Build on Sui

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

Powered by

CAC

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。

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

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。

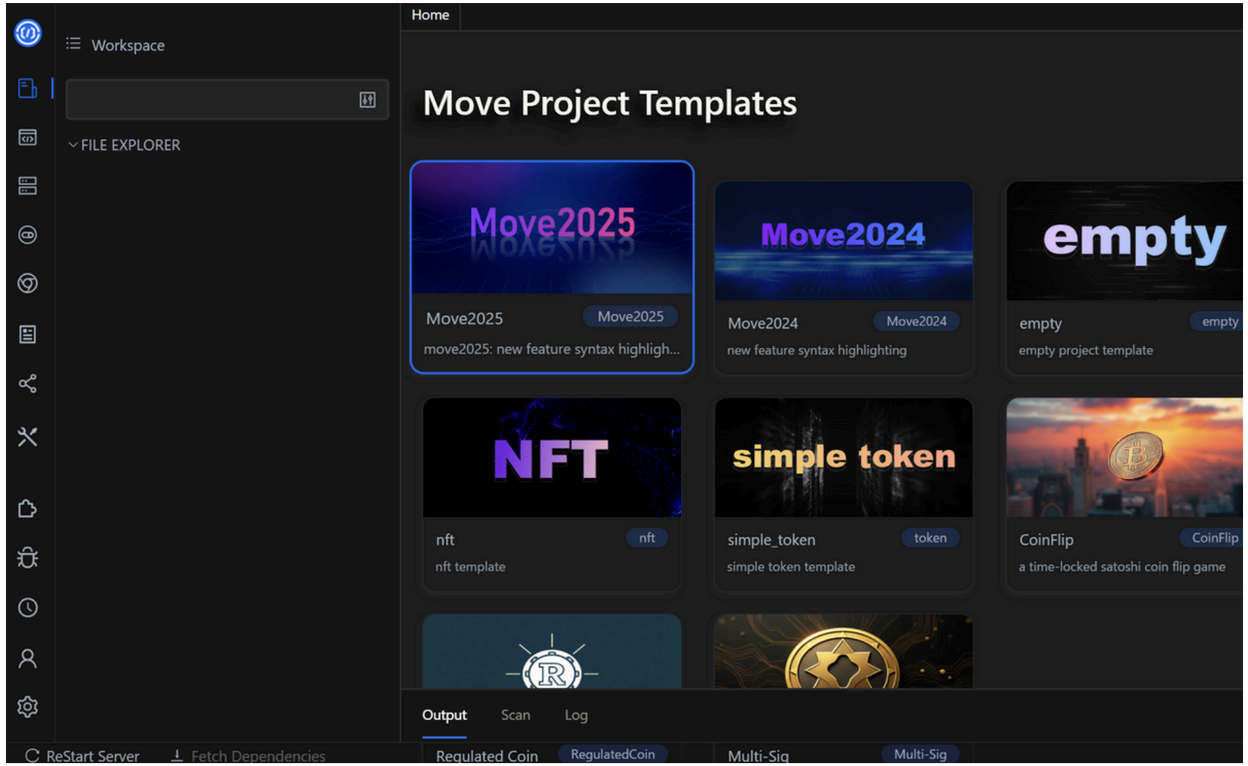
目次

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

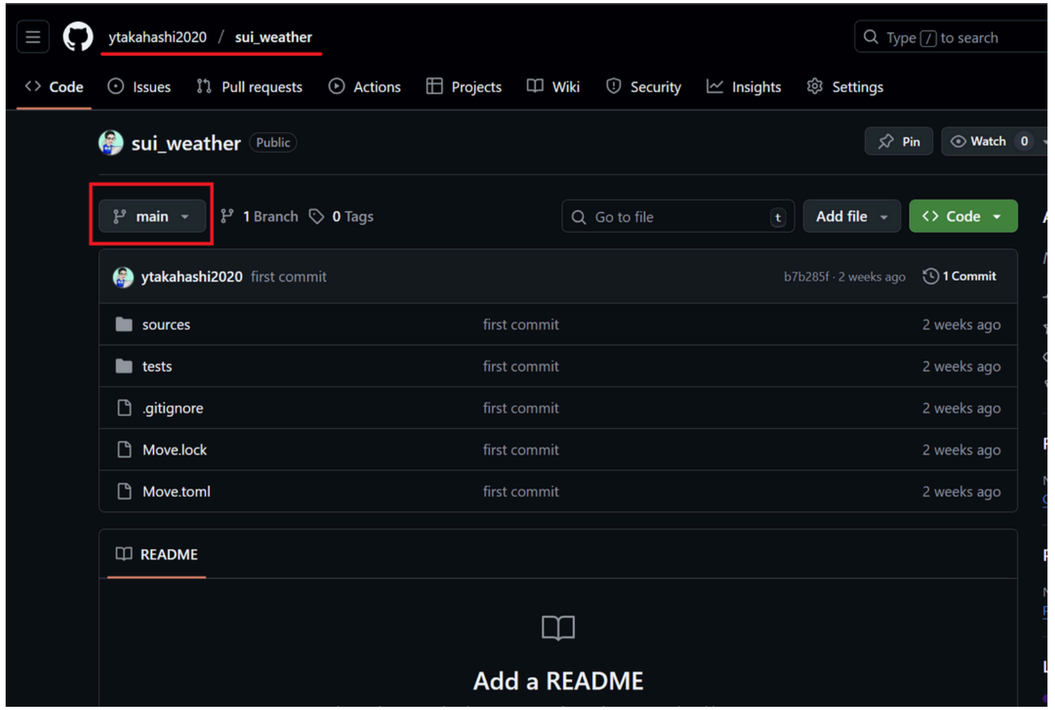
Web IDE（Web Integrated Development Environment）を利用することでブラウザ上で開発をすることができます。
今回は BitsLabを利用します。



https://ide.bitslab.xyz/

2 Githubの利用

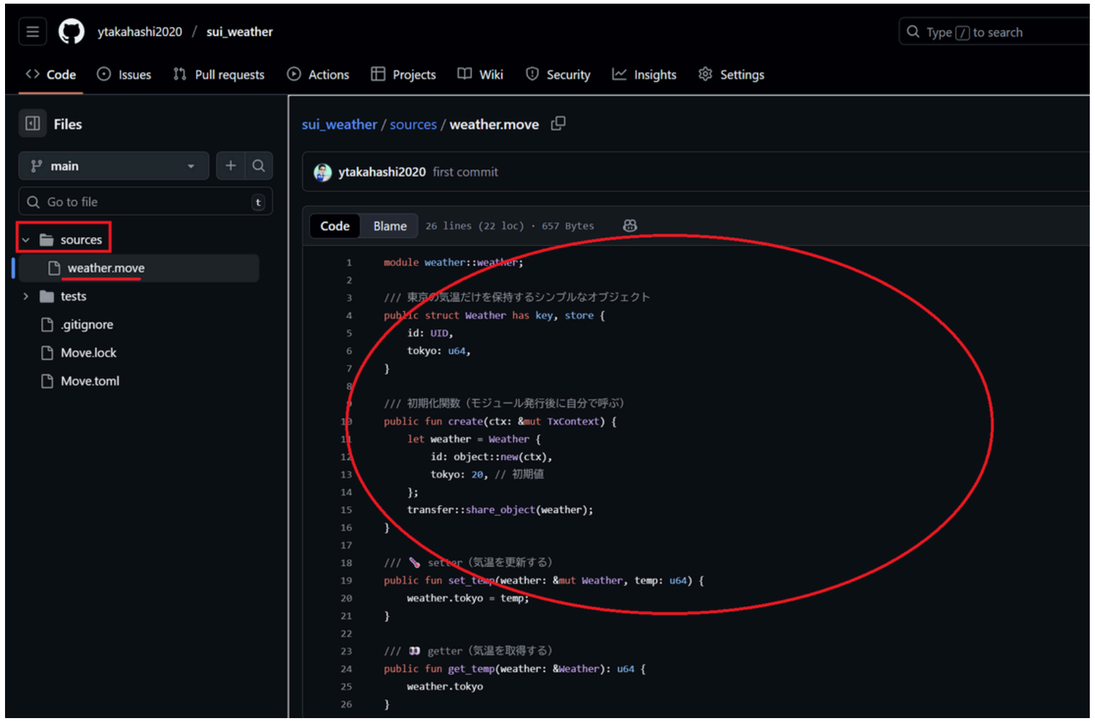
GitHubは、プログラムのコードを保存・共有し、複数の人で開発するための
サービスです。
まずはこのコードを使ってみましょう。



https://github.com/ytakahashi2020/sui_weather

2 Githubの利用

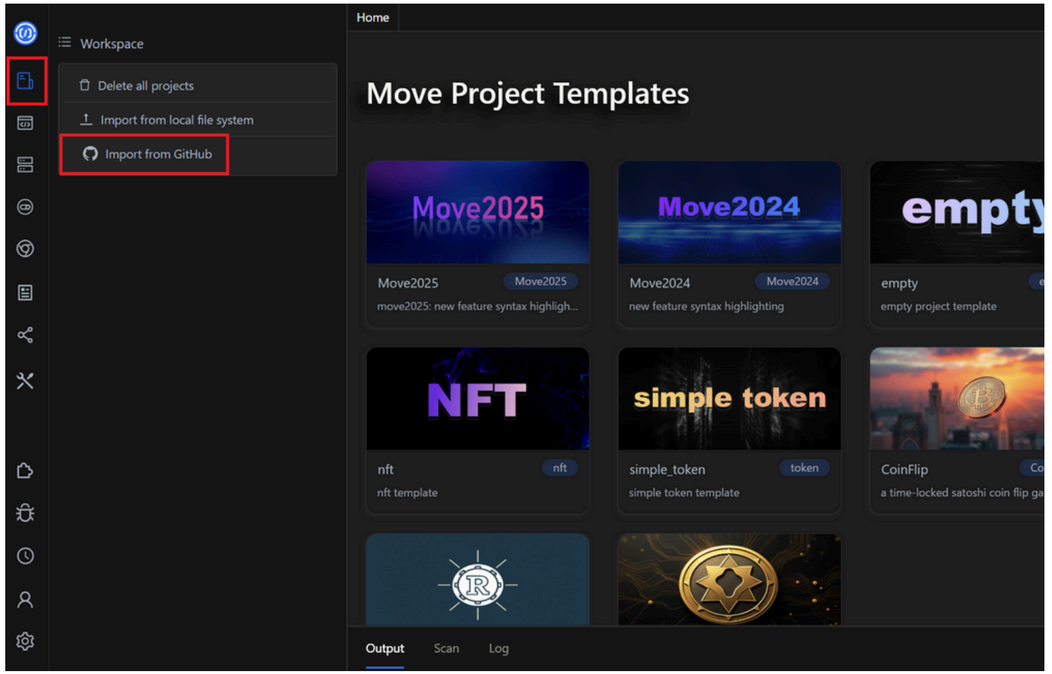
ファイルを開いてみましょう。
こちらは、「Move」というプログラミング言語で書かれています。



https://github.com/ytakahashi2020/sui_weather

３ Githubからのインポート

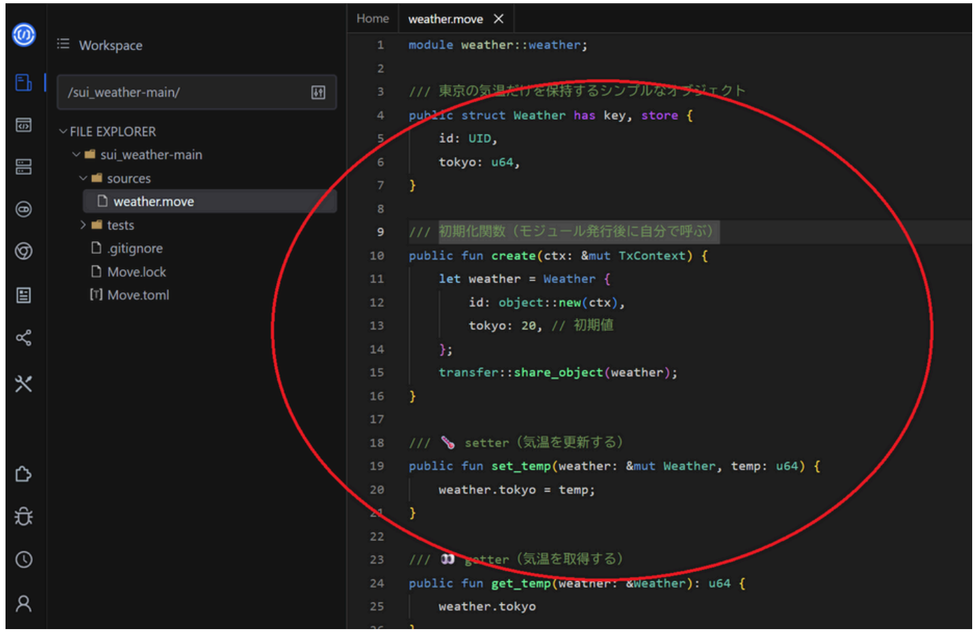
GitHubにあるコードやプロジェクトを、自分の開発環境などに取り込むこと
をインポートと言います。
先ほどのコードをインポートしてみましょう。



https://ide.bitslab.xyz/

3 Githubからのインポート

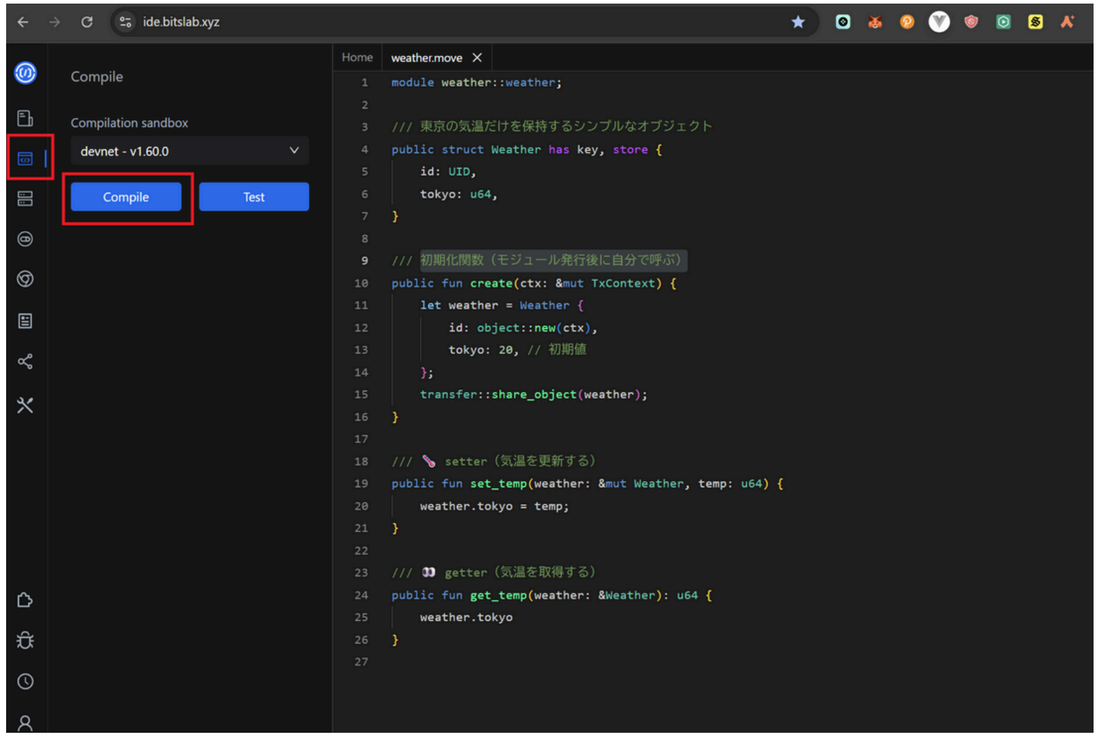
「https://github.com/ytakahashi2020/sui_weather」を設定します。
下のように、インポートを行うことができました。



https://ide.bitslab.xyz/

４ コンパイルの実行

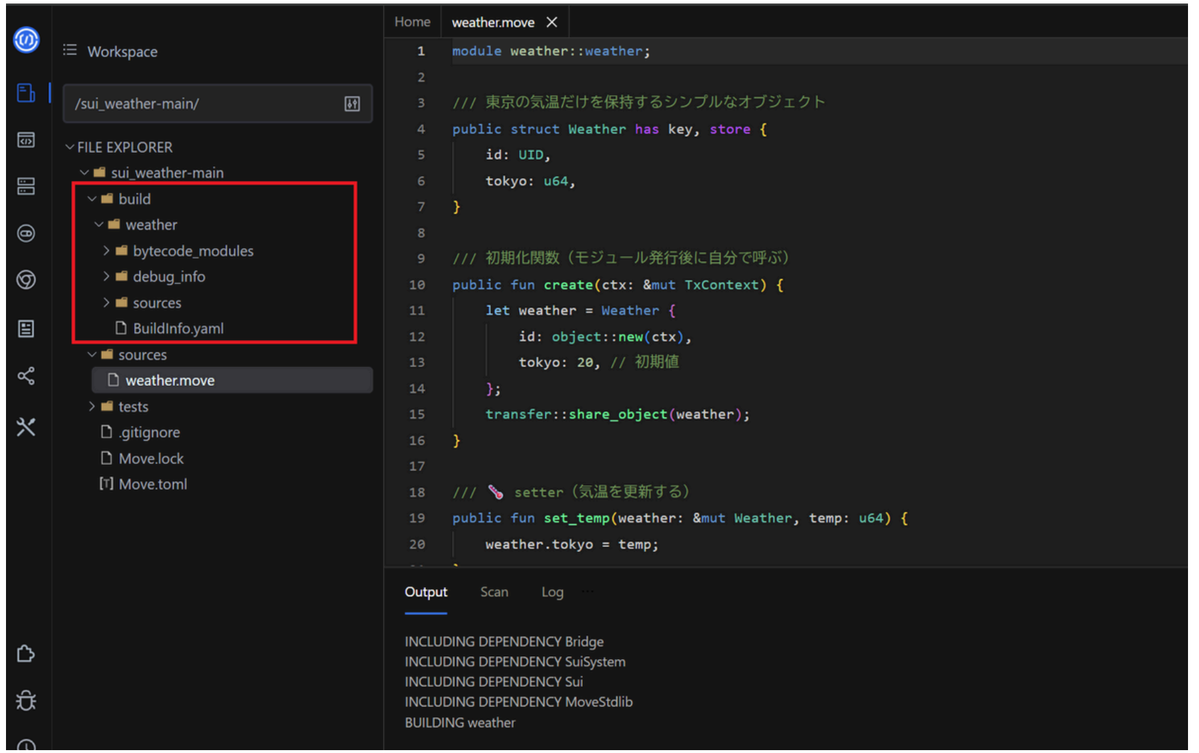
人間が読める形式から機械が読める形式への変換であるコンパイルを行います。



https://ide.bitslab.xyz/

４ コンパイルの実行

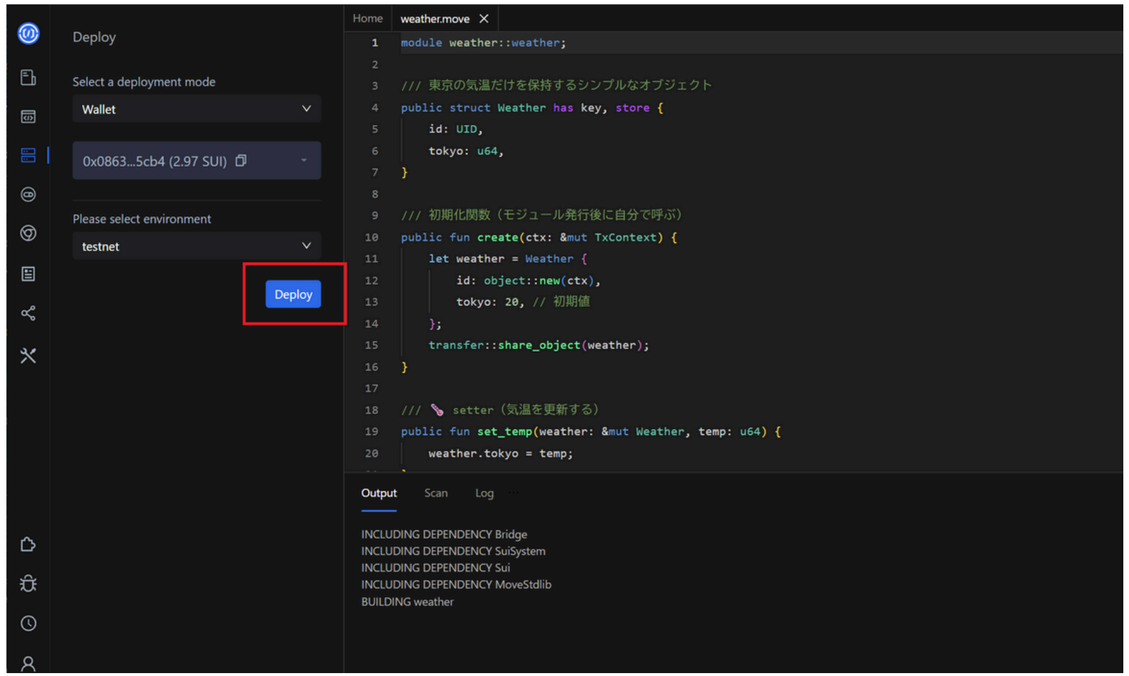
このように「build」というフォルダが作られました。



https://ide.bitslab.xyz/

5 デプロイの実行

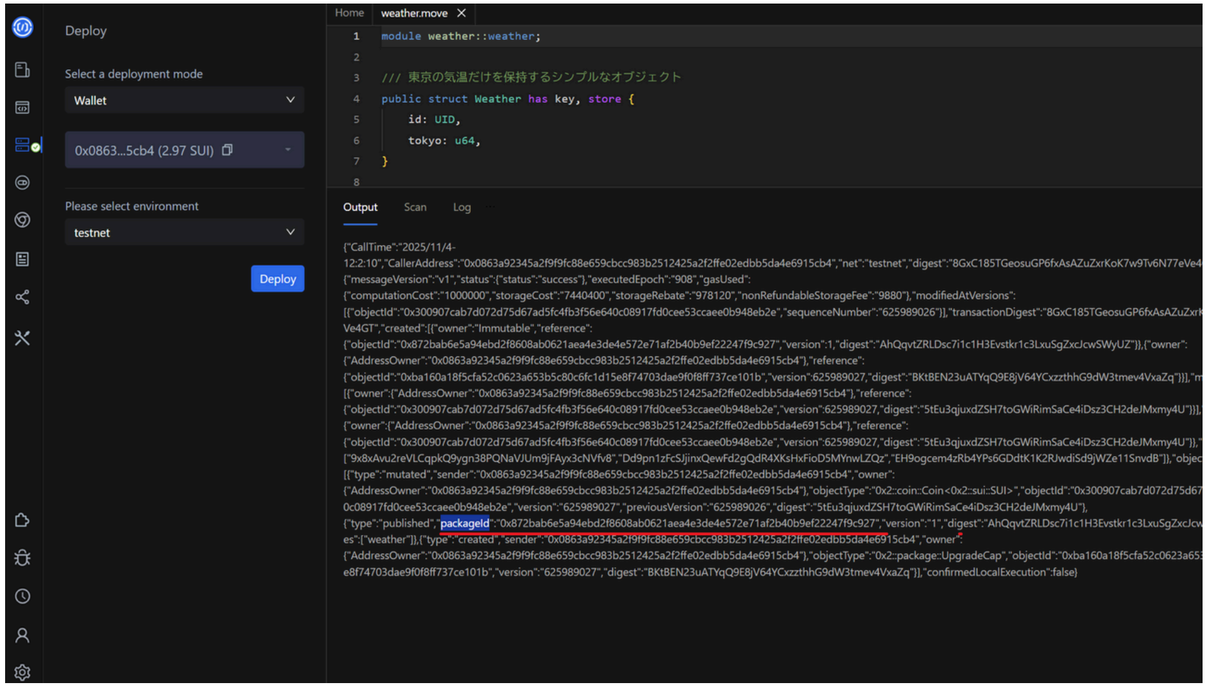
ブロックチェーン上に実装する処理である「デプロイ」を行います。



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。

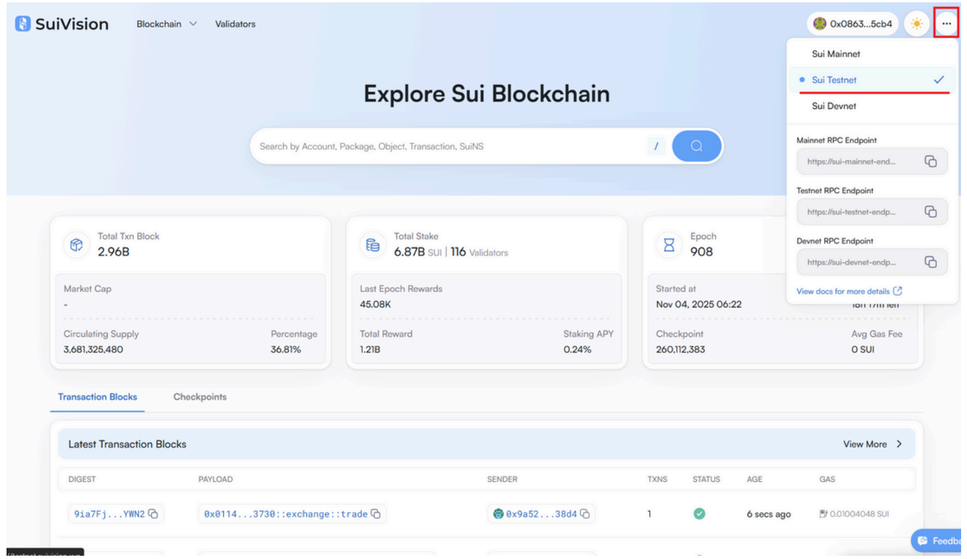


https://ide.bitslab.xyz/

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。

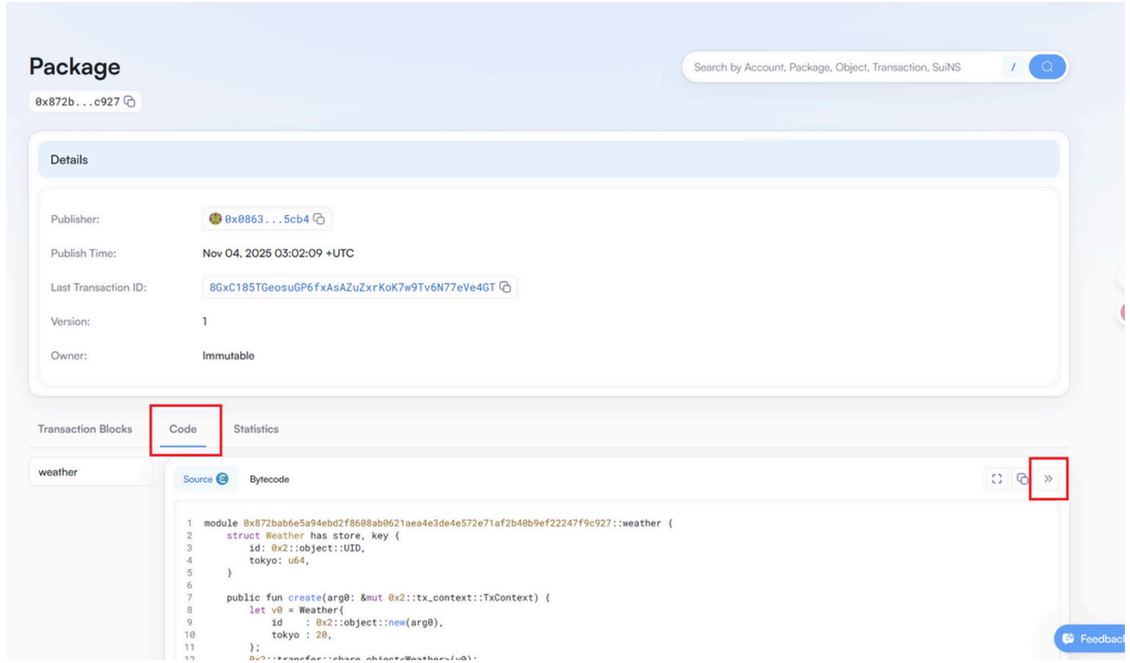


https://testnet.suivision.xyz/

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。

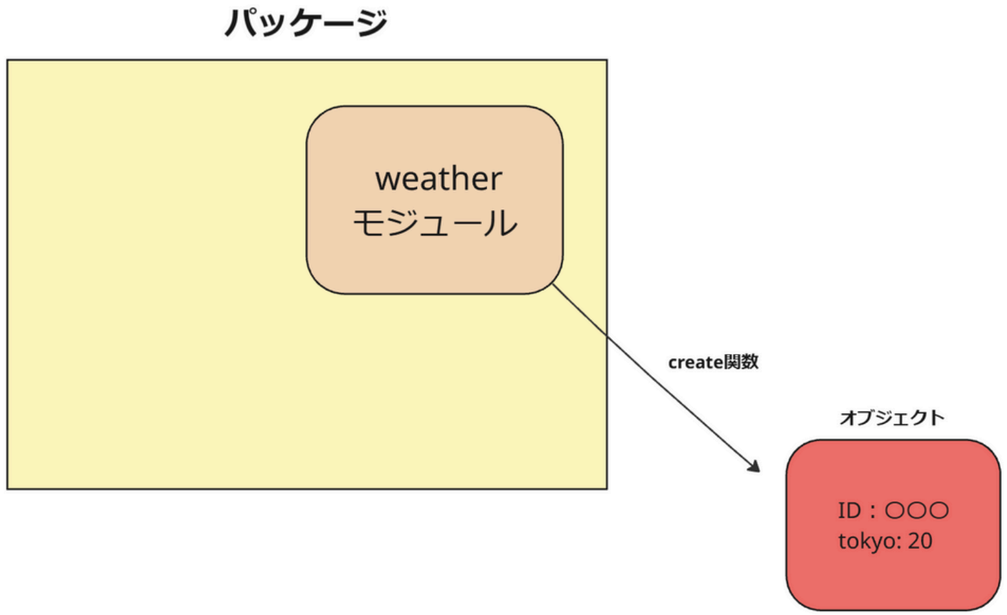
コードも確認ができます。

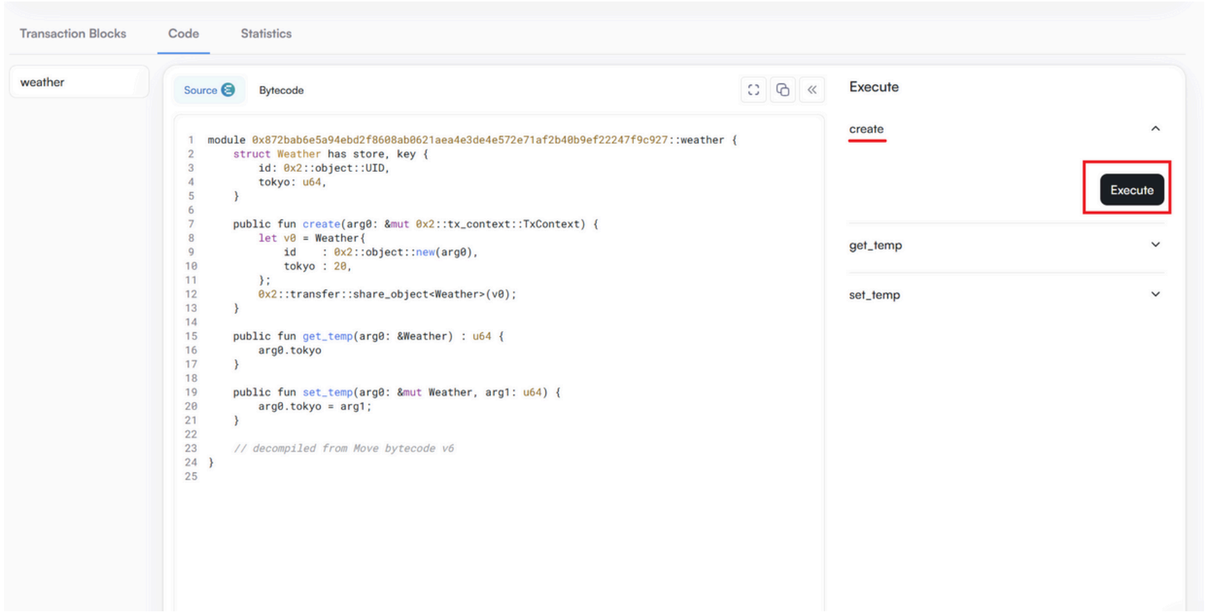


https://testnet.suivision.xyz/

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。

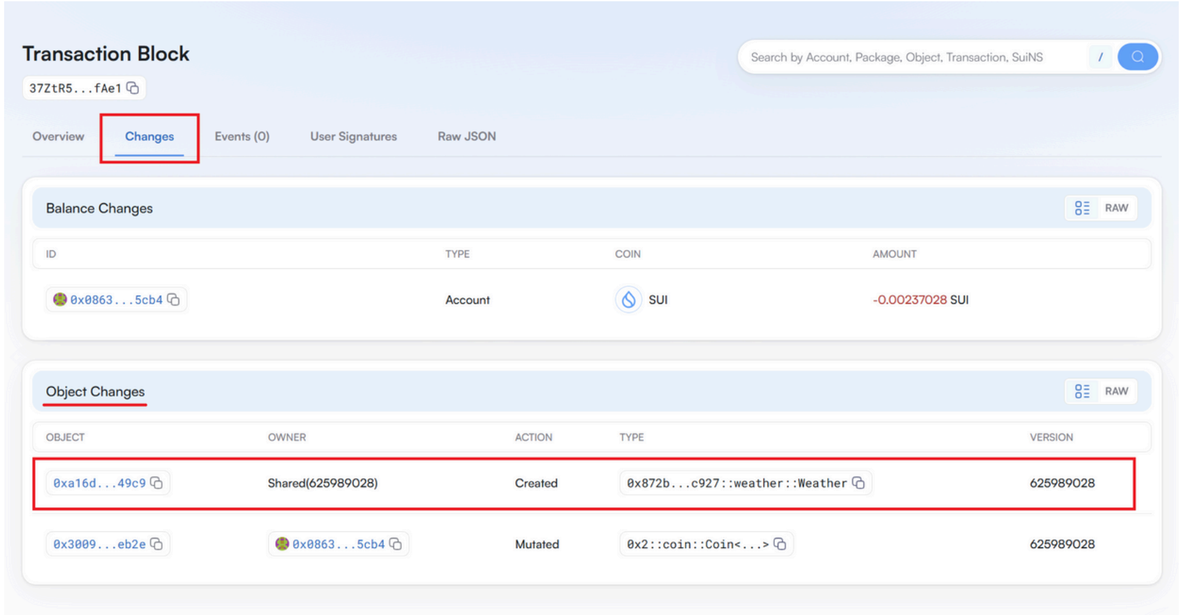




https://testnet.suivision.xyz/

９ トランザクションの確認

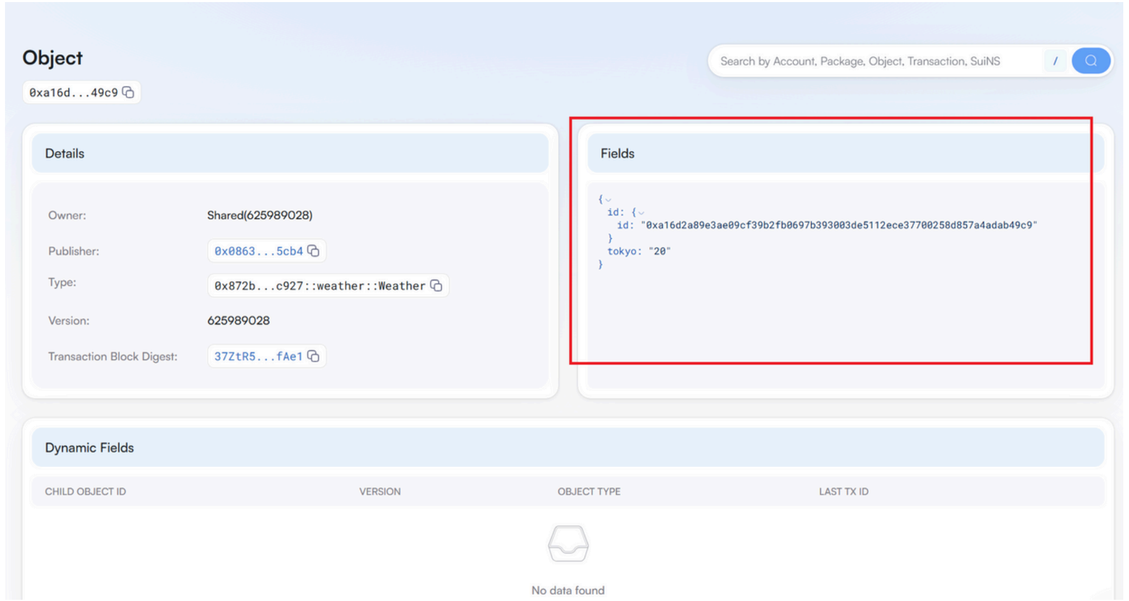
「Changes」からオブジェクトの変更を確認できます。
今回は新しく作られた「Created」を確認します。



https://testnet.suivision.xyz/

９ トランザクションの確認

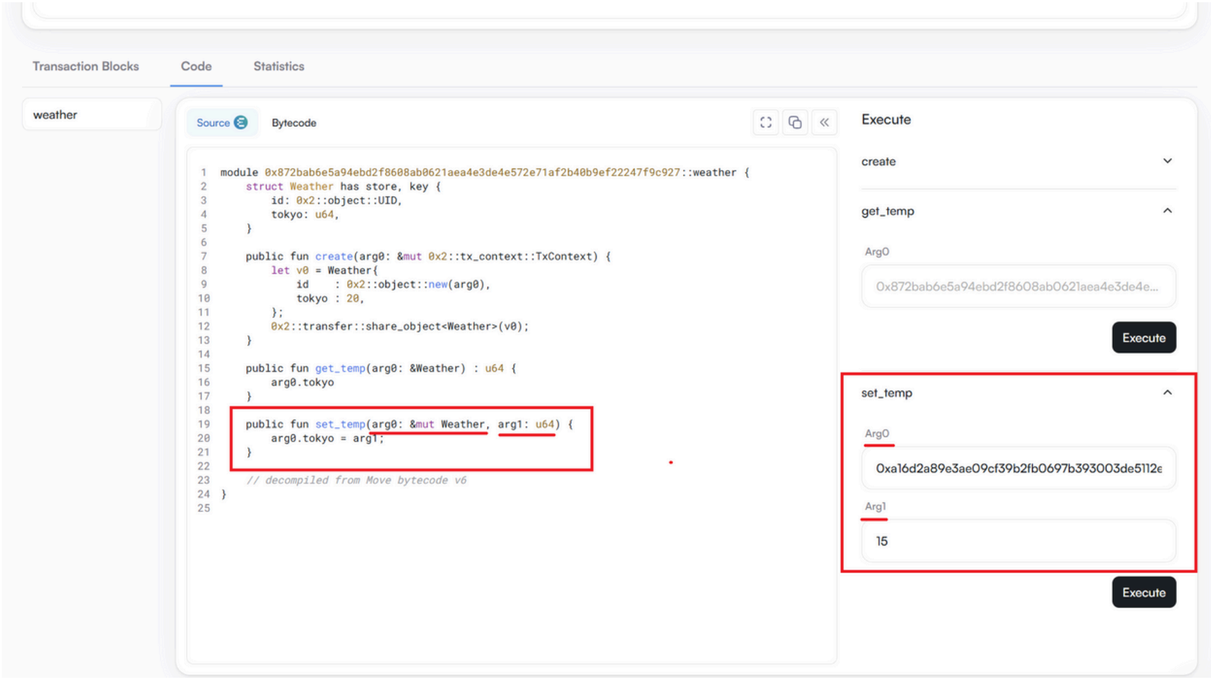
下のように「Fields」を確認すると、IDとtokyoの値が格納されていることがわかります。



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。

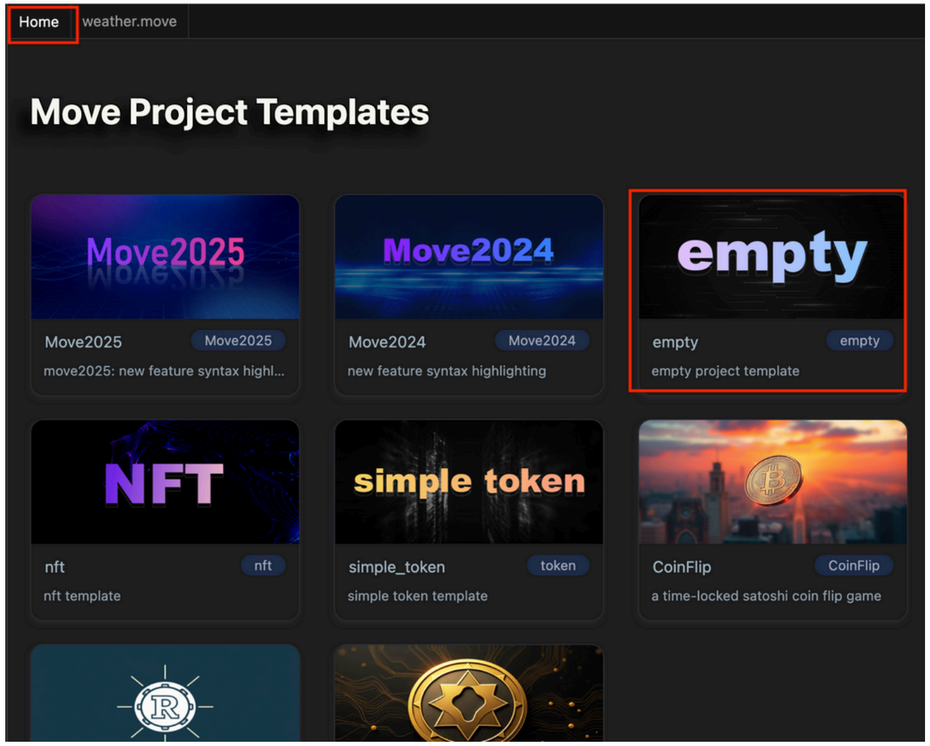


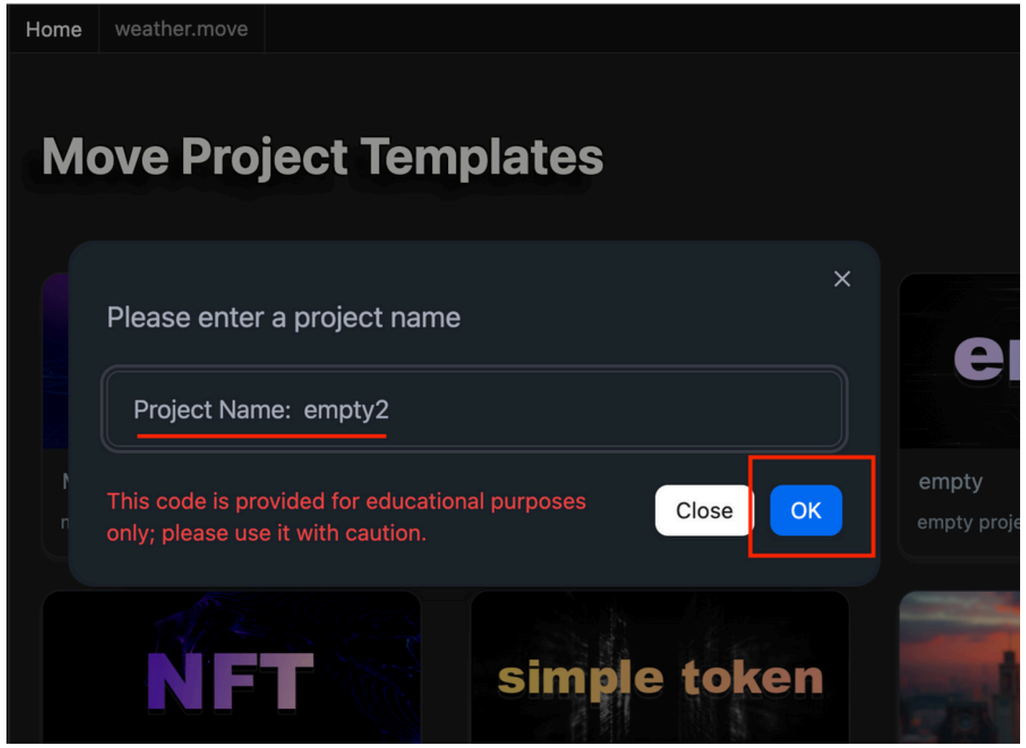
https://testnet.suivision.xyz/

2 実装体験

１ プロジェクトの作成

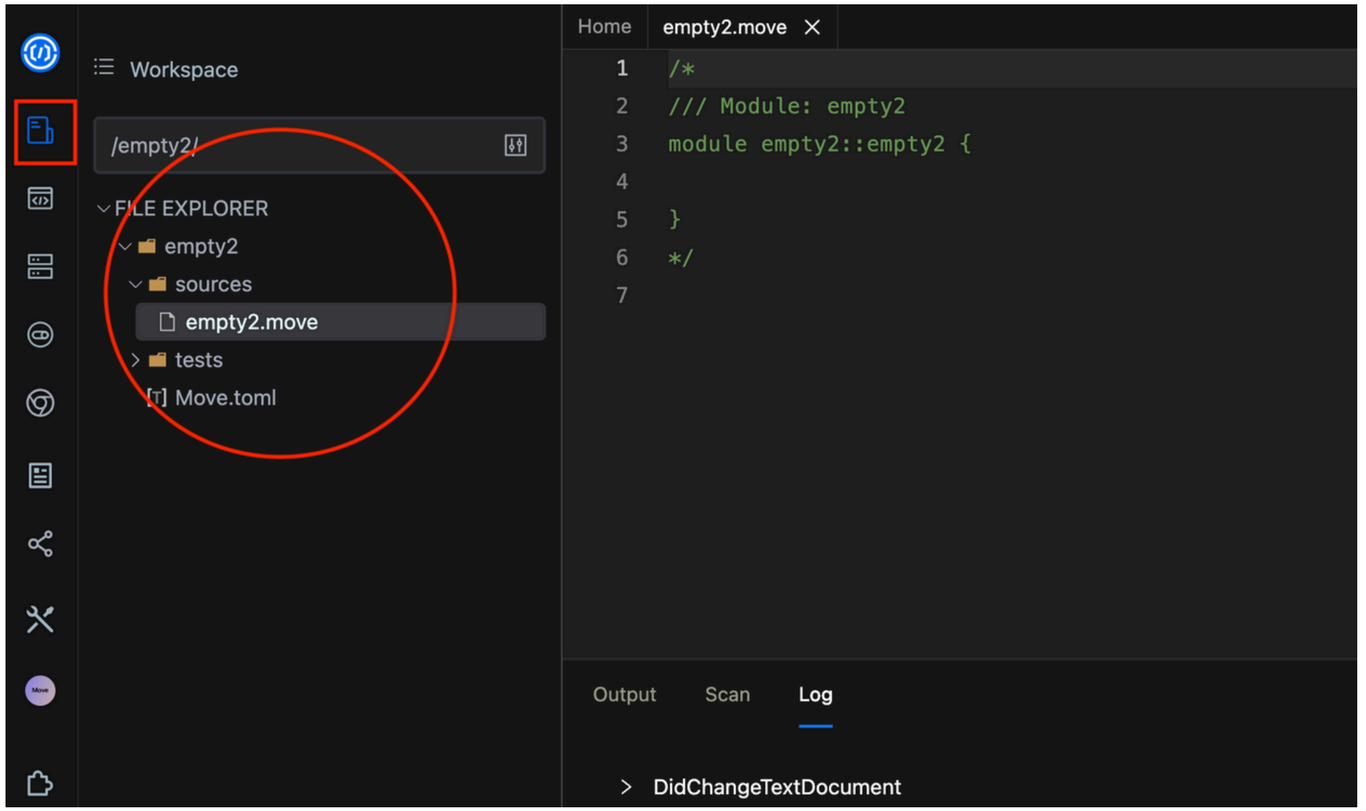
では、１からプロジェクトを作りましょう。
Homeから「empty」を選択します。





１ プロジェクトの作成

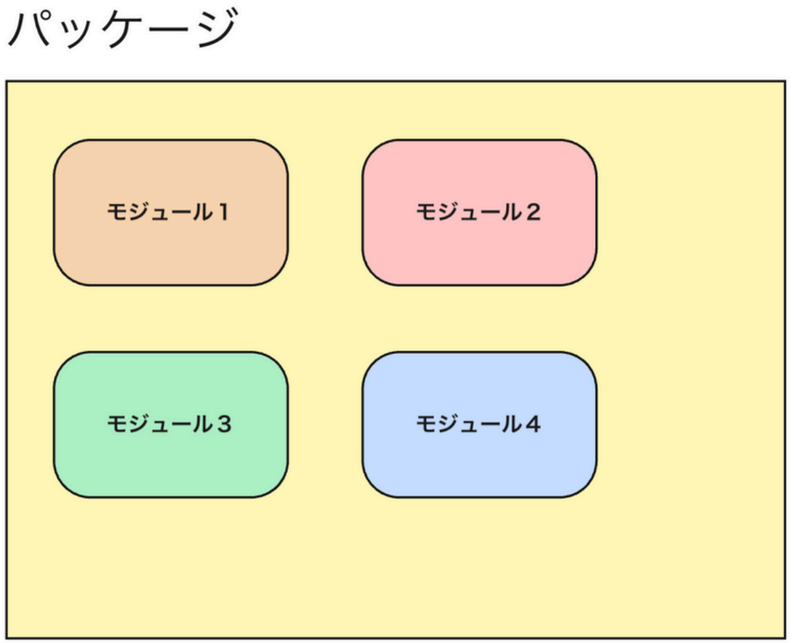
このように空のプロジェクトの雛形ができました。



2 パッケージとモジュール

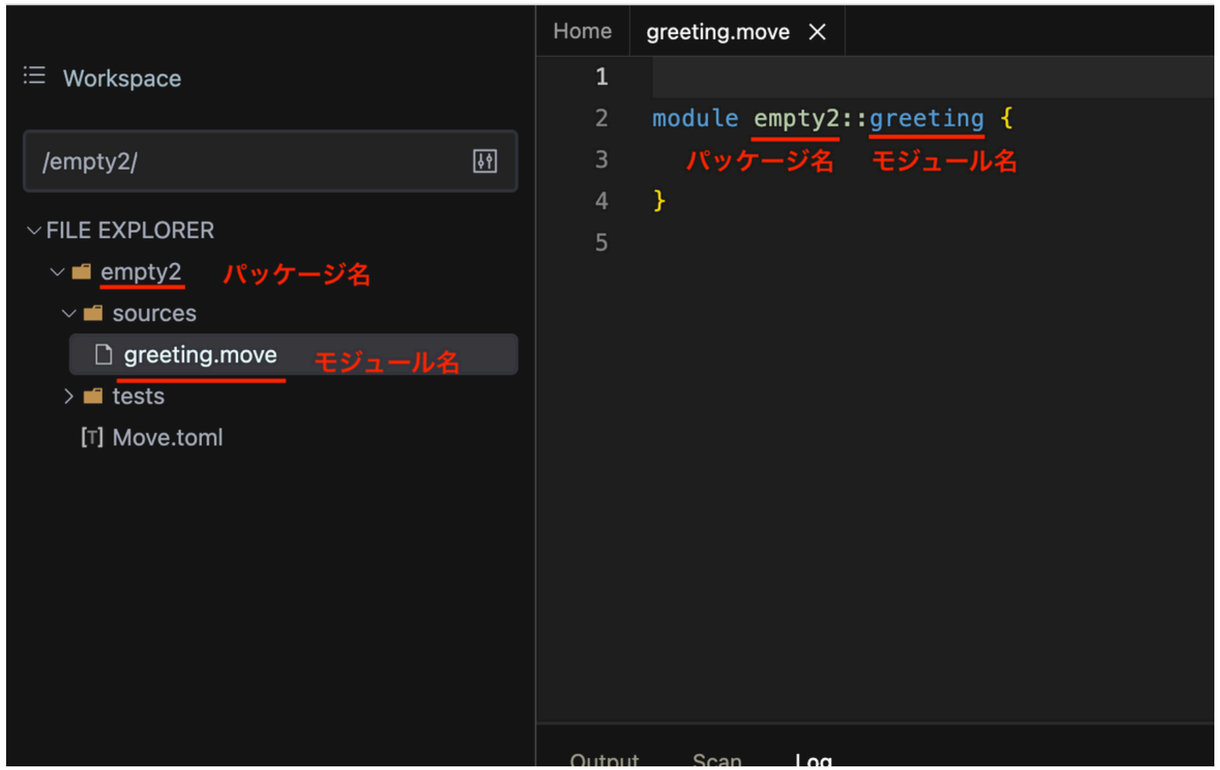
Suiではパッケージの中に複数のモジュールが配置されます。
（とはいえ、多くの場合はモジュールは一つです。）

このモジュールがいわゆるスマートコントラクトです。



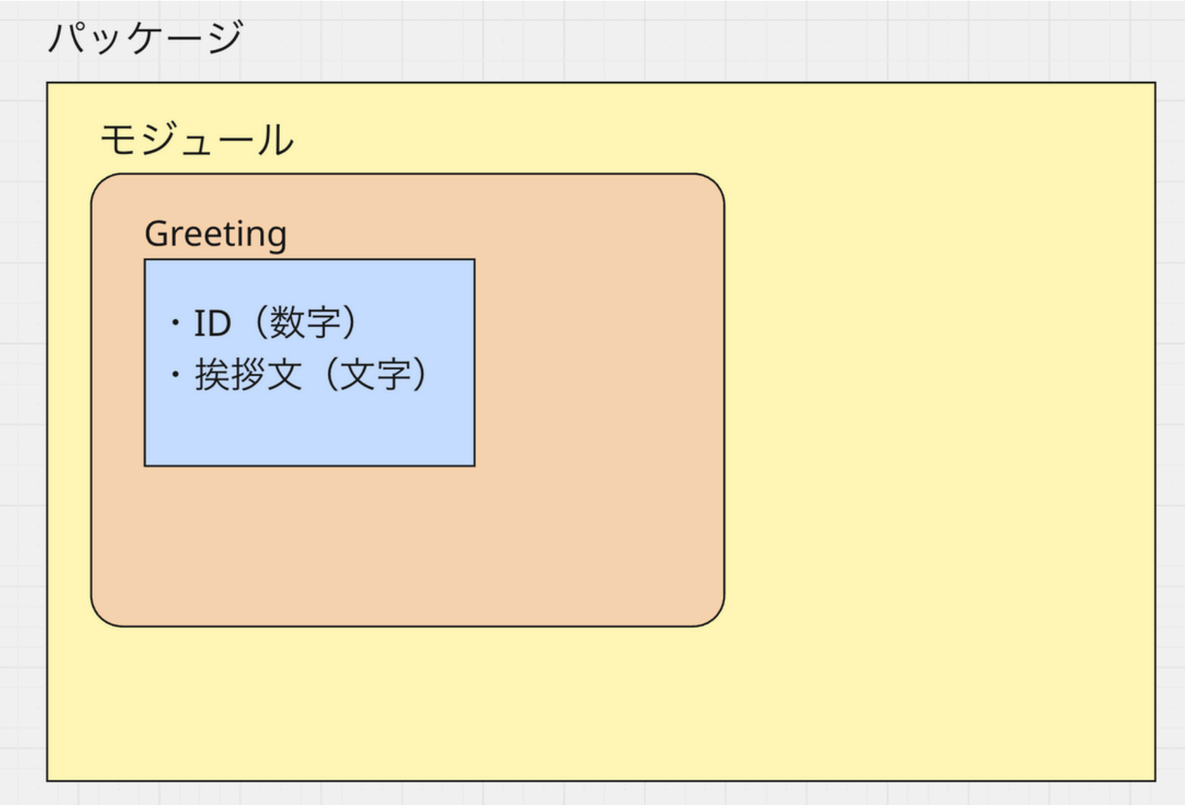
2 パッケージとモジュール

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。



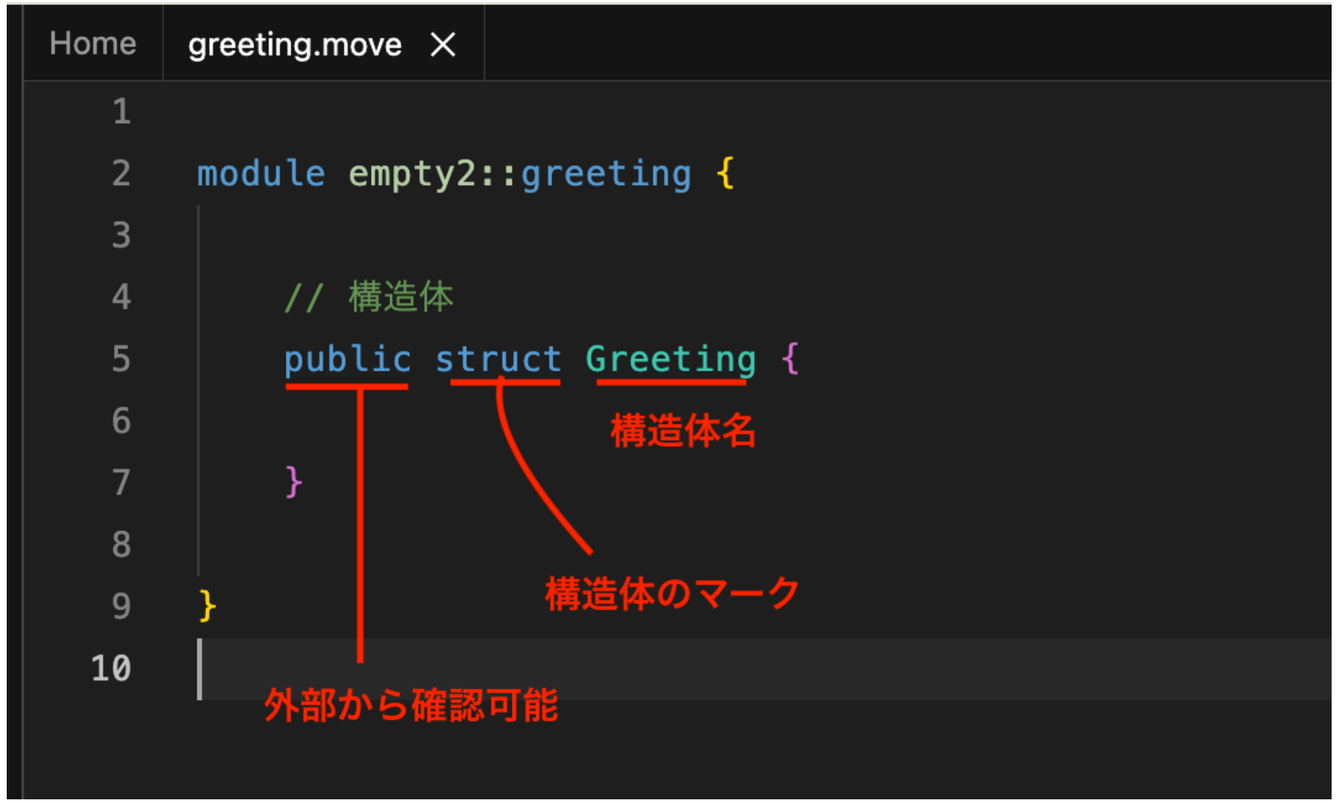
３ 構造体

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。



４ Keyの設定

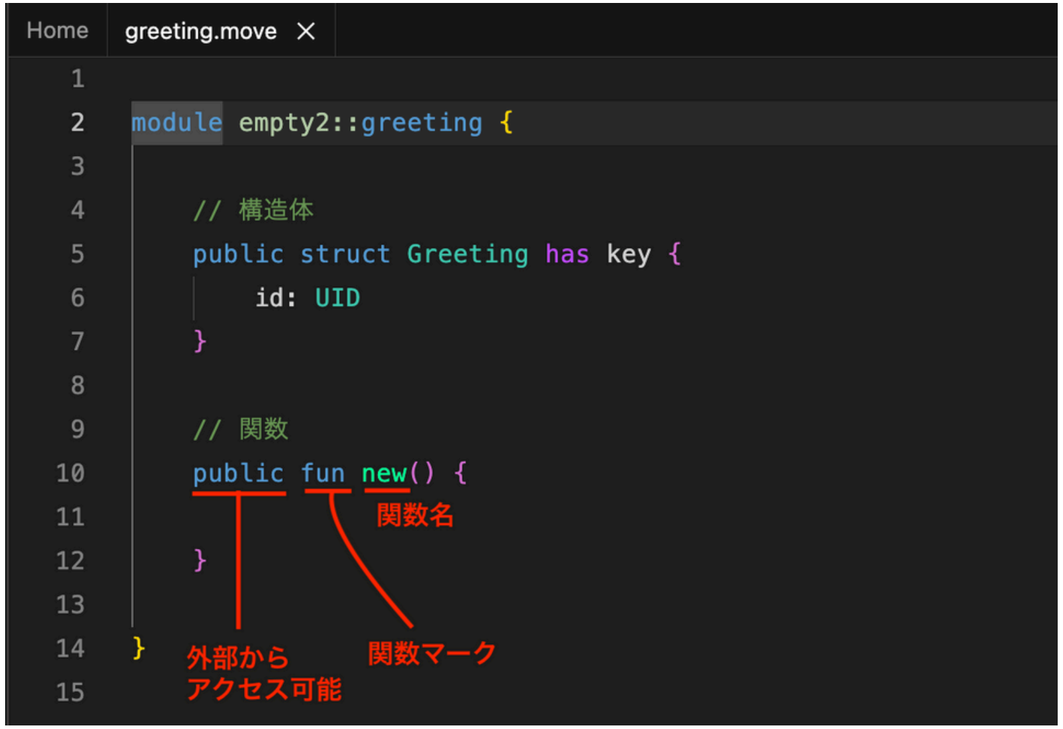
この構造体には４つのアビリティの一つである「key」を設定します。
これでこの構造体が世界で唯一のものとなります。

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。

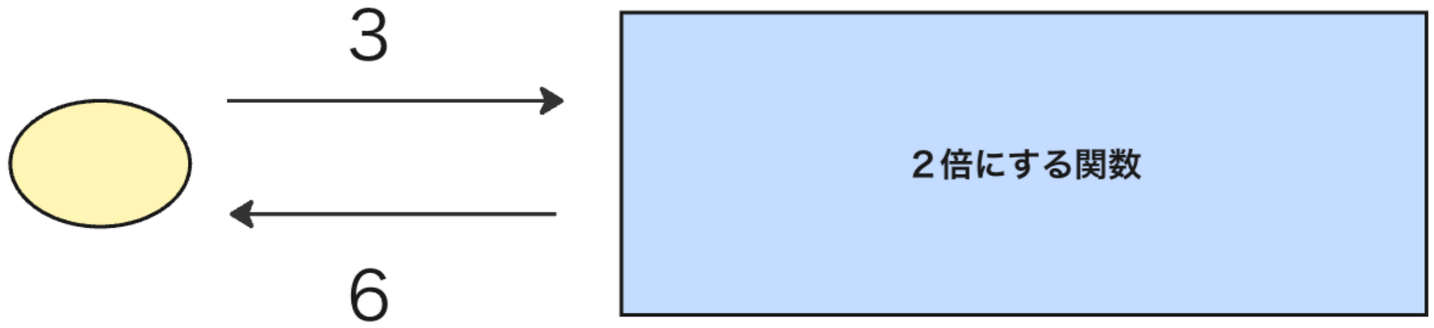
5 関数の作成

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。

```
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。

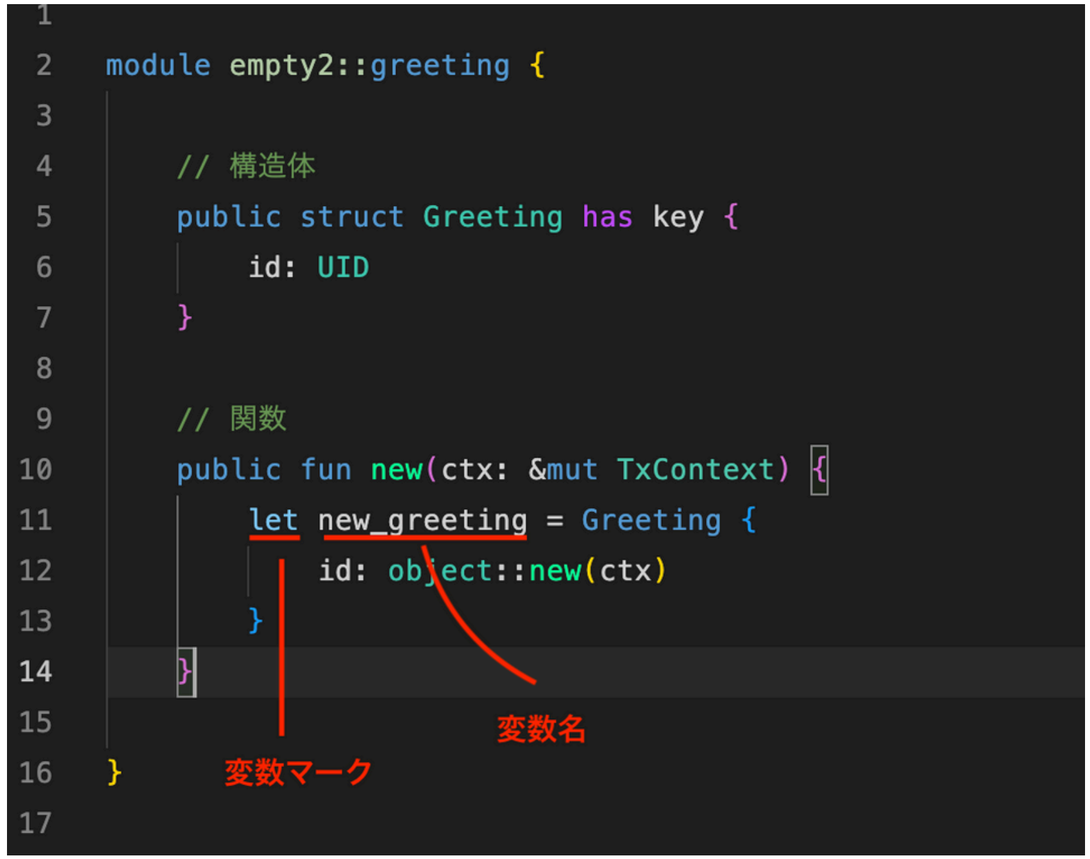
```
1
2  module empty2::greeting {
3
4      // 構造体
5      public struct Greeting has key {
6          id: UID
7      }
8
9      // 関数
10     public fun new(ctx: &mut TxContext) {
11         Greeting {
12             id: 12345
13         }
14     }
15
16 }
17
```

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。

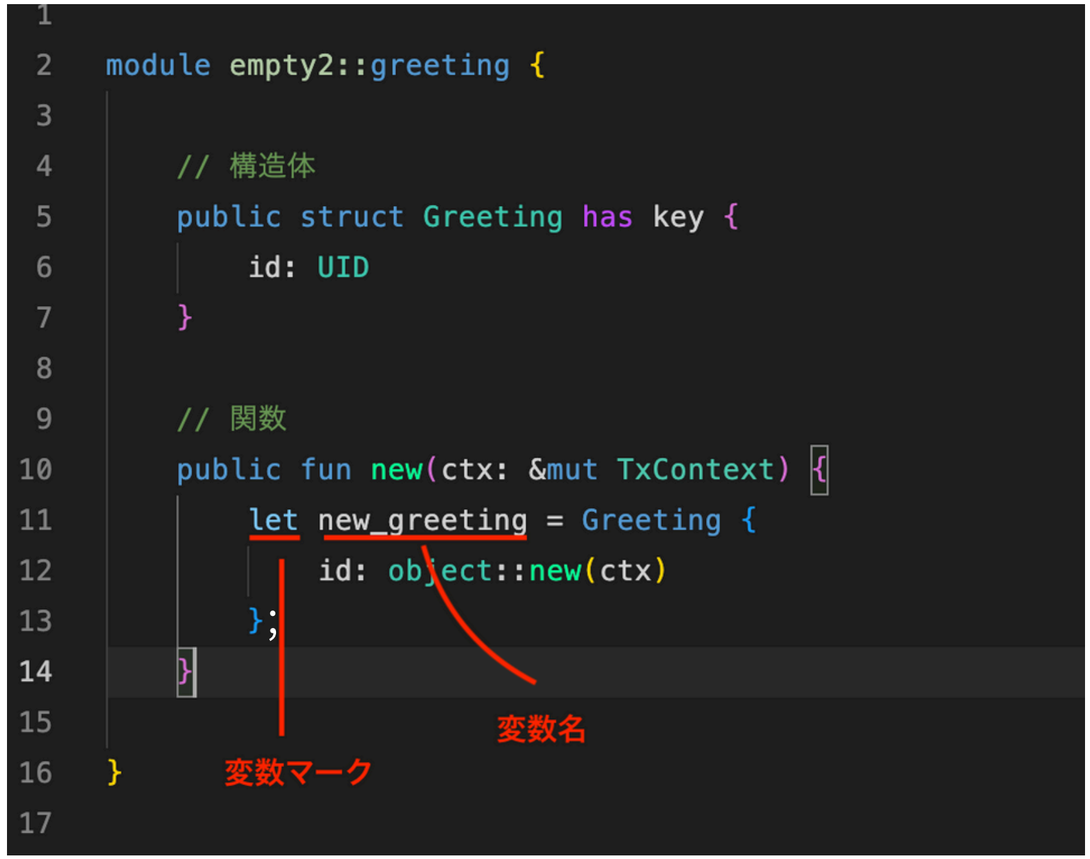
5 関数の作成

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

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

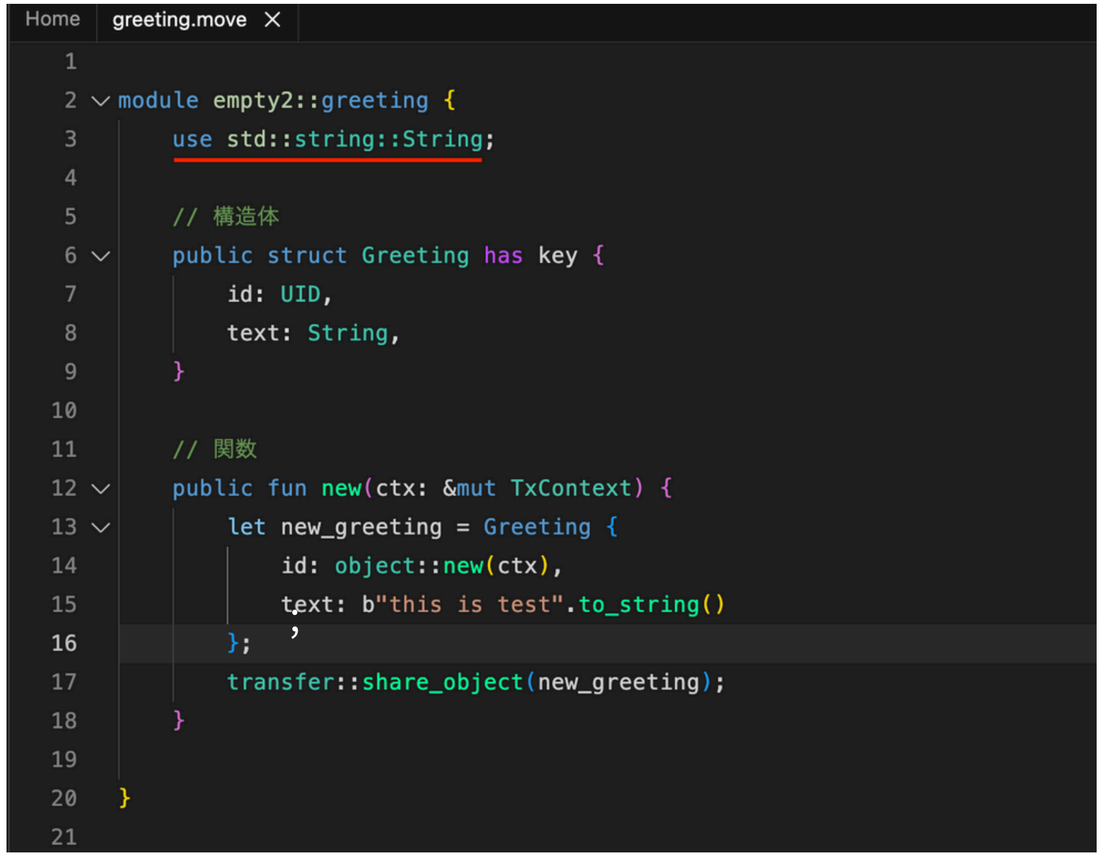
最後に、この構造体の所有者を設定します。
ここでは、共有所有とします。

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 }
18
```

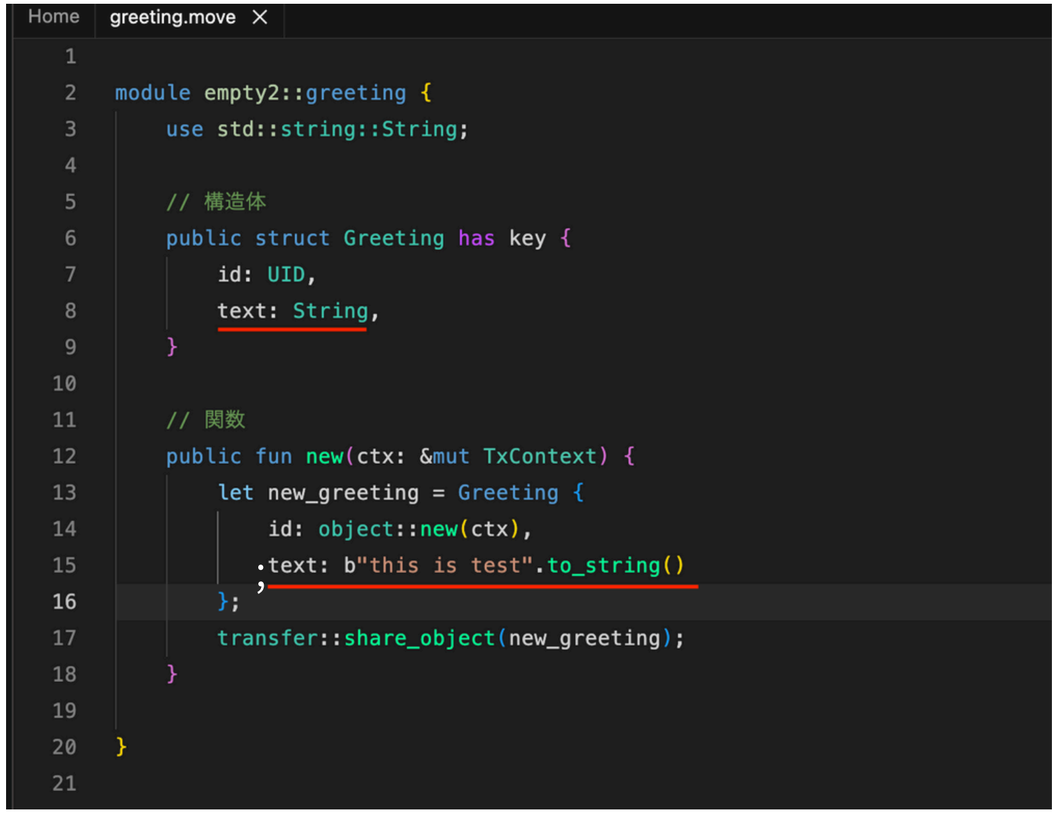
6 文字列の設定

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。



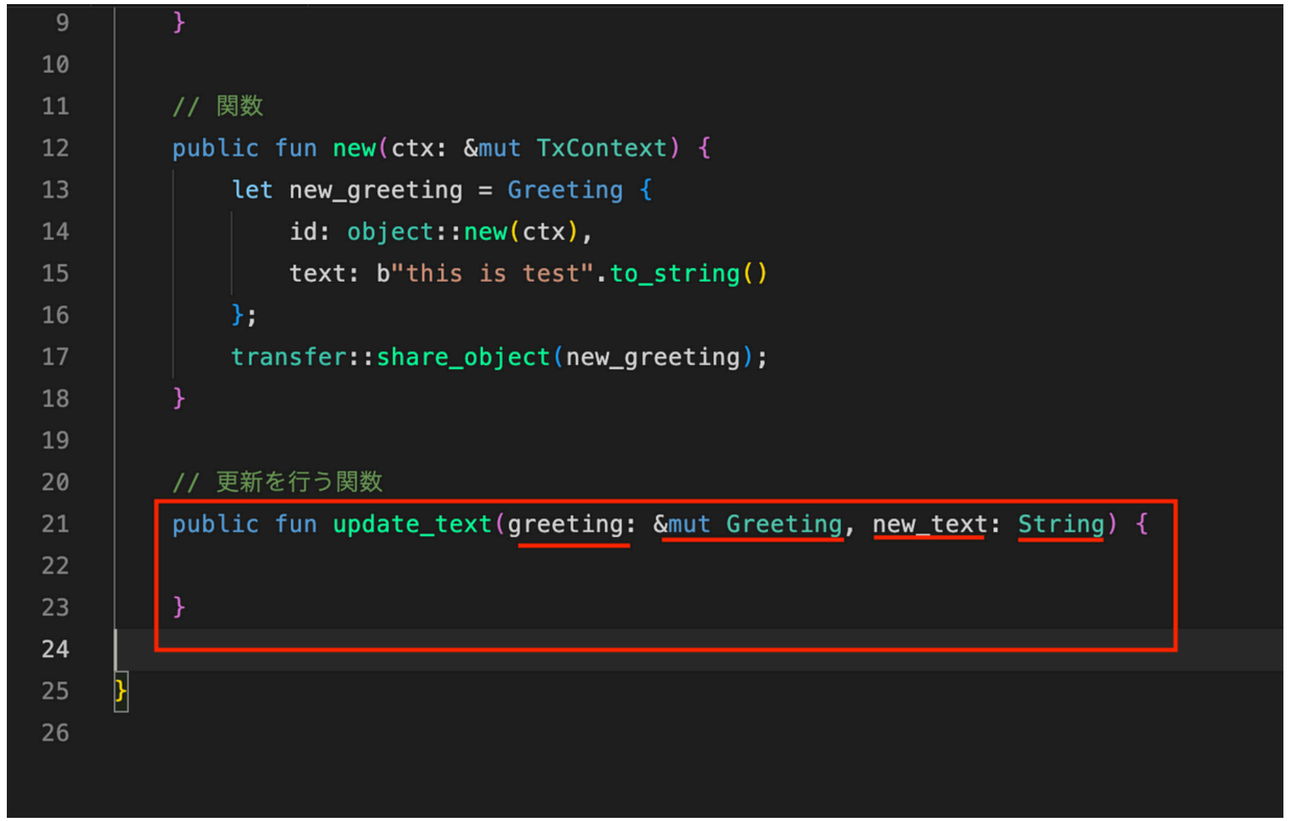
6 文字列の設定

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。



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

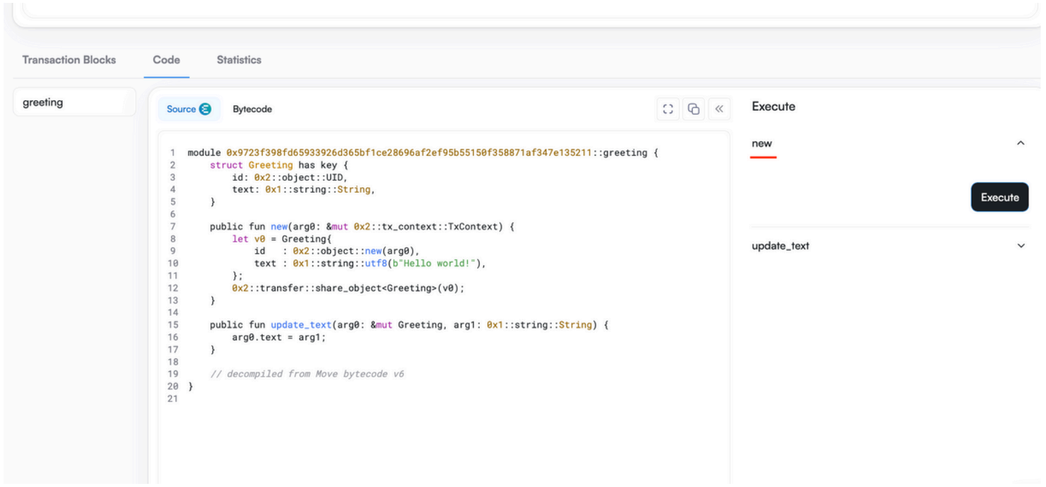
引数のnew_textをgreetingの「text」に代入します。
「=」は代入を表します。

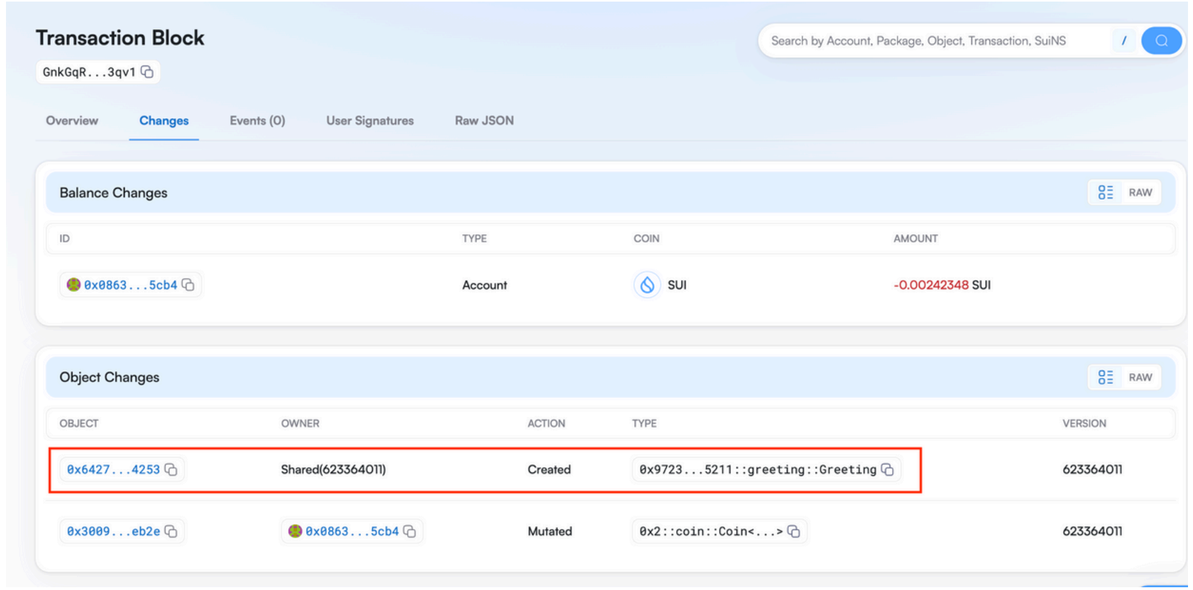
```
4
5       // 構造体
6       public struct Greeting has key {
7           id: UID,
8           text: String,
9       }
10
11      // 関数
12      public fun new(ctx: &mut TxContext) {
13          let new_greeting = Greeting {
14              id: object::new(ctx),
15              text: b"this is test".to_string()
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。

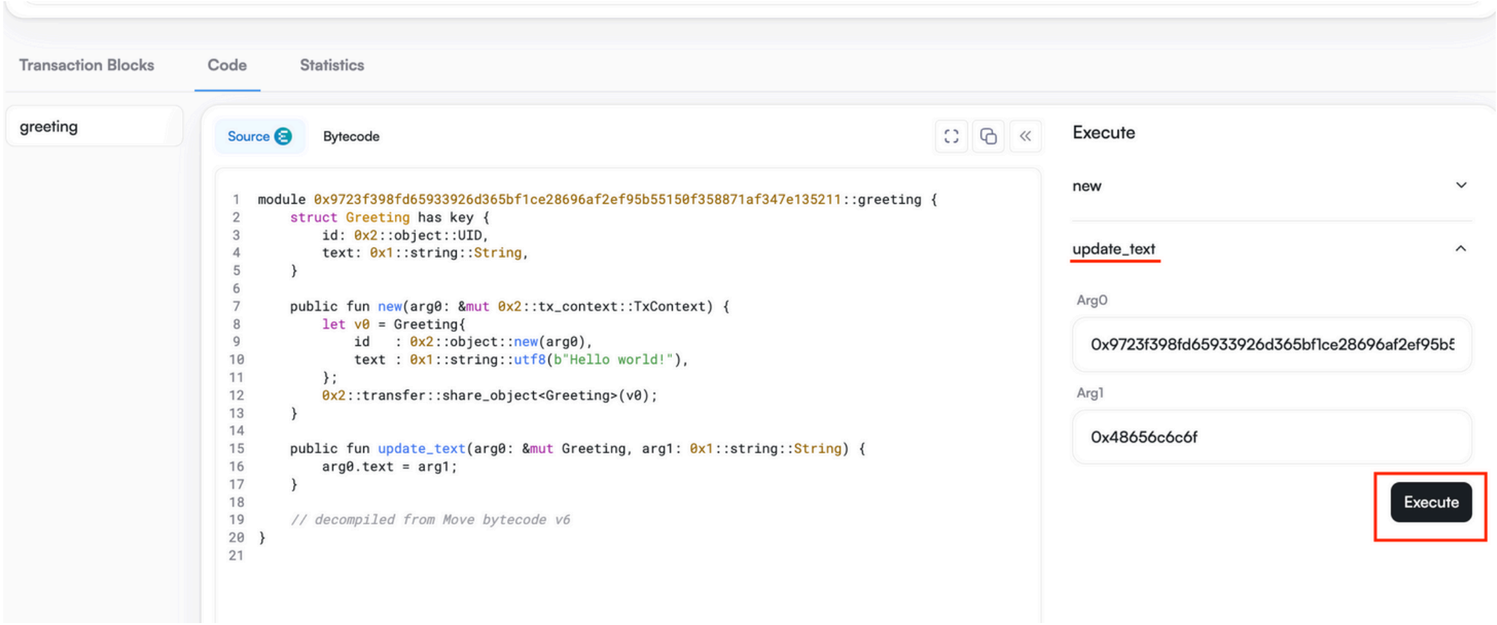




９ 共有オブジェクトの確認

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。

```
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、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。

```
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;
    }
}
```

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。

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

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

    // 作成者 = admin
    public fun new(ctx: &mut TxContext) {
        let board = NoticeBoard {
            id: object::new(ctx),
            admin: ctx.sender(),
            message: string::utf8(b"Welcome to our project!"),
        };
        transfer::share_object(board);
    }

    // admin だけ更新可能
    public fun update_message(
        board: &mut NoticeBoard,
        new_message: string::String,
        ctx: &TxContext,
    ) {
        assert!(ctx.sender() == board.admin, 1);
        board.message = new_message;
    }
}
```

assert!!(条件, エラーコード)

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。

```
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

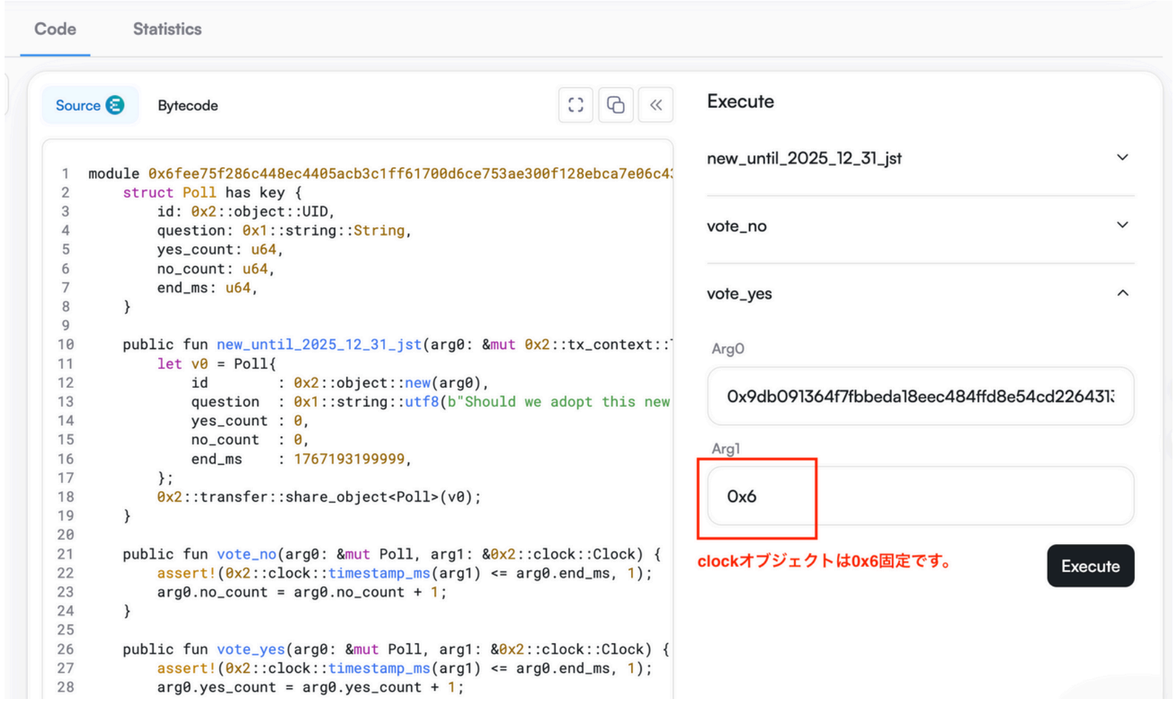
@をつけることで直接アドレスを指定する事ができます。

```
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         };
```

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。

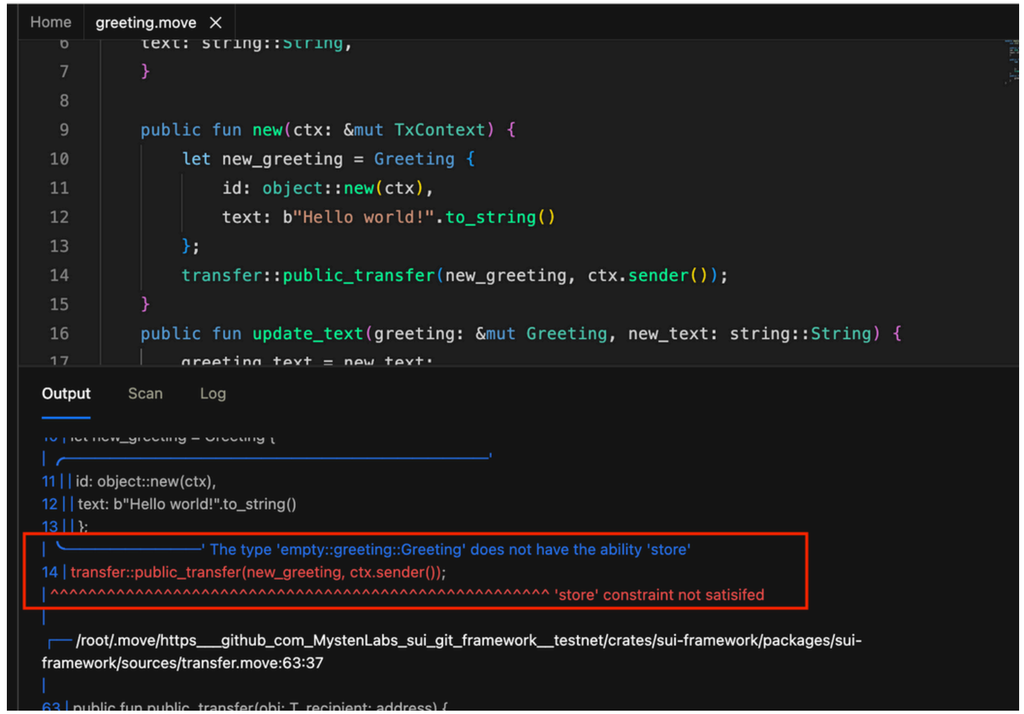
```move
module poll4::simple_poll {
    use std::string;
    use sui::clock::{Self, Clock};

    // 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;
    }
```



予備資料

開発11　public transferとstore

コンパイルしようとすると、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？



開発11　public transferとstore

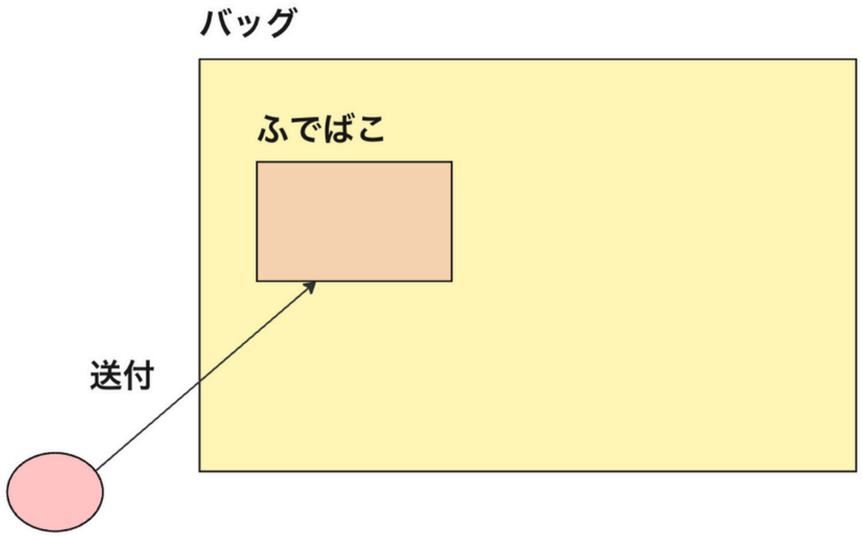
ルール上、送れないようです。

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

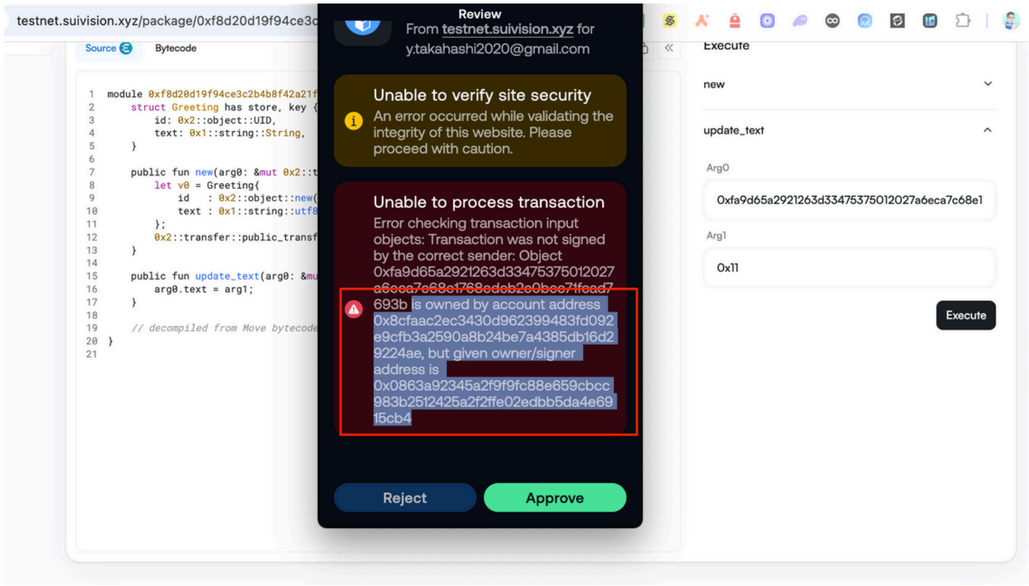
開発12　所有者以外で実行できない確認

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

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

    public struct Greeting has key, store {
    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;
    }
}
```



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

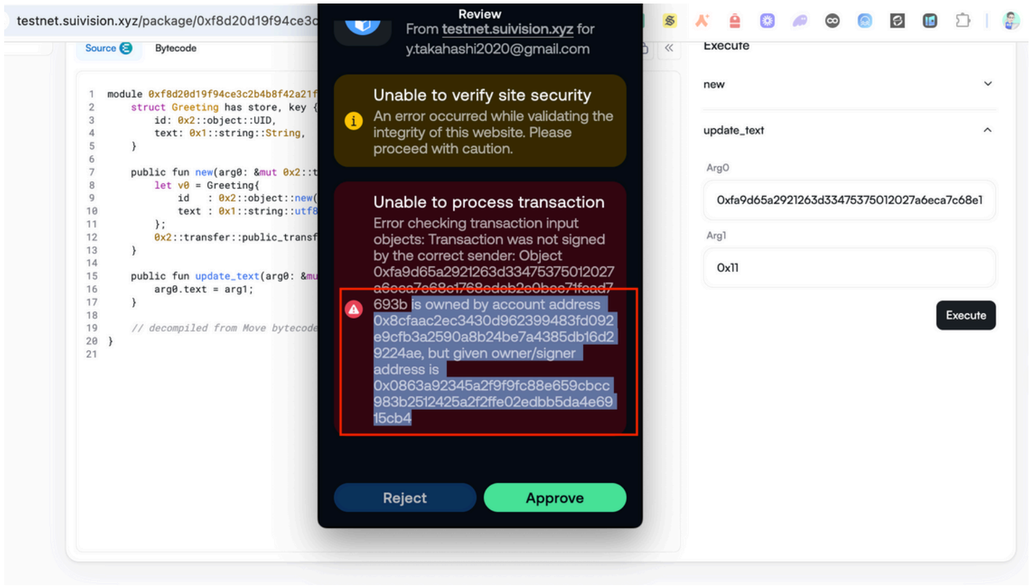
開発12　所有者以外で実行できない確認

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

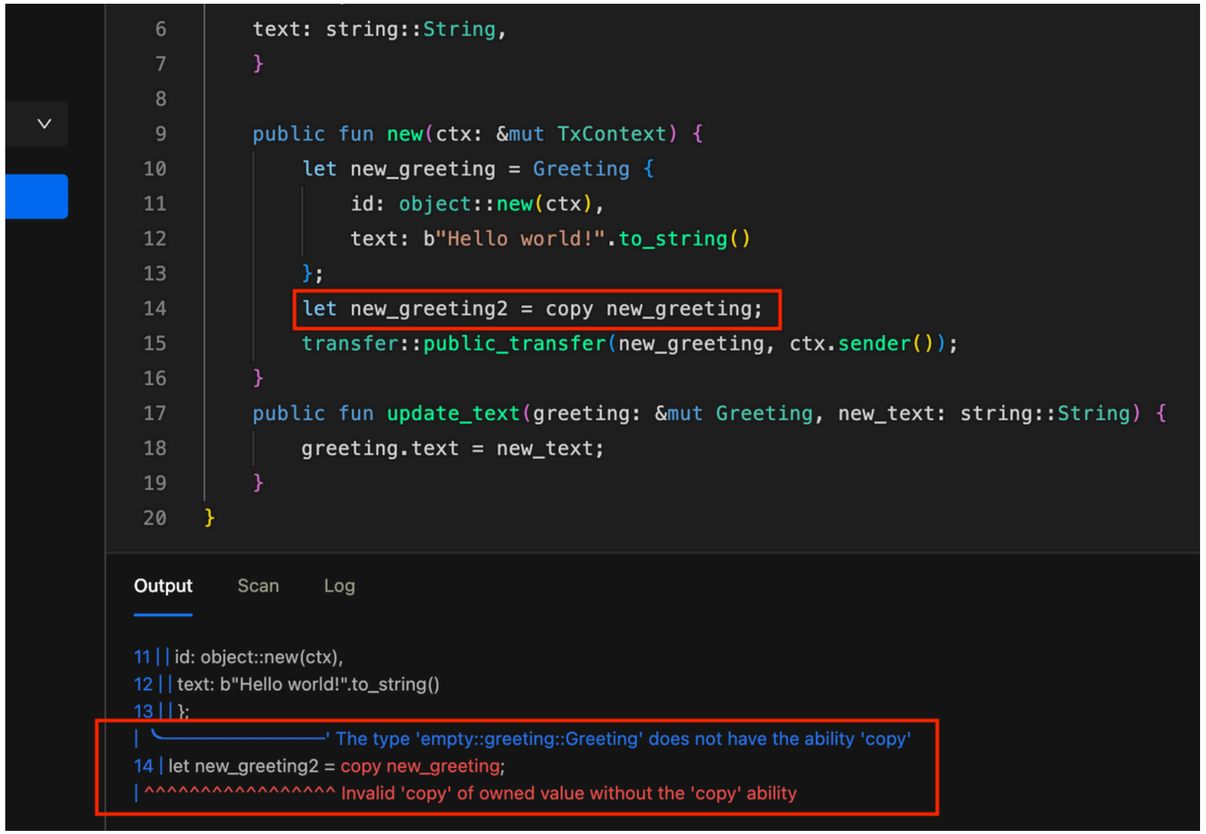
```move
module empty::greeting {
    use std::string;

    public struct Greeting has key, store {
    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;
    }
}
```



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

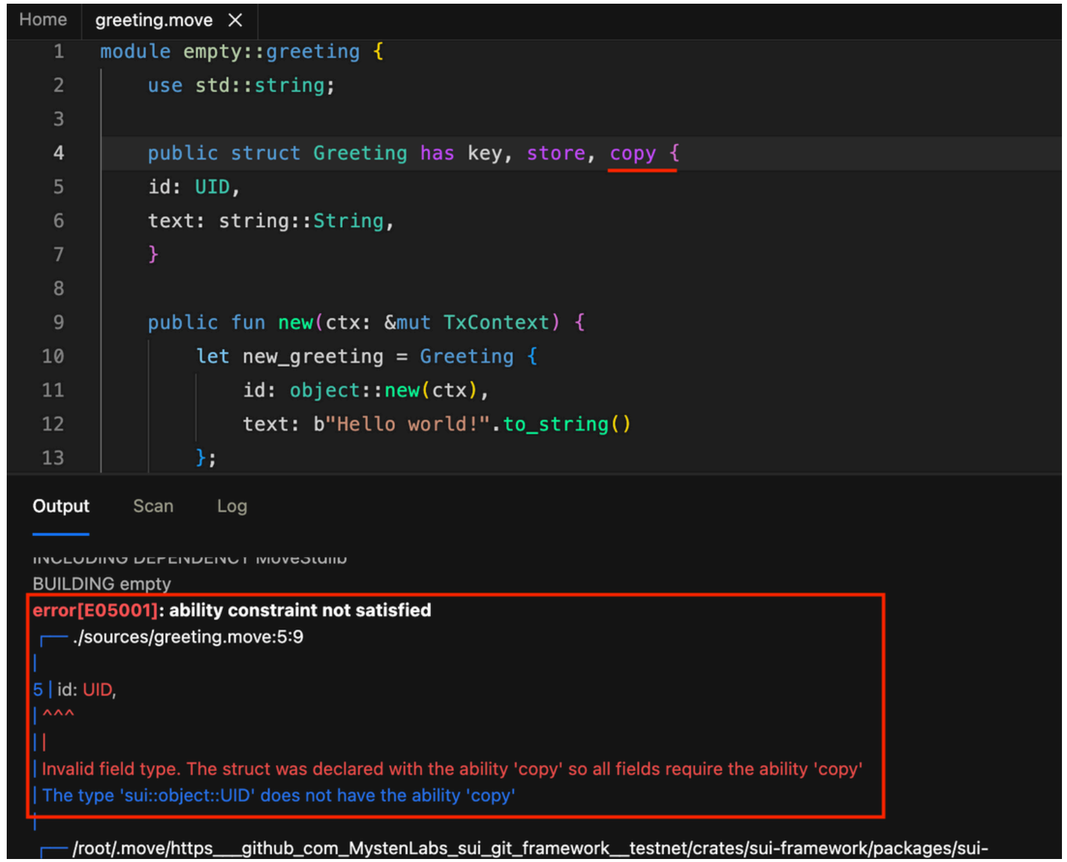
開発13　コピーができないことの確認

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



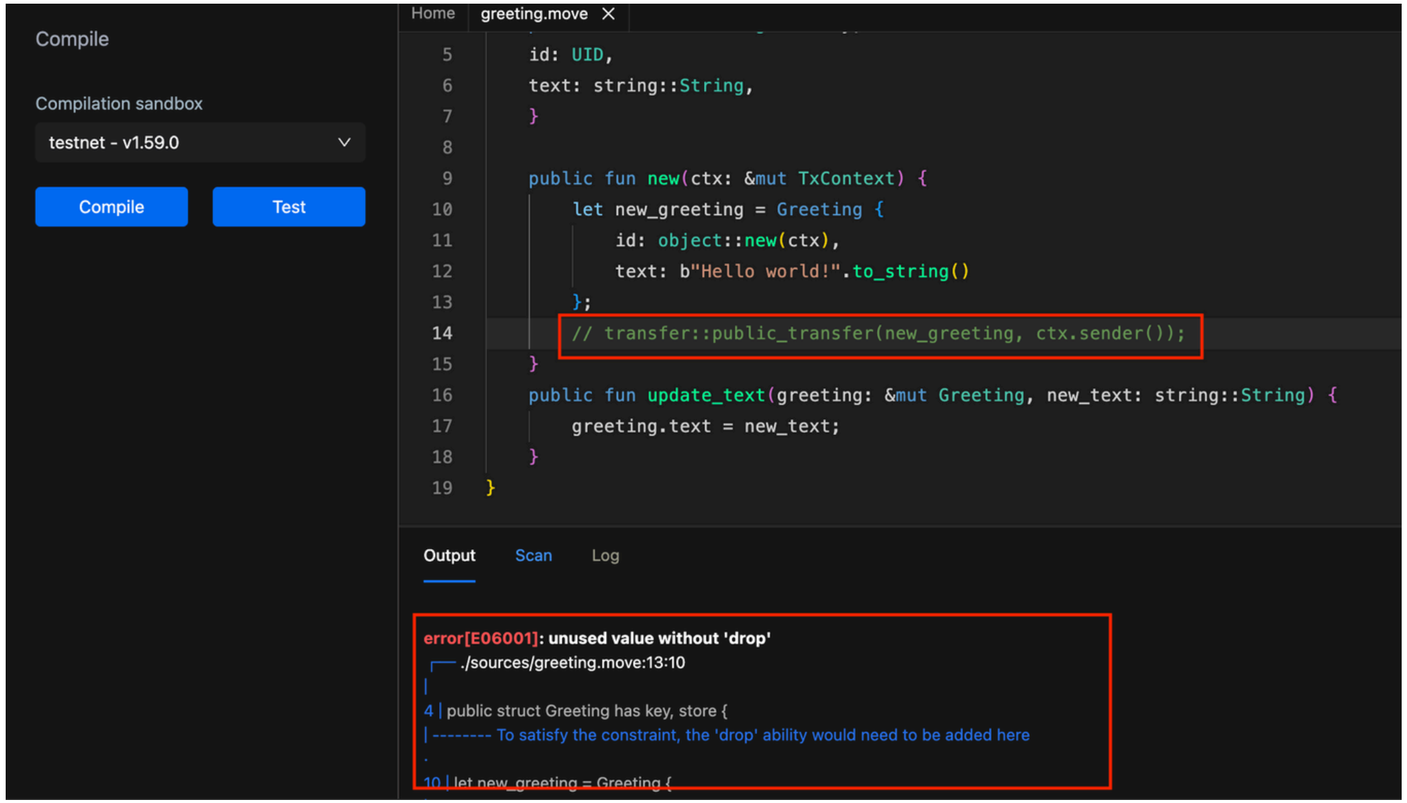
開発14　コピーとキーが両立しない確認

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**。



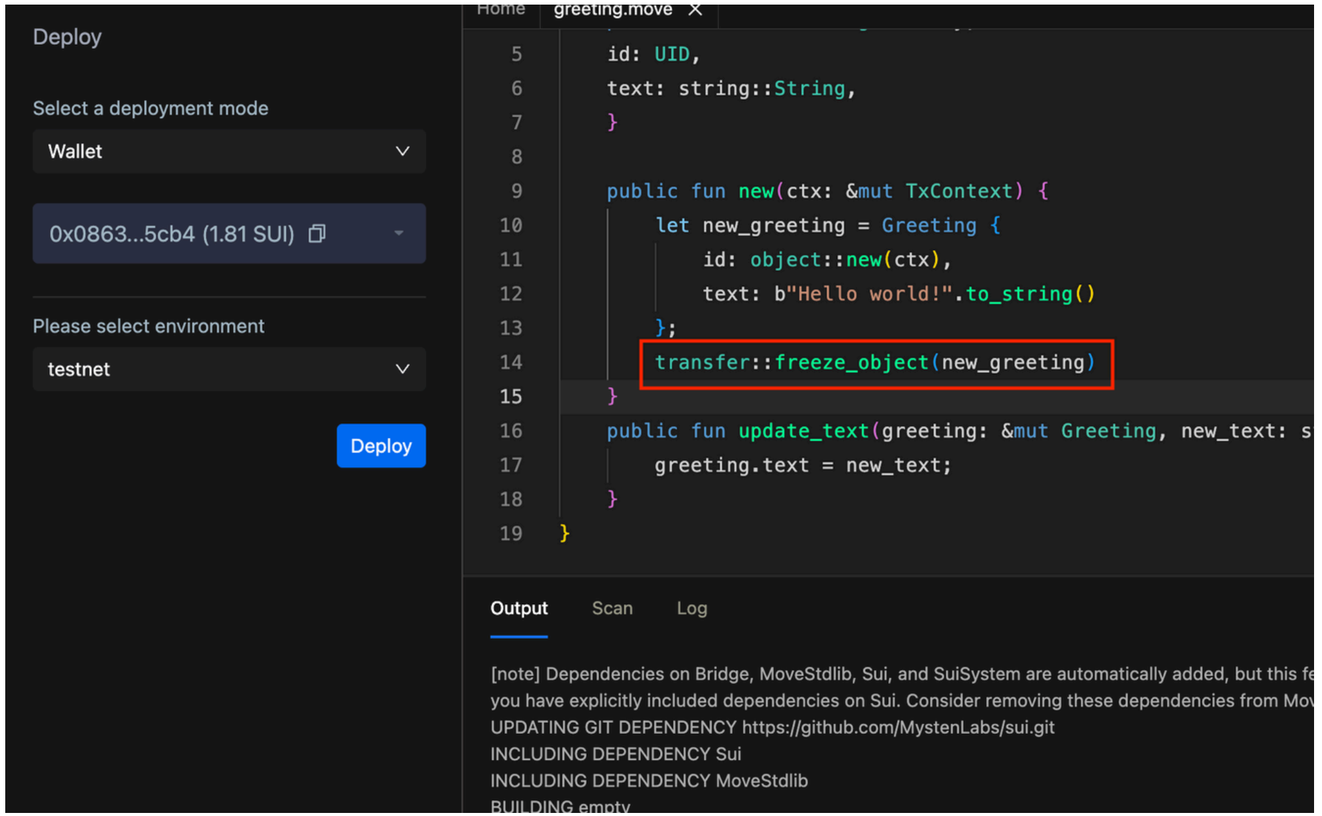
開発15　dropできないことの確認

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。



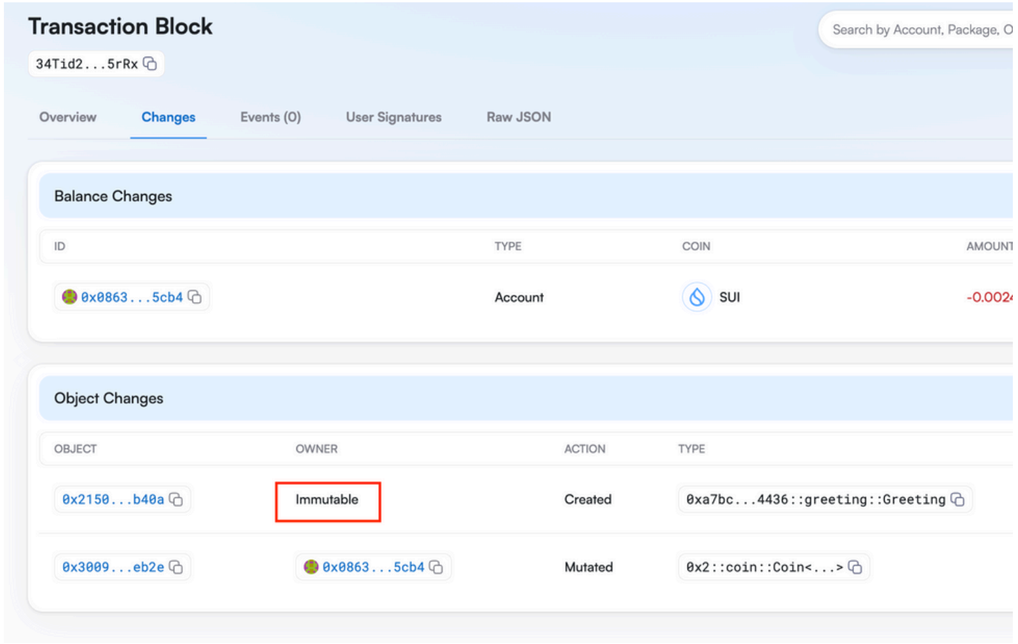
開発16　Immutableの設定

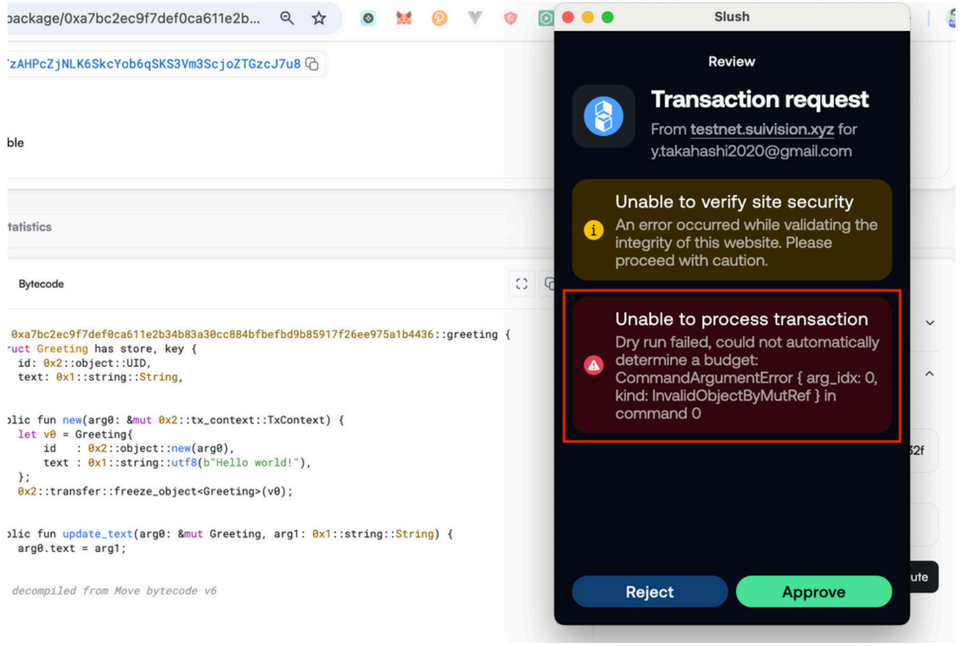
Immutableにすることは所有者を設定しないことではありません。
「所有者なし」という所有の状態に設定しています。



開発17　Immutableの確認

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





型からの派生　②オブジェクトと所有権

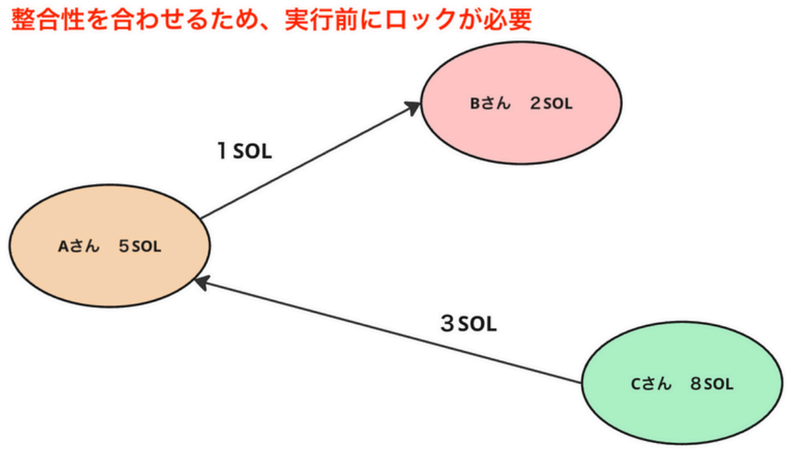
Suiはデータの値ではなく、**オブジェクトという形**で判断します。

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。

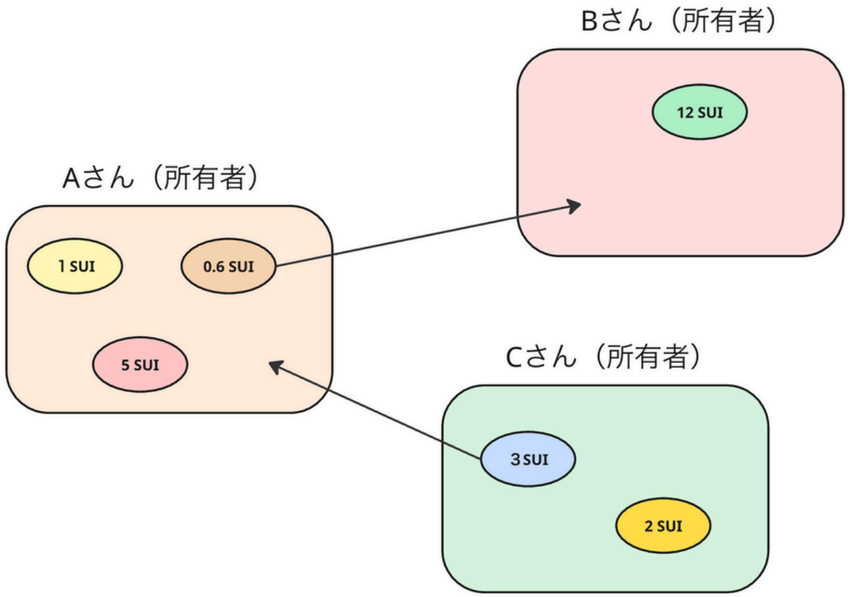


型からの派生　②オブジェクトと所有権

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。

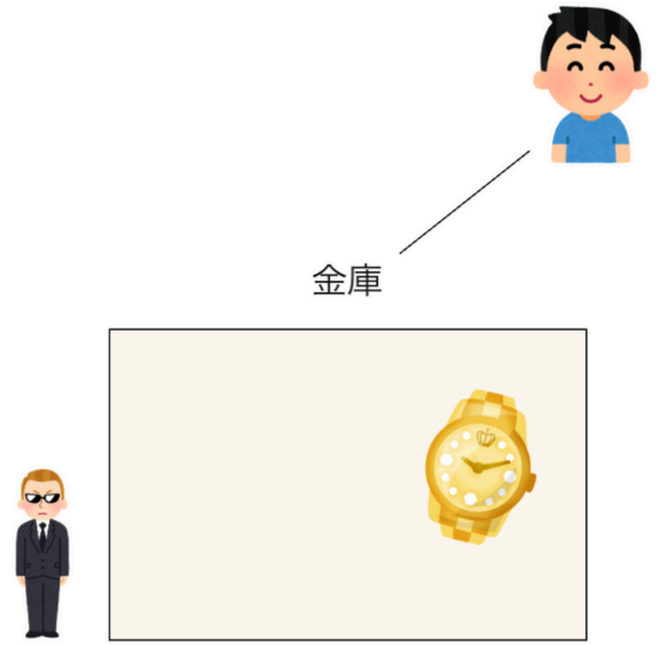
型からの派生　③ラップと単一所有の強制

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。

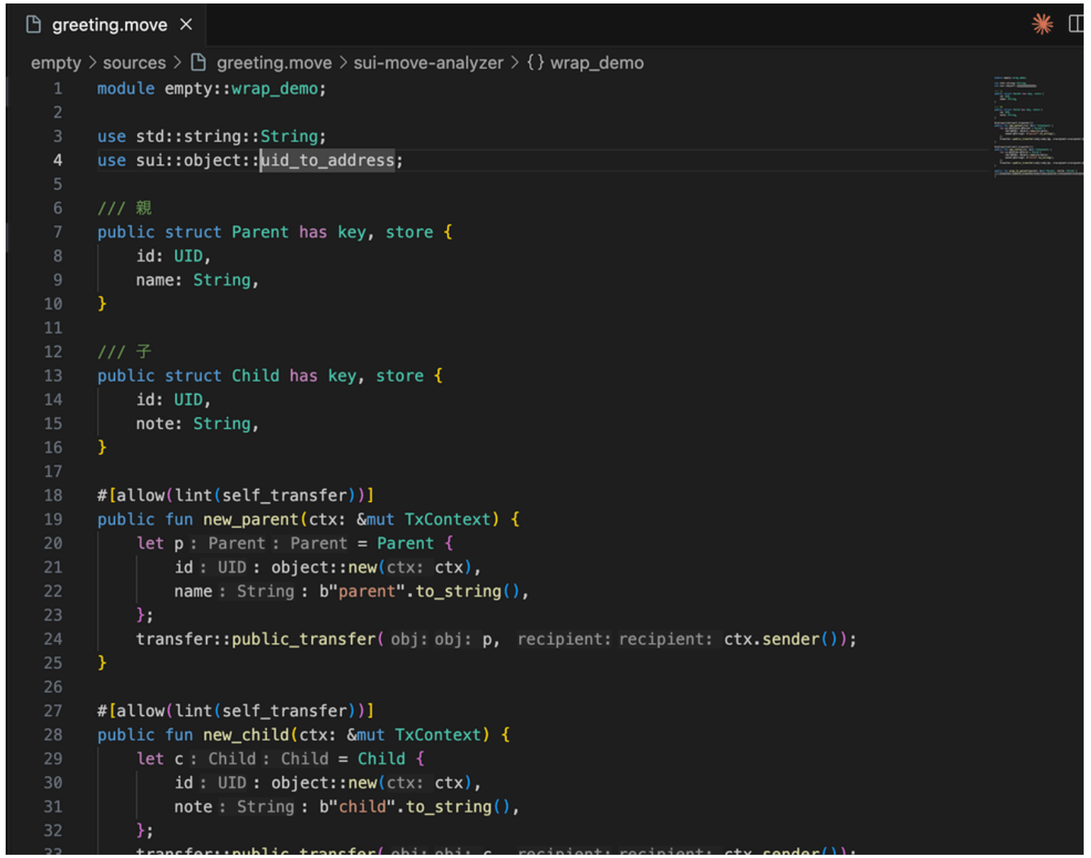
動かしたいときはバッグの所有者のあなたが
バッグごと動かせば良いのです。

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。



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

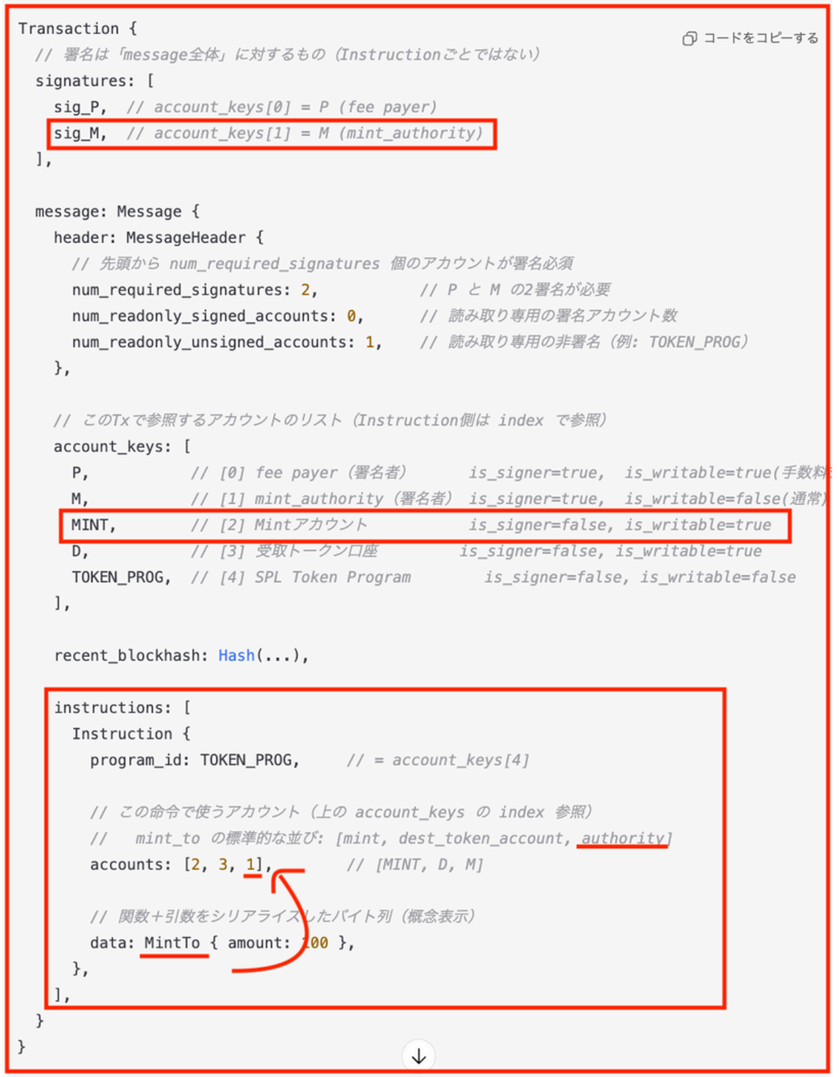
Solanaの場合、権限チェックは**実行時**に行われます。

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。



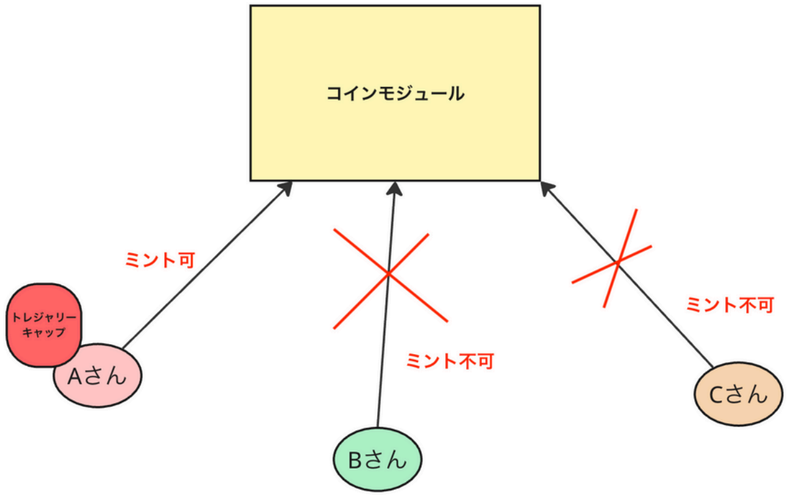
型からの派生　④Capabilityによる権限

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。

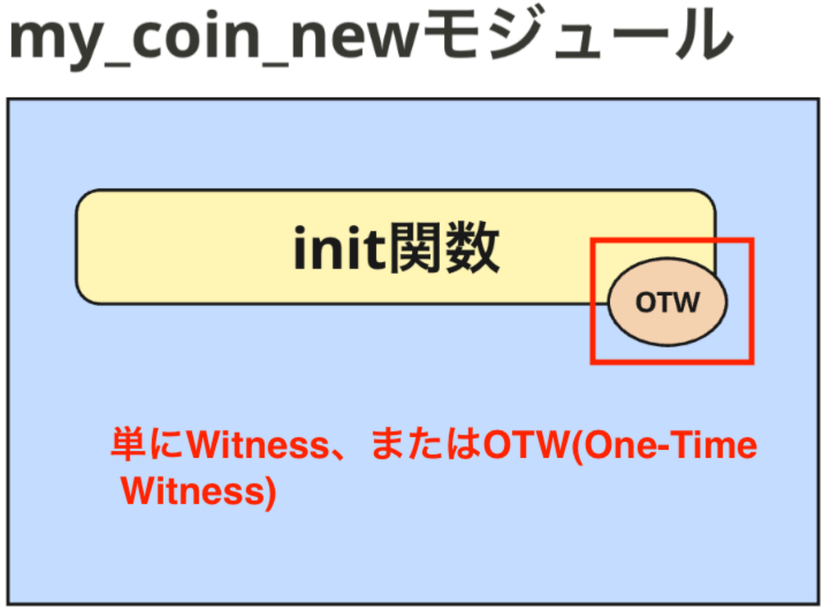


型からの派生　⑤OTWによる一度実行

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。

```move
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 ico
    // 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());
}
```



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。

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

```
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

/home

```
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 }
```

開発24　Capabilityの送付

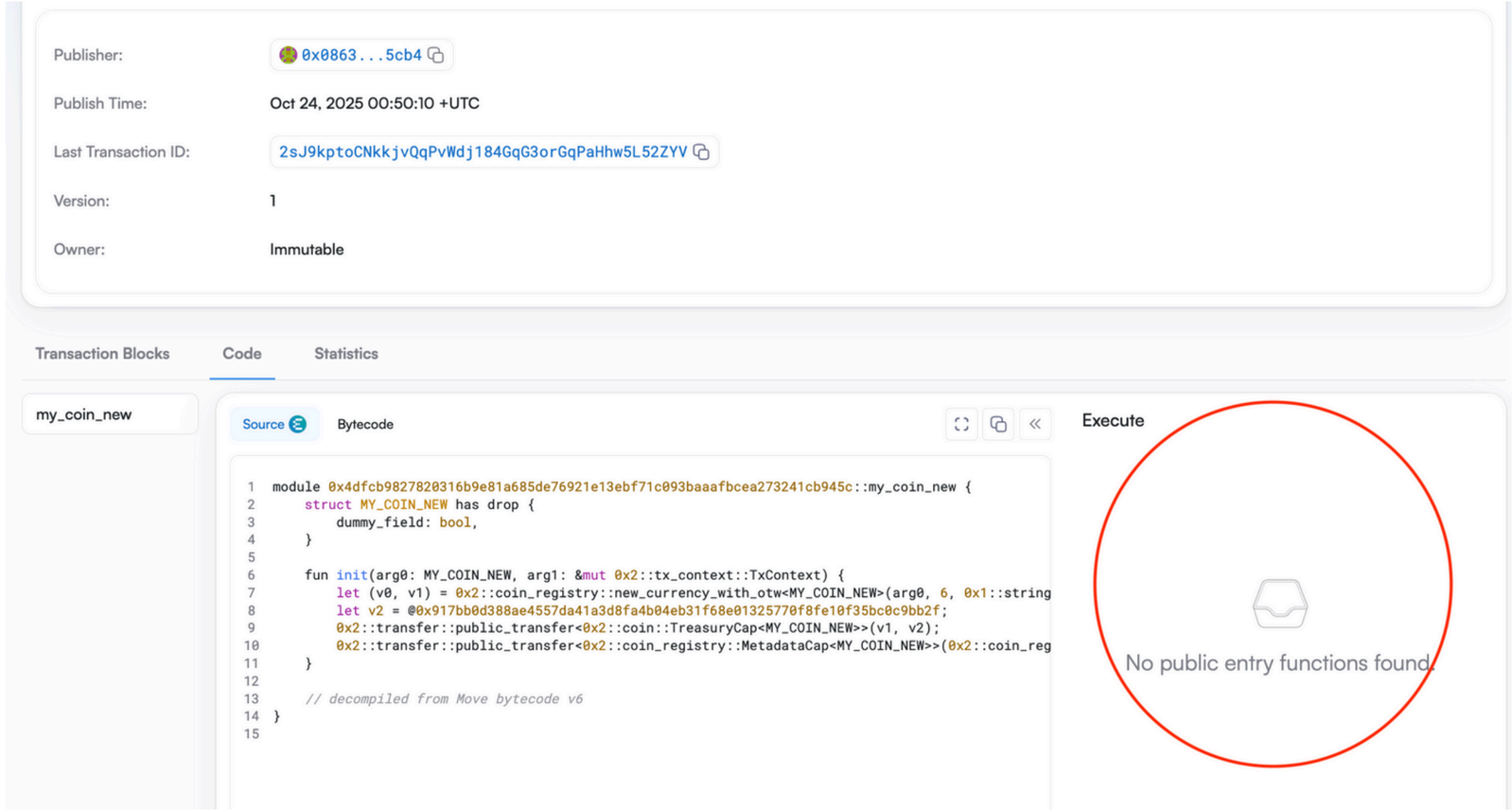
発行権限、メタデータの修正権限を実行者に送ります。

Home　my_coin_new.move

```
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

mintを実行しようとしたのですが、**public entry関数がない**と言われました。



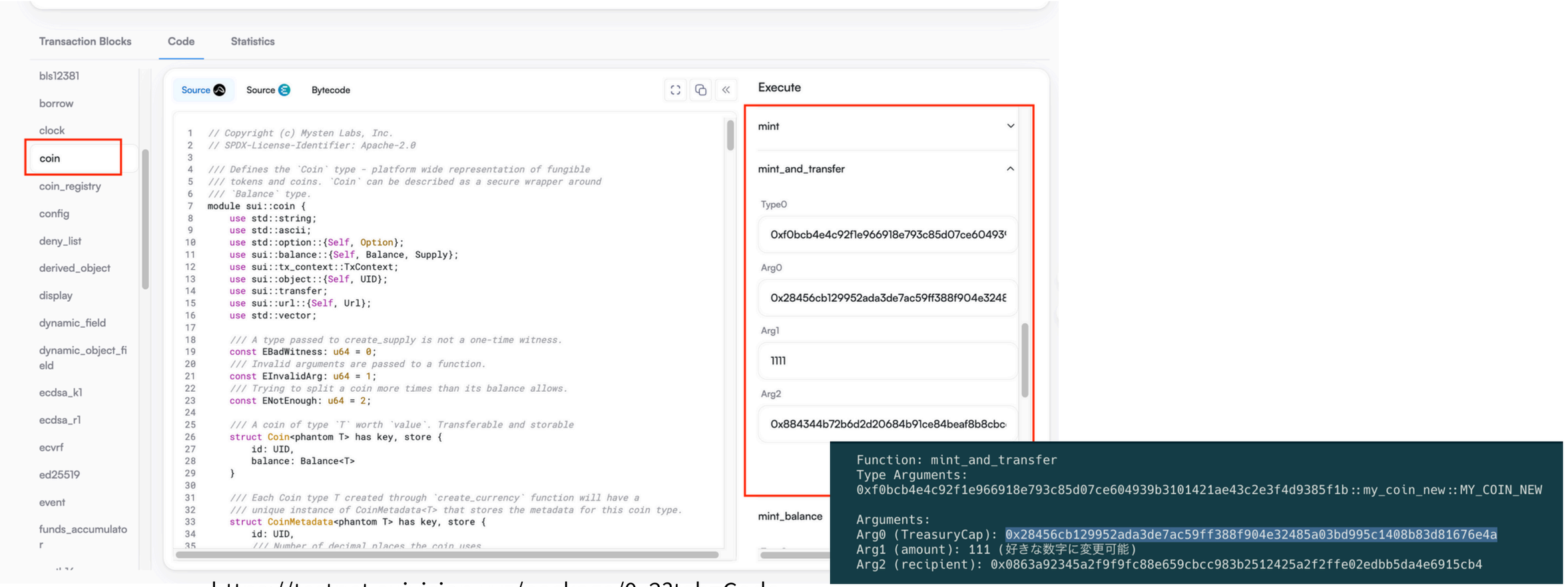
開発25　public, entry関数

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 0x917bb0d388ae4557da41a3d8fa
4b04eb31f68e01325770f8fe10f35bc0c9bb2f
[warning] Client/Server api version mismatch, client api version : 1.57.2, server api version : 1.59.0
Transaction Digest: CcccL7wsRq41VjGPQViPVzXXXxzc7z3WgG2uG1WqgK99
╭──────────────────────────────────────────────────────────────────────────────────────────────────────╮
│ 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

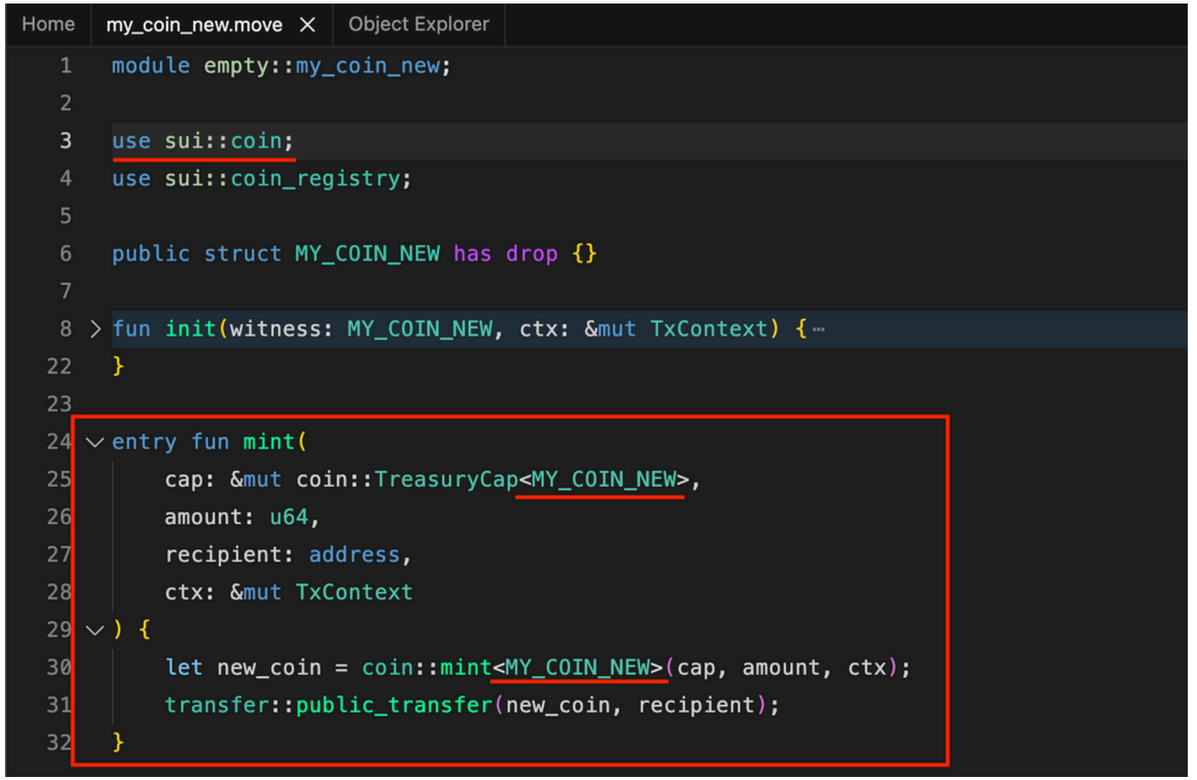
0x2パッケージで実行してみましょう。



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。



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。