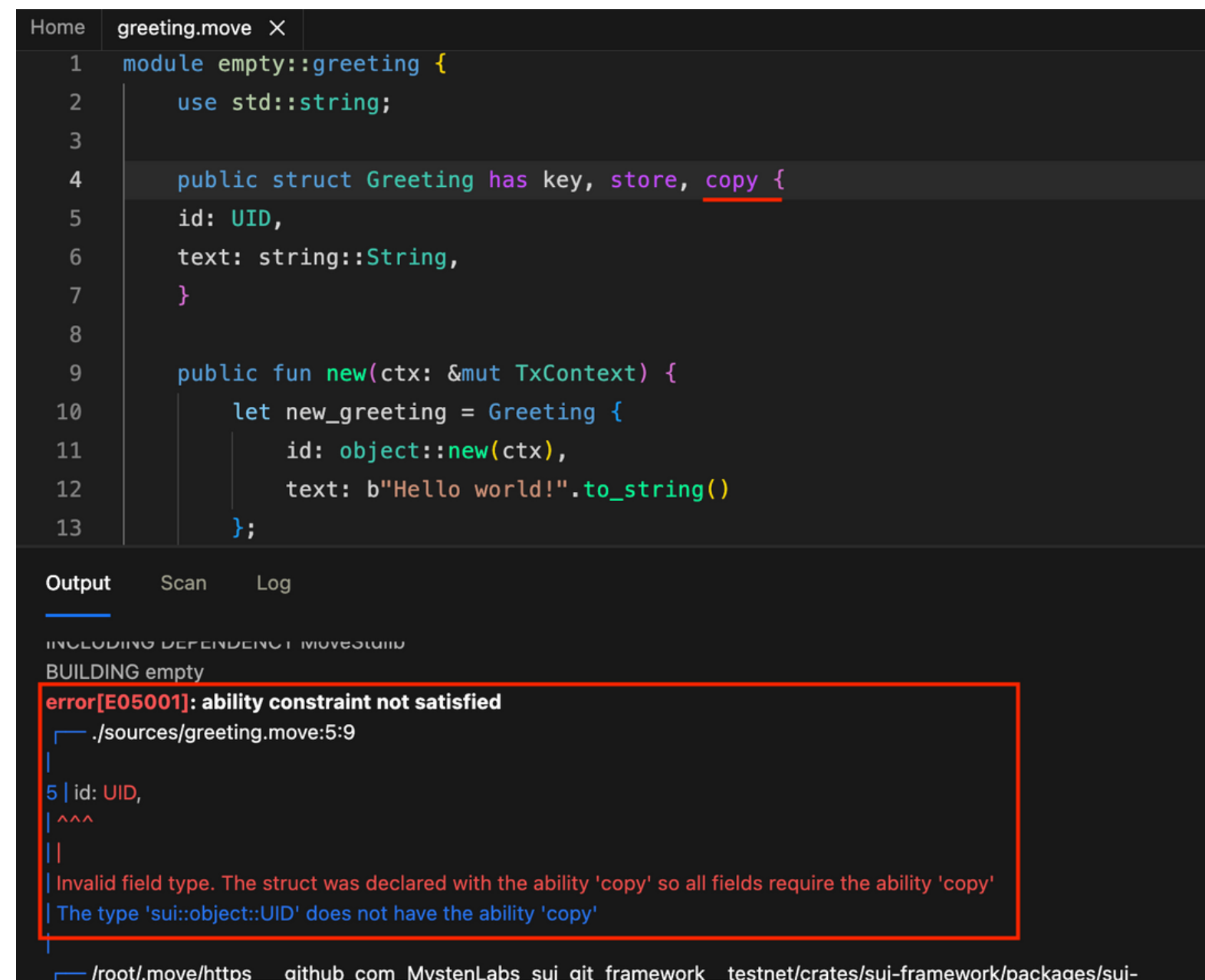
```
6      text: string::String,  
7  }  
8  
9  public fun new(ctx: &mut TxContext) {  
10      let new_greeting = Greeting {  
11          id: object::new(ctx),  
12          text: b"Hello world!".to_string()  
13      };  
14      let new_greeting2 = copy new_greeting;  
15      transfer::public_transfer(new_greeting, ctx.sender());  
16  }  
17  public fun update_text(greeting: &mut Greeting, new_text: string::String) {  
18      greeting.text = new_text;  
19  }  
20 }
```

Output Scan Log

```
11 || id: object::new(ctx),  
12 || text: b"Hello world!".to_string()  
13 || };  
14 | let new_greeting2 = copy new_greeting;  
   | ^^^^^^^^^^^^^^^^^^^^^ Invalid 'copy' of owned value without the 'copy' ability
```

開発14 コピーとキーが両立しない確認

Keyはグローバル上の唯一性です。以下のように**コピーとは両立しません**。



```
1 module empty::greeting {
2   use std::string;
3
4   public struct Greeting has key, store, copy {
5     id: UID,
6     text: string::String,
7   }
8
9   public fun new(ctx: &mut TxContext) {
10     let new_greeting = Greeting {
11       id: object::new(ctx),
12       text: b"Hello world!".to_string()
13     };
14   }
```

Output Scan Log

INCLUDING DEPENDENCY movestd
BUILDING empty

error[E05001]: ability constraint not satisfied
└─ ./sources/greeting.move:5:9
5 | id: UID,
 | ^^^
 |
 | Invalid field type. The struct was declared with the ability 'copy' so all fields require the ability 'copy'
 | The type 'sui::object::UID' does not have the ability 'copy'

└─ /root/.move/https_github_com_MystenLabs_sui_git_framework_testnet/crates/sui-framework/packages/sui-

開発15 dropできないことの確認

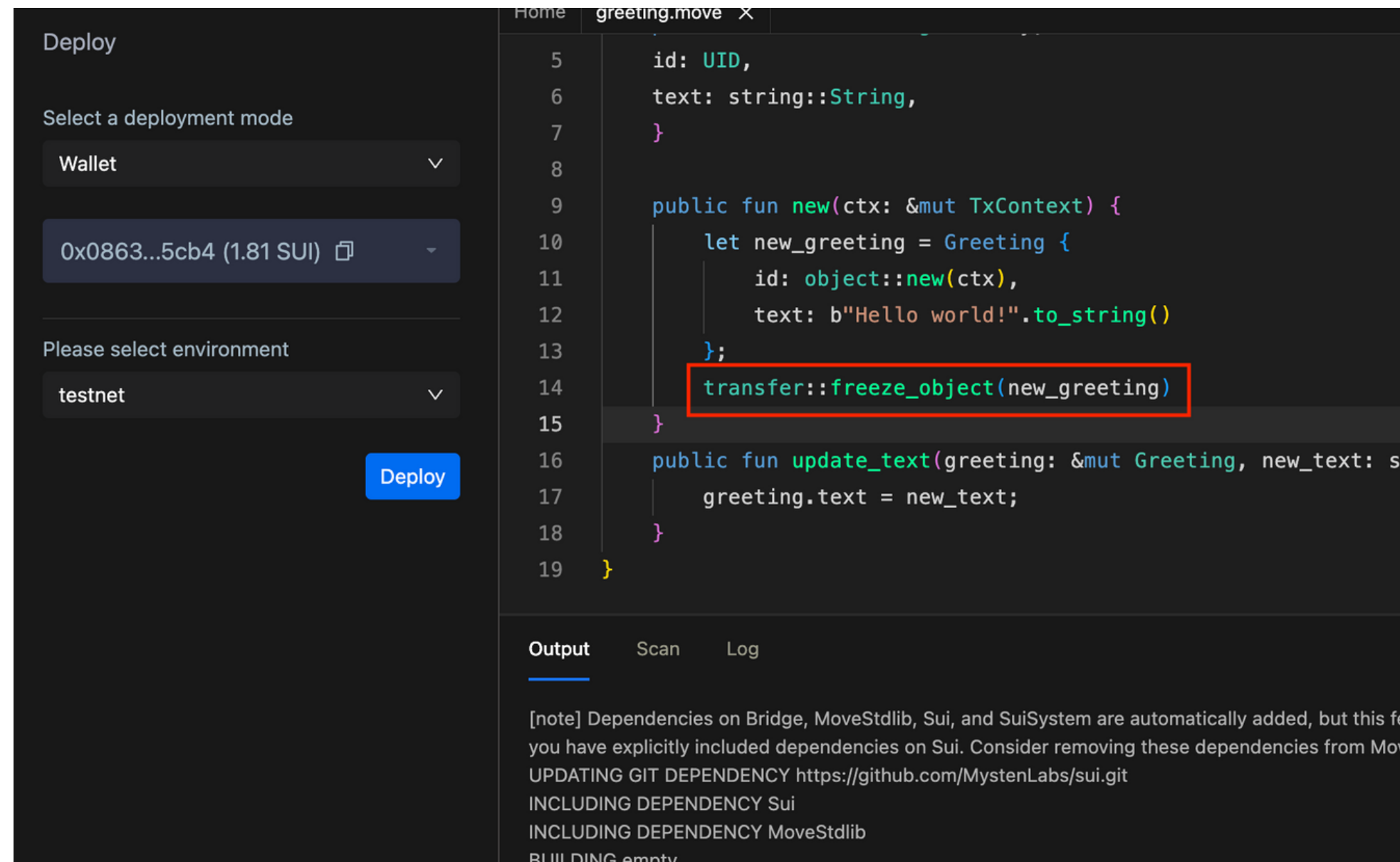
所有者を設定せずに終わろうとすると、**dropのエラー**になります。

```
5     id: UID,  
6     text: string::String,  
7 }  
8  
9 public fun new(ctx: &mut TxContext) {  
10     let new_greeting = Greeting {  
11         id: object::new(ctx),  
12         text: b"Hello world!".to_string()  
13     };  
14     // transfer::public_transfer(new_greeting, ctx.sender());  
15 }  
16 public fun update_text(greeting: &mut Greeting, new_text: string::String) {  
17     greeting.text = new_text;  
18 }  
19 }
```

error[E06001]: unused value without 'drop'
└─ ./sources/greeting.move:13:10
|
4 | public struct Greeting has key, store {
| ----- To satisfy the constraint, the 'drop' ability would need to be added here
.
10 | let new_greeting = Greeting {

開発16 Immutableの設定

Immutableにすることは所有者を設定しないことではありません。
「所有者なし」という所有の状態に設定しています。



開発17 Immutableの確認

オブジェクトがImmutableになることを確認し、**操作しようとするエラーになることを確認しましょう。**

Transaction Block

34Tid2...5rRx

Overview

Changes

Events (0)

User Signatures

Raw JSON

Balance Changes

ID	TYPE	COIN	AMOUNT
0x0863...5cb4	Account	SUI	-0.0024

Object Changes

OBJECT	OWNER	ACTION	TYPE
0x2150...b40a	Immutable	Created	0xa7bc...4436::greeting::Greeting
0x3009...eb2e	0x0863...5cb4	Mutated	0x2::coin::Coin<...>

package/0xa7bc2ec9f7def0ca611e2b...

zAHPcZjNLK6SkcYob6qSKS3Vm3ScjoZTGzcJ7u8

ble

statistics

Bytecode

```
0xa7bc2ec9f7def0ca611e2b34b83a30cc884bfbebd9b85917f26ee975a1b4436::greeting {  
  mut Greeting has store, key {  
    id: 0x2::object::UID,  
    text: 0x1::string::String,  
  }  
}  
  
lic fun new(arg0: &mut 0x2::tx_context::TxContext) {  
  let v0 = Greeting {  
    id : 0x2::object::new(arg0),  
    text : 0x1::string::utf8(b"Hello world!"),  
  };  
  0x2::transfer::freeze_object<Greeting>(v0);  
}  
  
lic fun update_text(arg0: &mut Greeting, arg1: 0x1::string::String) {  
  arg0.text = arg1;  
}
```

decompiled from Move bytecode v6

Review

Transaction request

From testnet.suivision.xyz for y.takahashi2020@gmail.com

Unable to verify site security

An error occurred while validating the integrity of this website. Please proceed with caution.

Unable to process transaction

Dry run failed, could not automatically determine a budget:
CommandArgumentError { arg_idx: 0, kind: InvalidObjectByMutRef } in command 0

Reject

Approve

型からの派生 ②オブジェクトと所有権

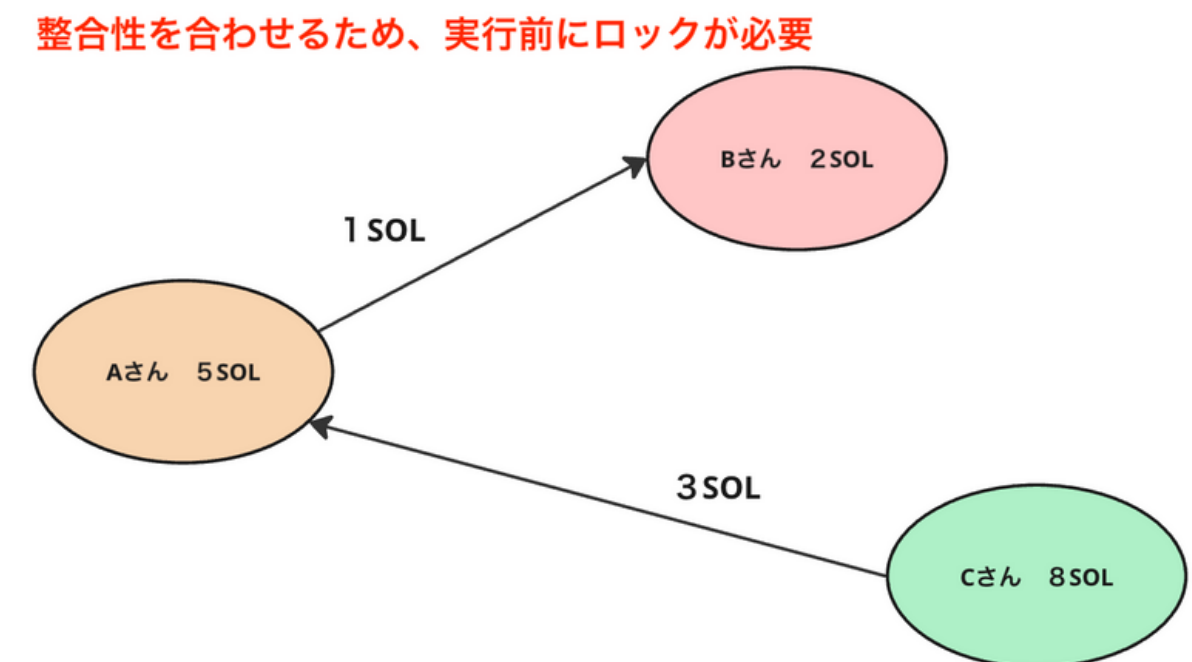
Suiはデータの値ではなく、**オブジェクトという形**で判断します。

また、**所有者**が外から明確に分かるようになっています。

まずは比較のためにSolanaで考えてみましょう。

下のようにAさんのアカウントが同時に利用されようとする場合です。

そのため、**実行前に書き込みが行われるアカウントを確認し、事前時にロック**をすることで整合性を保ちます。

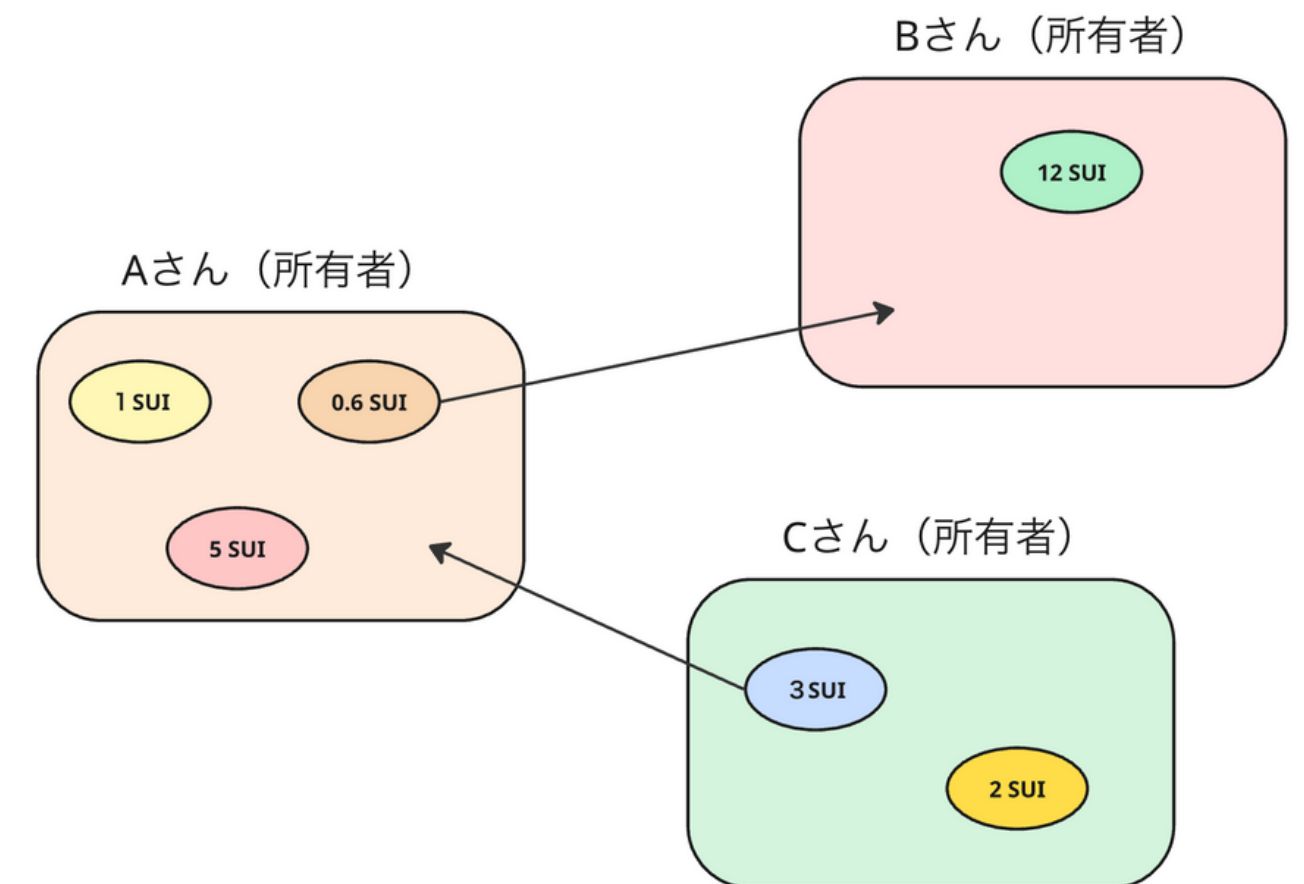


型からの派生 ②オブジェクトと所有権

Suiの場合はオブジェクトを元にした所有権の設計です。

あるコインがアドレス所有である場合、**そのコインを移動できるのは所有者だけ**のため、**整合性の問題がそもそも発生しません。**

そのため**機械的に並列処理**を行うことができます。



※簡易化のため、コインの分割は行っていませんが、行っても本質は同じです。

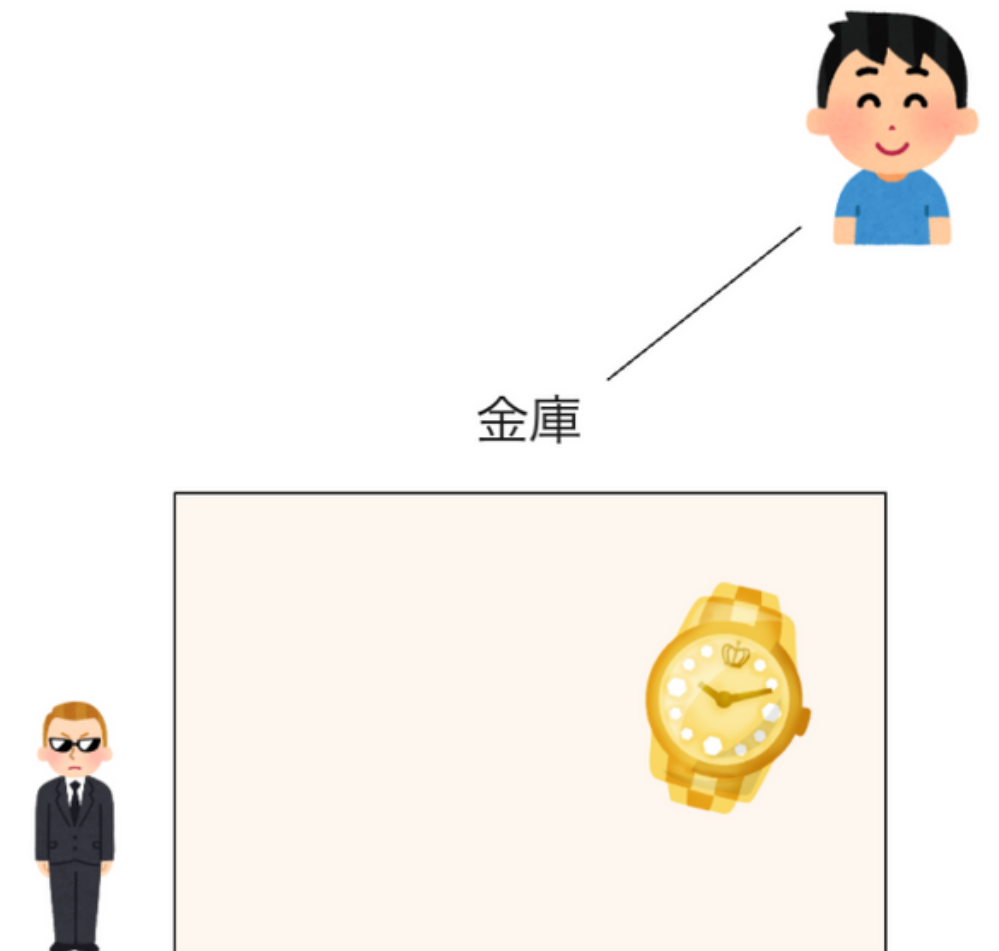
型からの派生 ③ラップと単一所有の強制

現実世界でいうところの所有と占有の話を考えてみましょう。

あなたの高級時計が金庫に保管され、大切に守られています。

この高級時計、動かせるのは誰でしょう。

守っている側からすると、**知らない間にあなたが動かせると
ちょっとびっくりしてしまいそう**ですね。



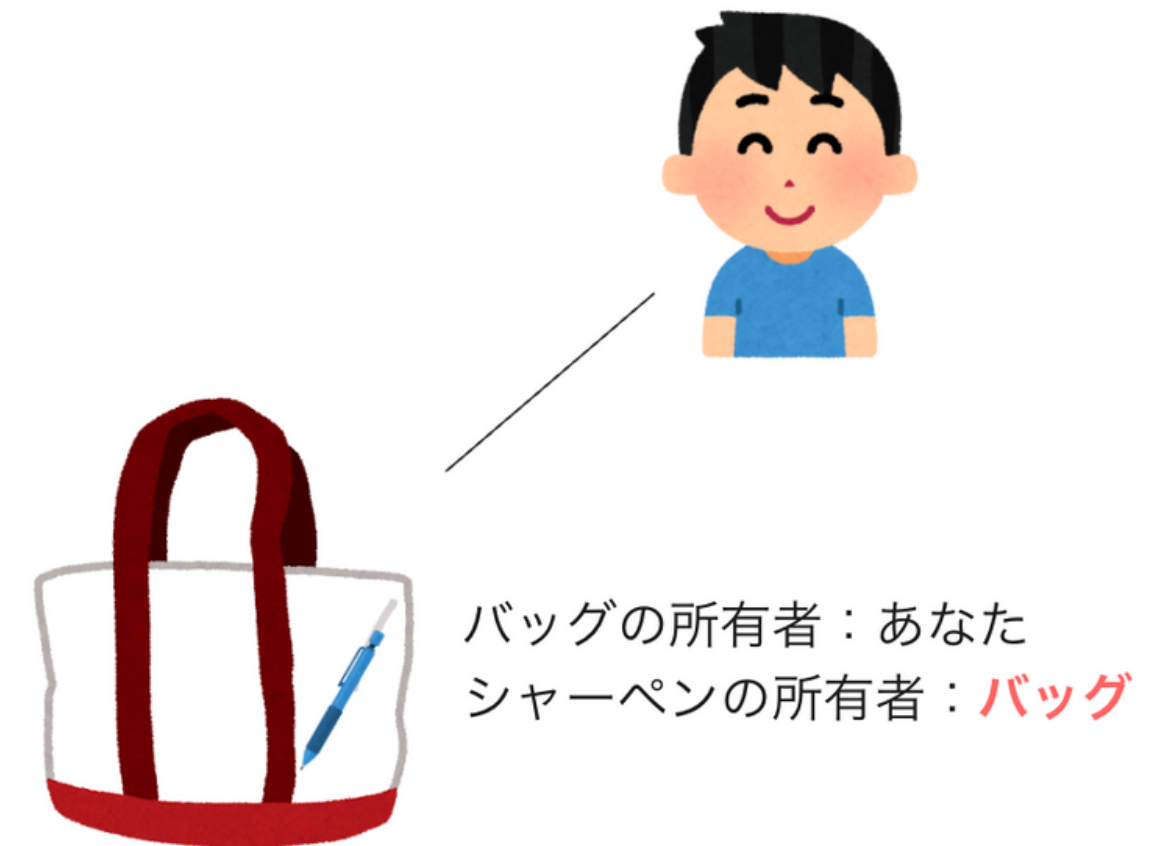
型からの派生 ③ラップと単一所有の強制

移動のイメージにしたいので、金庫でなく、バッグとシャーペンで考えます。

バッグにシャーペンがある場合、シャーペンの所有者は**あなたでなく、バッグのみ**です。

これによって**所有者が一意**になり、明確です。

動かしたいときはバッグの所有者のあなたが**バッグごと動かせば良い**のです。

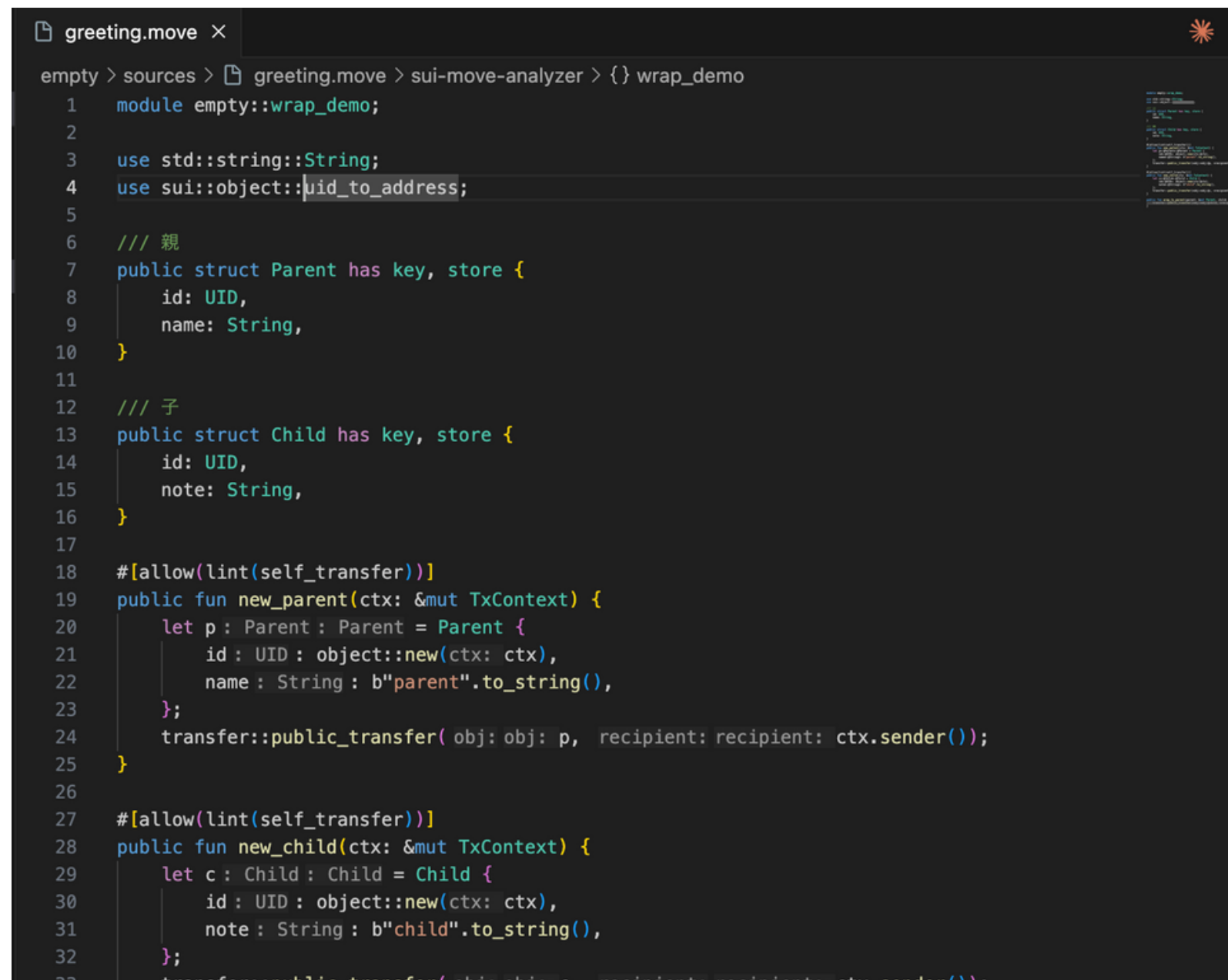


バッグの所有者：あなた
シャーペンの所有者：**バッグ**

**あなたはシャーペンだけを動かすことはできません。
所有者はバッグなのでバッグごと移動させることになります。**

開発18 Wrapと単一所有の確認

実際に試してみましょう。



```
greeting.move ×
empty > sources > greeting.move > sui-move-analyzer > {} wrap_demo
1  module empty::wrap_demo;
2
3  use std::string::String;
4  use sui::object::uid_to_address;
5
6  /// 親
7  public struct Parent has key, store {
8      id: UID,
9      name: String,
10 }
11
12 /// 子
13 public struct Child has key, store {
14     id: UID,
15     note: String,
16 }
17
18 #[allow(lint(self_transfer))]
19 public fun new_parent(ctx: &mut TxContext) {
20     let p: Parent: Parent = Parent {
21         id: UID: object::new(ctx: ctx),
22         name: String: b"parent".to_string(),
23     };
24     transfer::public_transfer( obj: obj: p, recipient: recipient: ctx.sender());
25 }
26
27 #[allow(lint(self_transfer))]
28 public fun new_child(ctx: &mut TxContext) {
29     let c: Child: Child = Child {
30         id: UID: object::new(ctx: ctx),
31         note: String: b"child".to_string(),
32     };
33     transfer::public_transfer( obj: obj: c, recipient: recipient: ctx.sender());
34 }
```

https://github.com/ytakahashi2020/sui_send

代表的な設計パターン

データをオブジェクトとして扱う

Suiのモデルはデータをオブジェクト（所有または共有）として保存します。

SuiのCapability（権限）

Capabilityによってアクセスを制御し、許可された操作のみを許容します。

One-time Witnessパターン

一意に作成された証明オブジェクトにより、操作を一度だけ実行可能にします。

型からの派生 ④Capabilityによる権限

Solanaの場合、権限チェックは**実行時**に行われます。

例えば、「**MintTo**」という関数の場合、**配列の3番目**のアカウントが**ミント権限者**です。

そして、その者の署名があるか確認します。

大事なのは関数ごとに**権限者かが何か異なる**点です。

そのため、**機械的なチェックができず**、実行時に確認します。

```
Transaction {
  // 署名は「message全体」に対するもの (Instructionごとではない)
  signatures: [
    sig_P, // account_keys[0] = P (fee payer)
    sig_M, // account_keys[1] = M (mint authority)
  ],

  message: Message {
    header: MessageHeader {
      // 先頭から num_required_signatures 個のアカウントが署名必須
      num_required_signatures: 2, // P と M の2署名が必要
      num_readonly_signed_accounts: 0, // 読み取り専用の署名アカウント数
      num_readonly_unsigned_accounts: 1, // 読み取り専用の非署名 (例: TOKEN_PROG)
    },

    // このTxで参照するアカウントのリスト (Instruction側は index で参照)
    account_keys: [
      P, // [0] fee payer (署名者) is_signer=true, is_writable=true(手数料)
      M, // [1] mint authority (署名者) is_signer=true, is_writable=false(通常)
      MINT, // [2] Mintアカウント is_signer=false, is_writable=true
      D, // [3] 受取トークン口座 is_signer=false, is_writable=true
      TOKEN_PROG, // [4] SPL Token Program is_signer=false, is_writable=false
    ],

    recent_blockhash: Hash(...),

    instructions: [
      Instruction {
        program_id: TOKEN_PROG, // = account_keys[4]

        // この命令で使うアカウント (上の account_keys の index 参照)
        // mint_to の標準的な並び: [mint, dest_token_account, authority]
        accounts: [2, 3, 1], // [MINT, D, M]

        // 関数+引数をシリアライズしたバイト列 (概念表示)
        data: MintTo { amount: 100 },
      },
    ],
  },
}
```

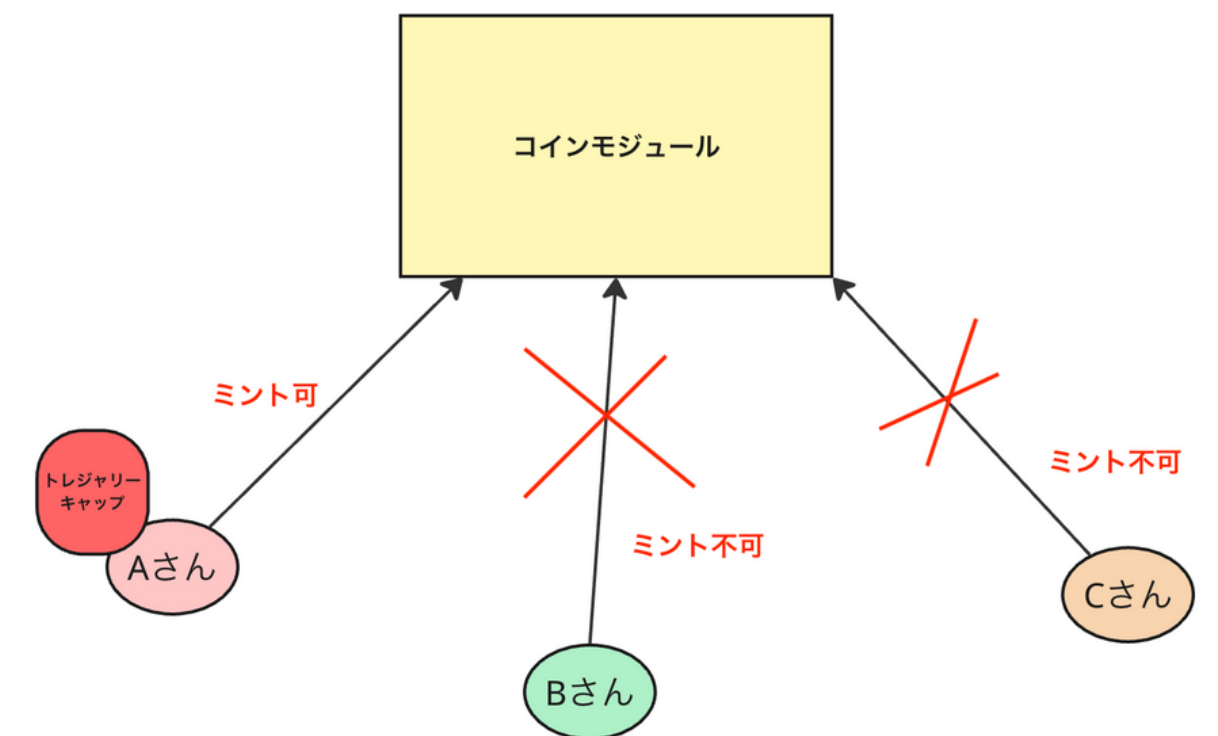
型からの派生 ④Capabilityによる権限

Suiの場合、権限チェックは**コンパイル時**に行われます。

Suiの場合は**権限オブジェクトを持っている人が権限者**というシンプルな構造です。

そのため、実行時に形式的に判断ができます。

また、コンパイル時に「**ミント関数に権限オブジェクトが引数に設定されているか**」の形式的なチェックで済みます。



型からの派生 ⑤OTWによる一度実行

一度きりの処理には**OTW (One-Time Witness)** を引数として渡します。

これにより、一度きりであることが客観的にわかります。

```
module examples::my_coin_new;

use sui::coin_registry;

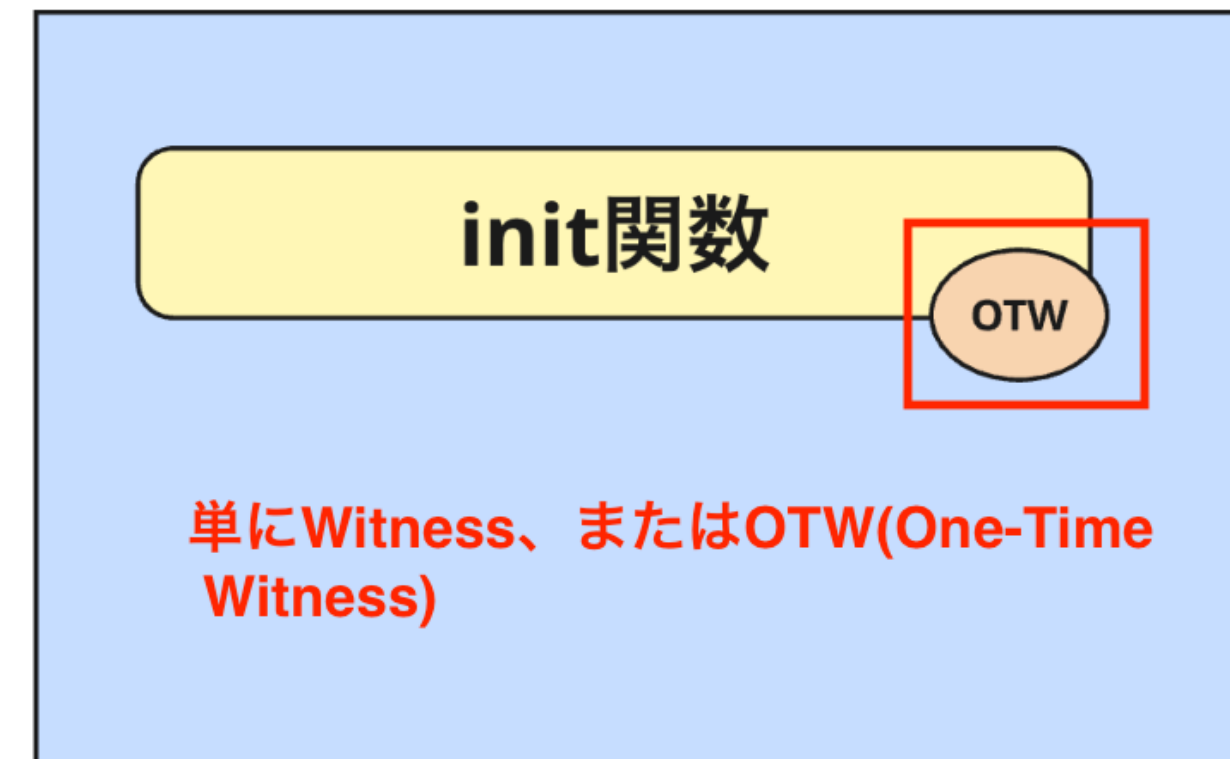
// The type identifier of coin. The coin will have a type
// tag of kind: `Coin<package_object::mycoin::MYCOIN>`
// Make sure that the name of the type matches the module's name.
public struct MY_COIN_NEW has drop {}

// Module initializer is called once on module publish. A `TreasuryCap` is sent
// to the publisher, who then controls minting and burning. `MetadataCap` is also
// sent to the Publisher
fun init(witness: MY_COIN_NEW, ctx: &mut TxContext) {
  let (builder, treasury_cap) = coin_registry::new_currency_with_otw(
    witness,
    6, // Decimals
    b"MY_COIN".to_string(), // Symbol
    b"My Coin".to_string(), // Name
    b"Standard Unregulated Coin".to_string(), // Description
    b"https://example.com/my_coin.png".to_string(), // Icon URL
    ctx,
  );

  let metadata_cap = builder.finalize(ctx);

  // Freezing this object makes the metadata immutable, including the title, name, and icon
  // If you want to allow mutability, share it with public_share_object instead.
  transfer::public_transfer(treasury_cap, ctx.sender());
  transfer::public_transfer(metadata_cap, ctx.sender());
}
```

my_coin_newモジュール



<https://docs.sui.io/guides/developer/currency>

開発19 OTWの作成

コイン作成の例で先ほどの内容を学びましょう。

まずは、otwを作成します。 **dropアビリティ**がついています。

```
Home my_coin_new.move ×  
1  module empty::my_coin_new;  
2  
3  
4  public struct MY_COIN_NEW has drop {}  
5  
6
```

開発20 Structと可視性

現在、**Structは必ずpublicが必要**です。
将来の可視性に備えているためです。

Struct declarations 構造体宣言

Currently, struct declarations can only be `public`, so it's not necessary to include the visibility modifier. To make room for a future where struct types have visibility other than `public`, the 2024 edition of Move will require `public` on all struct declarations.

現在、構造体宣言は パブリック にしかできないため、可視性修飾子を含める必要はありません。構造体型がパブリック以外の可視性を持つ将来に備える余地を確保するために、2024 年版の Move では、すべての構造体宣言でパブリック が義務付けられます。

<https://blog.sui.io/move-edition-2024-update>

開発21 init関数

公開時に一度だけ実行されるのが**init関数**です。
witnessを引数に取ることで**一度のみの実行**であることを示します。

```
Home my_coin_new.move ✕  
1  module empty::my_coin_new;  
2  
3  use sui::coin_registry;  
4  
5  public struct MY_COIN_NEW has drop {}  
6  
7  fun init(witness: MY_COIN_NEW, ctx: &mut TxContext) {  
8  
9  }
```

<https://docs.sui.io/guides/developer/currency>

開発22 new_currency_with_otw

コインを作成してみましょう。

「**treasury_cap**」が権限である、**Capability**です。

```
Home my_coin_new.move X
1  module empty::my_coin_new;
2
3  use sui::coin_registry;
4
5  public struct MY_COIN_NEW has drop {}
6
7  fun init(witness: MY_COIN_NEW, ctx: &mut TxContext) {
8      let (builder, treasury_cap) = coin_registry::new_currency_with_otw(
9          witness,
10         6, // Decimals
11         b"MY_COIN".to_string(), // Symbol
12         b"My Coin".to_string(), // Name
13         b"Standard Unregulated Coin".to_string(), // Description
14         b"https://example.com/my_coin.png".to_string(), // Icon URL
15         ctx,
16     );
17
18 }
```

開発23 metadata_capの取得

finalize関数によって、メタデータの修正権限である、「**metadata_cap**」を取得します。

```
Home my_coin_new.move ×
1  module empty::my_coin_new;
/home
3  use sui::coin_registry;
4
5  public struct MY_COIN_NEW has drop {}
6
7  fun init(witness: MY_COIN_NEW, ctx: &mut TxContext) {
8      let (builder, treasury_cap) = coin_registry::new_currency_with_otw(
9          witness,
10         6, // Decimals
11         b"MY_COIN".to_string(), // Symbol
12         b"My Coin".to_string(), // Name
13         b"Standard Unregulated Coin".to_string(), // Description
14         b"https://example.com/my_coin.png".to_string(), // Icon URL
15         ctx,
16     );
17     let metadata_cap = builder.finalize(ctx);
18
19 }
```

開発24 Capabilityの送付

発行権限、メタデータの修正権限を実行者に送ります。

```
Home my_coin_new.move X
1  module empty::my_coin_new;
2
3  use sui::coin_registry;
4
5  public struct MY_COIN_NEW has drop {}
6
7  fun init(witness: MY_COIN_NEW, ctx: &mut TxContext) {
8      let (builder, treasury_cap) = coin_registry::new_currency_with_otw(
9          witness,
10         6, // Decimals
11         b"MY_COIN".to_string(), // Symbol
12         b"My Coin".to_string(), // Name
13         b"Standard Unregulated Coin".to_string(), // Description
14         b"https://example.com/my_coin.png".to_string(), // Icon URL
15         ctx,
16     );
17     let metadata_cap = builder.finalize(ctx);
18
19     transfer::public_transfer(treasury_cap, ctx.sender());
20     transfer::public_transfer(metadata_cap, ctx.sender());
21
22 }
```


開発25 public, entry関数

でもこんなふうにやればできます。
これってSuiVisionでできないのでしょうか？

```
base ~ (5.455s)
sui client call \
  --package 0x2 \
  --module coin \
  --function mint_and_transfer \
  --type-args 0x4dfcb9827820316b9e81a685de76921e13ebf71c093baaafbcea273241cb945c::my_coin_new::MY_COIN_NEW \
  --args 0x402ebf3fab8c8a64bf1c2faa1db9818d179e9bab3dc8fe2897ba20d1ea405f45 10000 0x917bb0d388ae4557da41a3d8fa4b04eb31f68e01325770f8fe10f35bc0c9bb2f

[warning] Client/Server api version mismatch, client api version : 1.57.2, server api version : 1.59.0
Transaction Digest: CcccL7wsRq41VjGPQViPVzXXXzxc7z3WgG2uG1WqgK99

Transaction Data

Sender: 0x917bb0d388ae4557da41a3d8fa4b04eb31f68e01325770f8fe10f35bc0c9bb2f
Gas Owner: 0x917bb0d388ae4557da41a3d8fa4b04eb31f68e01325770f8fe10f35bc0c9bb2f
Gas Budget: 4434888 MIST
Gas Price: 1000 MIST
Gas Payment:
```

開発26 0x2パッケージ

0x2パッケージで実行してみましょう。

The screenshot displays the Suivision testnet interface. On the left, a sidebar lists various transaction blocks, with 'coin' highlighted. The main area shows the source code for the 'coin' package, which defines a fungible token system. The code includes comments in Japanese and uses Sui's native types and functions. On the right, the 'Execute' tab is active, showing the execution of the 'mint_and_transfer' function. The arguments are: Type0 (0xf0bcb4e4c92f1e966918e793c85d07ce604939b3101421ae43c2e3f4d9385f1b::my_coin_new::MY_COIN_NEW), Arg0 (TreasuryCap: 0x28456cb129952ada3de7ac59ff388f904e32485a03bd995c1408b83d81676e4a), Arg1 (amount: 111), and Arg2 (recipient: 0x0863a92345a2f9f9fc88e659cbcc983b2512425a2f2ffe02edbb5da4e6915cb4).

```
1 // Copyright (c) Mysten Labs, Inc.
2 // SPDX-License-Identifier: Apache-2.0
3
4 /// Defines the `Coin` type - platform wide representation of fungible
5 /// tokens and coins. `Coin` can be described as a secure wrapper around
6 /// `Balance` type.
7 module sui::coin {
8     use std::string;
9     use std::ascii;
10    use std::option::{Self, Option};
11    use sui::balance::{Self, Balance, Supply};
12    use sui::tx_context::TxContext;
13    use sui::object::{Self, UID};
14    use sui::transfer;
15    use sui::url::{Self, Url};
16    use std::vector;
17
18    /// A type passed to create_supply is not a one-time witness.
19    const EBadWitness: u64 = 0;
20    /// Invalid arguments are passed to a function.
21    const EInvalidArg: u64 = 1;
22    /// Trying to split a coin more times than its balance allows.
23    const ENotEnough: u64 = 2;
24
25    /// A coin of type `T` worth `value`. Transferable and storable
26    struct Coin<phantom T> has key, store {
27        id: UID,
28        balance: Balance<T>
29    }
30
31    /// Each Coin type T created through `create_currency` function will have a
32    /// unique instance of CoinMetadata<T> that stores the metadata for this coin type.
33    struct CoinMetadata<phantom T> has key, store {
34        id: UID,
35        /// Number of decimal places the coin uses
36    }
```

<https://testnet.suivision.xyz/package/0x2?tab=Code>

開発27 entry関数の作成

以下のようにmint関数を作ります。

<MY_COIN_NEW>で型を設定しています。

```
Home my_coin_new.move X Object Explorer
1  module empty::my_coin_new;
2
3  use sui::coin;
4  use sui::coin_registry;
5
6  public struct MY_COIN_NEW has drop {}
7
8  > fun init(witness: MY_COIN_NEW, ctx: &mut TxContext) { ...
22 }
23
24 ∨ entry fun mint(
25     cap: &mut coin::TreasuryCap<MY_COIN_NEW>,
26     amount: u64,
27     recipient: address,
28     ctx: &mut TxContext
29 ∨ ) {
30     let new_coin = coin::mint<MY_COIN_NEW>(cap, amount, ctx);
31     transfer::public_transfer(new_coin, recipient);
32 }
```

開発28 SuiVisionでの実行

SuiVisionで実行ができることを確認しましょう。

Publisher:

0x0863...5cb4

Publish Time:

Oct 24, 2025 02:22:05 +UTC

Last Transaction ID:

FfmB8pvnpKDAHGMfu3PSHarMTWALjULdzV1HGbfMgtLe

Version:

1

Owner:

Immutable

Transaction Blocks

Code

Statistics

my_coin_new

Source

Bytecode

```
1 module 0xbde479c784c71eb37fed2005cfbdb9f555adff2469ac9791524a94a0ae0e7865::my_coin_new {
2   struct MY_COIN_NEW has drop {
3     dummy_field: bool,
4   }
5
6   entry fun mint(arg0: &mut 0x2::coin::TreasuryCap<MY_COIN_NEW>, arg1: u64, arg2: address, arg3:
7     0x2::transfer::public_transfer<0x2::coin::Coin<MY_COIN_NEW>>(0x2::coin::mint<MY_COIN_NEW>{
8     }
9   }
10  fun init(arg0: MY_COIN_NEW, arg1: &mut 0x2::tx_context::TxContext) {
11    let (v0, v1) = 0x2::coin_registry::new_currency_with_otw<MY_COIN_NEW>(arg0, 6, 0x1::string
12      0x2::transfer::public_transfer<0x2::coin::TreasuryCap<MY_COIN_NEW>>(v1, 0x2::tx_context::s
13      0x2::transfer::public_transfer<0x2::coin_registry::MetadataCap<MY_COIN_NEW>>(0x2::coin_reg
14    )
15  }
16  // decompiled from Move bytecode v6
17 }
18 }
```

Execute

mint

Arg0

0x6af242069e8718fa4fbd54f78527d64c4937193b:

Arg1

1111

Arg2

0x884344b72b6d2d20684b91ce84beaf8b8cbce8:

Execute

SuiVisionBlockchainValidators

Ox0863...5cb4

Object

0x96dd...5dfb

Search by Account, Package, Object, Transaction, SuiNS

Details

Owner:

0x8843...d33b

Publisher:

0x0863...5cb4

Type:

0x2::coin::Coin<0xbde4...7865::my_coin_new::MY_COIN_NEW>

Version:

623364027

Transaction Block Digest:

G1wKwP...ePp8

Fields

```
{
  balance: "1111"
  id: {
    id: "0x96dd0504599d940dab5d783bf6121ffb60dc9057685732d39becaece5fdf5dfb"
  }
}
```

Dynamic Fields

CHILD OBJECT ID	VERSION	OBJECT TYPE	LAST TX ID
-----------------	---------	-------------	------------