Build on Sui

Weekend Move Workshop in Tokyo // Day 1

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。

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、**Corda**を利用したサービスを手がけ、**Sui、Solana、Avalanche**３基盤の技術選定を行なっています。

事前準備

以下の４パターンで進めていきます。

- ローカルにSuiが入っている
- Dockerを準備
- BUTASANのGithubCodeを利用
  https://zenn.dev/bububutasan/articles/65059536febaa1
- BitsLabを利用
  https://ide.bitslab.xyz/

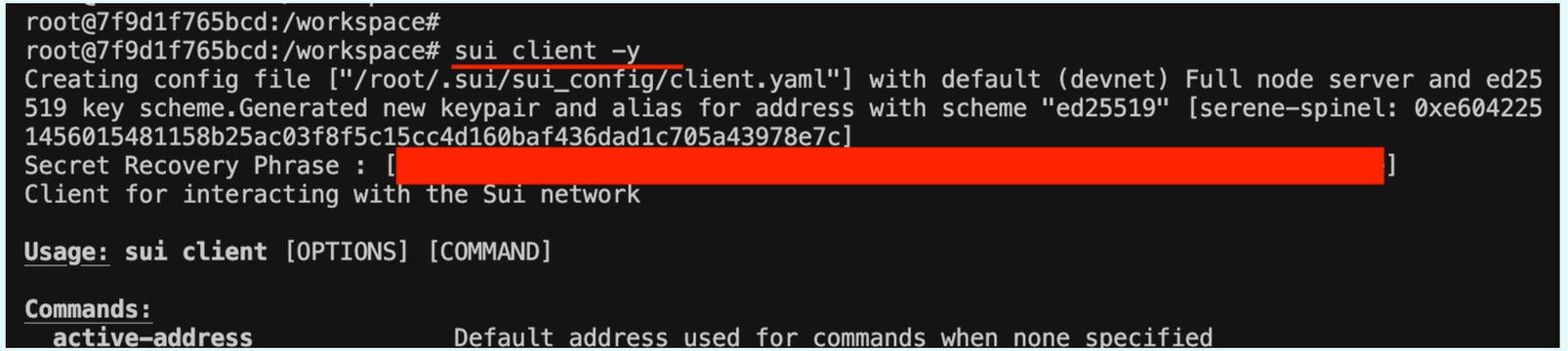
事前準備

１　クライアントの作成

Suiとやりとりするためのクライアントを作成します。

sui client -y

これで**アドレスを作成**できます。



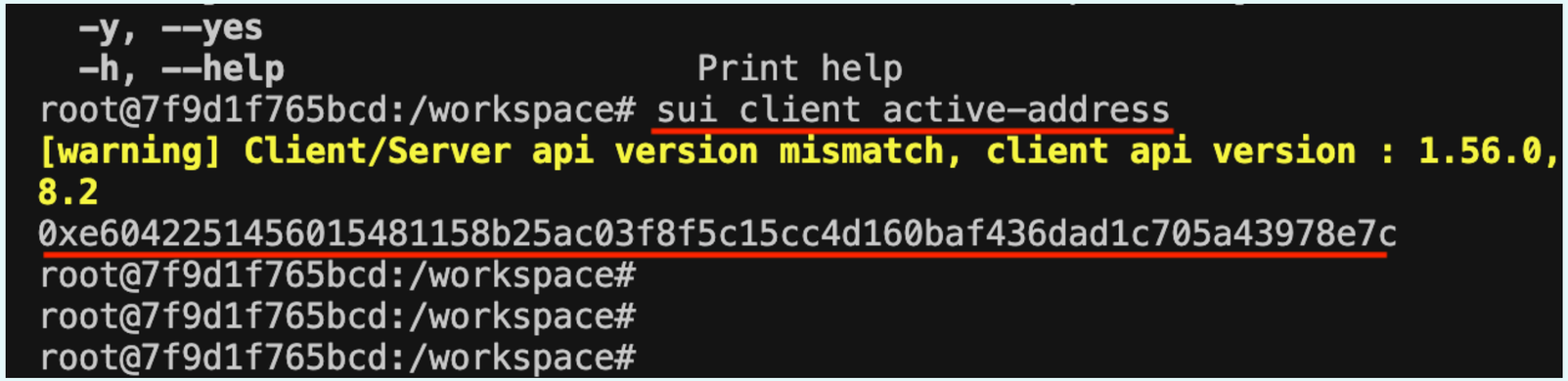
事前準備

２　アクティブアドレスの確認

以下のコマンドで**アクティブアドレスが確認**できます。

sui client active-address

手持ちのウォレットからテストトークンを送りましょう。(0.02Suiなど少量でOK)



事前準備

３　テストネットの指定

以下のコマンドで**テストネットに設定**し、結果を確認しましょう。

sui client switch --env testnet
sui client envs

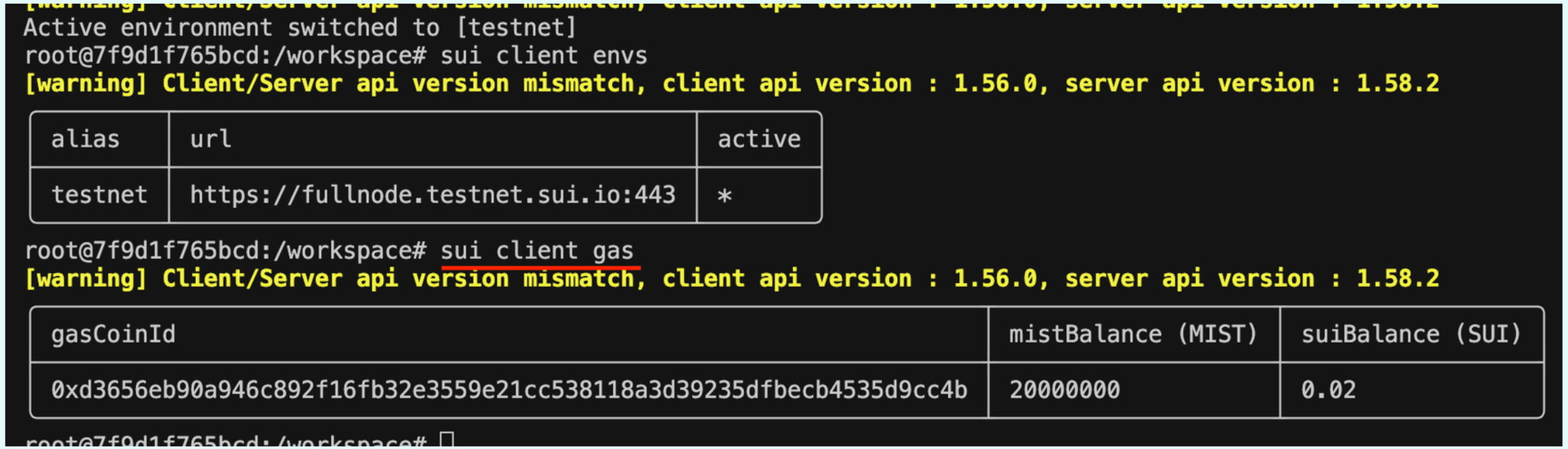
```
root@7f9d1f765bcd:/workspace#
root@7f9d1f765bcd:/workspace# sui client switch --env testnet
[warning] Client/Server api version mismatch, client api version : 1.56.0
8.2
Active environment switched to [testnet]
root@7f9d1f765bcd:/workspace# sui client envs
[warning] Client/Server api version mismatch, client api version : 1.56.0
8.2
```

alias	url	active
testnet	https://fullnode.testnet.sui.io:443	*

事前準備

４　残高の確認

以下のコマンドで**残高を確認**しましょう。
sui client gas



Moveとは？

安全かつ柔軟

Moveは、Sui上で安全かつ柔軟に動作するスマートコントラクトのために設計されています。

リソース指向

資産を安全に管理し、リエントランシーのような一般的な脆弱性を防ぎます。

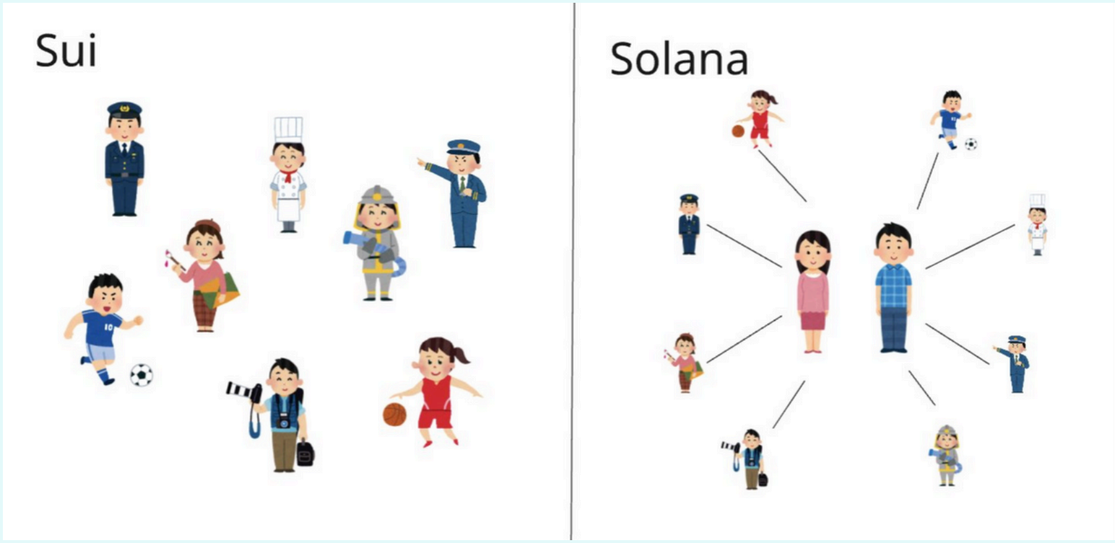
Suiでの利用

Moveはオンチェーンのオブジェクトを制御し、効率的かつ安全なデータ管理を実現します。

Suiの本質は？　型を重視した設計

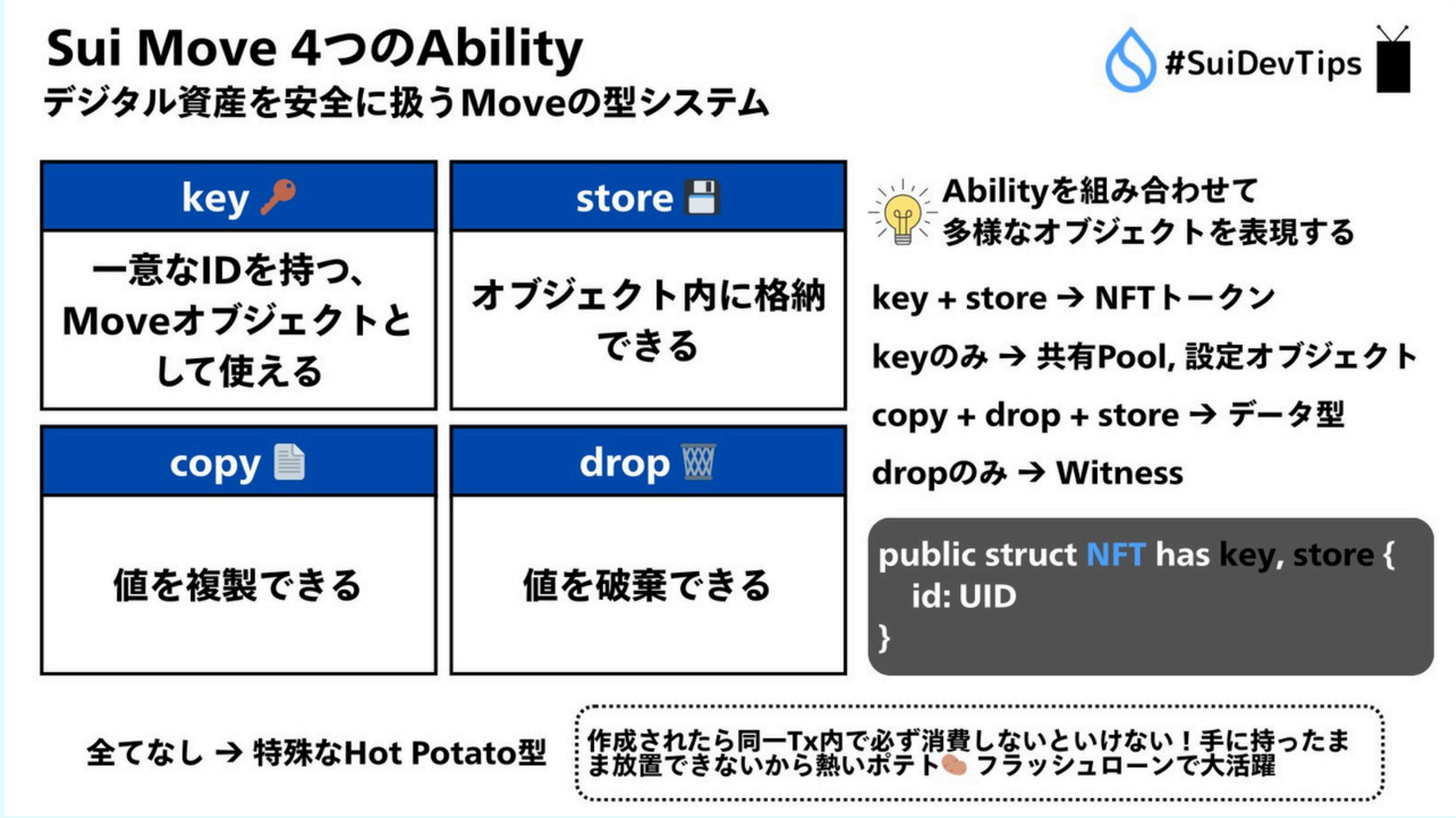
Suiは**型**を非常に重視します。
コックさんの格好をしていれば、コックさんだと確定します。

一方、例えば、**Solana**の場合は、**実行時の挙動**によって変わります。
厨房で料理しているという挙動を持ってコックさんのような人だとわかります。



型からの派生　①能力（Ability）

Moveには**「Key」「Copy」「Drop」「Store」**の４つの能力があります。
型を重視するため、**客観的にこのマークがついているかどうか**で判断します。



https://x.com/0x_onigiri/status/1964982038962081834

型からの派生　①能力（Ability）

能力の例として、カービィのコピー能力とハンターハンターの念能力を考えてみましょう。Suiはどちらだと思いますか。前のページを元に考えてみましょう。

https://www.reddit.com/r/Kirby/comments/ufs0jw/i_made_kirby_and_the_forgotten_land_copy/?tl=ja

https://jumpcs.shueisha.co.jp/shop/g/g4530430522989/?srsltid=AfmBOop7lqDSN_8CqP9zy98iL_up-6kgpKNotRGuAWsJPsY16rCIqnpp

型からの派生　①能力（Ability）1 Key

私の何の変哲もないシャーペンは全く同じものが存在しない**唯一のもの**です。

私の千円札も他の千円札と価値は同じですが、**唯一のもの**です。

このような唯一のものに対し、Suiは**「key」**というキーワードをつけます。

世界で唯一の**グローバルID**がつきます。

型からの派生　①能力（Ability）2 Copy

デジタルの世界ではコピペなどは頻繁に行われますが、実際の世界では誰かにものを渡すときにコピーは行われません。

移動するだけです。

なので、原則コピーはできず、できるようにするには**「Copy」**をつけます。

型からの派生　①能力（Ability）3 Drop

デジタルの世界ではデリート（削除）は簡単に行えますが、現実の世界では捨てたとしても、**存在自体は無くなりません**ね。

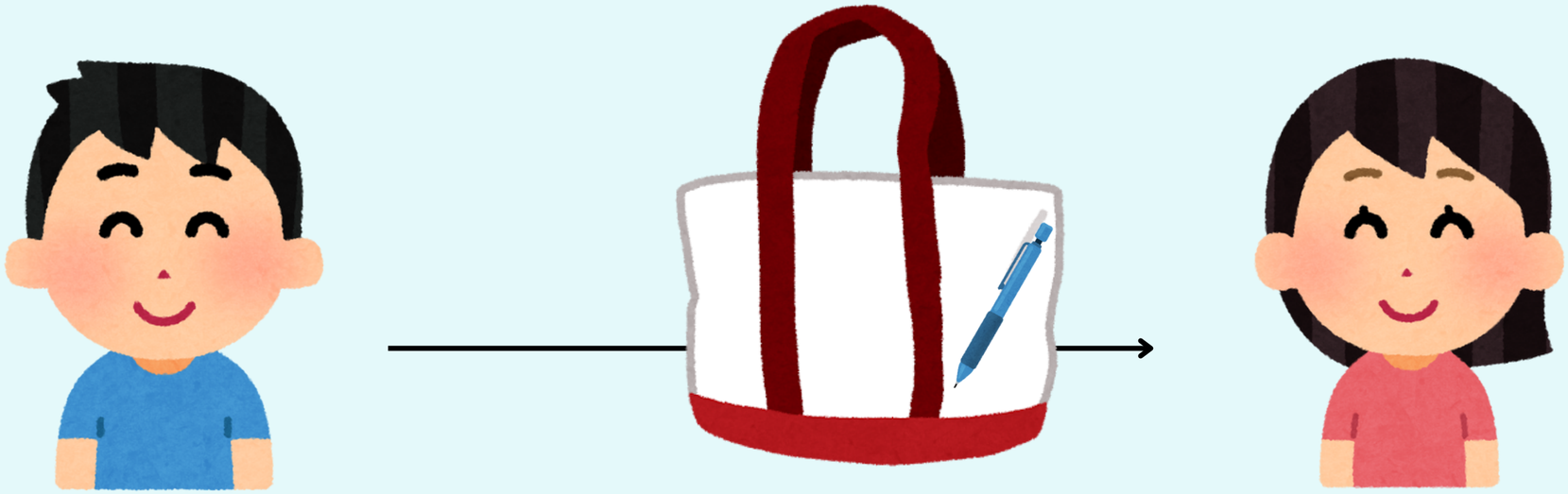
なので、原則は破棄ができず、できるようにするには**「drop」**をつけます。

型からの派生　①能力（Ability）4 Store

Moveは「移動」に焦点を当てた言語です。

もしもシャーペンがバッグに入っている（オブジェクトのフィールド）場合、
バッグが移動すれば、**その中のシャーペンも移動**します。

そのため、格納できるかどうかで**「store」**をつけます。



補足　The DAO事件とは

再入（リエントランシー）の脆弱性をついて不正に引き出しを行おうとした事件です。

Photo by **ekakinonakagawa**

【完全保存版】The DAO事件を通じて、Re-entrancy攻撃やCEIパターンを学ぼう！

22

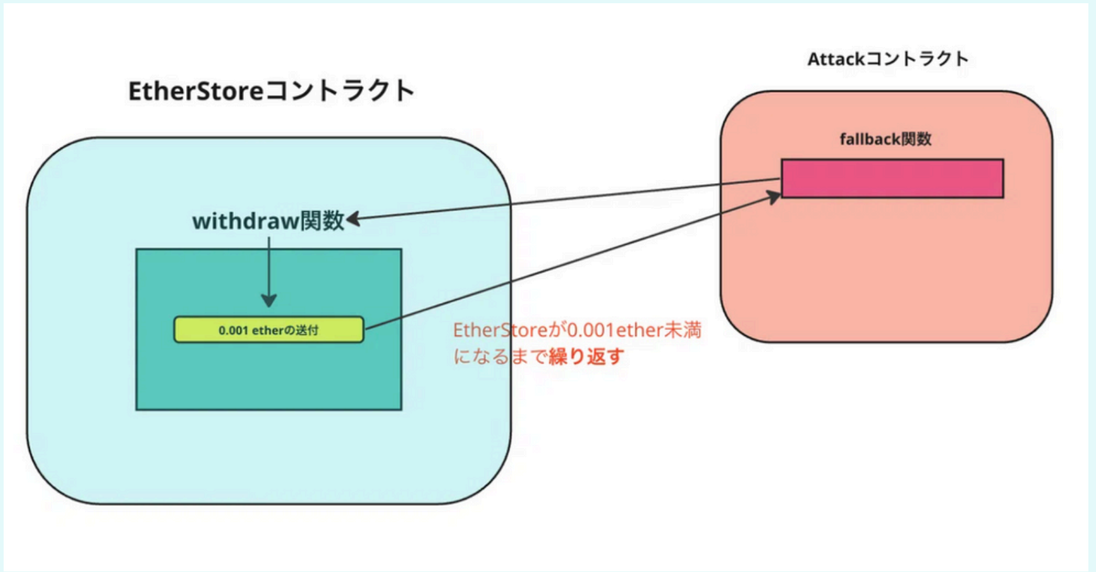
ユウキ
2023年7月15日 07:37

https://note.com/standenglish/n/ncd433393aca3

補足　The DAO事件とは

下のように、スマートコントラクトの脆弱性（順番が違うなど）によって攻撃を受けてしまった。

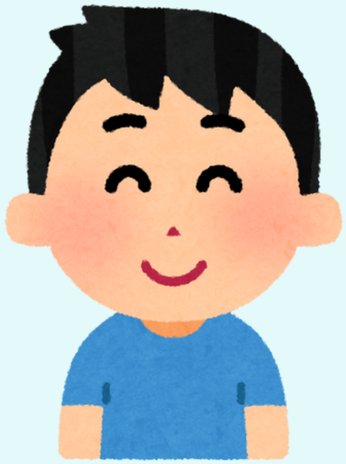
```solidity
// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

contract EtherStore {
    mapping(address => uint) public balances;

    function deposit() public payable {
        balances[msg.sender] += msg.value;
    }

    function withdraw() public {
        uint bal = balances[msg.sender];
        require(bal > 0);   ①残高確認

                                        ②送付
        (bool sent, ) = msg.sender.call{value: bal}("");
        require(sent, "Failed to send Ether");

        balances[msg.sender] = 0;  ③値の反映
    }

    function getBalance() public view returns (uint) {
        return address(this).balance;
    }
}
```



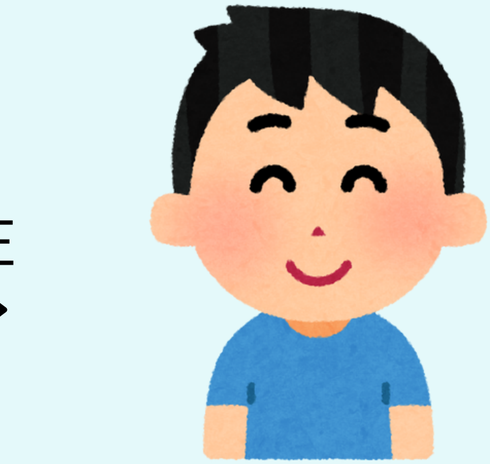
補足　リソース指向とリエントランシー

リソース指向であればこのようなケースは起こりえません。
例えば、コインにはそれぞれに一意の**Key**がついており、**Copy不可**です。

EVMのように金額を数値としてあつかうのではなく、**物（オブジェクト）**として扱っているためです。
（他にもいろいろな理由でリエントランシーを防いでいますがここでは省略）

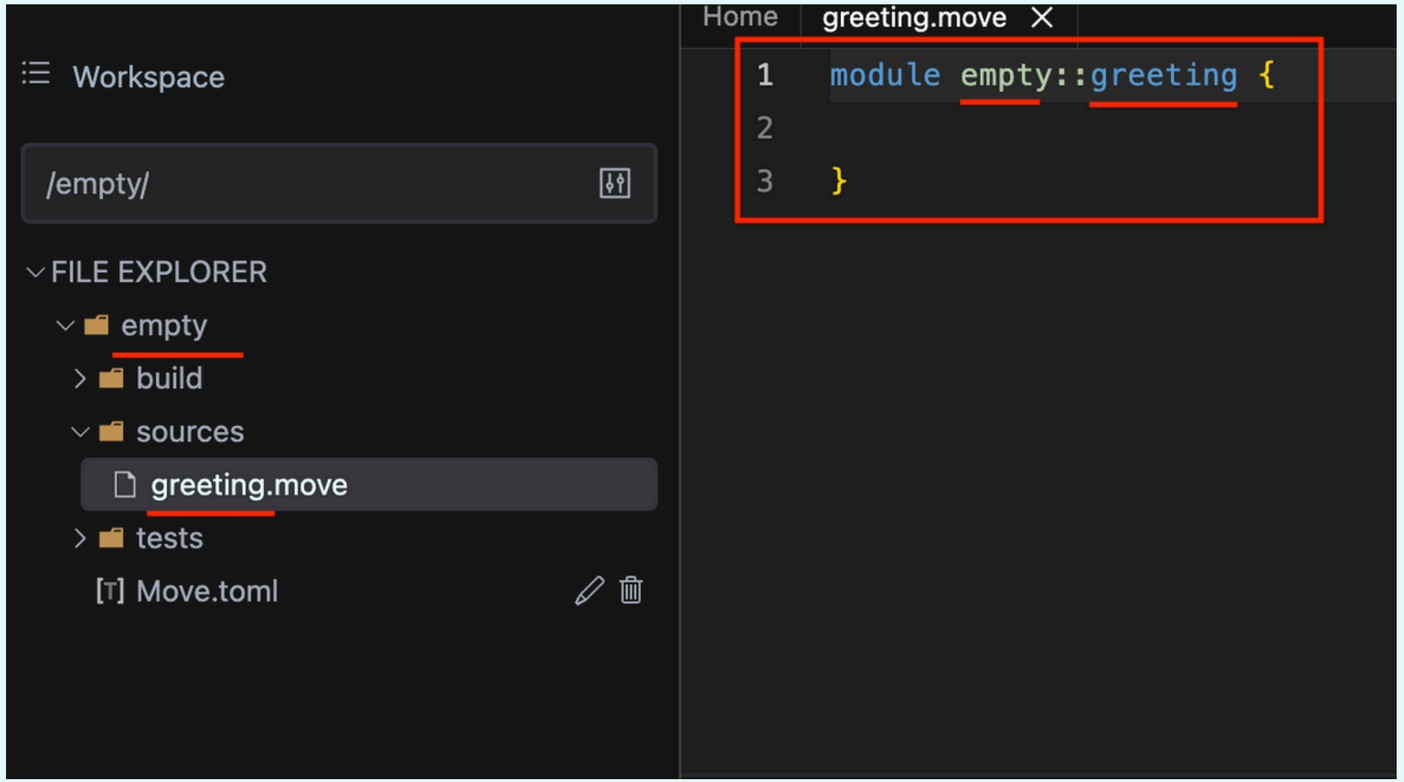


AE0119AE011973AE



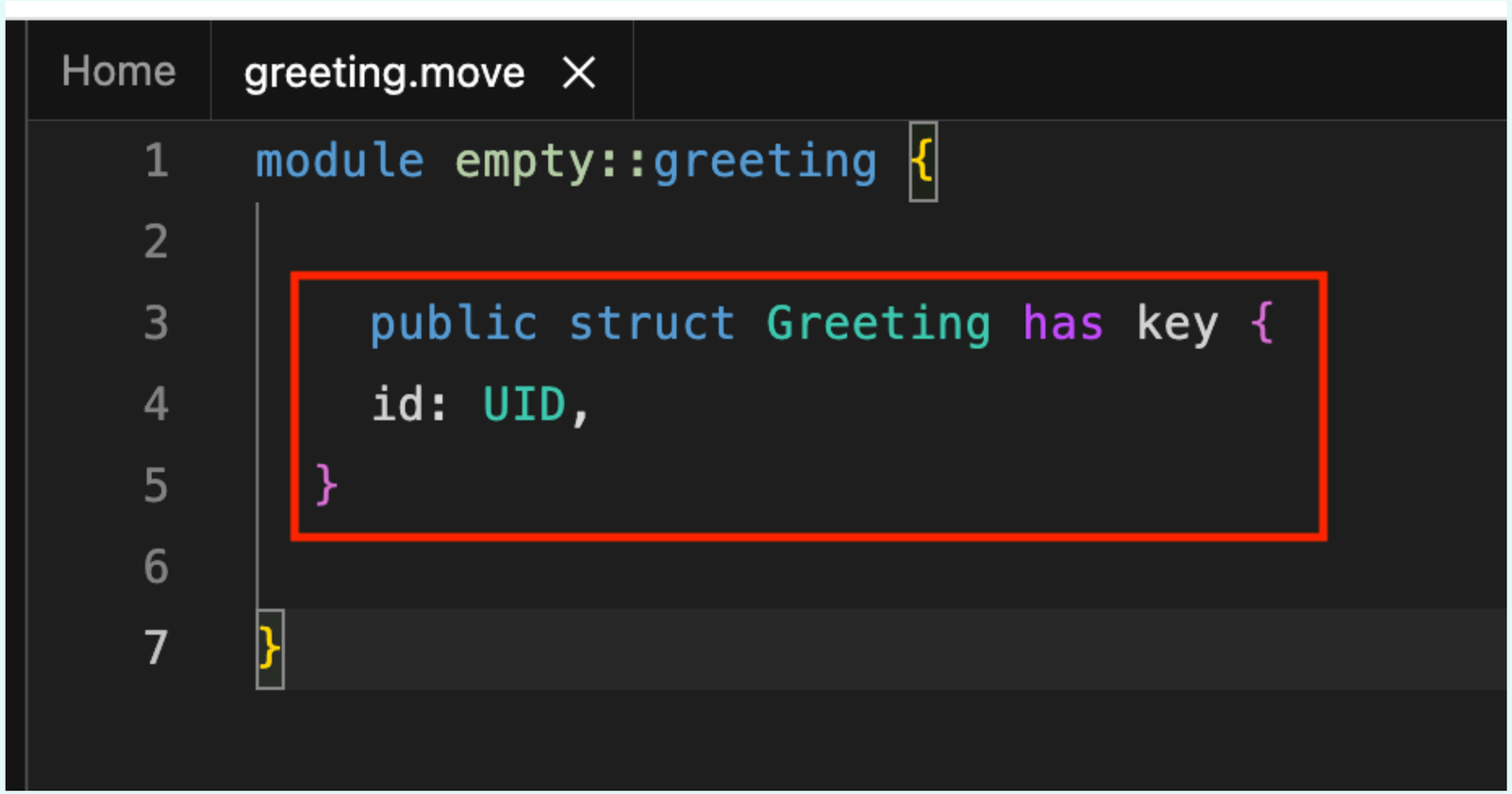
開発①　モジュールの作成

sui move new <プロジェクト名>で作成後
下のように、外側を作ってみましょう。



開発②　keyアビリティを持つ構造体

グローバル唯一の構造体を持ちます。



開発③　Contextの理解

コンテキスト：プログラム実行時に利用可能な定義済み変数の集合

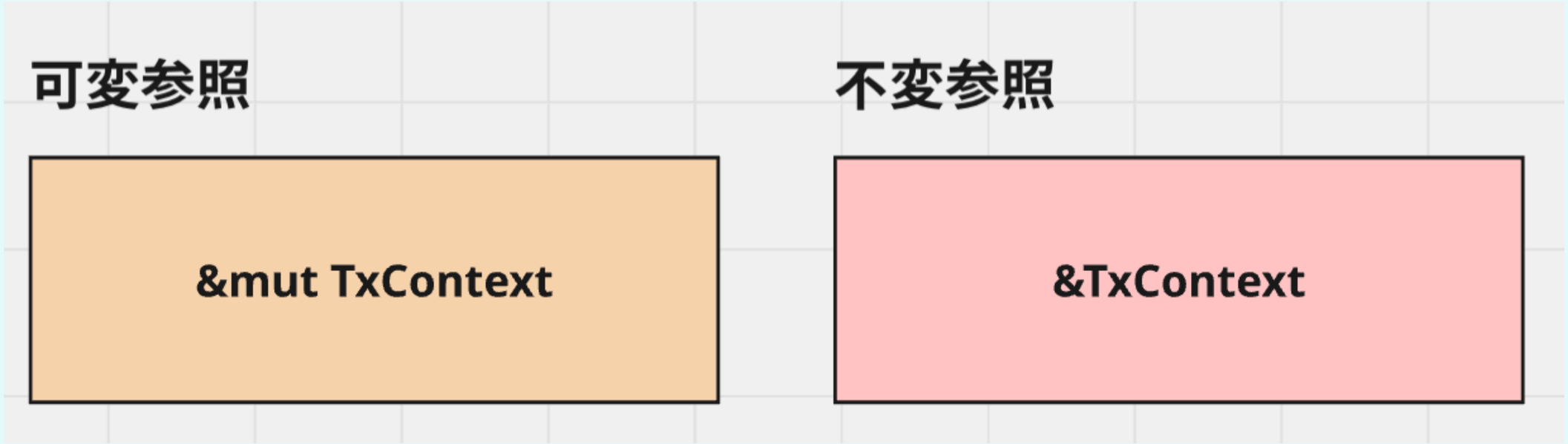
```
module sui::tx_context;

/// 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

開発③　Contextの理解

可変参照か不変参照かで渡し方が異なります。



開発③　Contextの理解

fresh_object_address関数を見ると、ids_createdの+1が確認できる

```
module sui::tx_context;

/// Create an `address` that has not been used. As it is an object address, it will never
/// occur as the address for a user.
/// In other words, the generated address is a globally unique object ID.
public fun fresh_object_address(ctx: &mut TxContext): address {
    let ids_created = ctx.ids_created;
    let id = derive_id(*&ctx.tx_hash, ids_created);
    ctx.ids_created = ids_created + 1;
    id
}
```

tx_hashとids_createdを元にID作成

ids_createdは+1されていく

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

開発④　オブジェクトの作成

object::new(ctx)でIDを作成します。
なぜ可変参照かも考えてみましょう。

Home　greeting.move

```
1  module empty::greeting {
2
3      public struct Greeting has key {
4      id: UID,
5      }
6
7      public fun new(ctx: &mut TxContext) {
8          Greeting {
9              id: object::new(ctx),
10         };
11     }
12
13 }
```

開発⑤　所有者の設定

Suiでは所有者の設定が必要です。
ここでは、**共有**という所有にしましょう。

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

開発⑥　文字の設定

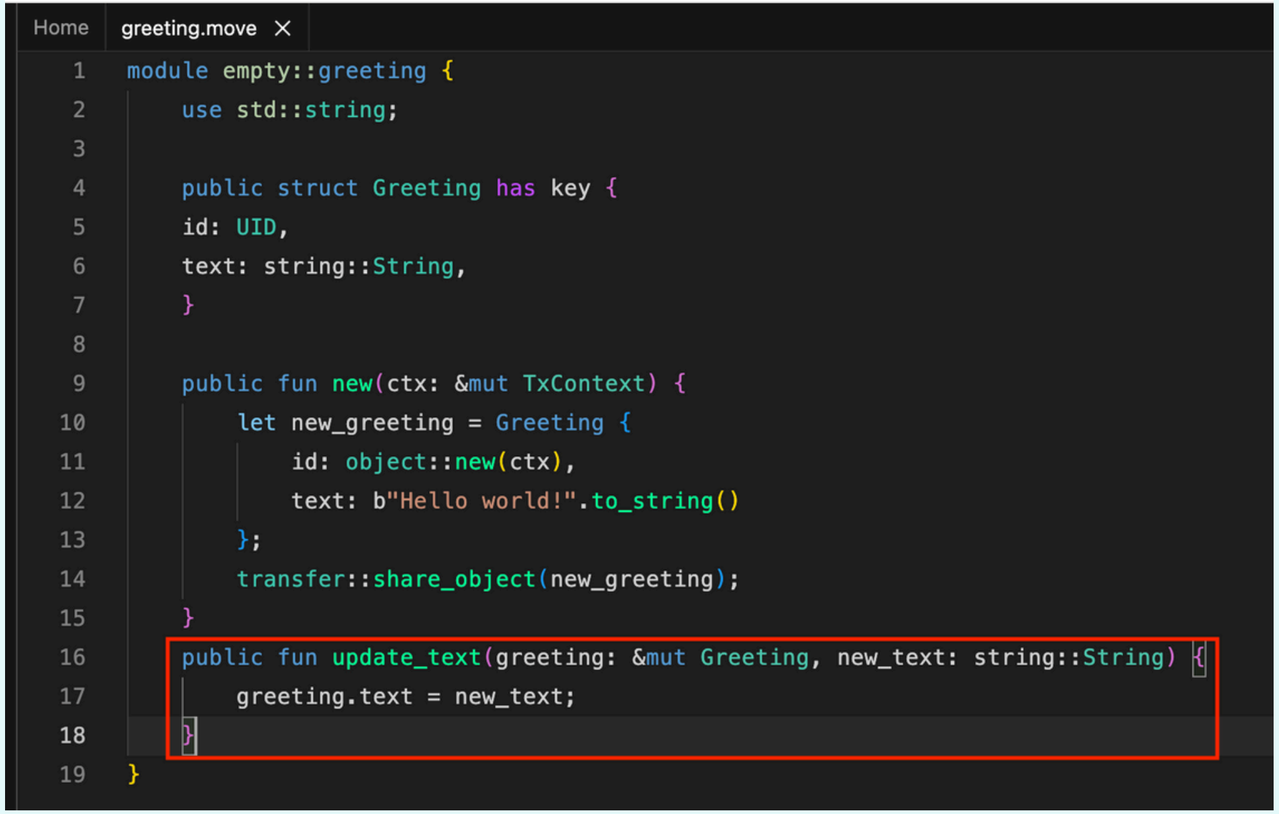
Stringを用いて**文字を設定**しましょう。

Home　greeting.move

```
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::share_object(new_greeting);
15     }
16
17 }
```

開発⑦　変更を行う関数

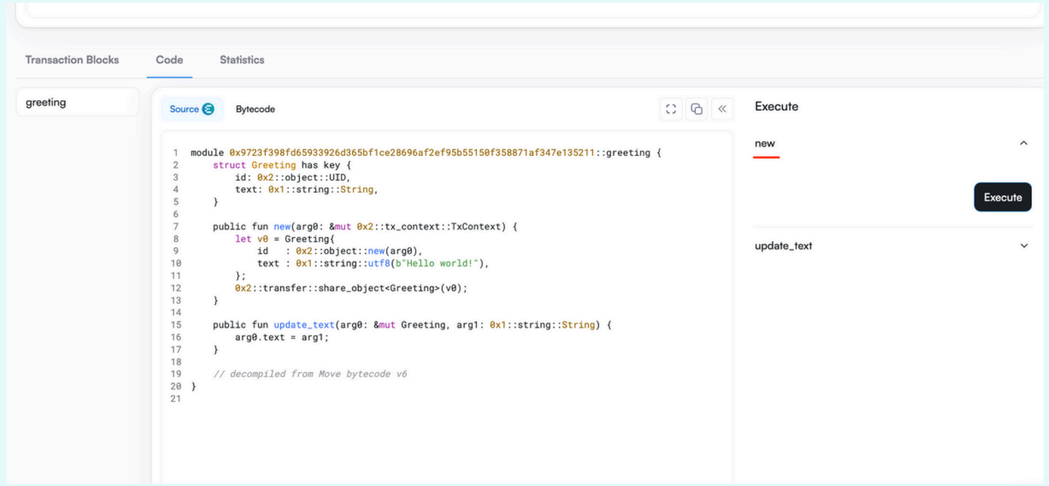
値を変更する関数を作りましょう。

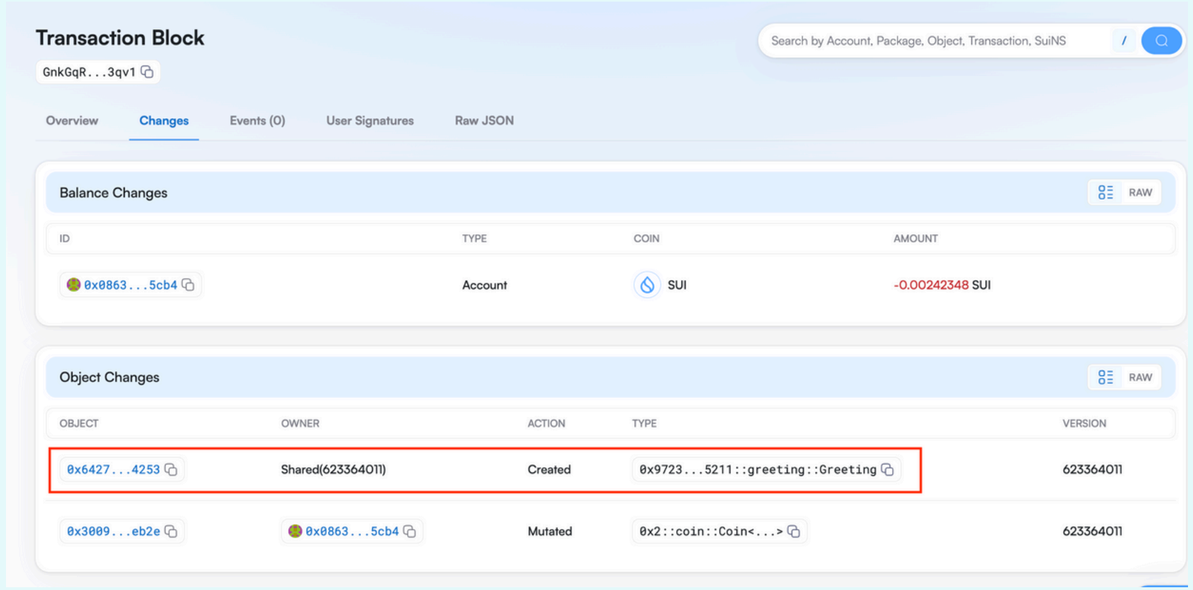


開発⑧　SuiVisionでの実行

SuiVisionで作成しましょう。

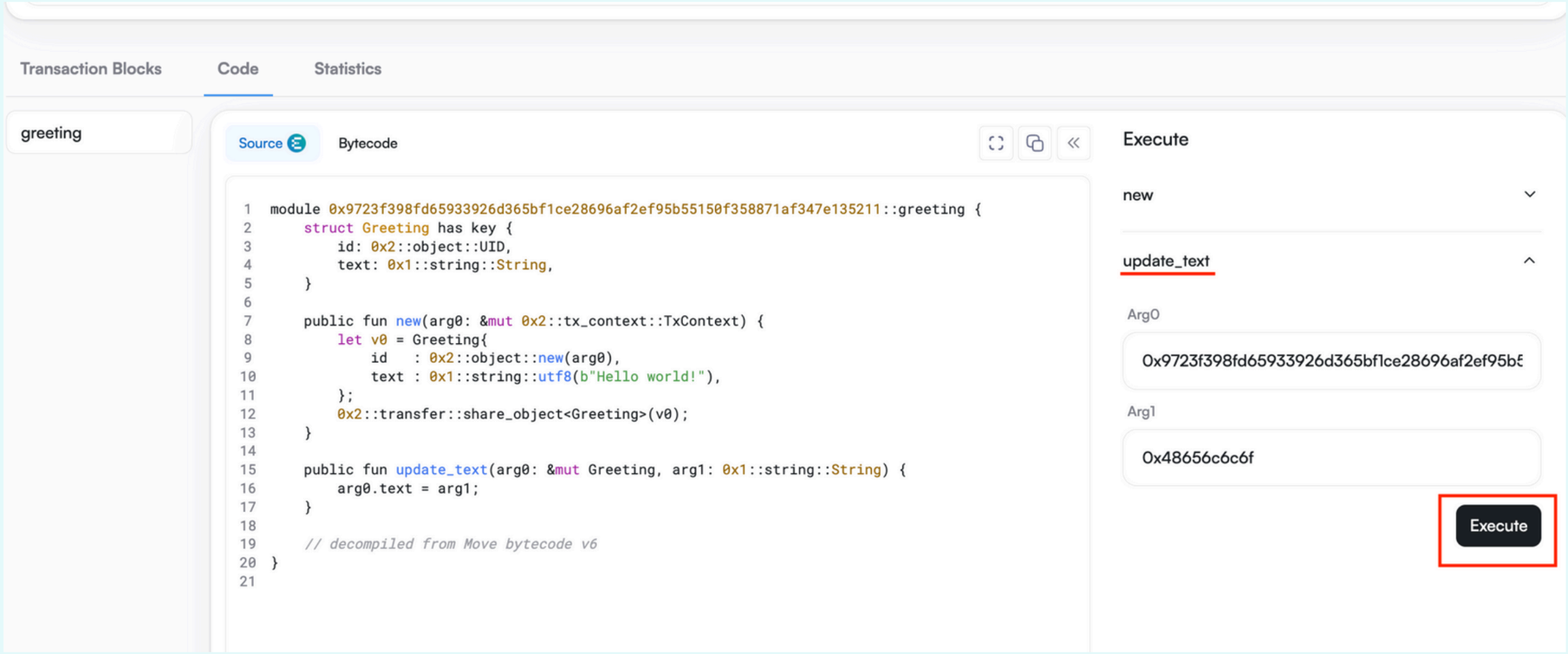
また、実行できた**ObjectにIDがついている**ことも確認しましょう。





開発⑨　共有オブジェクトの確認

実行アドレスを変えて、うまくいくことを確認してみましょう。



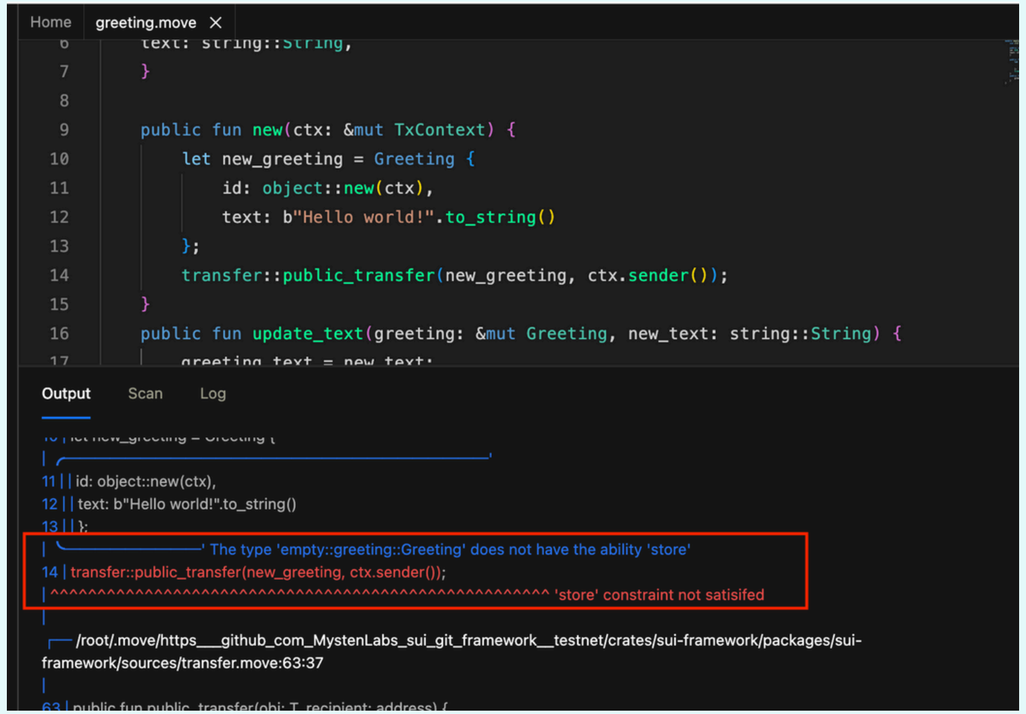
開発10　実行者への所有者の設定

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;
    }
}
```

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。

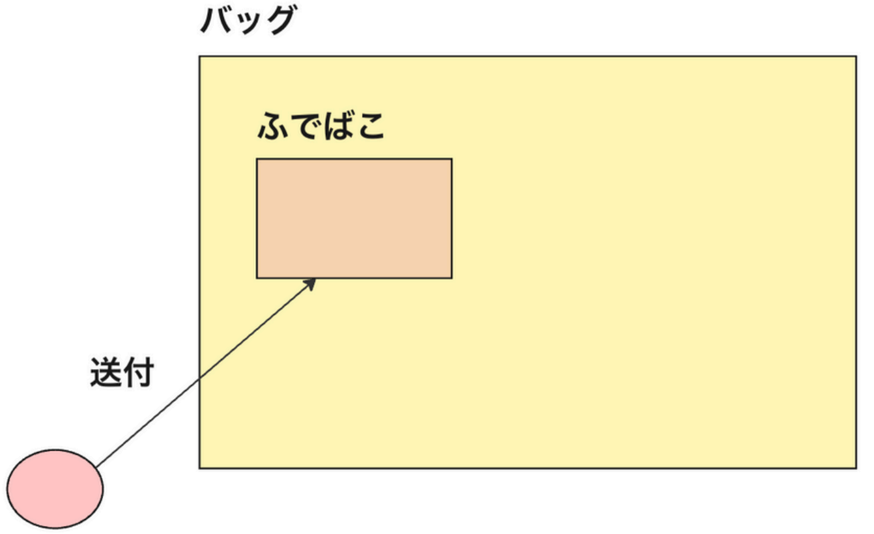
私の仮説は右です。これが許されると、すでに何かに格納されているものに送られようとした場合、整合性が取れなくなるからではと考えています。

```
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



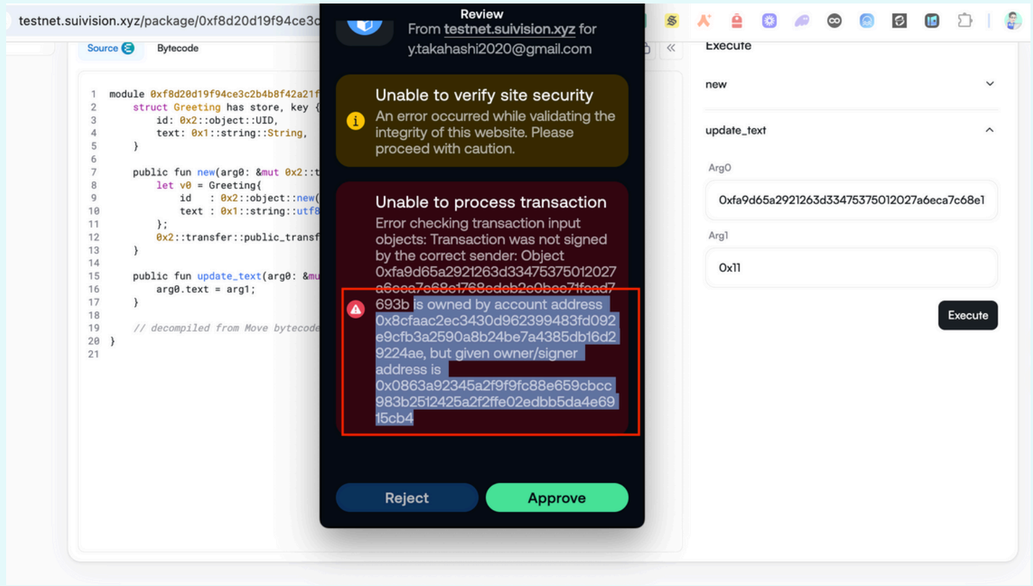
開発12　所有者以外で実行できない確認

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

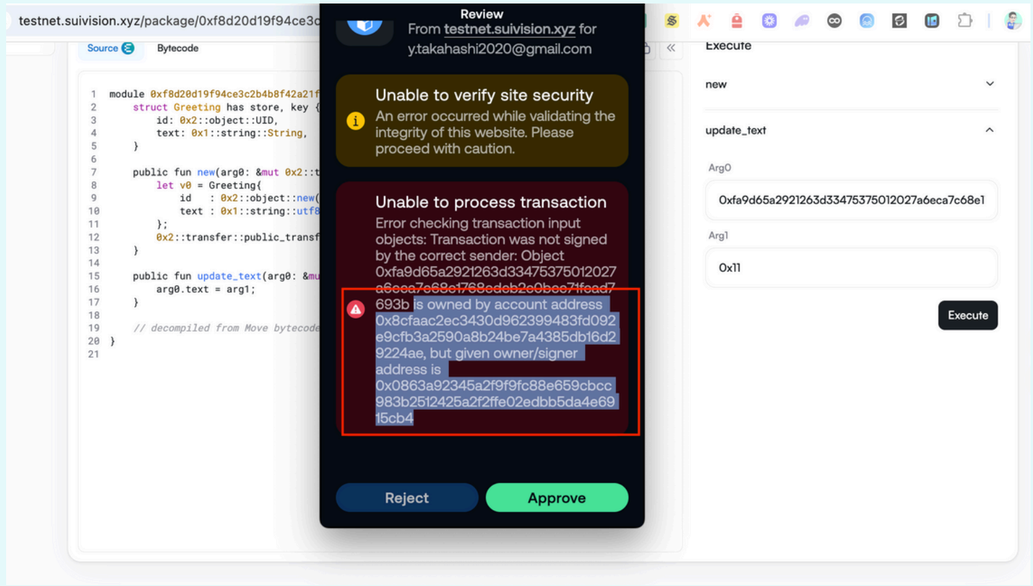
開発12　所有者以外で実行できない確認

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

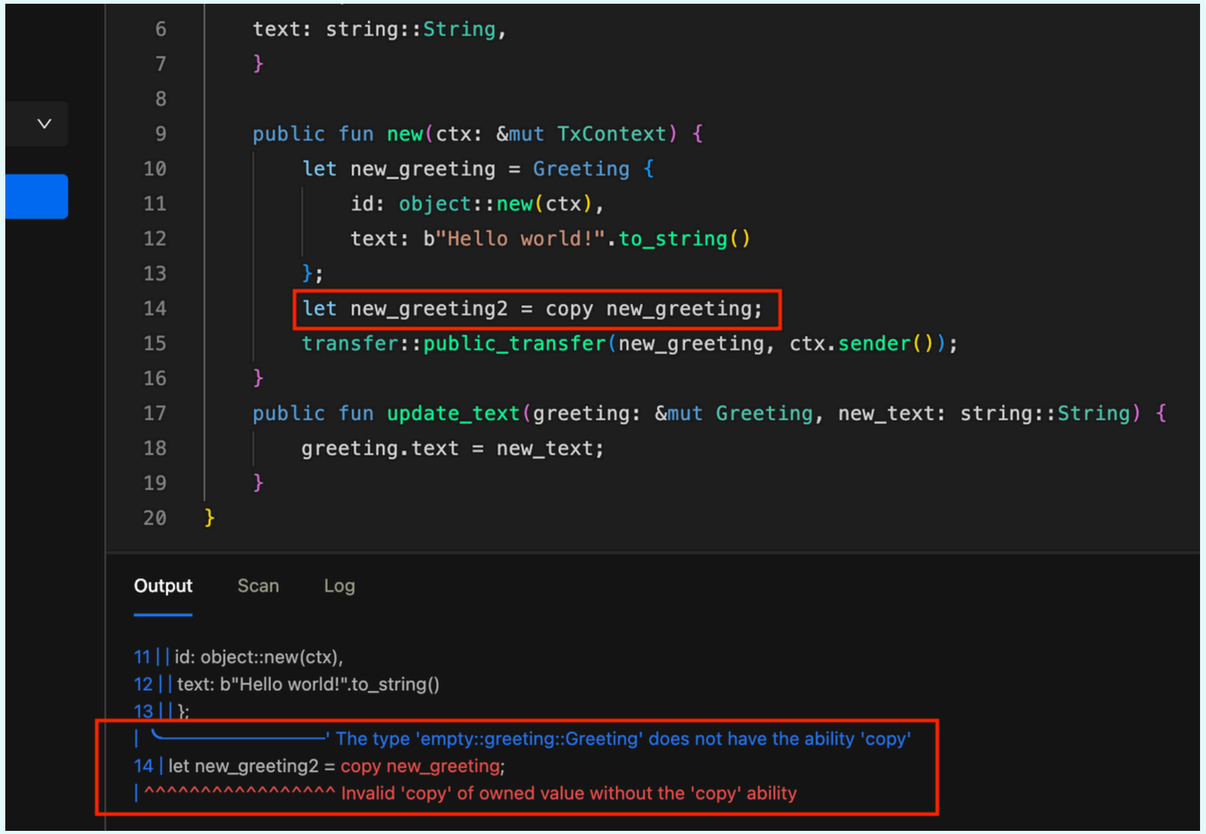
```move
module empty::greeting {
    use std::string;

    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

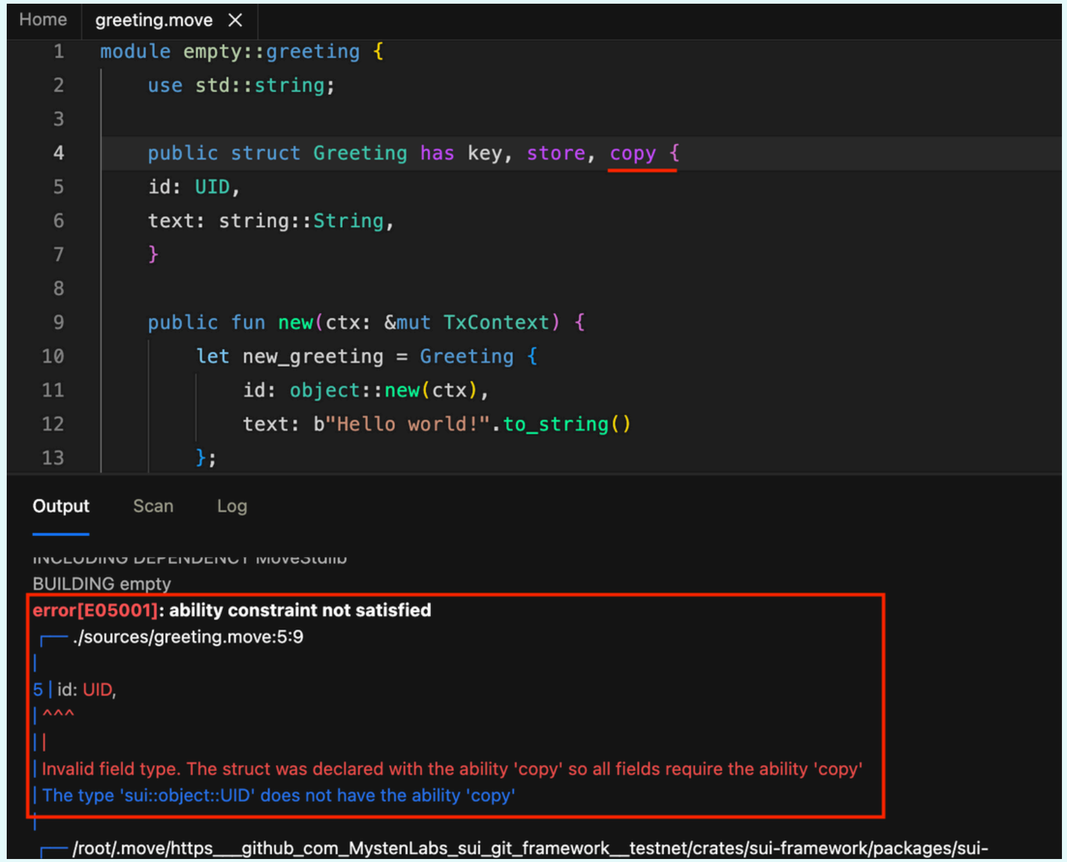
開発13　コピーができないことの確認

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。



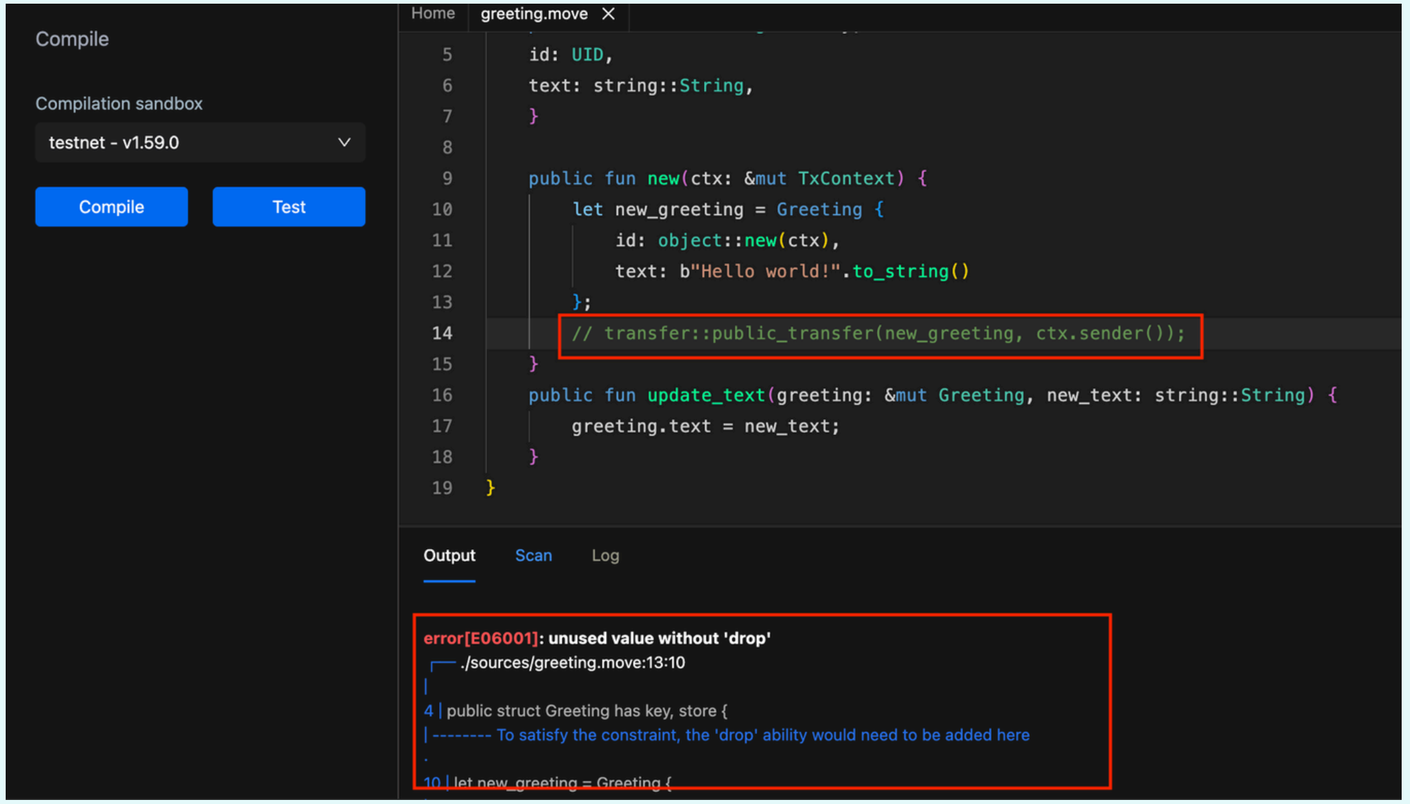
開発14　コピーとキーが両立しない確認

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**。



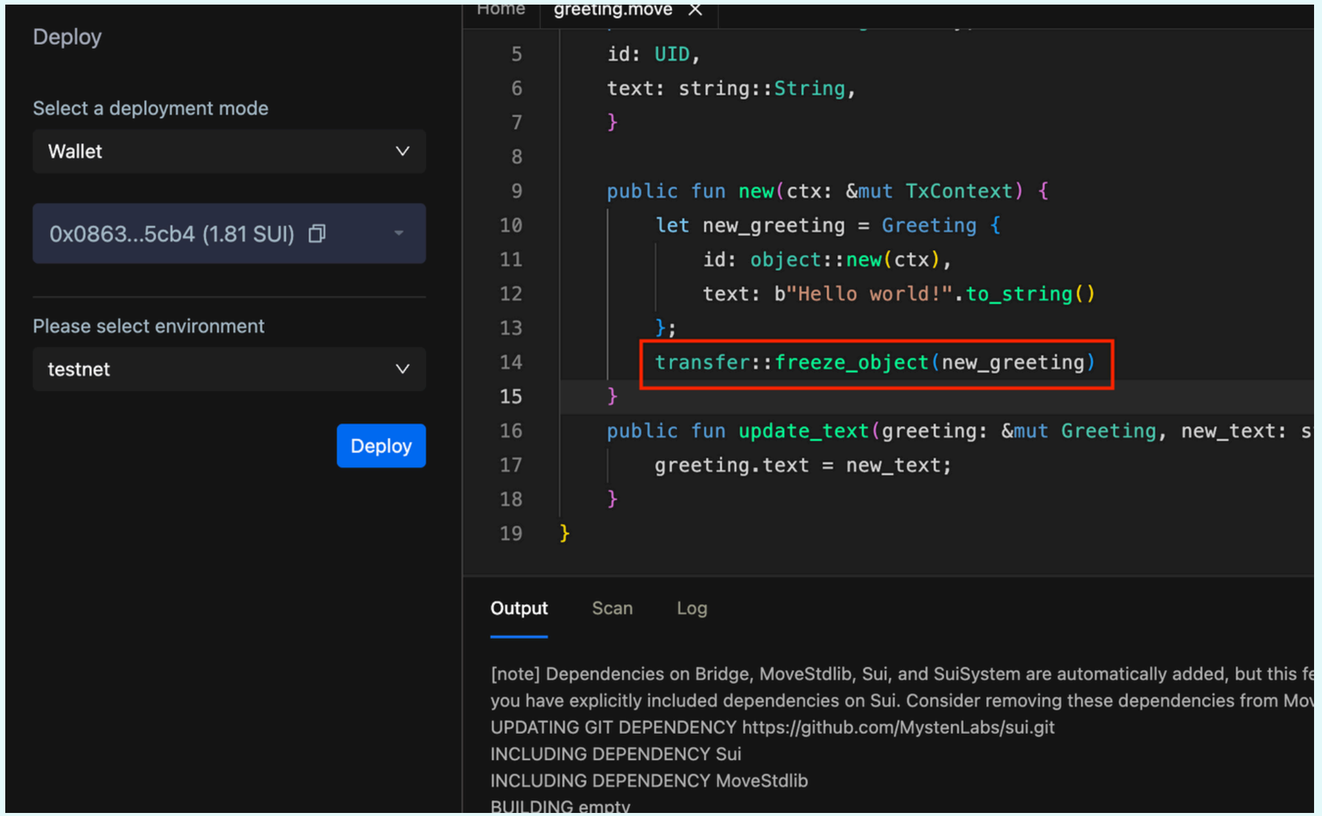
開発15　dropできないことの確認

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。



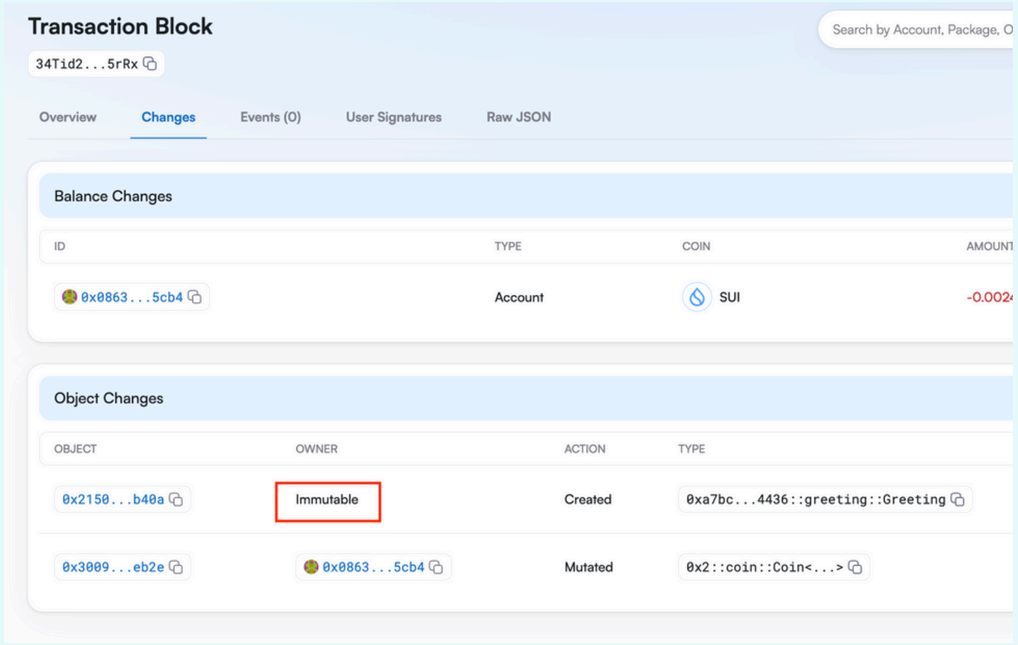
開発16　Immutableの設定

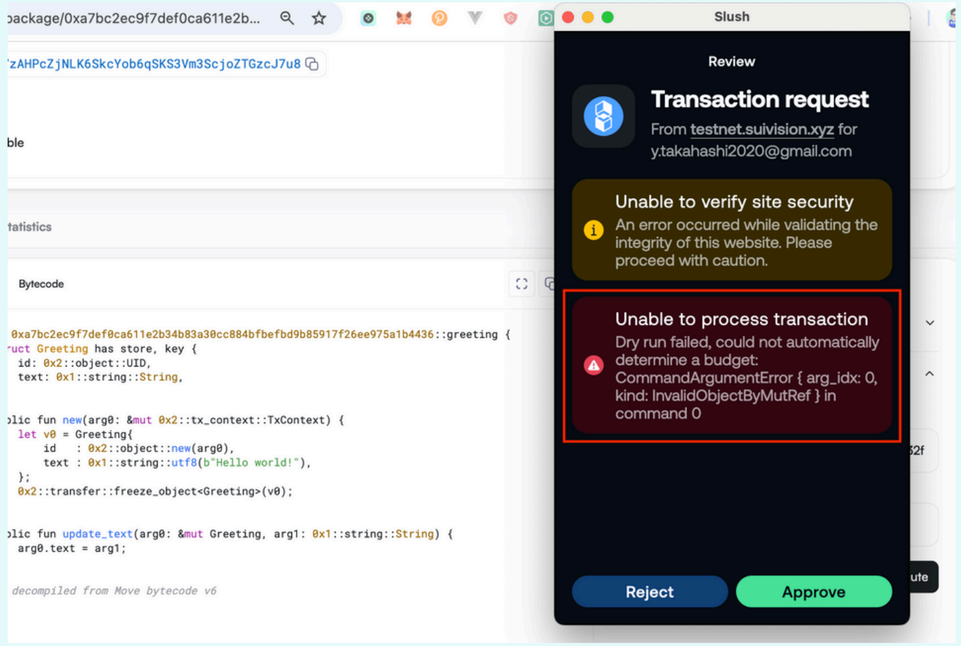
Immutableにすることは所有者を設定しないことではありません。
「所有者なし」という所有の状態に設定しています。



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。





リソースとオブジェクト

リソース

希少性を保証し、デジタル資産の複製を防ぎます。

オブジェクト

Suiの中核データ構造。所有、共有、不変のいずれかになり得ます。

所有権

Wrapping（ラップ）により単一所有を強制し、不正な複製を防ぎます。

移転（Transfer）

`transfer::transfer` を用いて、オブジェクトの所有権を安全に移転します。

型からの派生　②オブジェクトと所有権

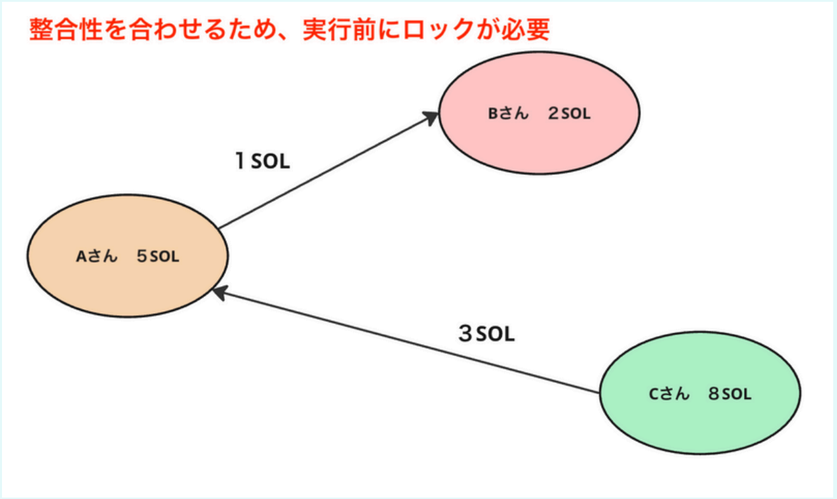
Suiはデータの値ではなく、**オブジェクトという形**で判断します。

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。

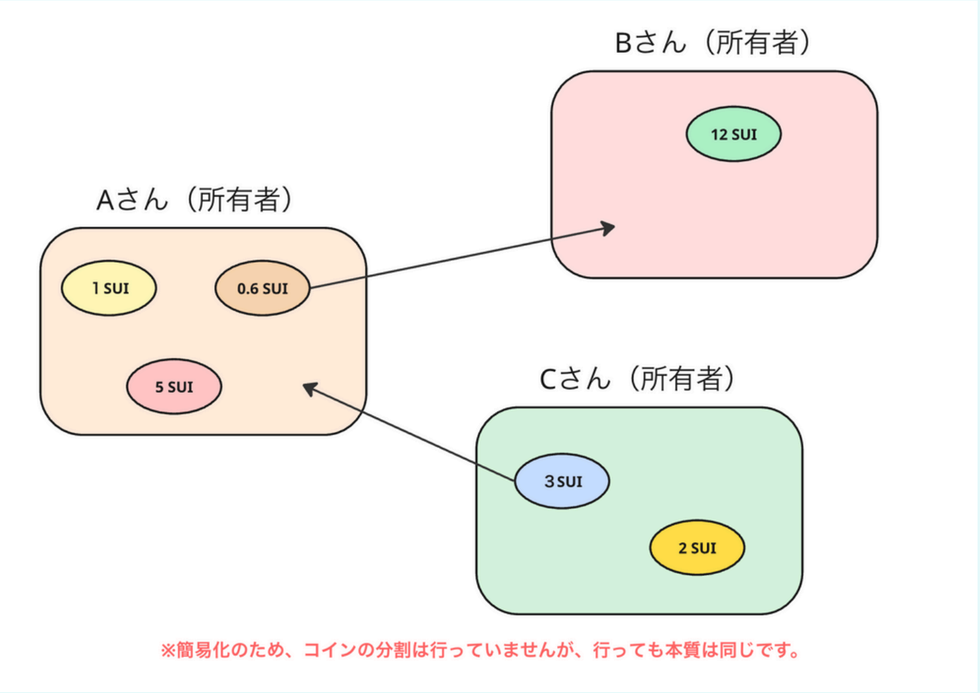


型からの派生　②オブジェクトと所有権

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。



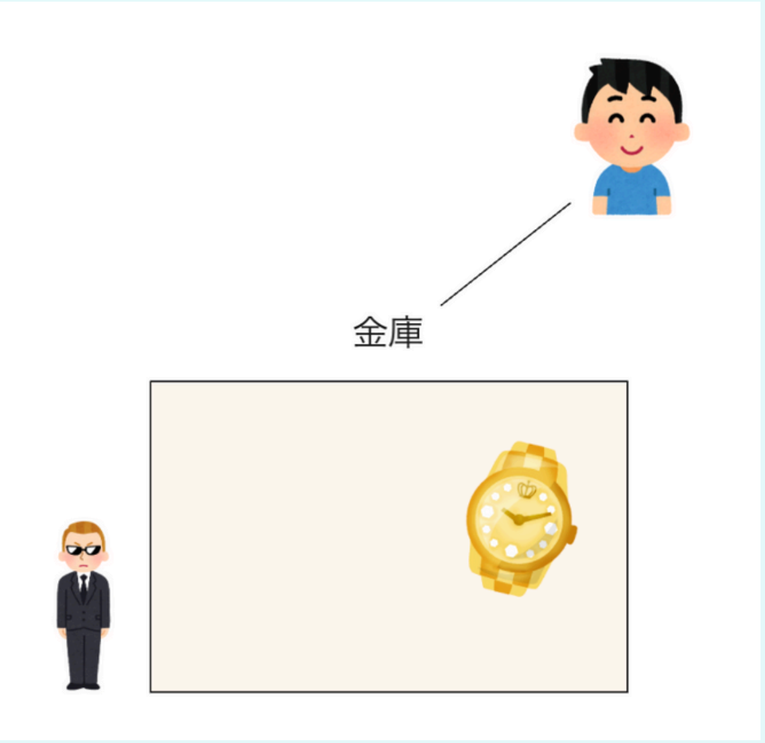
型からの派生　③ラップと単一所有の強制

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。

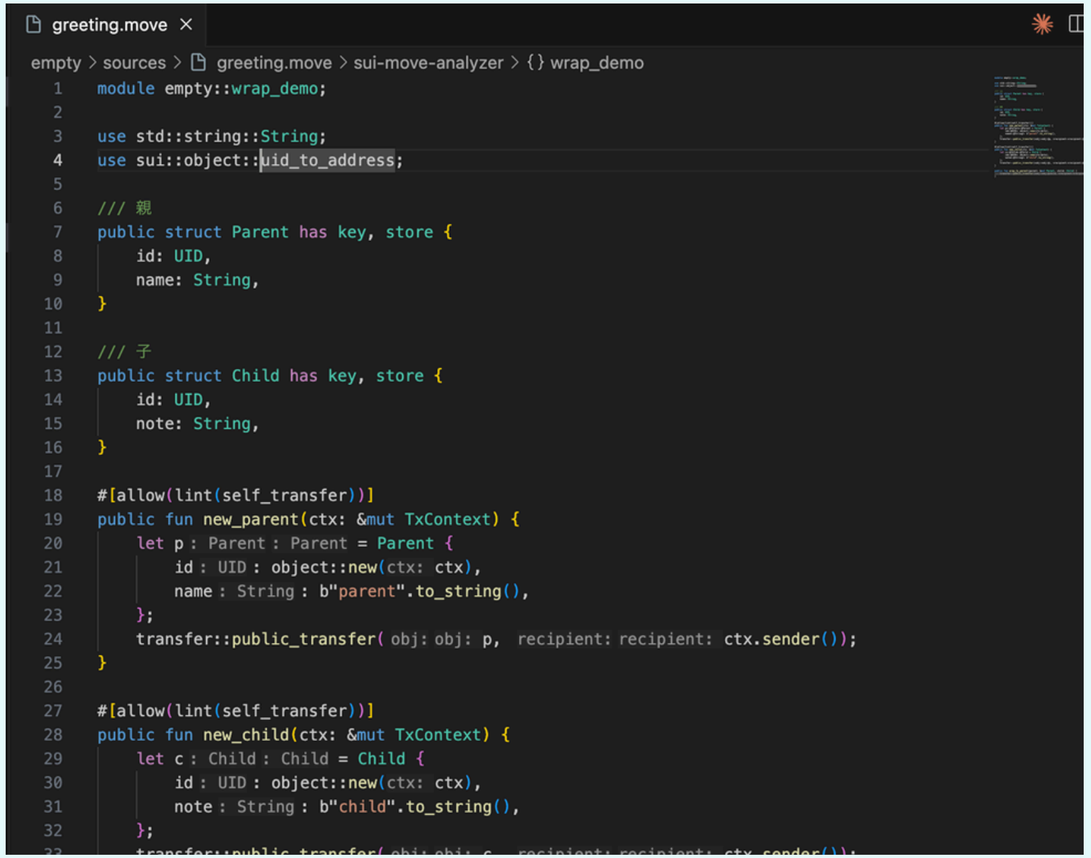
動かしたいときはバッグの所有者のあなたが
バッグごと動かせば良いのです。

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

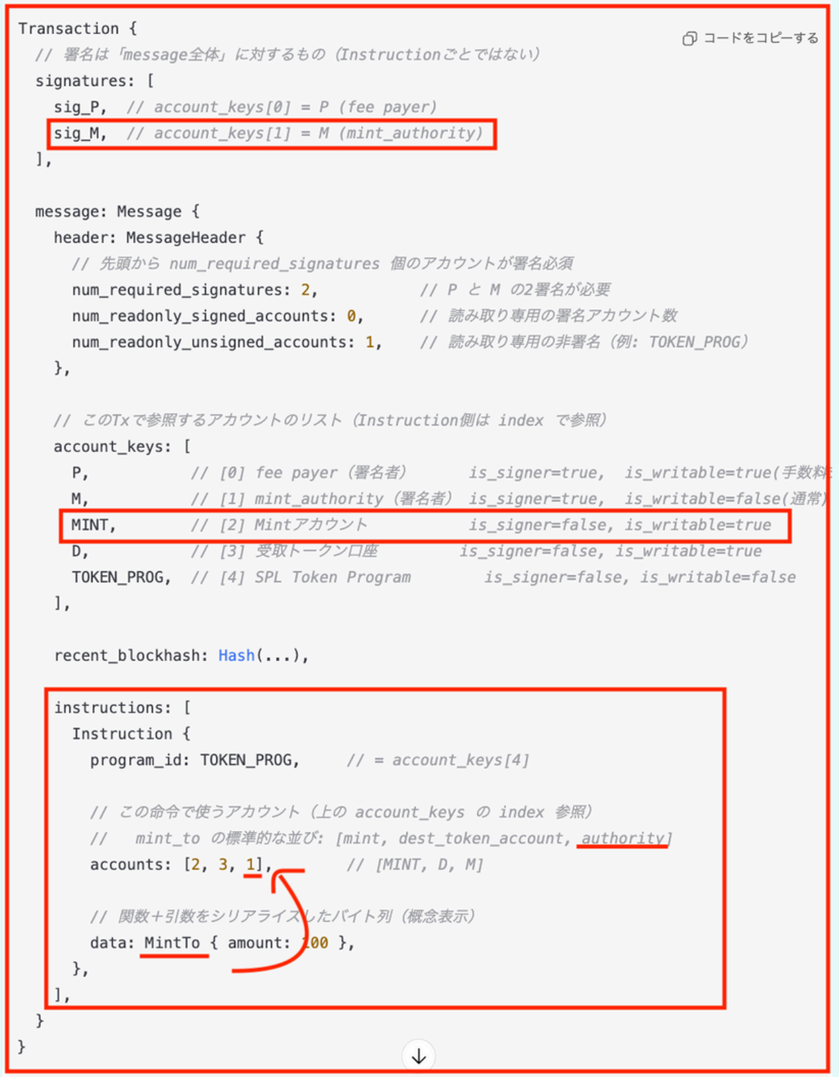
Solanaの場合、権限チェックは**実行時**に行われます。

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。



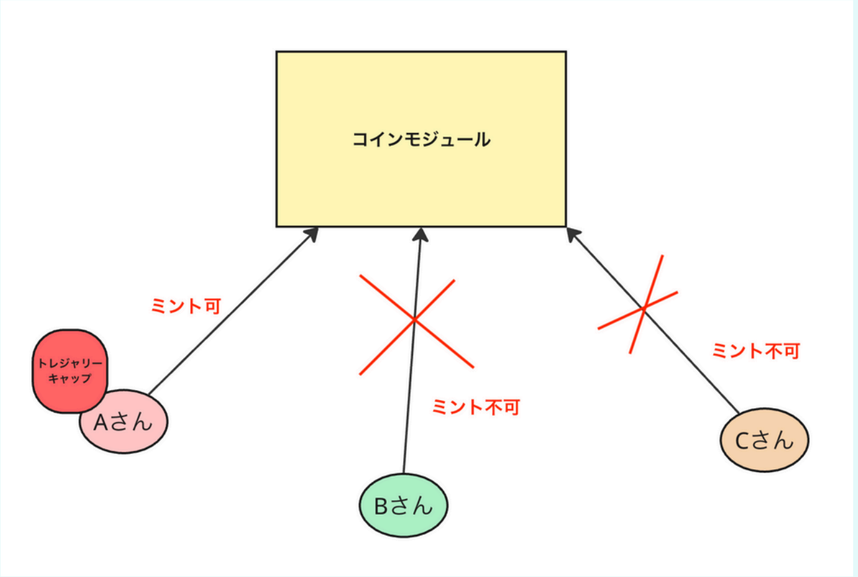
型からの派生　④Capabilityによる権限

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());
}
```

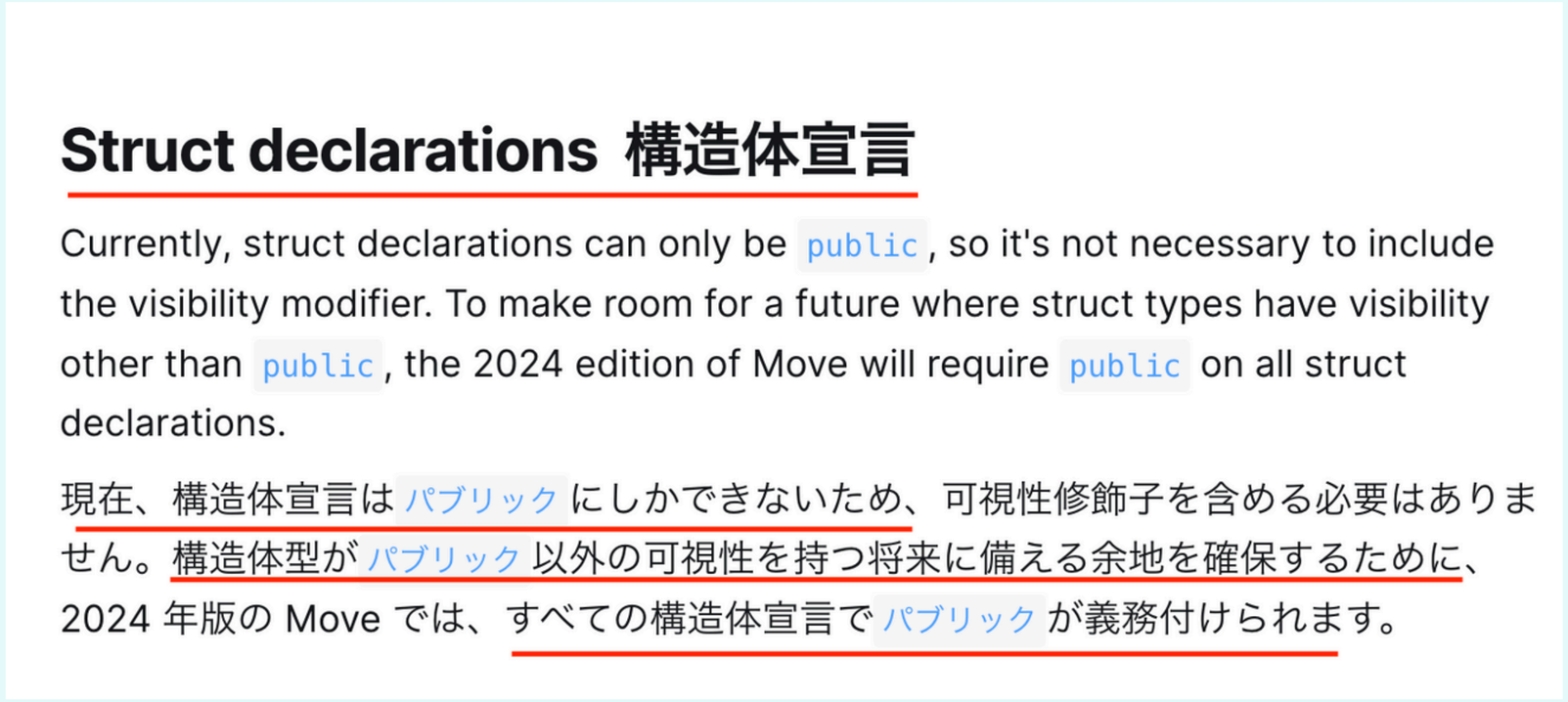
https://docs.sui.io/guides/developer/currency

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

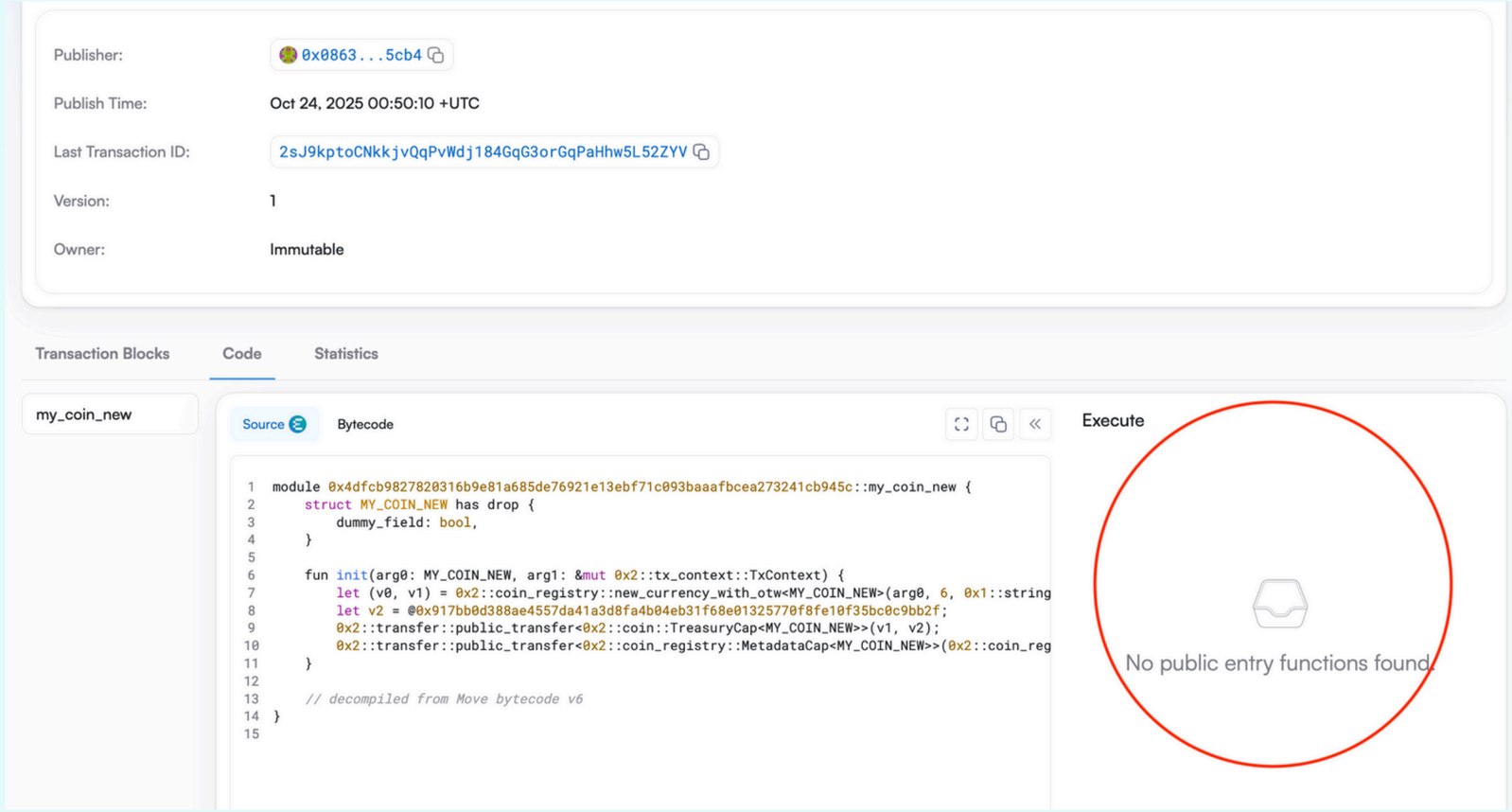
発行権限、メタデータの修正権限を実行者に送ります。

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。



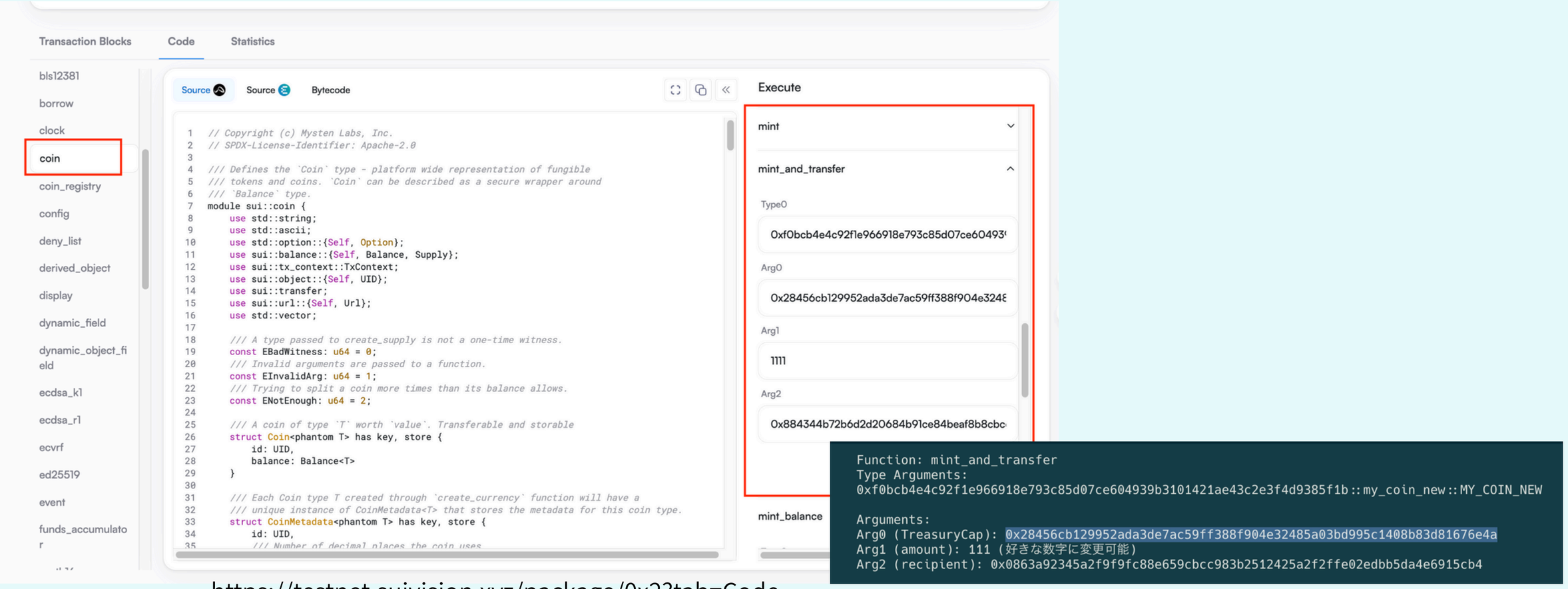
開発25　public, entry関数

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

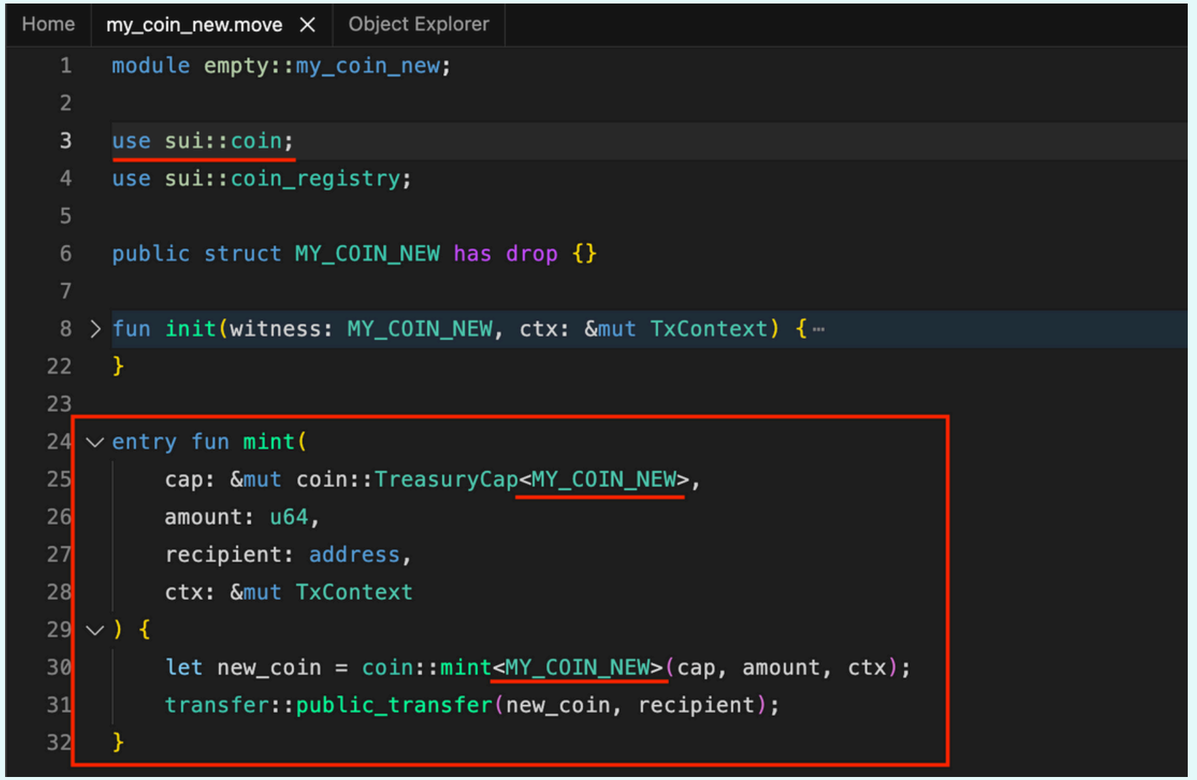
0x2パッケージで実行してみましょう。



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

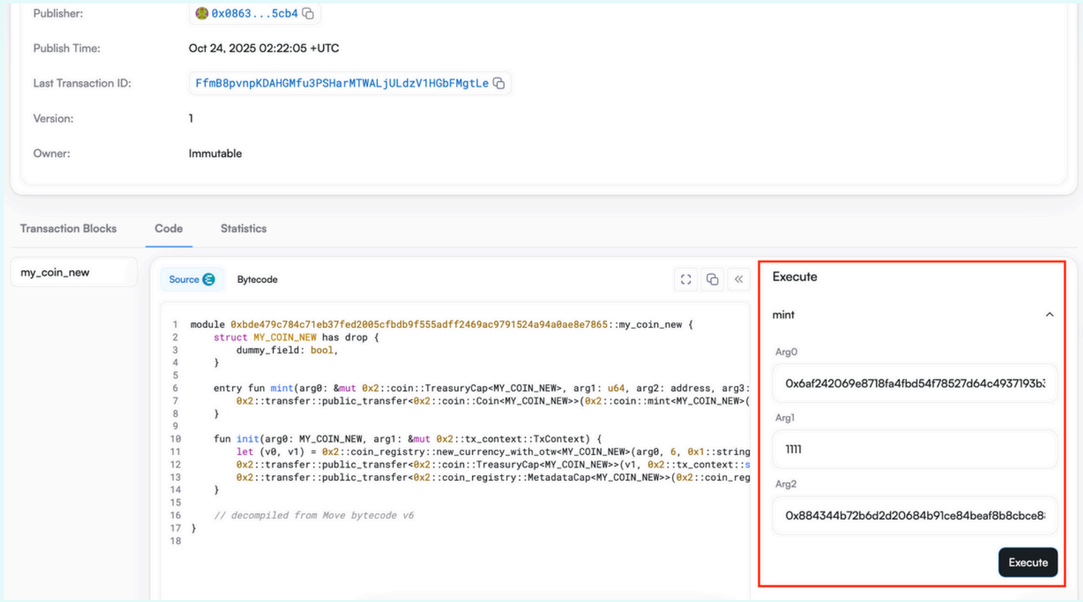
開発27　entry関数の作成

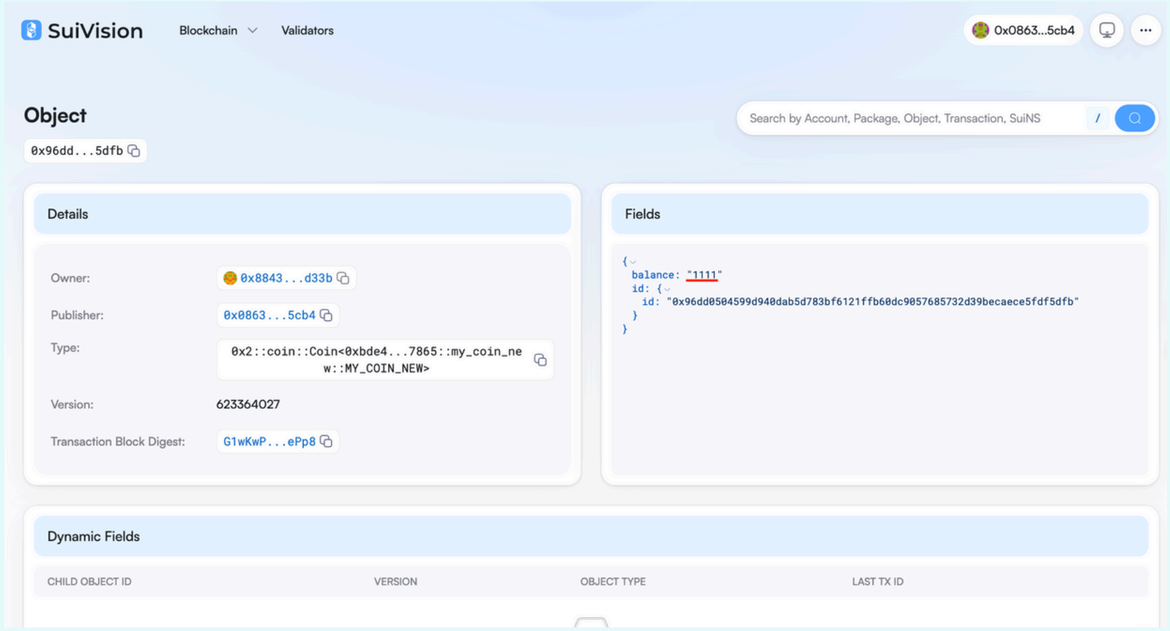
以下のようにmint関数を作ります。
<MY_COIN_NEW>で型を設定しています。



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。





セキュリティの考慮事項

ガス最適化

高コストなトランザクションによるサービス拒否を避けるため、コストを最小化します。

リエントランシー対策

リソースモデルによりリエントランシー攻撃を効率的に阻止します。

形式検証

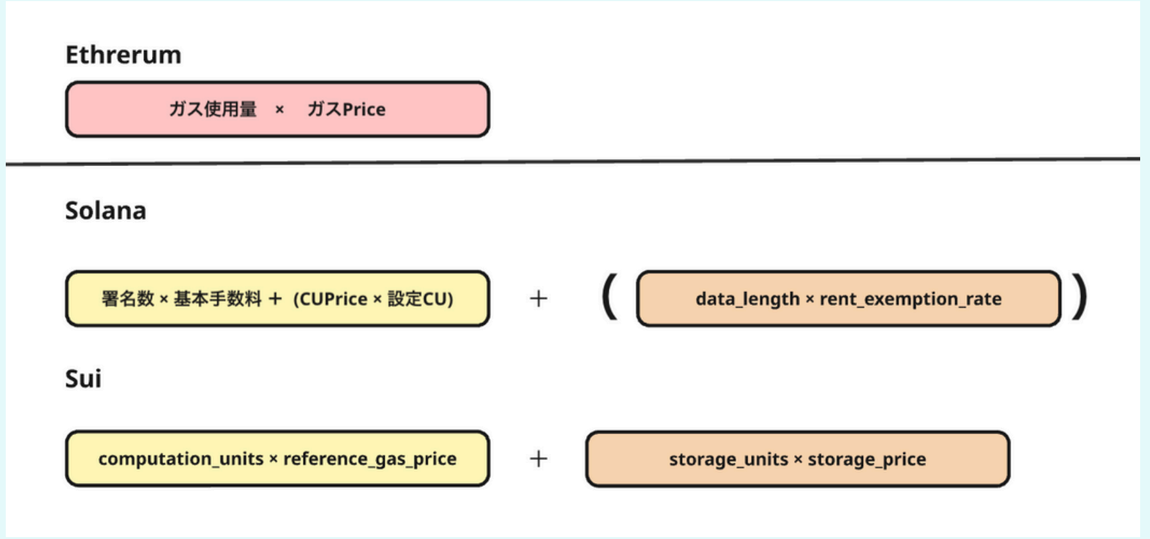
重要なコントラクトロジックの正しさを数学的に証明します。

型からの派生　⑥ガス最適化

Ethereumの場合、計算コストとストレージコストが合算してガスPriceにかけていますが、**Solana・Suiは単価を分けて**います。

Solanaの場合は、事前にレンタル代として確保するので、
書き方を変えています。

Suiでは**オブジェクトで所有者が決まっている**ので、**ストレージの計算が明瞭**です。



https://docs.sui.io/concepts/tokenomics/gas-in-sui

型からの派生　⑦形式的検証

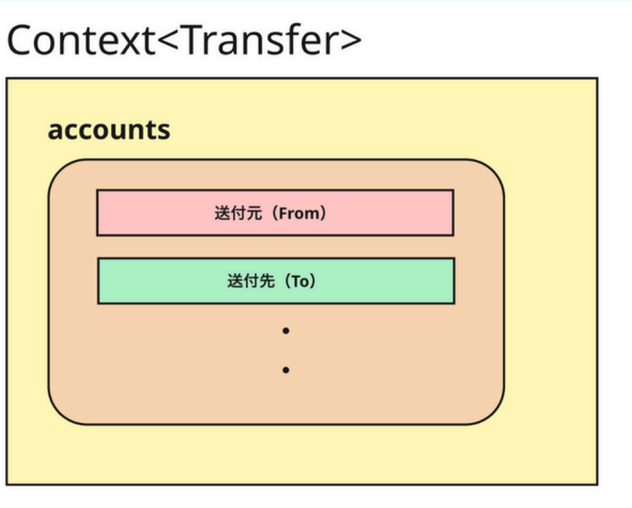
Suiは**客観的に数学的にロジックが正しいことを検証**できます。

例えば、Solanaで**送付時に総量が変わらないこと**を客観的に証明してみましょう。

動的に中身が変わったり、**他のトランザクションの影響もありうる**ので、証明すべきパターンが多くなり過ぎてしまいます。

```
fn transfer(ctx: Context<Transfer>, amount: u64) -> Result<()> {
    let from = &mut ctx.accounts.from;
    let to = &mut ctx.accounts.to;
    from.amount -= amount;
    to.amount += amount;
    Ok(())
}
```

関数にContextという箱を渡し、
その中に入っているアカウントを操作する

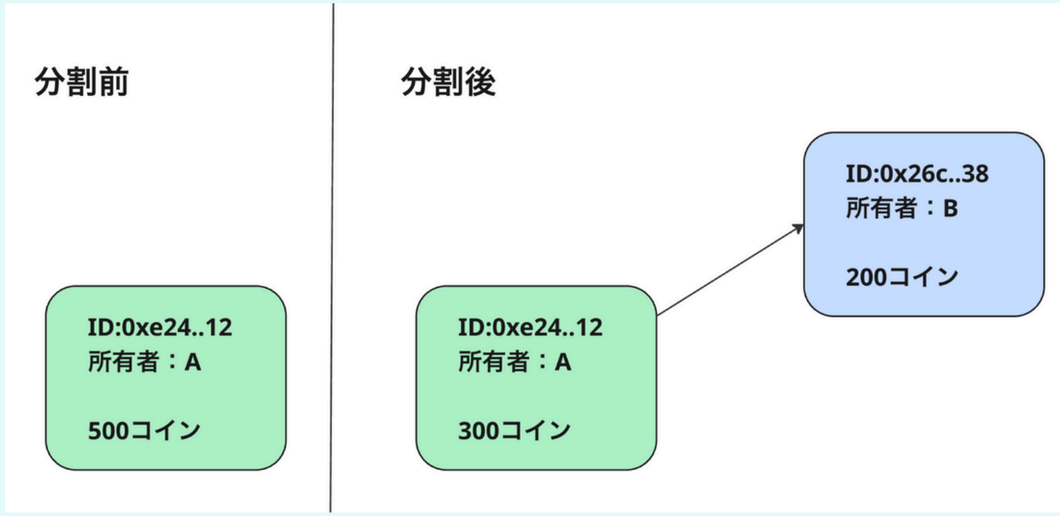


型からの派生　⑦形式的検証

Suiの場合は「何を」「いくら」「誰に」渡すかを具体的に渡すため、**証明すべき範囲が明確**です。

それだけでなくコインの場合、**複製・破棄がルール上不可能**であり、**ミント権限（Capability）を持っていないことが客観的に判断**できるので、形式上問題ないかを検証することができます。

```
module transfer_demo::utils {
    use sui::coin::{Self as coin, Coin};
    use sui::tx_context::TxContext;

    public entry fun transfer<T>(
        c: &mut Coin<T>,        // 何を（どのコインを）
        amount: u64,            // いくら
        to: address,            // 誰に
        ctx: &mut TxContext
    ) {
        // 新しいコインを生成するので ctx が必要
        let part = coin::split(c, amount, ctx);
        // 所有権の移動は ctx 不要
        coin::transfer(part, to);
    }
}
```



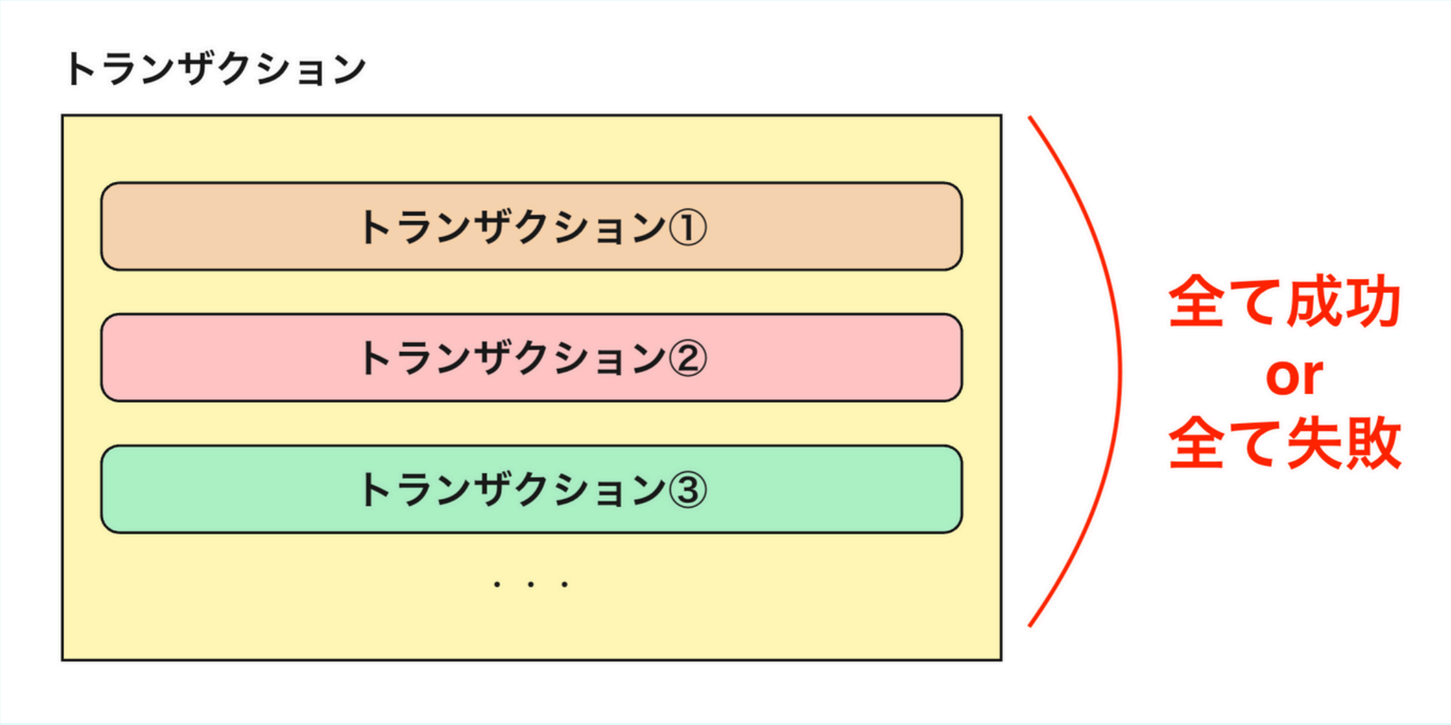
Build on Sui

Weekend Move Workshop in Tokyo // Day 1

Powered by

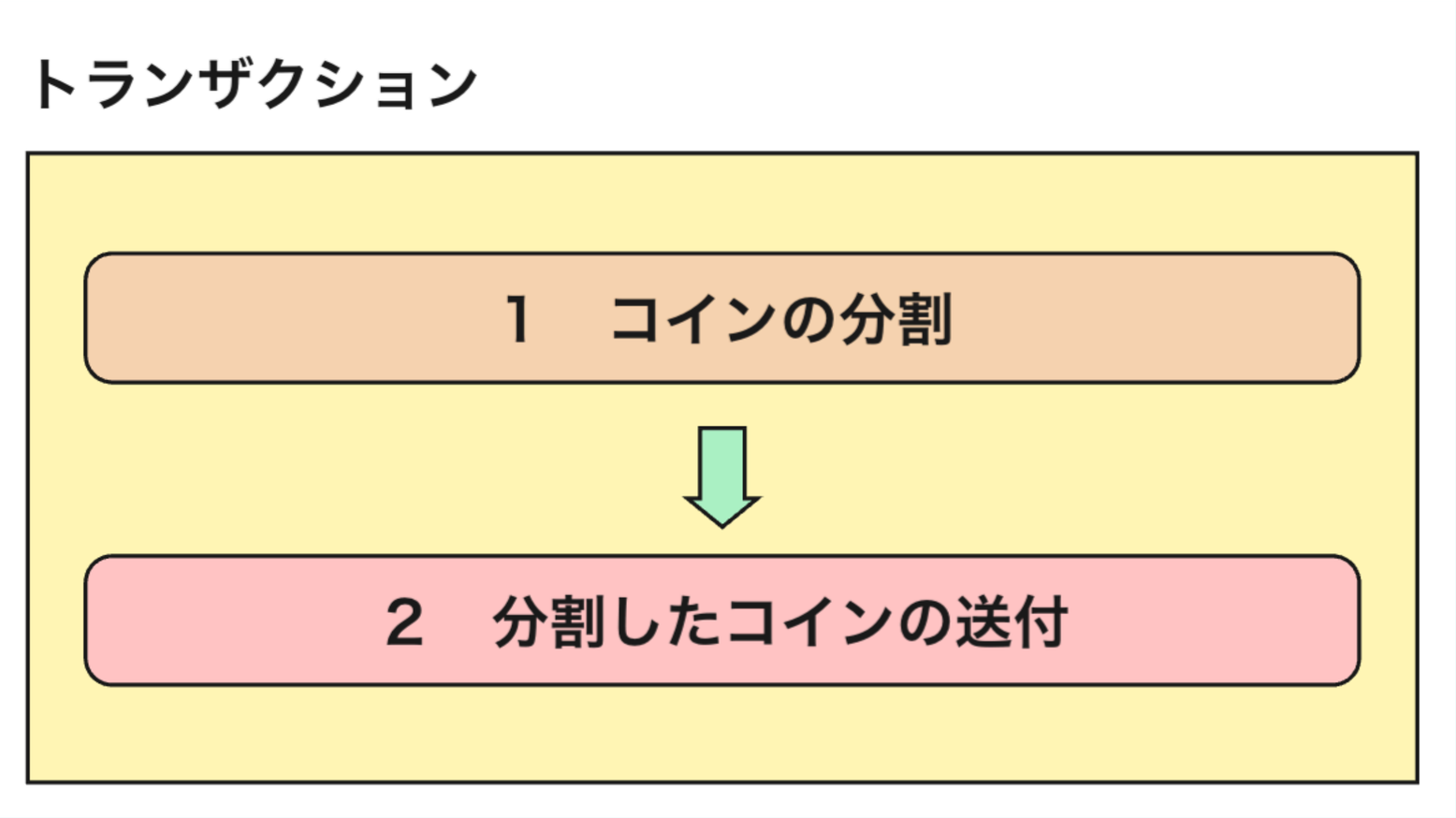
1 PTBとは

PTB（programmable transaction block）はプログラム可能なトランザクションのブロックです。
まとめることで**原子性（全て成功か失敗）**を確保できます。



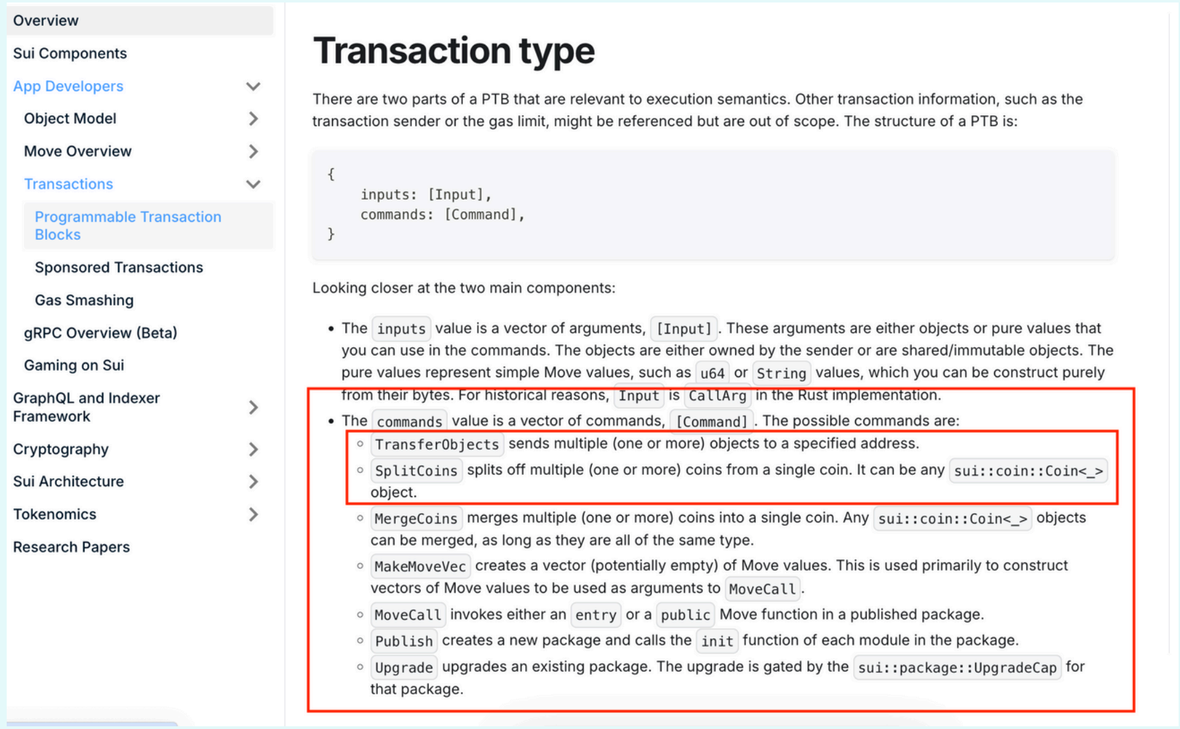
1 PTBとは

また、大きな特徴として、**前の結果をそのまま次に渡せ**ます。



1 PTBとは

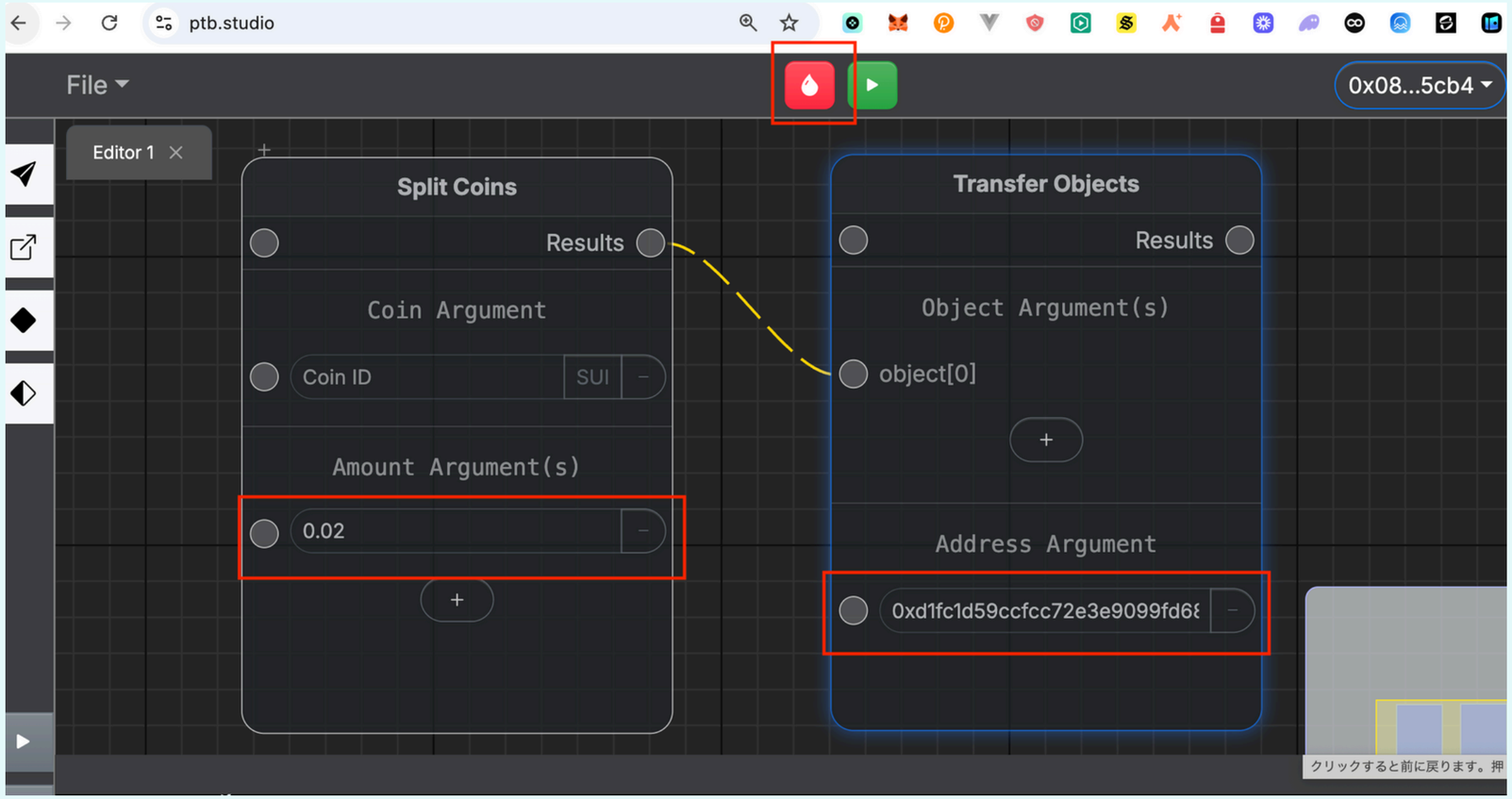
こちらが使用できるコマンドです。
今回は「TransferObjects」と「SplitCoins」を使います。



https://docs.sui.io/concepts/transactions/prog-txn-blocks

1 PTBとは

ptb.studioで体験してみましょう。

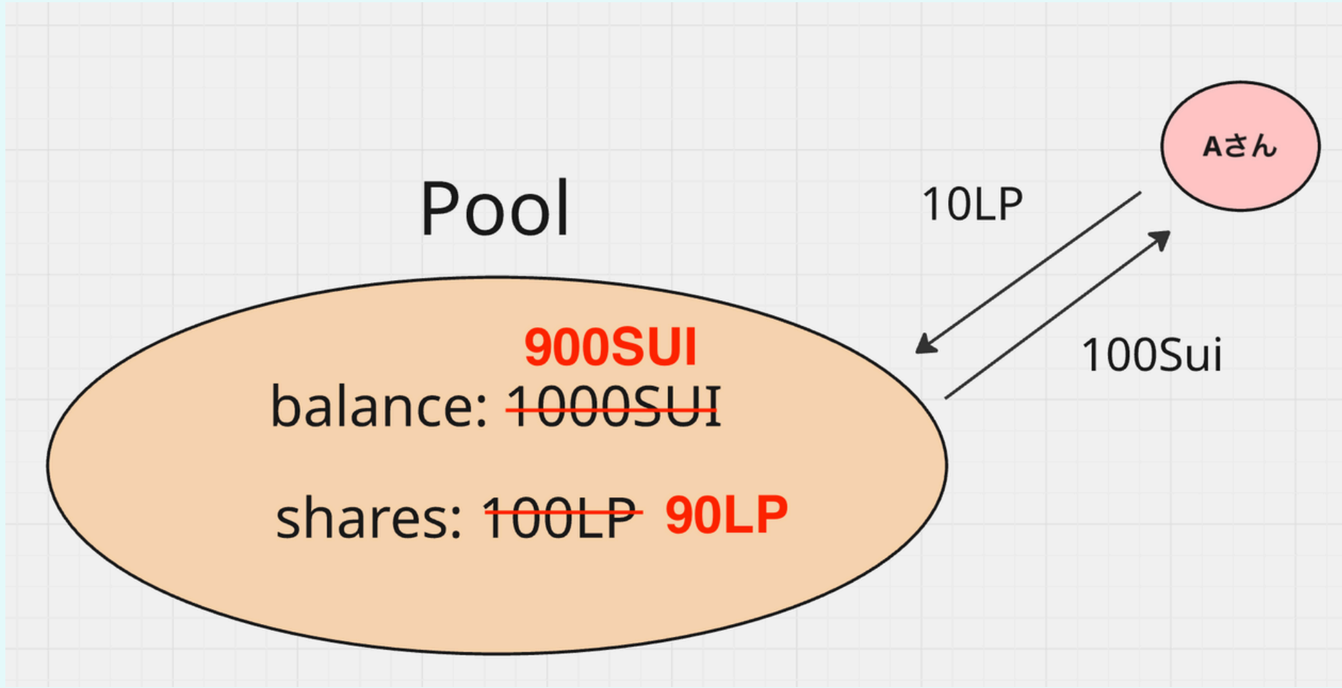


https://ptb.studio/

2 形式的検証　Sui Prover

プールからの引き出し（withdraw）について考えてみます。

例えば、所有している10LPを使用することで、その割合分である100SUIが取得できるとします。

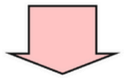


2 形式的検証　Sui Prover

今回は、引き出し後のLPの価値が引き出し前の価値より下がらないことを証明します。

割り算は使いたくないので、下のような掛け算の内容を証明します。

$$\frac{\text{引き出し前の総額（1000SUI）}}{\text{引き出し前のシェア（100LP）}} \leqq \frac{\text{引き出し後の総額（900SUI）}}{\text{引き出し後のシェア（90LP）}}$$



引き出し前の総額（1000SUI）× 引き出し後のシェア（90LP）

≦ 引き出し後の総額（900SUI）× 引き出し前のシェア（100LP）

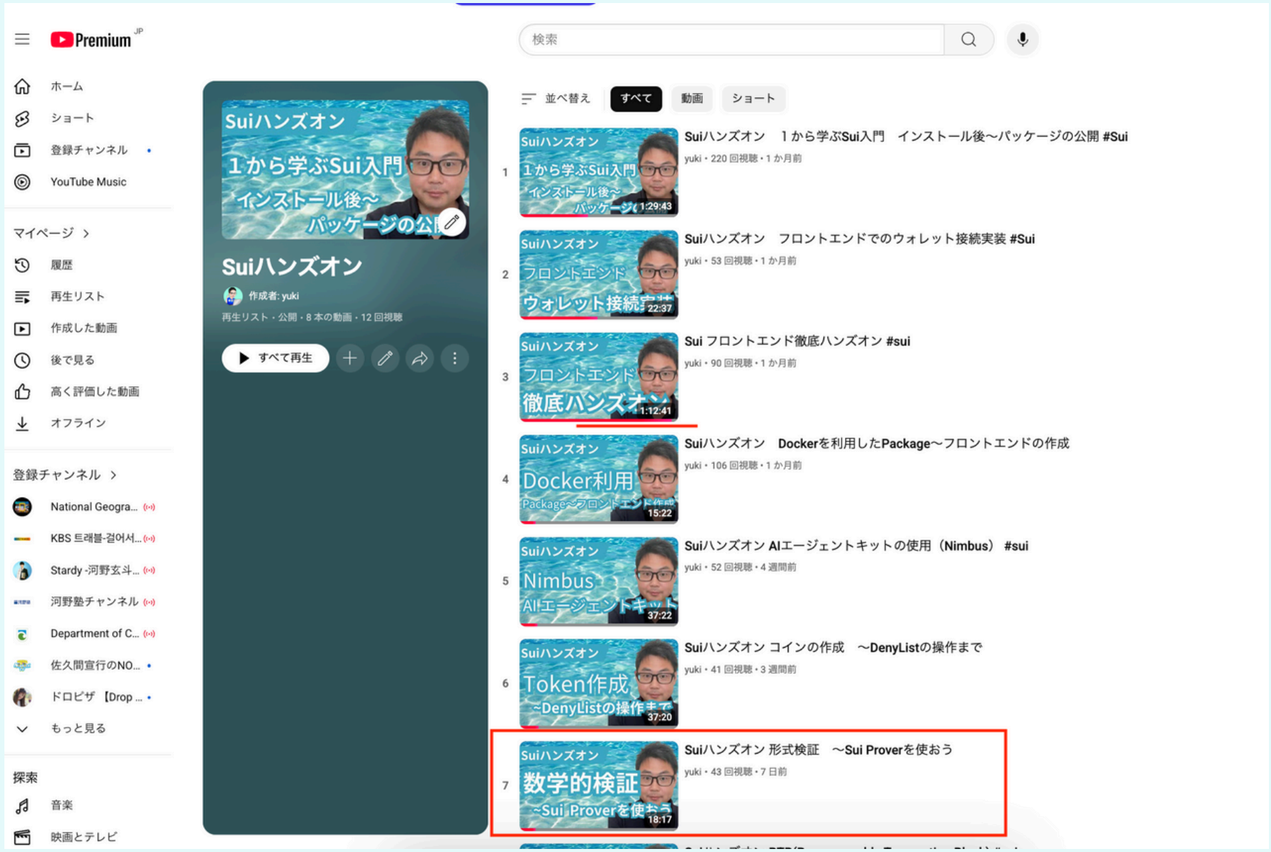
2 形式的検証　Sui Prover

以下のように、実行後に数学的に検証します。

https://github.com/SuiJapan/core-workshop/blob/main/chapter3/sui-prover/amm/sources/simple_lp.move#L31

2 形式的検証　Sui Prover

興味がありましたら、ぜひやってみてください



https://www.youtube.com/watch?v=wE_fu-nUdDE

3 Clockオブジェクト

ポイント
①引数のClockは不変で渡す
②時間の取得はclock.timestamp_ms()
③Clockのアドレスは0x6

examples/move/basics/sources/clock.move

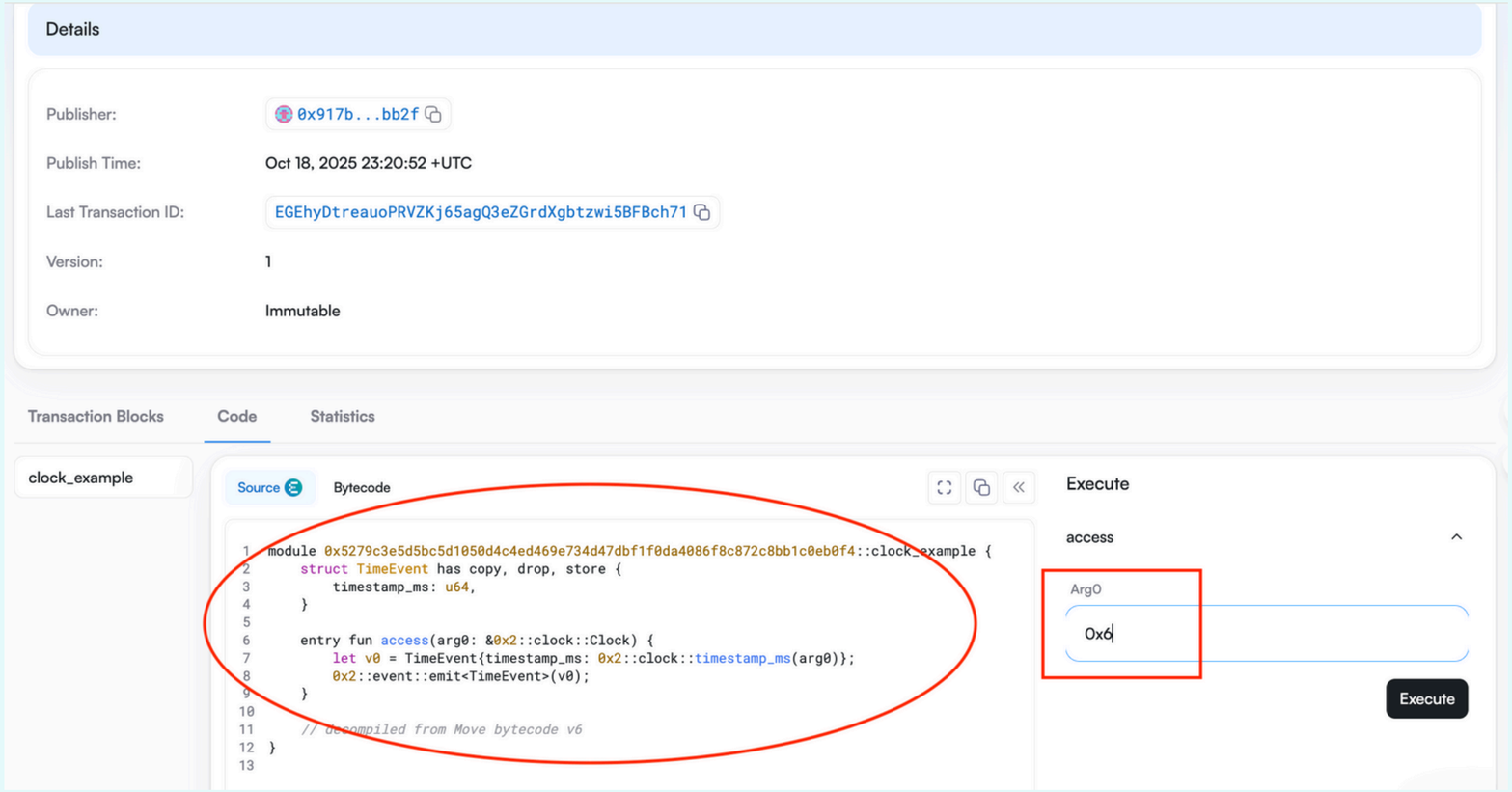
```
module basics::clock;

use sui::clock::Clock;
use sui::event;

public struct TimeEvent has copy, drop, store {
    timestamp_ms: u64,
}

entry fun access(clock: &Clock) {
    event::emit(TimeEvent { timestamp_ms: clock.timestamp_ms() });
}
```

A call to the previous entry function takes the following form, passing `0x6` as the address for the `Clock` parameter:

https://docs.sui.io/guides/developer/sui-101/access-time

3 Clockオブジェクト

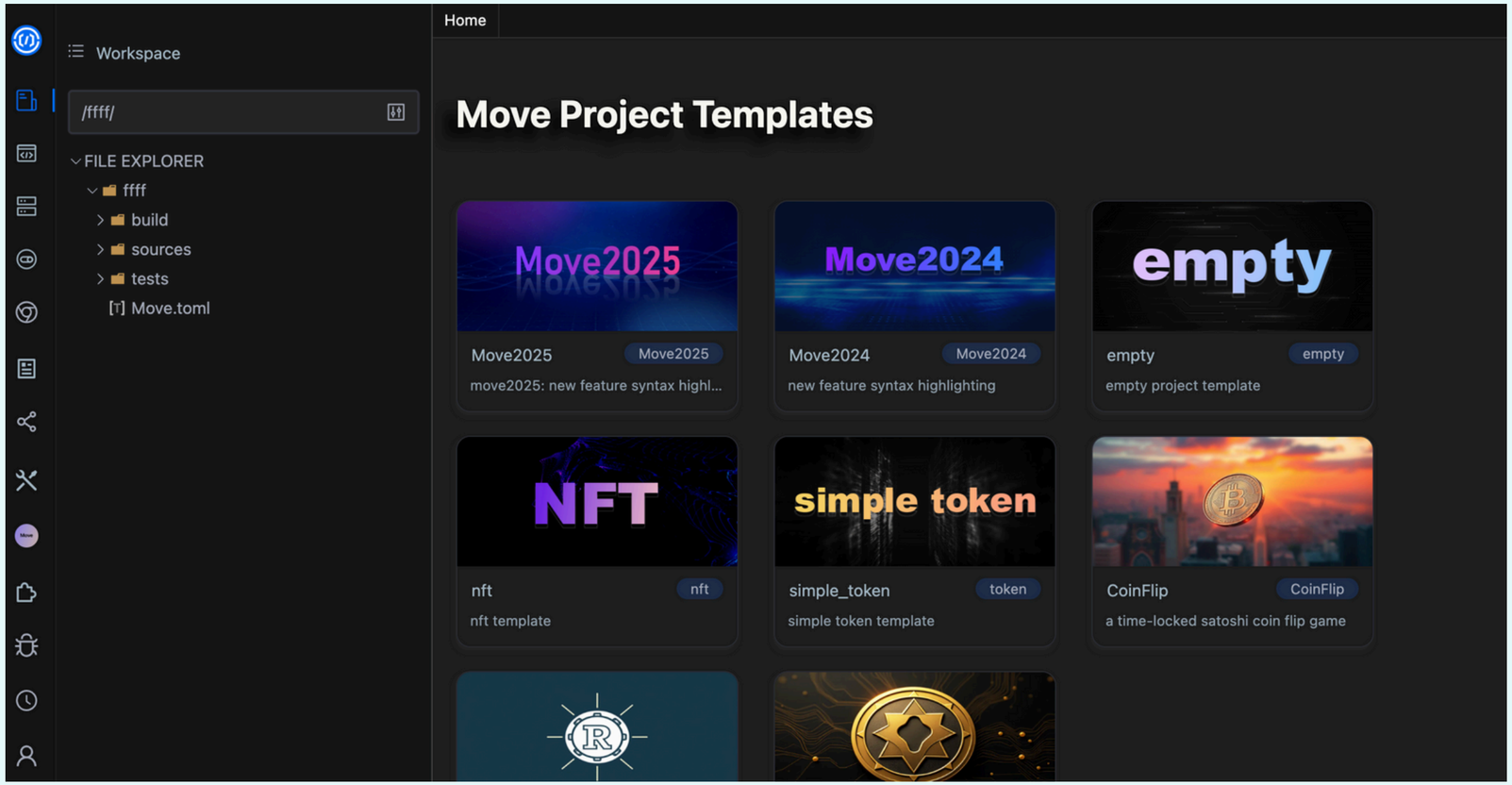
実際に処理を行ってみましょう。



https://testnet.suivision.xyz/package/0x5279c3e5d5bc5d1050d4c4ed469e734d47dbf1f0da4086f8c872c8bb1c0eb0f4?tab=Code

3 Clockオブジェクト

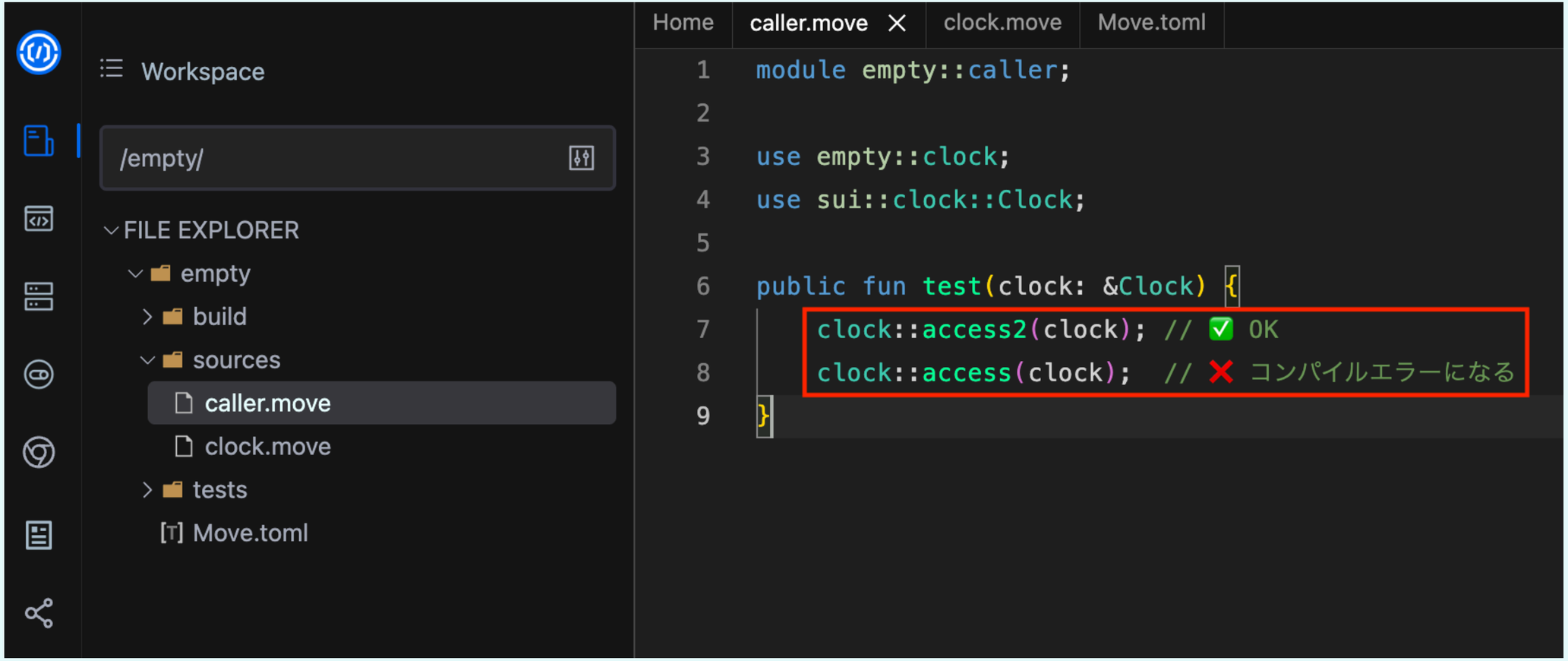
Bits Labを使ってみましょう。



https://ide.bitslab.xyz/

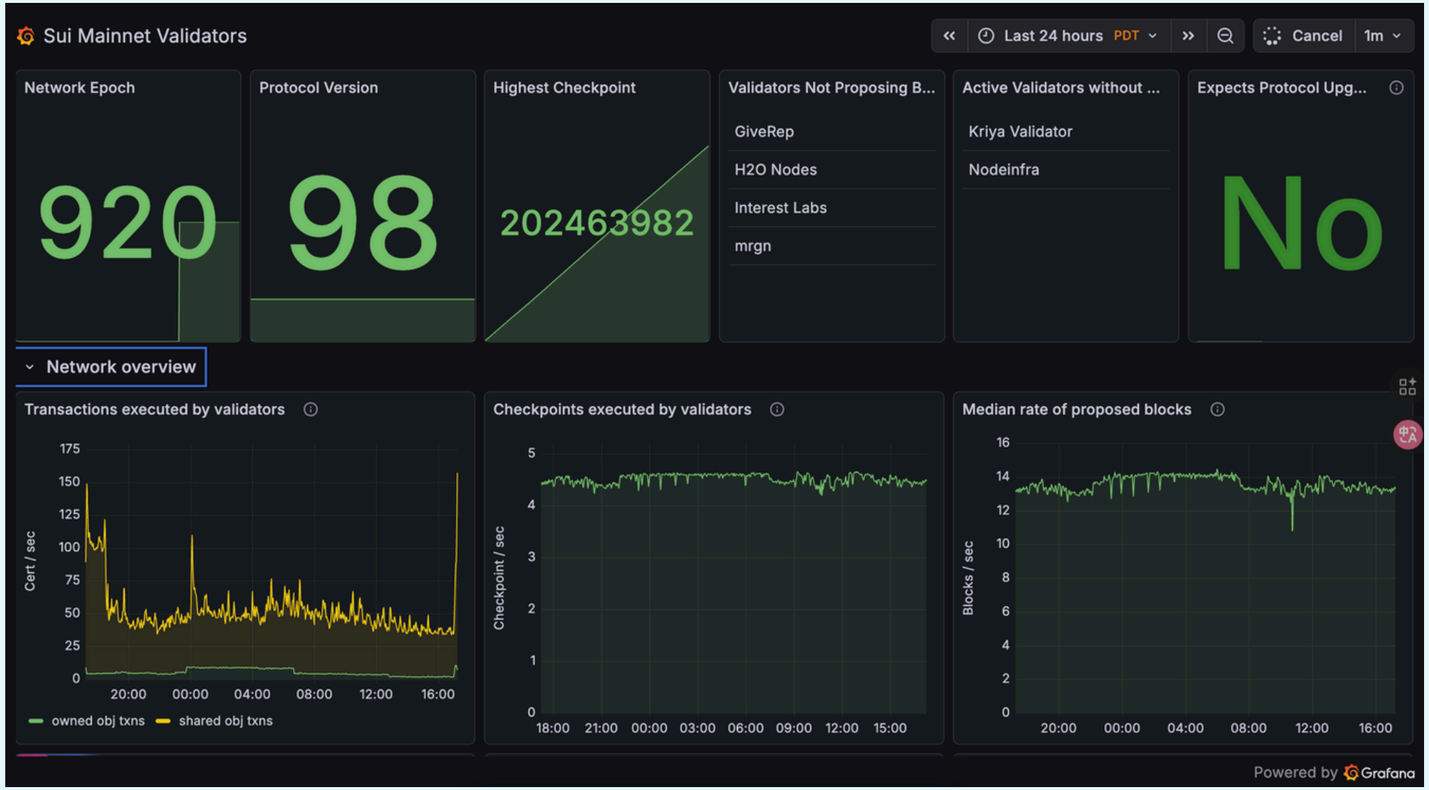
3 Clockオブジェクト

entryとpublicの違いを体験しましょう。



3 Clockオブジェクト

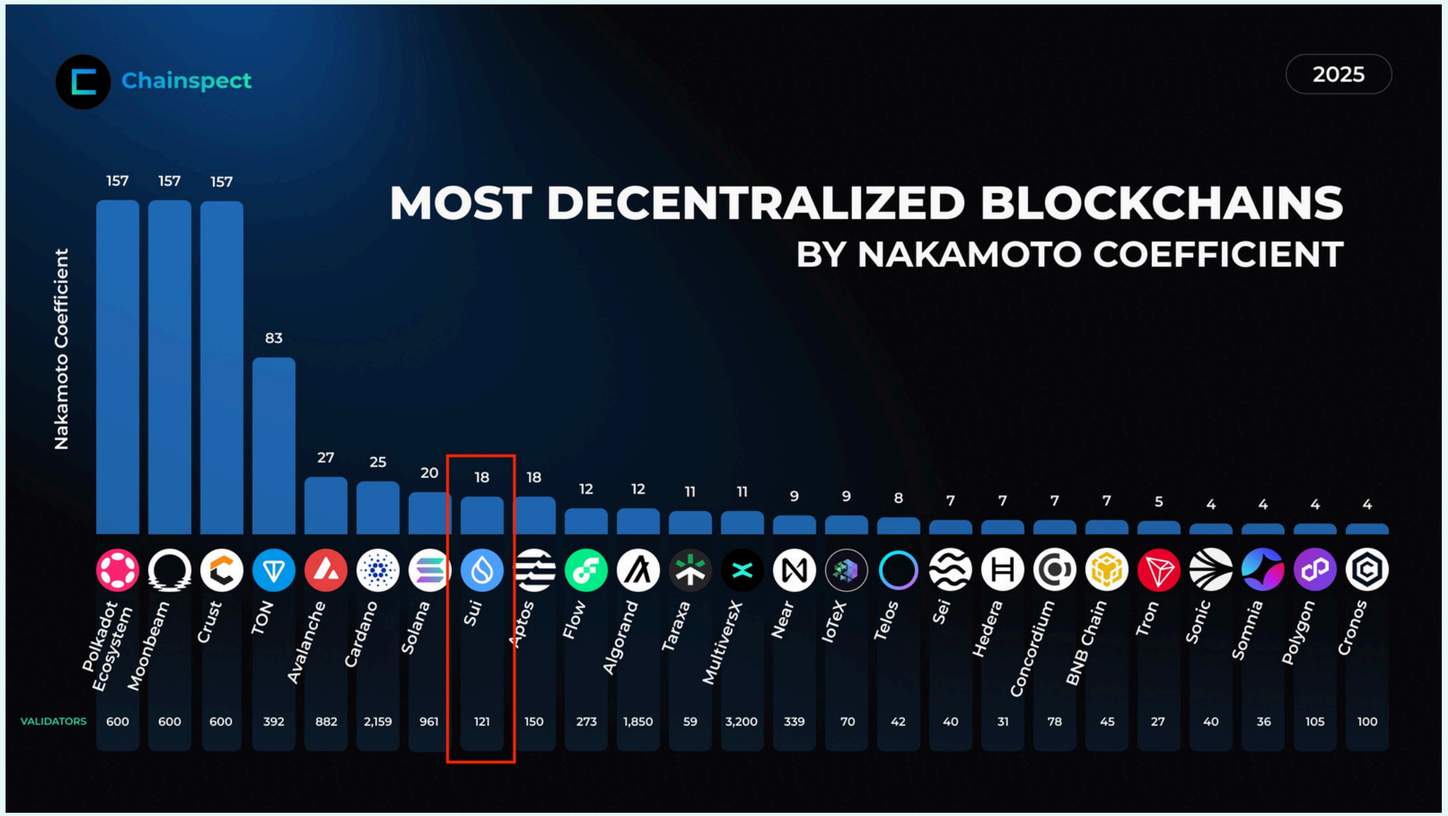
せっかくなので色々な指標を見てみましょう。



https://metrics.sui.io/public-dashboards/4ceb11cc210d4025b122294586961169?from=now-24h&to=now&timezone=America%2FLos_Angeles

余談　ナカモト指数について

何ノードが協力すれば、ネットワークを支配（50%）できるか？を示す数字です。



https://x.com/chainspect_app/status/1978096780580364414/photo/1

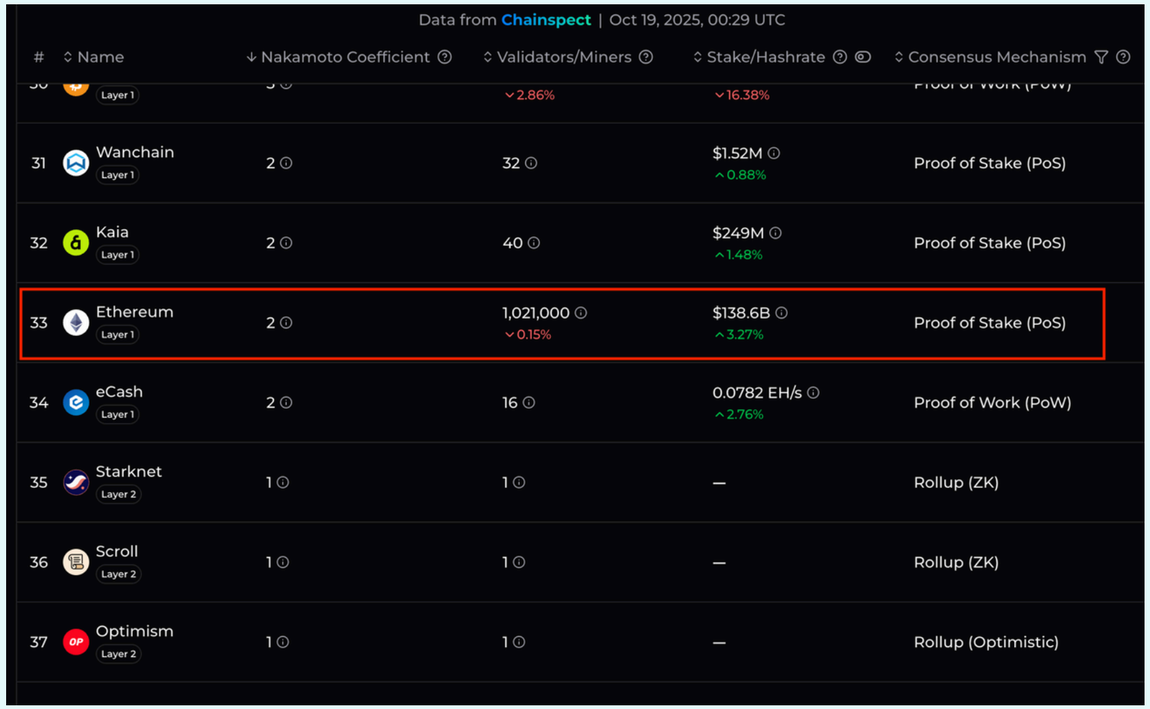
余談　ナカモト指数について

Suiの分散状況も見てみましょう。

https://metrics.sui.io/public-dashboards/4ceb11cc210d4025b122945869611 69?from=now-24h&to=now&timezone=America%2FLos_Angeles

余談　ナカモト指数について

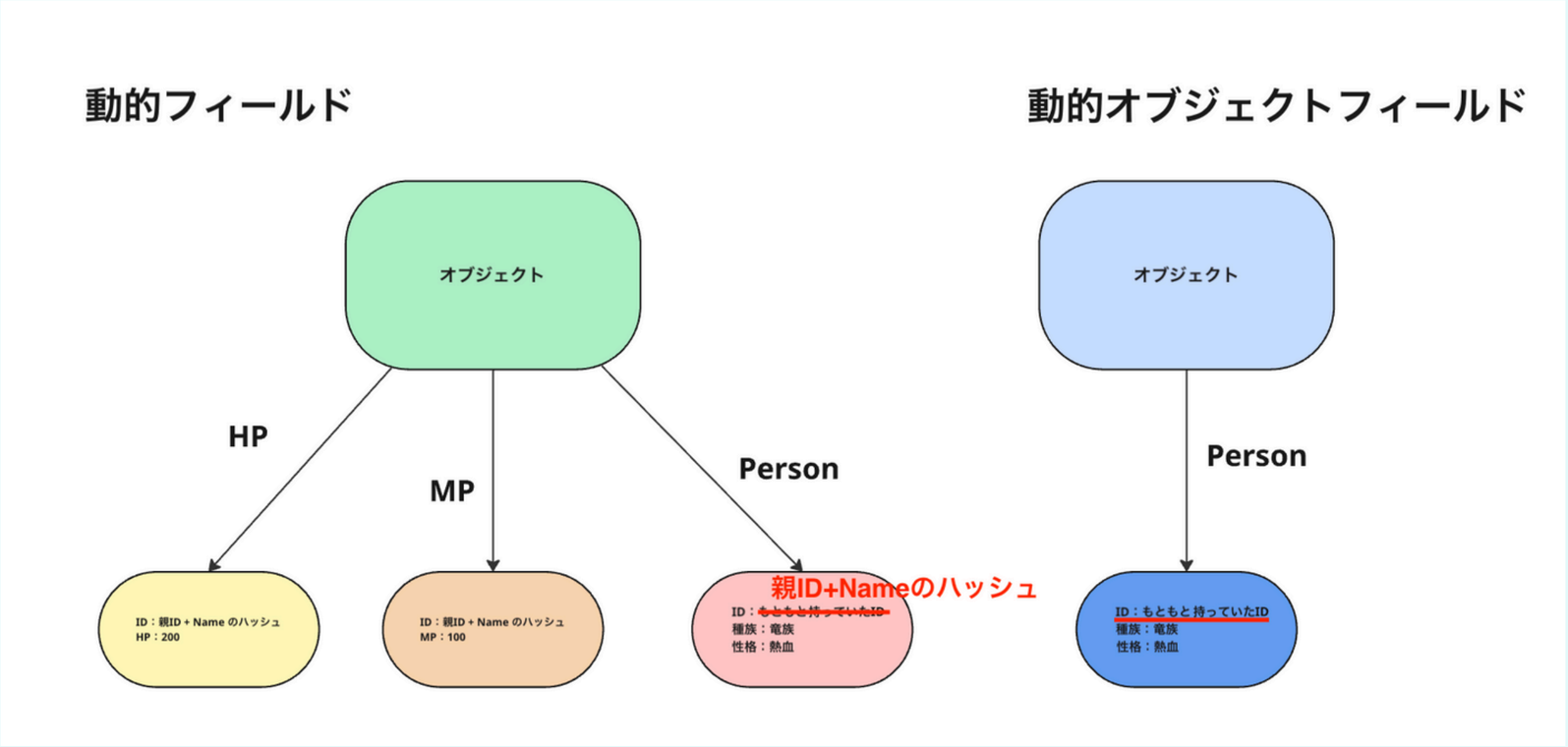
ちなみに、Etehreumは２です。
これは「Lido Finance」と「Coinbase」が大きなシェアを持つためです。



https://chainspect.app/dashboard/decentralization

4 動的（オブジェクト）フィールド

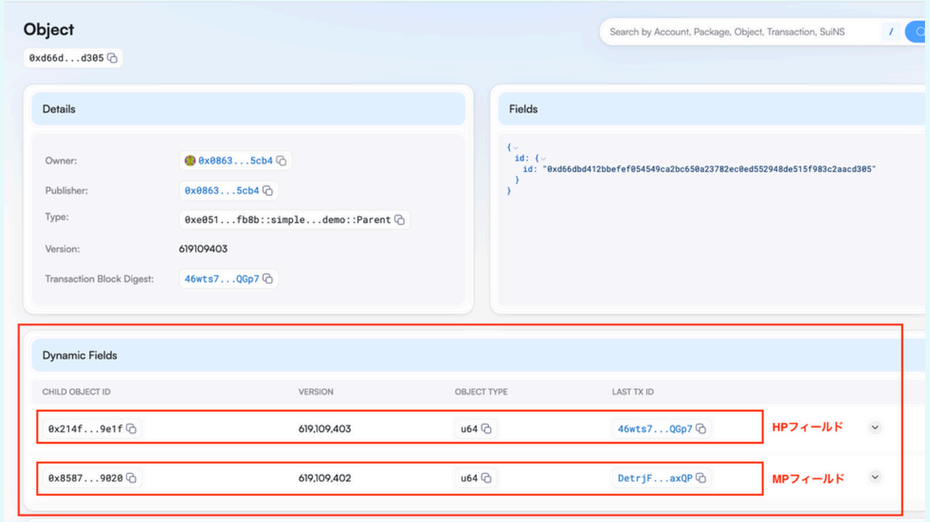
Suiではオブジェクトに動的にフィールドの設定ができます。
２種類がありますが、どちらもオブジェクトとして付属させます。



https://github.com/ytakahashi2020/dynamic_field/blob/main/my/sources/dynamic_field_demo.move

4 動的（オブジェクト）フィールド

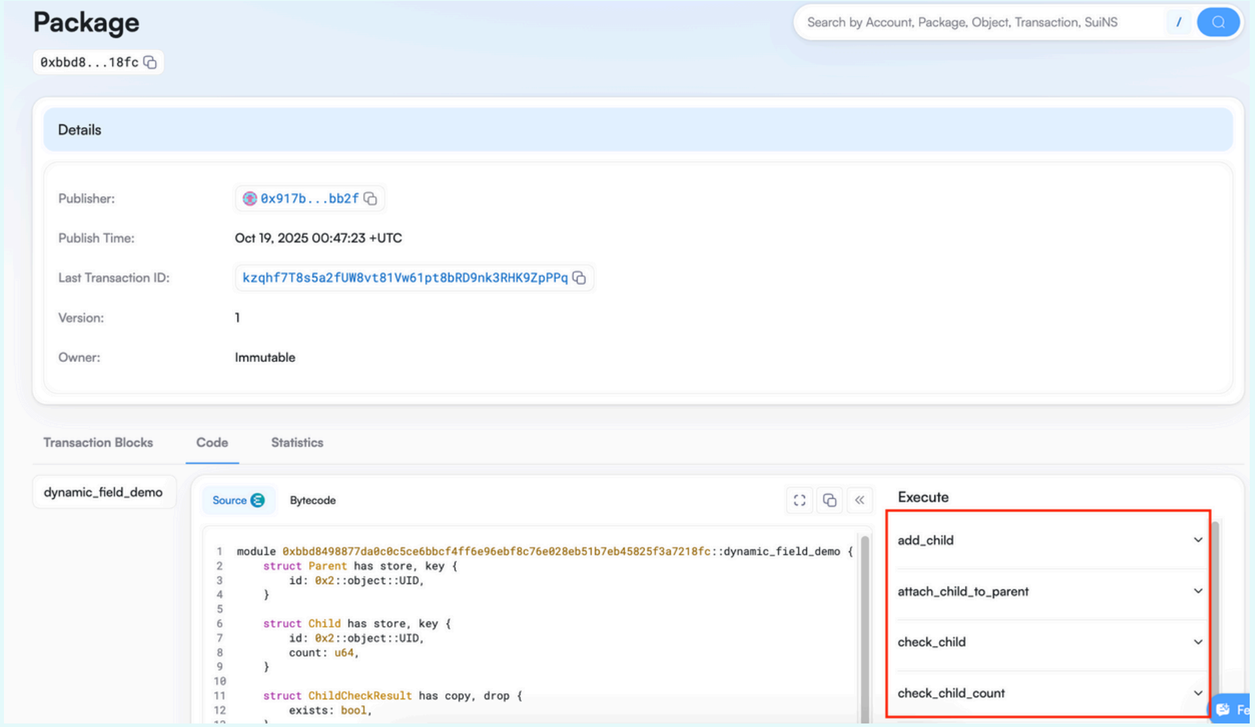
例えば、下のオブジェクトは「HP」,「MP」というフィールドを持ち、それぞれがオブジェクトです。



https://testnet.suivision.xyz/object/0xa8fc841714fef0a9dcbdca7ba7c98a678359180a6984bfcc4a552913d2d93e0f

４ー１ 動的フィールド

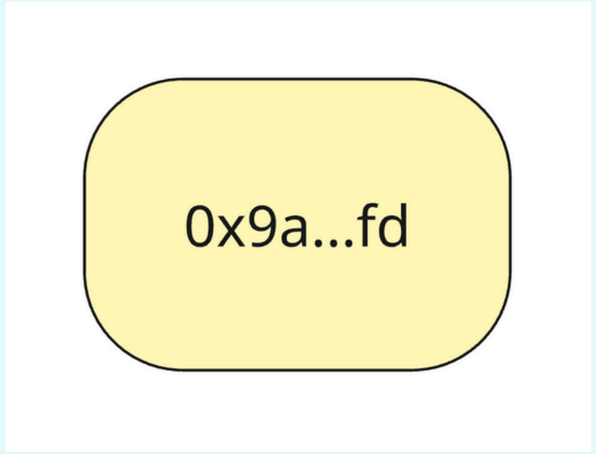
なお、動的フィールドの場合、新しくIDが生成されます。
そのため、すでにあるオブジェクトを子にした場合、元のIDからアクセスできず、親を通じてのみのアクセスとなります。

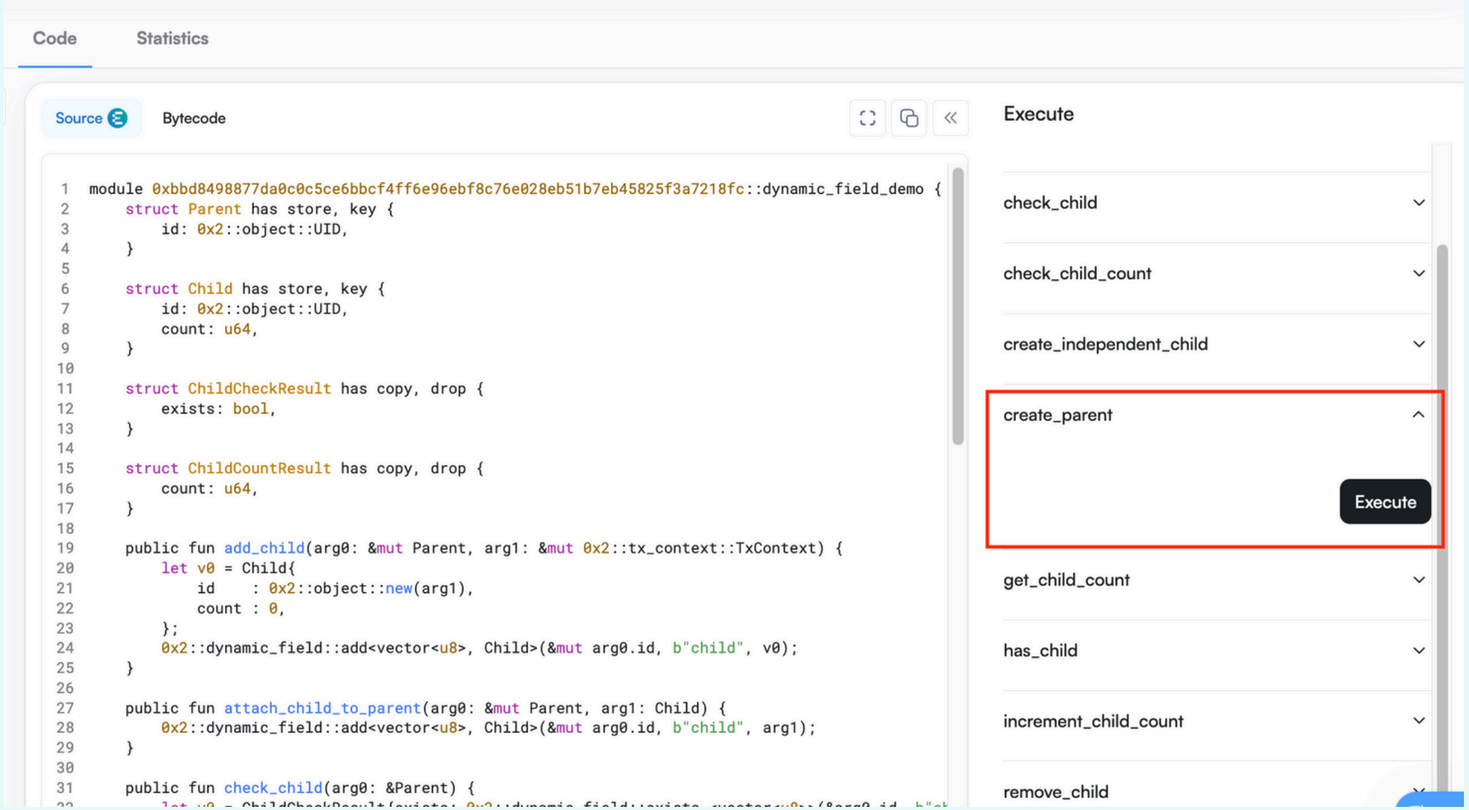


https://testnet.suivision.xyz/package/0xbbd8498877da0c0c5ce6bbcf4ff6e96ebf8c76e028eb51b7eb45825f3a7218fc?tab=Code

４ー１ 動的フィールド

まずは親のオブジェクトを作成します。



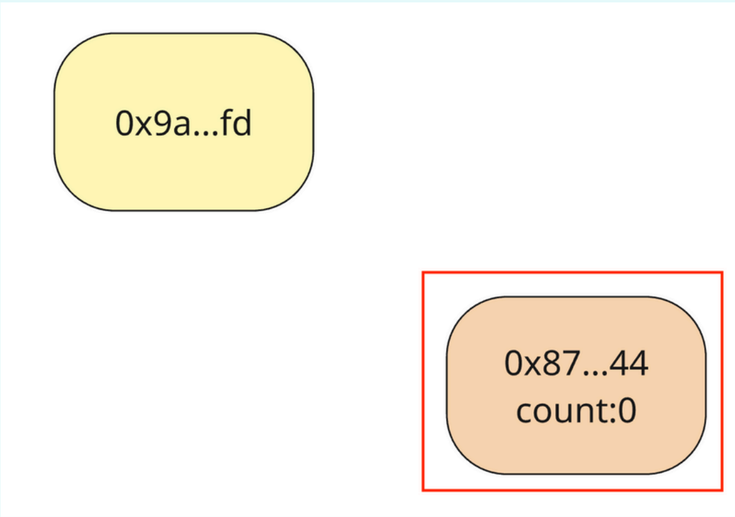


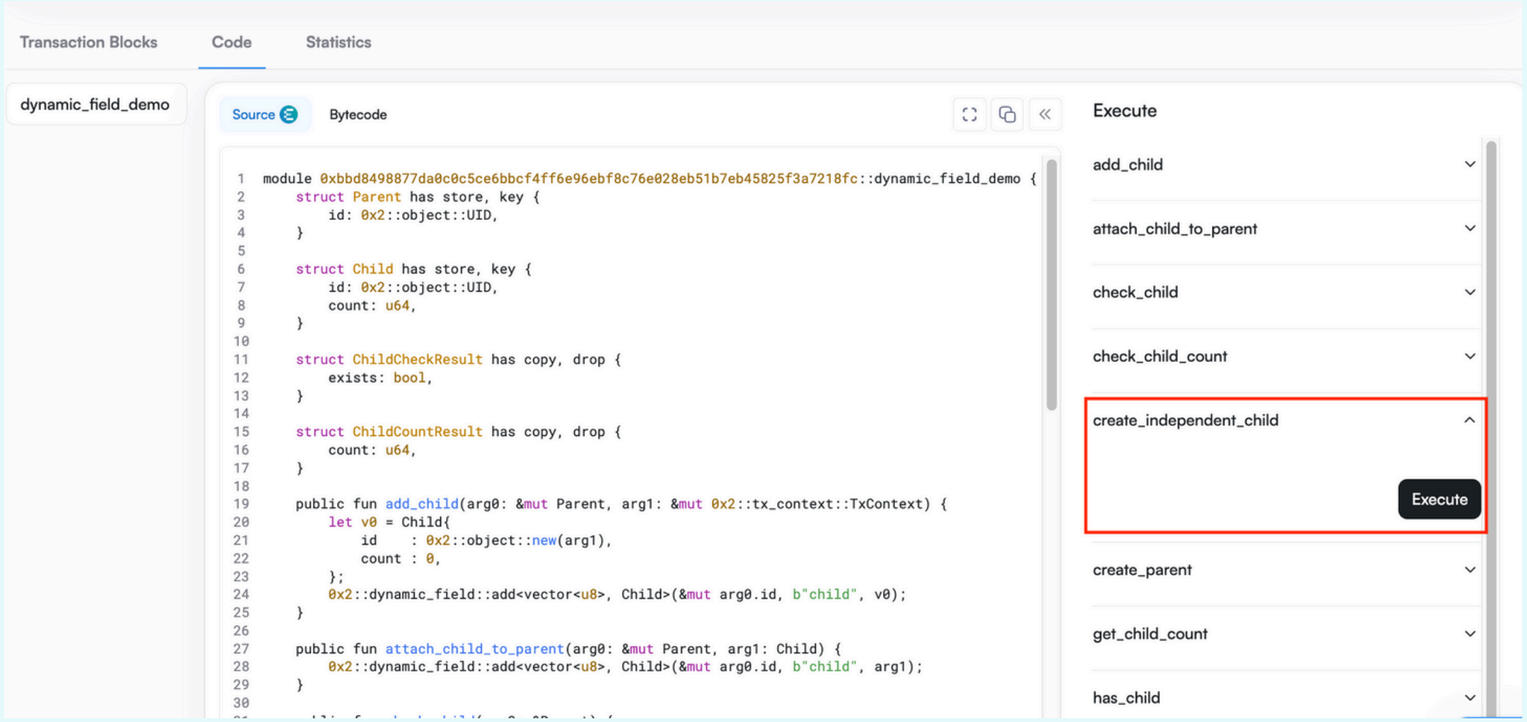

```
module 0xbbd8498877da0c0c5ce6bbcf4ff6e96ebf8c76e028eb51b7eb45825f3a7218fc::dynamic_field_demo {
    struct Parent has store, key {
        id: 0x2::object::UID,
    }

    struct Child has store, key {
        id: 0x2::object::UID,
        count: u64,
    }

    struct ChildCheckResult has copy, drop {
        exists: bool,
    }

    struct ChildCountResult has copy, drop {
        count: u64,
    }

    public fun add_child(arg0: &mut Parent, arg1: &mut 0x2::tx_context::TxContext) {
        let v0 = Child{
            id    : 0x2::object::new(arg1),
            count : 0,
        };
        0x2::dynamic_field::add<vector<u8>, Child>(&mut arg0.id, b"child", v0);
    }

    public fun attach_child_to_parent(arg0: &mut Parent, arg1: Child) {
        0x2::dynamic_field::add<vector<u8>, Child>(&mut arg0.id, b"child", arg1);
    }

    public fun check_child(arg0: &Parent) {
```

https://testnet.suivision.xyz/package/0xbbd8498877da0c0c5ce6bbcf4ff6e96ebf8c76e028eb51b7eb45825f3a7218fc?tab=Code

４ー１ 動的フィールド

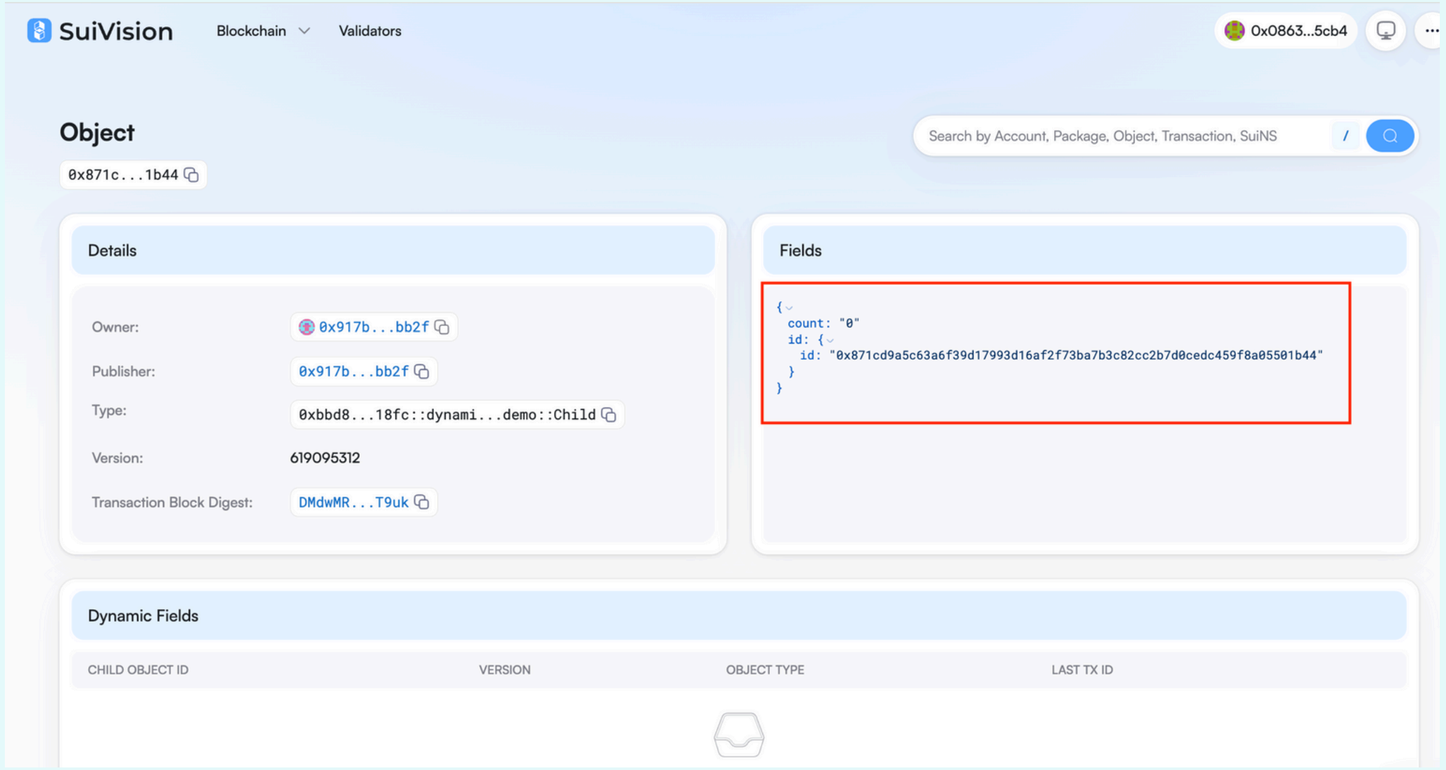
次に別のオブジェクトを作ります。この時点では関係性はありません。





４ー１ 動的フィールド

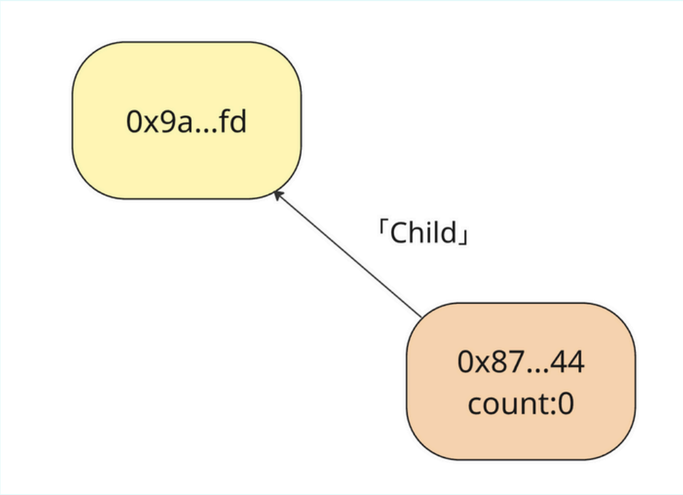
作成したオブジェクトにアクセスできることを確認します。

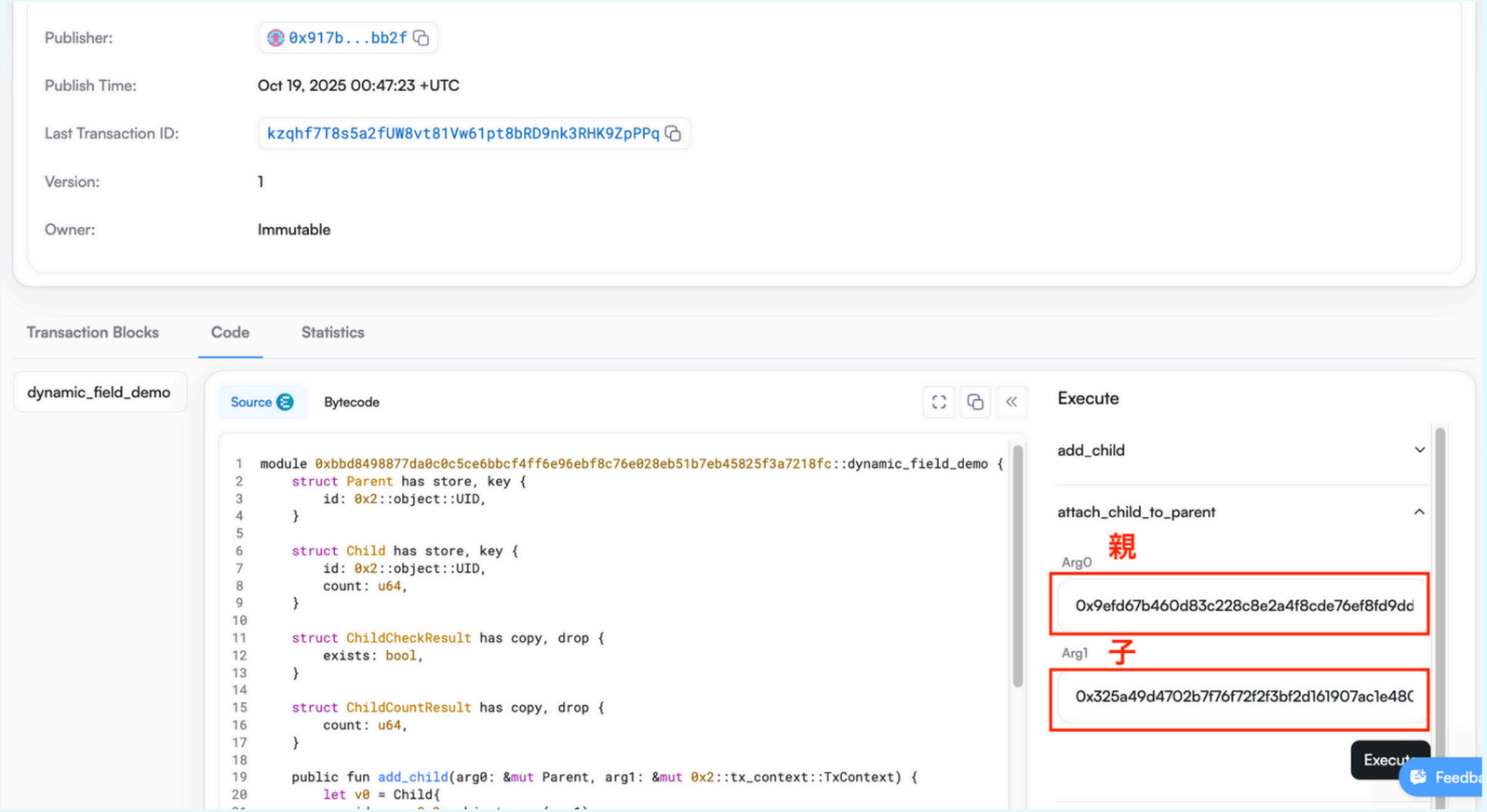


https://testnet.suivision.xyz/object/0x871cd9a5c63a6f39d17993d16af2f73ba7b3c82cc2b7d0cedc459f8a05501b44

４ー１ 動的フィールド

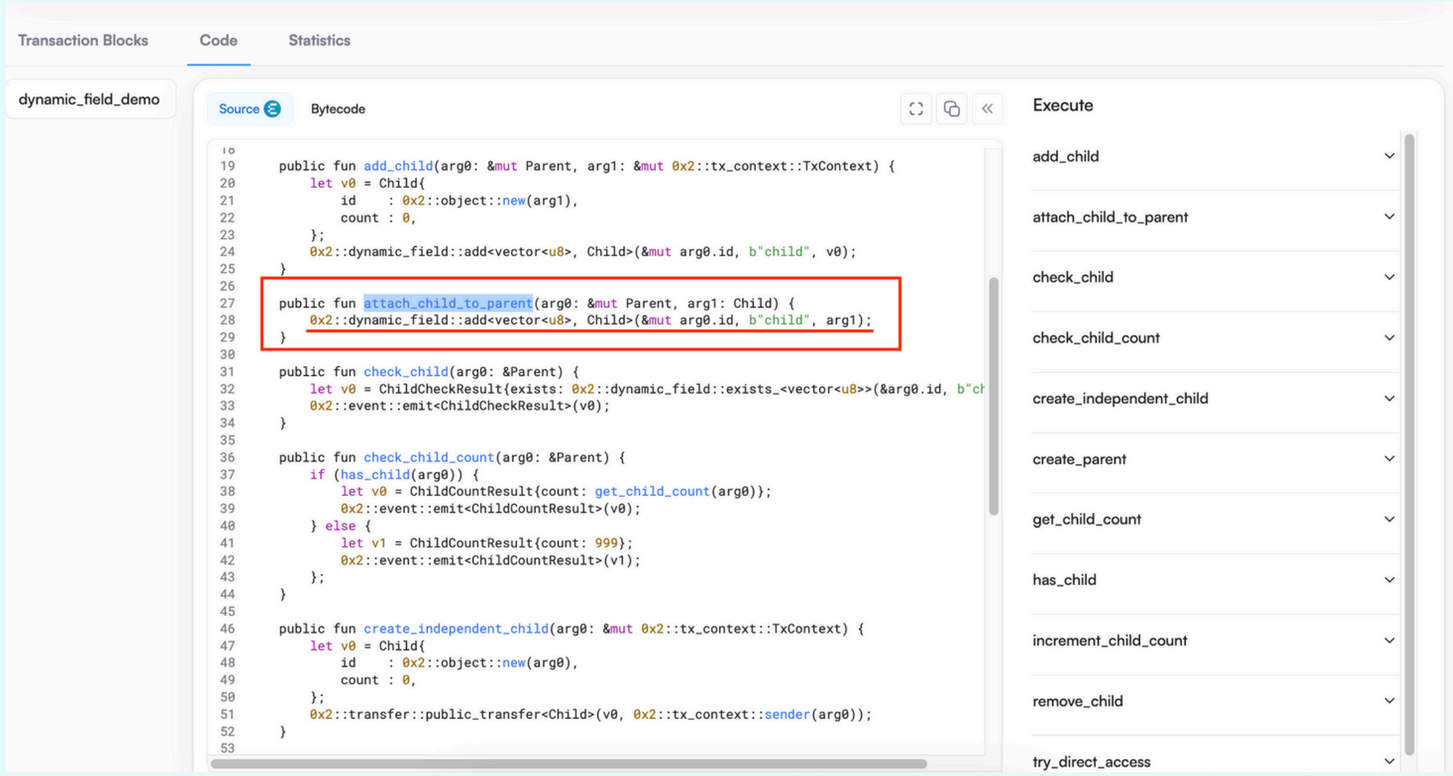
では子オブジェクトを「child」という名前で紐づけましょう。





４ー１ 動的フィールド

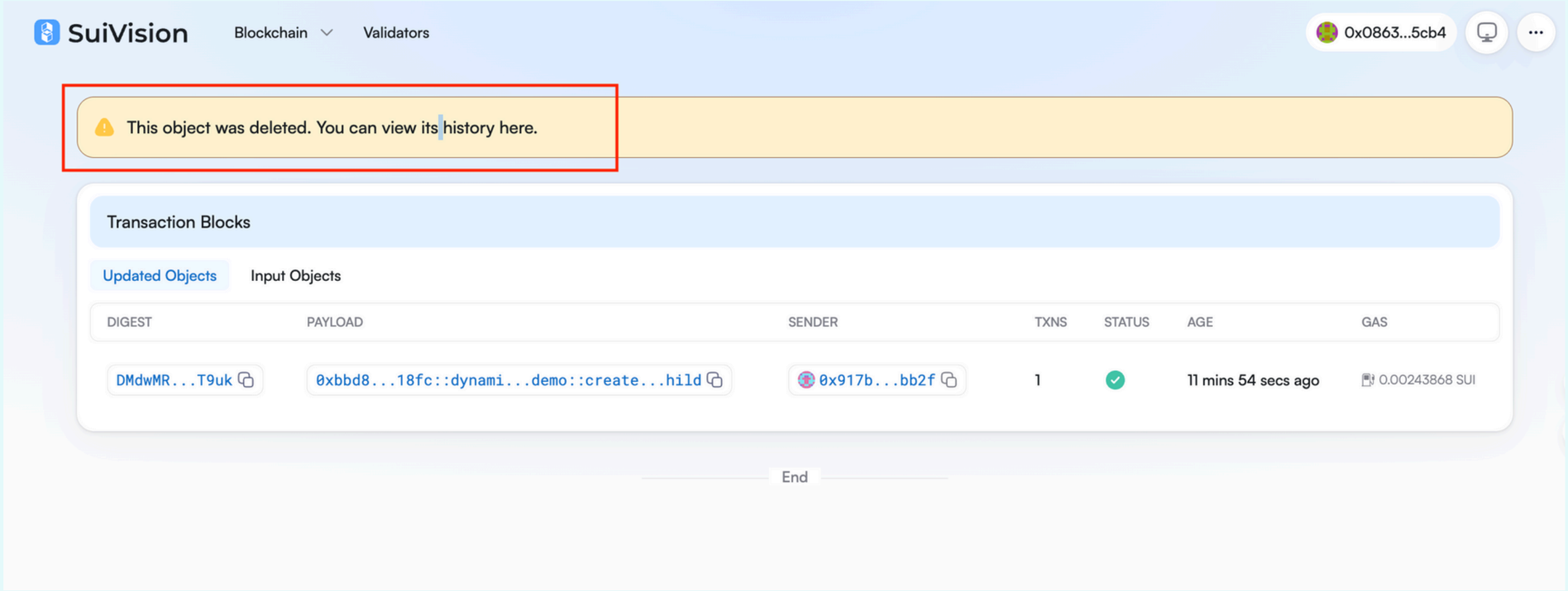
動的フィールドを作成し、子のオブジェクトが直接参照できないことを確認しましょう。



https://testnet.suivision.xyz/package/0xbbd8498877da0c0c5ce6bbcf4ff6e96ebf8c76e028eb51b7eb45825f3a7218fc?tab=Code

４ー１ 動的フィールド

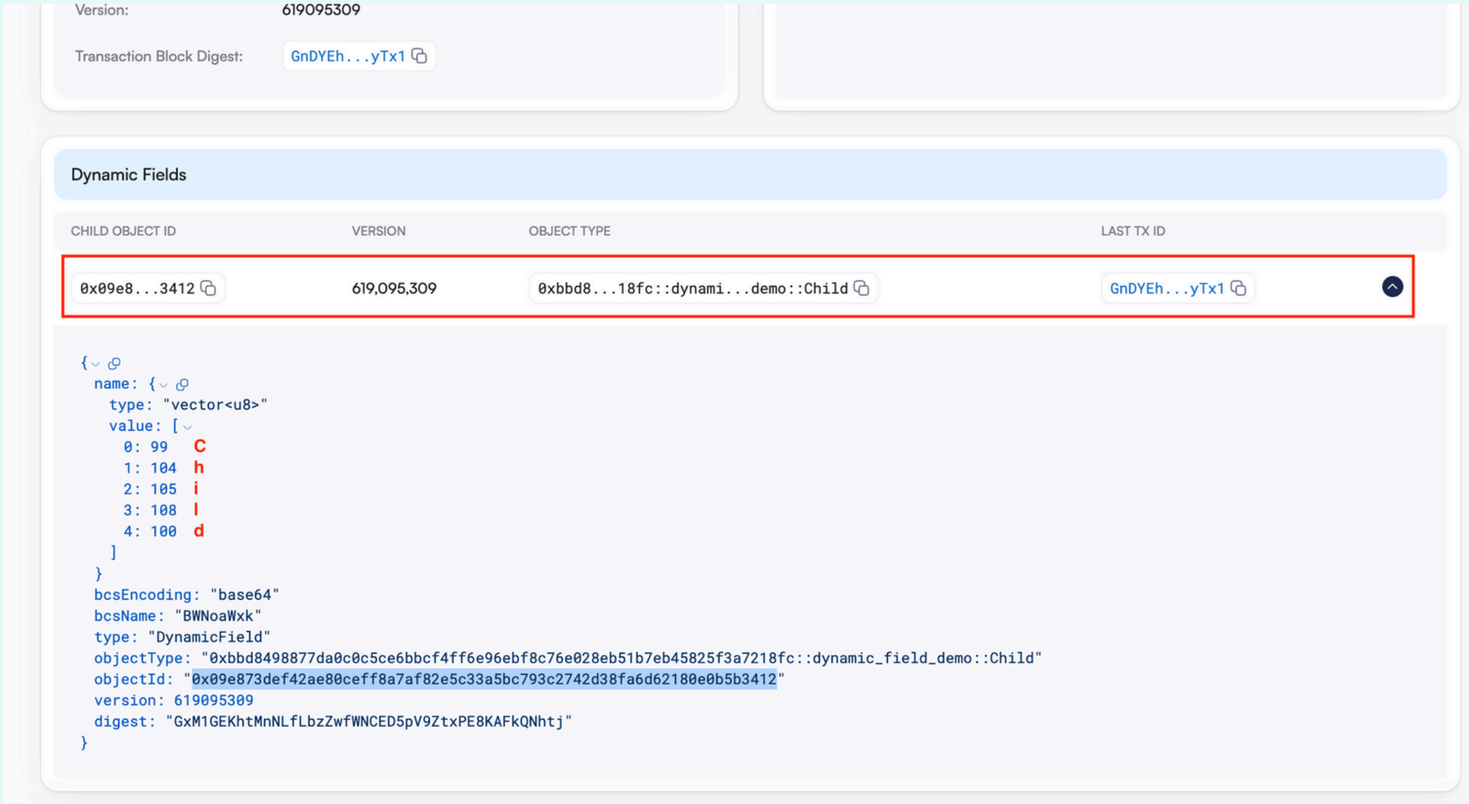
子オブジェクトにアクセスしようとすると、できないことがわかります。



https://testnet.suivision.xyz/object/0x871cd9a5c63a6f39d17993d16af2f73ba7b3c82cc2b7d0cedc459f8a05501b44

４ー１ 動的フィールド

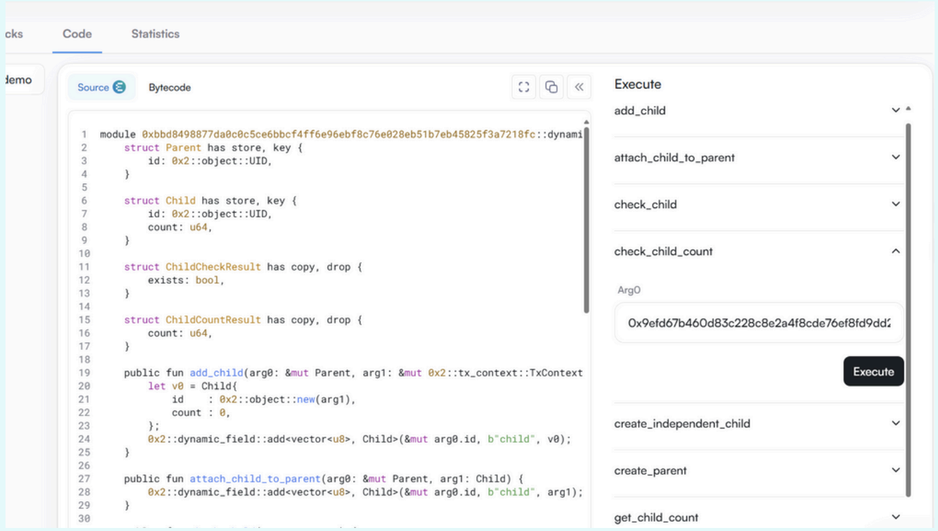
親オブジェクトを確認すると動的フィールドがあることが確認できます。

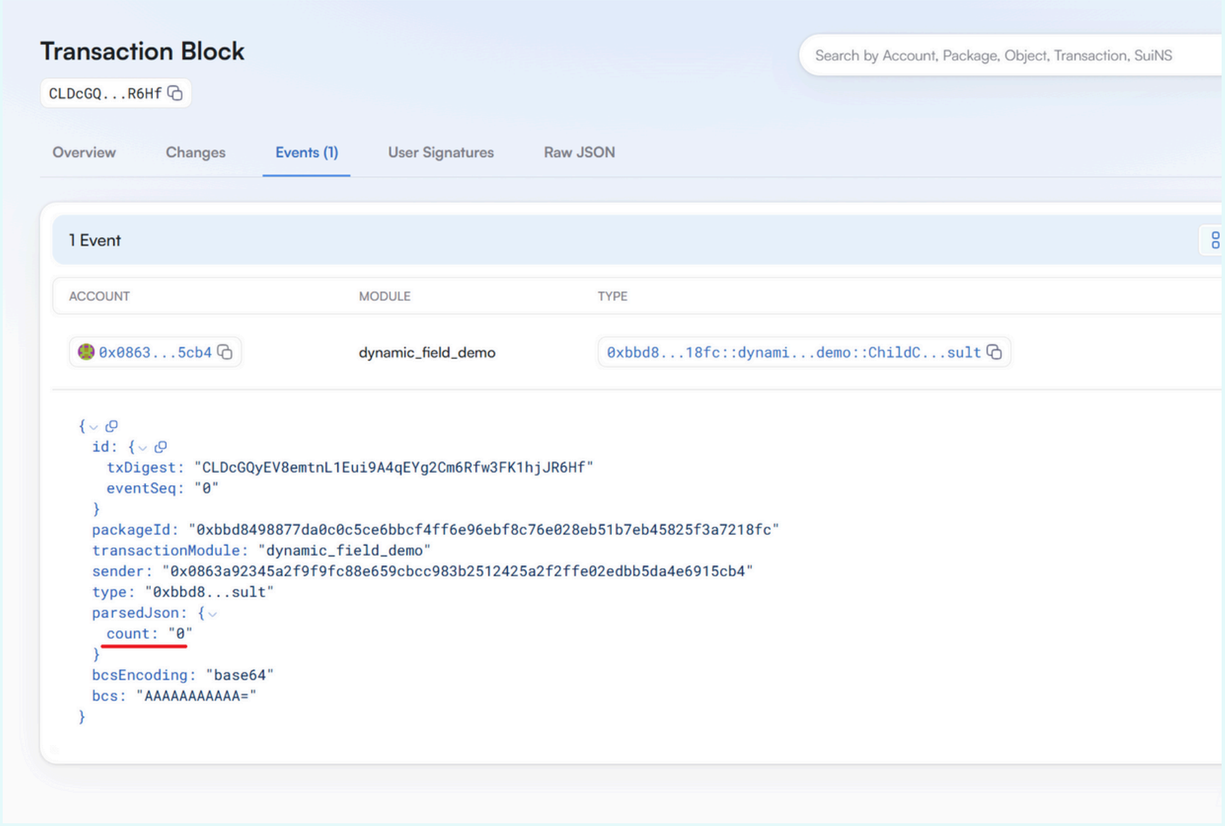


https://testnet.suivision.xyz/object/0x9aafb73cbde8227798ae87661a39f91d61035e88d77993cfd40f6fd6cbbf5dfd

４ー１ 動的フィールド

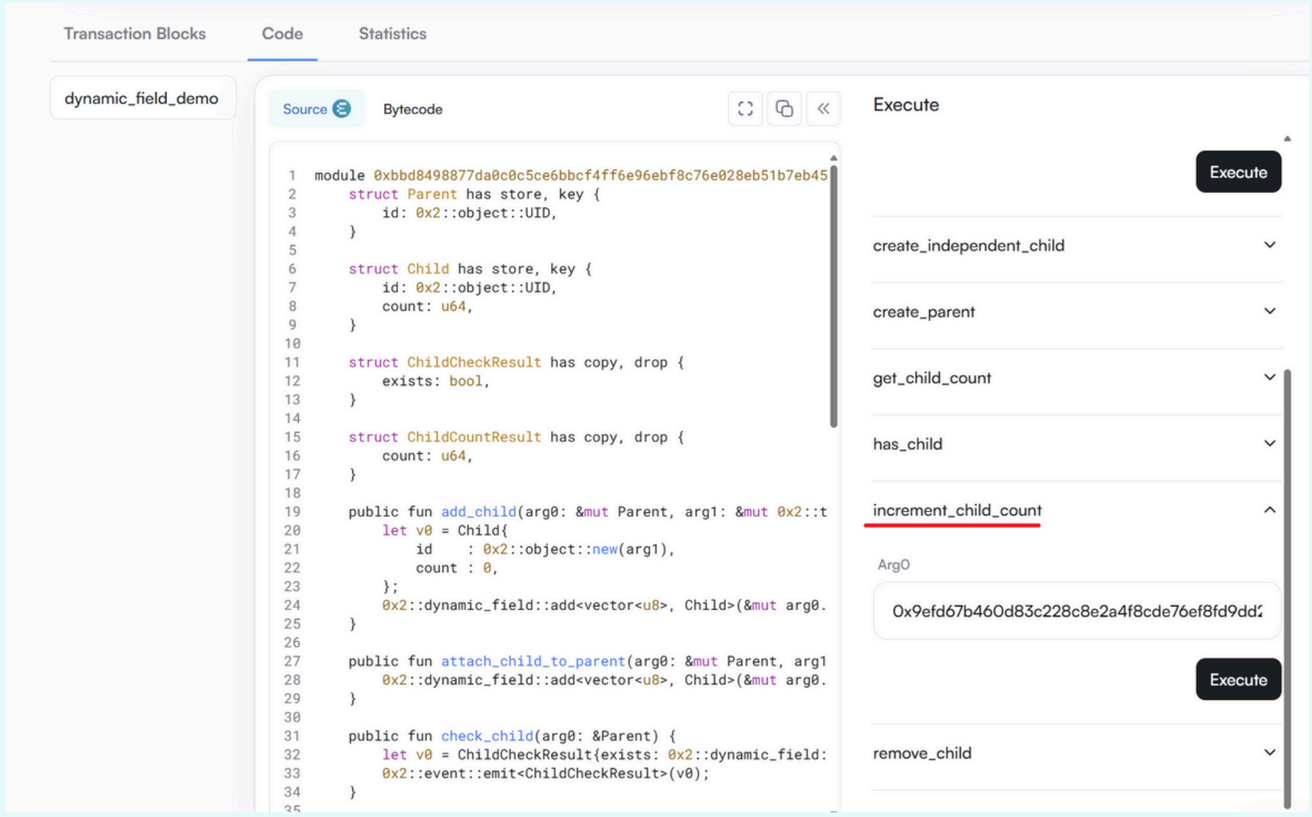
次に親オブジェクトから値を確認してみましょう。子IDはもはや使いません。





４ー１ 動的フィールド

子オブジェクトの値を操作してみましょう。＋１を行なっています。



４ー１ 動的フィールド

Deleting an object with dynamic fields
動的フィールド　を持つオブジェクトの削除

It is possible to delete an object that has (potentially non-`drop`) dynamic fields still defined on it. Because field values can be accessed only via the dynamic field's associated object and field name, deleting an object that has dynamic fields still defined on it renders them all inaccessible to future transactions. This is true regardless of whether the field's value has the `drop` ability. This might not be a concern when adding a small number of statically known additional fields to an object, but is particularly undesirable for on-chain collection types that could be holding unboundedly many key-value pairs as dynamic fields.

動的フィールドがまだ定義されている (`ドロップ`されない可能性がある) オブジェクトを削除することができます。フィールド値には、動的フィールドに関連付けられたオブジェクトとフィールド名を介してのみアクセスできるため、動的フィールドがまだ定義されているオブジェクトを削除すると、今後のトランザクションですべてのオブジェクトにアクセスできなくなります。これは、フィールドの値に`ドロップ`能力があるかどうかに関係なく当てはまります。これは、静的に知られている少数の追加フィールドをオブジェクトに追加する場合には問題にならないかもしれませんが、動的フィールドとして無制限に多くのキーと値のペアを保持する可能性のあるオンチェーンコレクションタイプには特に望ましくありません。

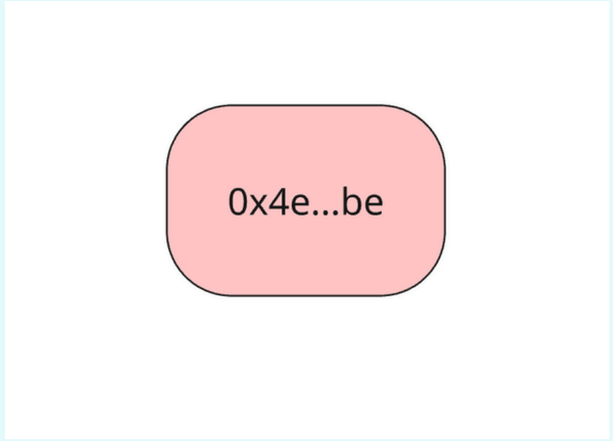
Sui provides `Table` and `Bag` collections built using dynamic fields, but with additional support to count the number of entries they contain to protect against accidental deletion when non-empty. To learn more, see Tables and Bags.

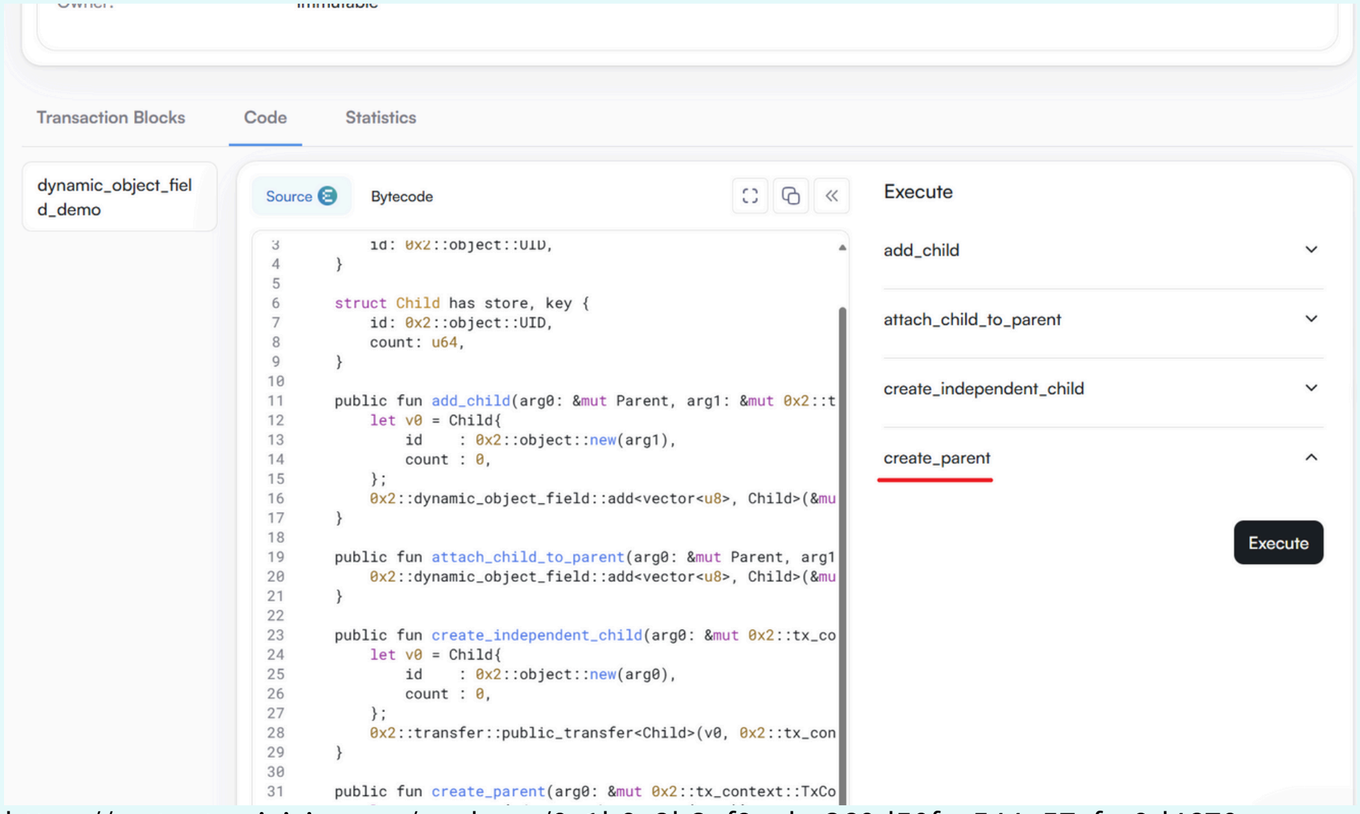
Sui は、動的フィールドを使用して構築された `Table` コレクションと `Bag` コレクションを提供しますが、空でない場合の誤った削除を防ぐために、含まれるエントリの数をカウントするための追加のサポートが備わっています。詳細については、「テーブルとバッグ」を参照してください。

https://docs.sui.io/concepts/dynamic-fields

4ー2 動的オブジェクトフィールド

まずは親のオブジェクトを作成します。

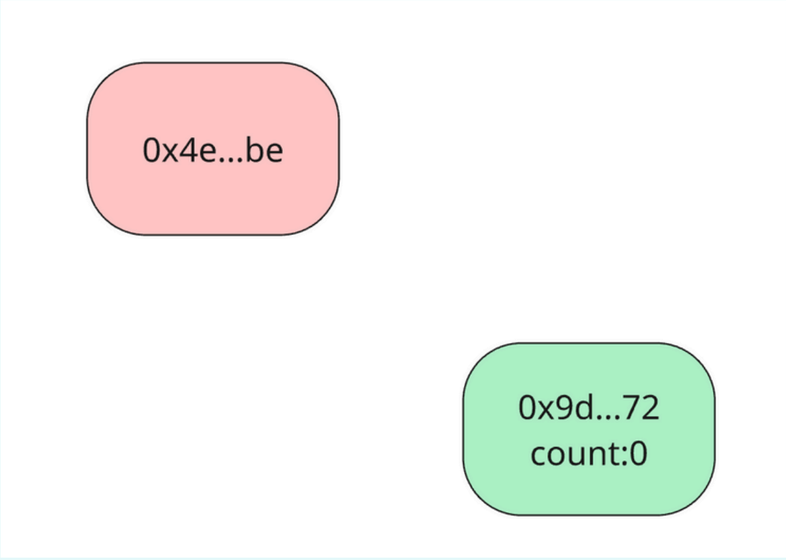


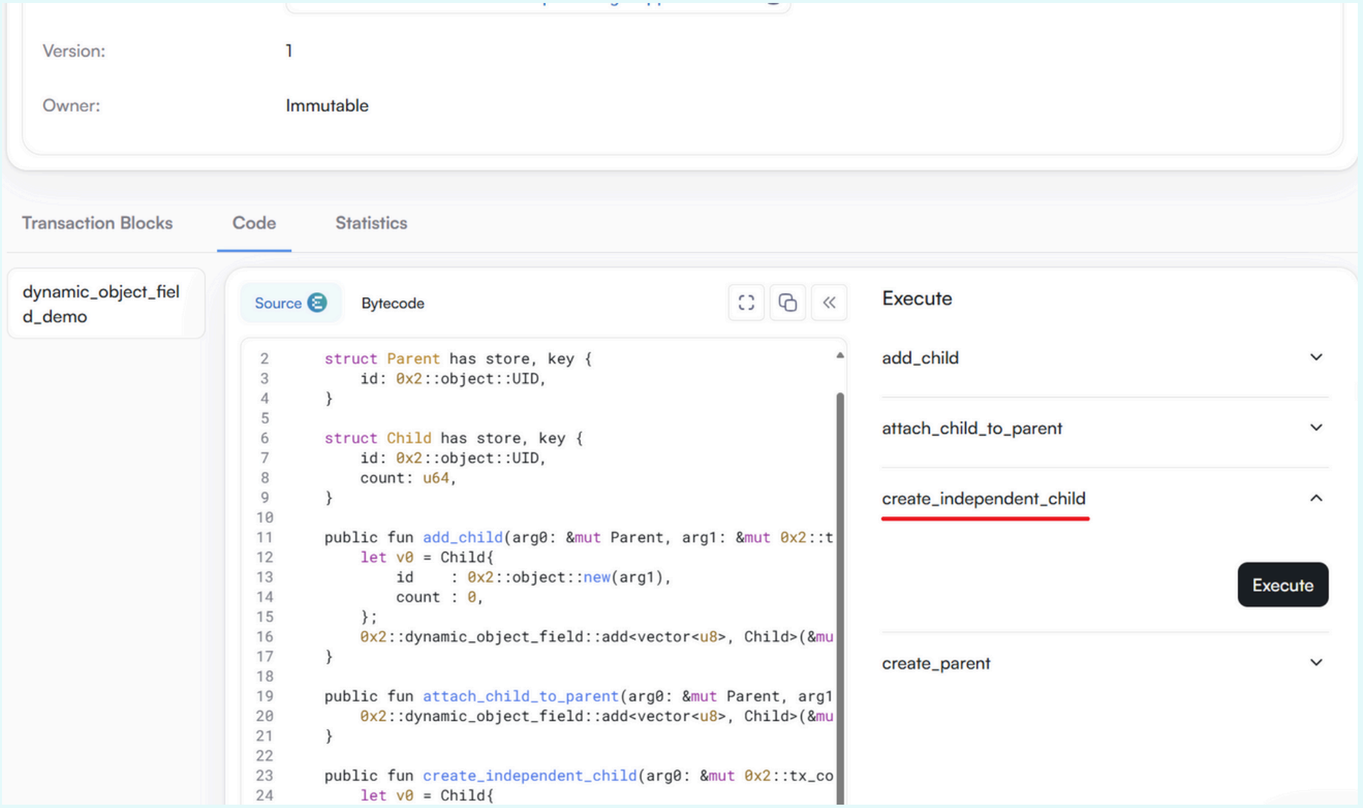


https://testnet.suivision.xyz/package/0x1b9c2b3af8aeba360d50fce544a57cfcc9d4670ad01ae7356a9ff9a9b975ba7a?tab=Code

4ー2 動的オブジェクトフィールド

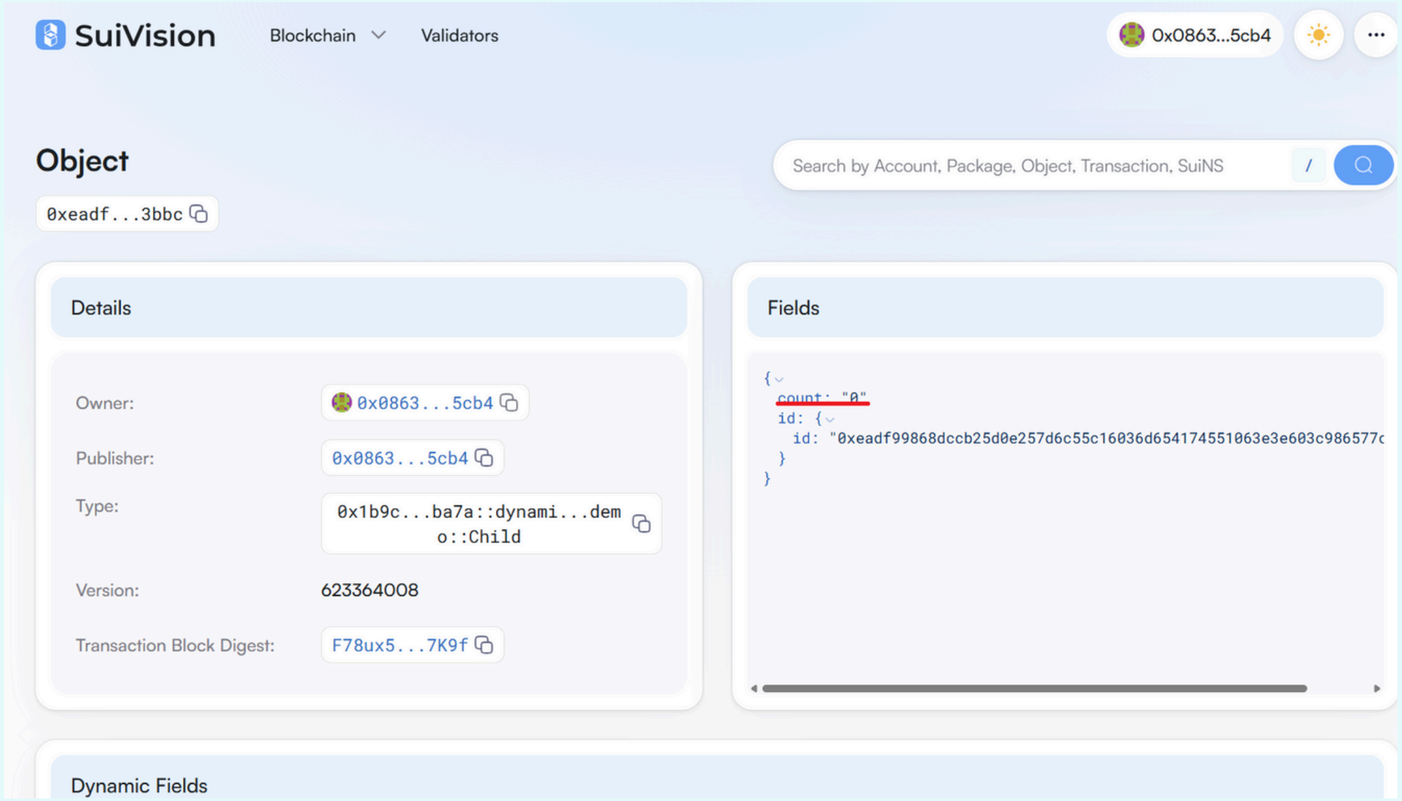
次に別のオブジェクトを作ります。この時点では関係性はありません。





4ー2 動的オブジェクトフィールド

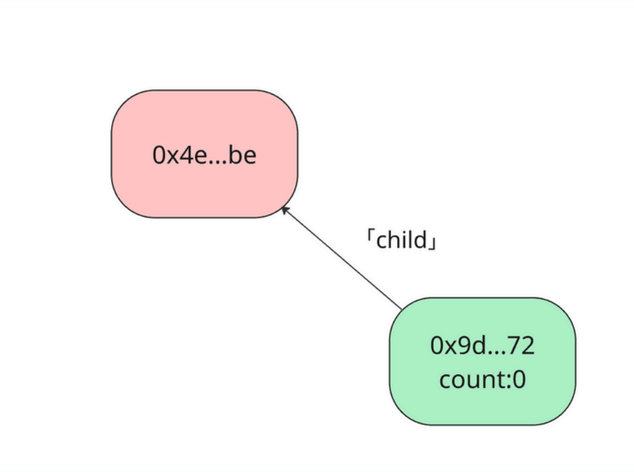
作成したオブジェクトにアクセスできることを確認します。

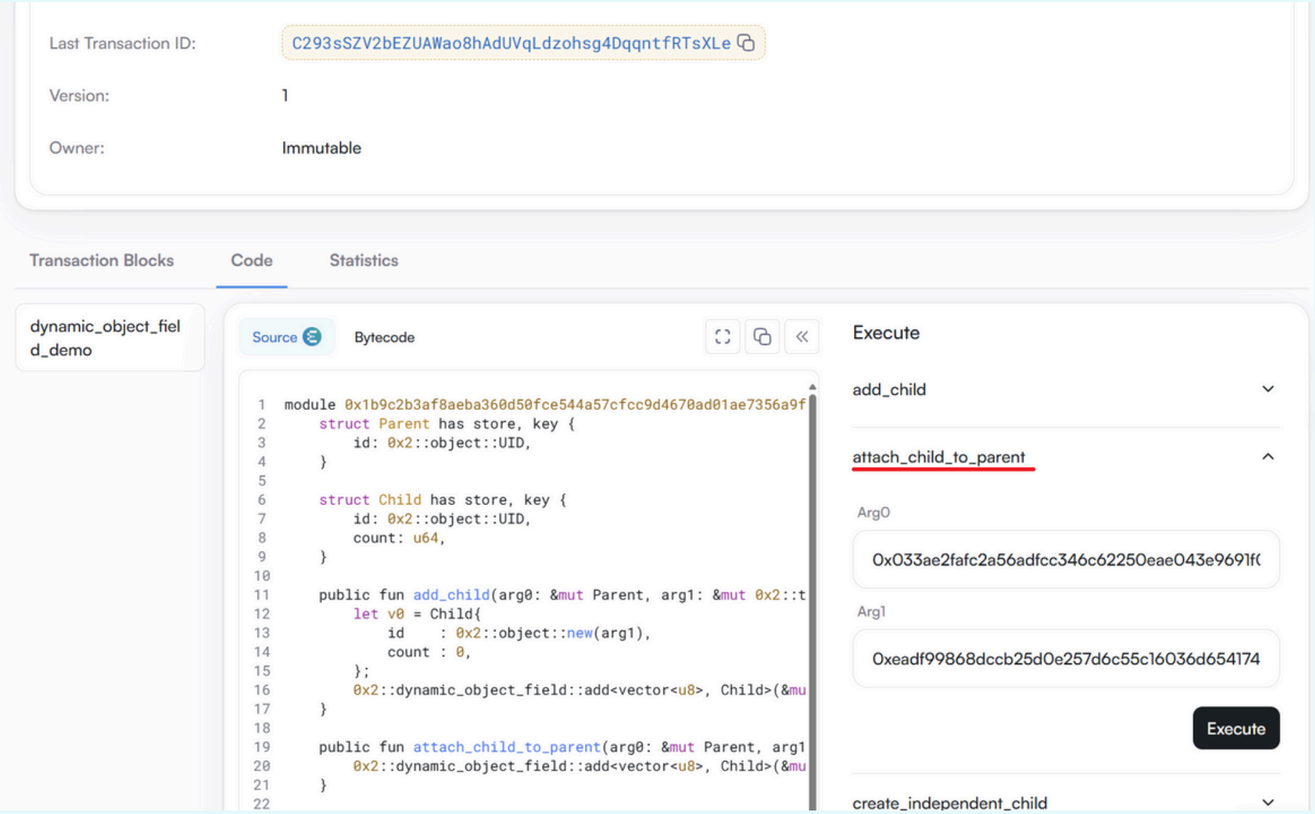


https://testnet.suivision.xyz/object/0xeadf99868dccb25d0e257d6c55c16036d654174551063e3e603c986577c03bbc

4ー2 動的オブジェクトフィールド

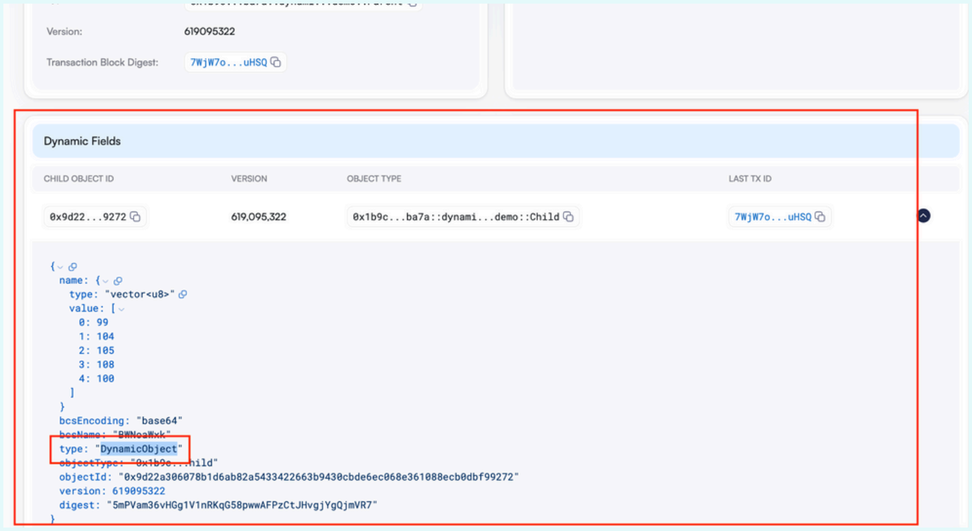
では子オブジェクトを「child」という名前で紐づけましょう。

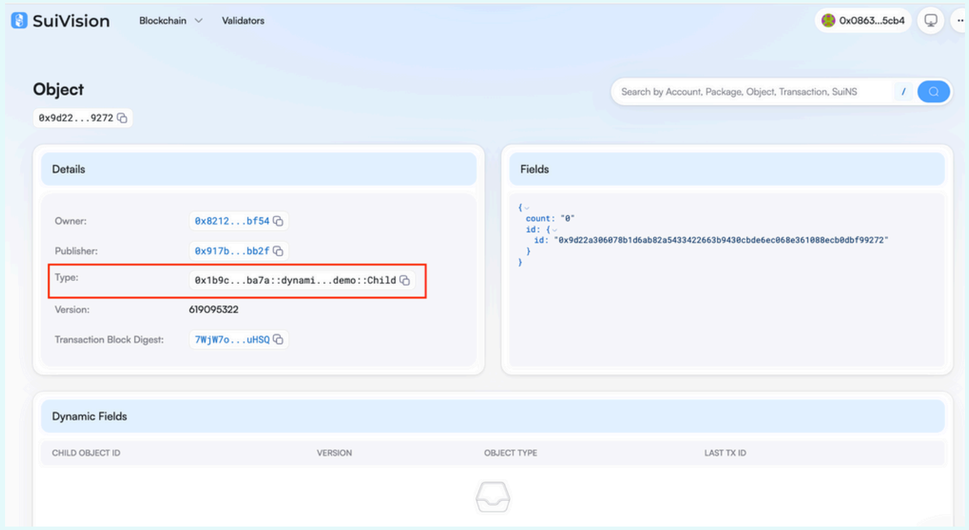




4ー2 動的オブジェクトフィールド

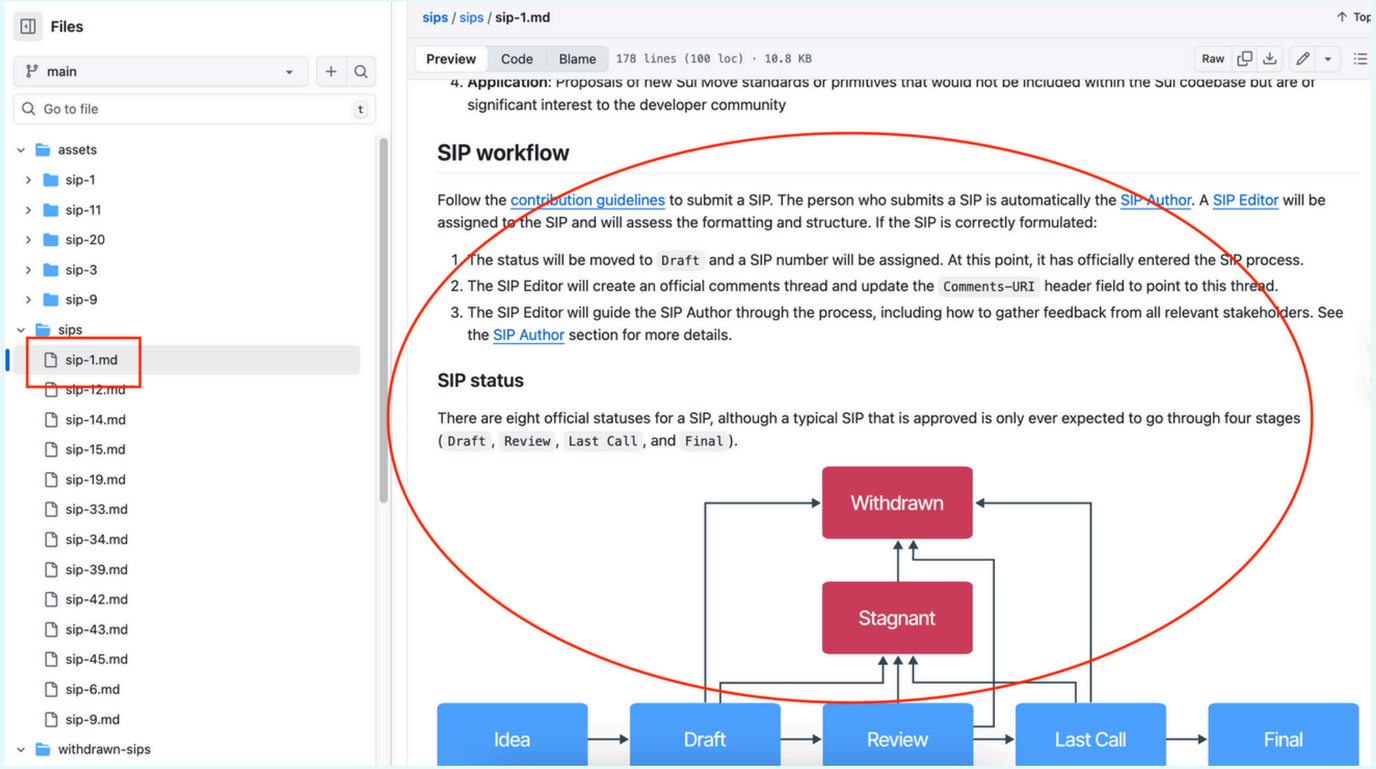
親オブジェクトからだけでなく、子オブジェクトからも確認できることがわかります。





5 SIPとは

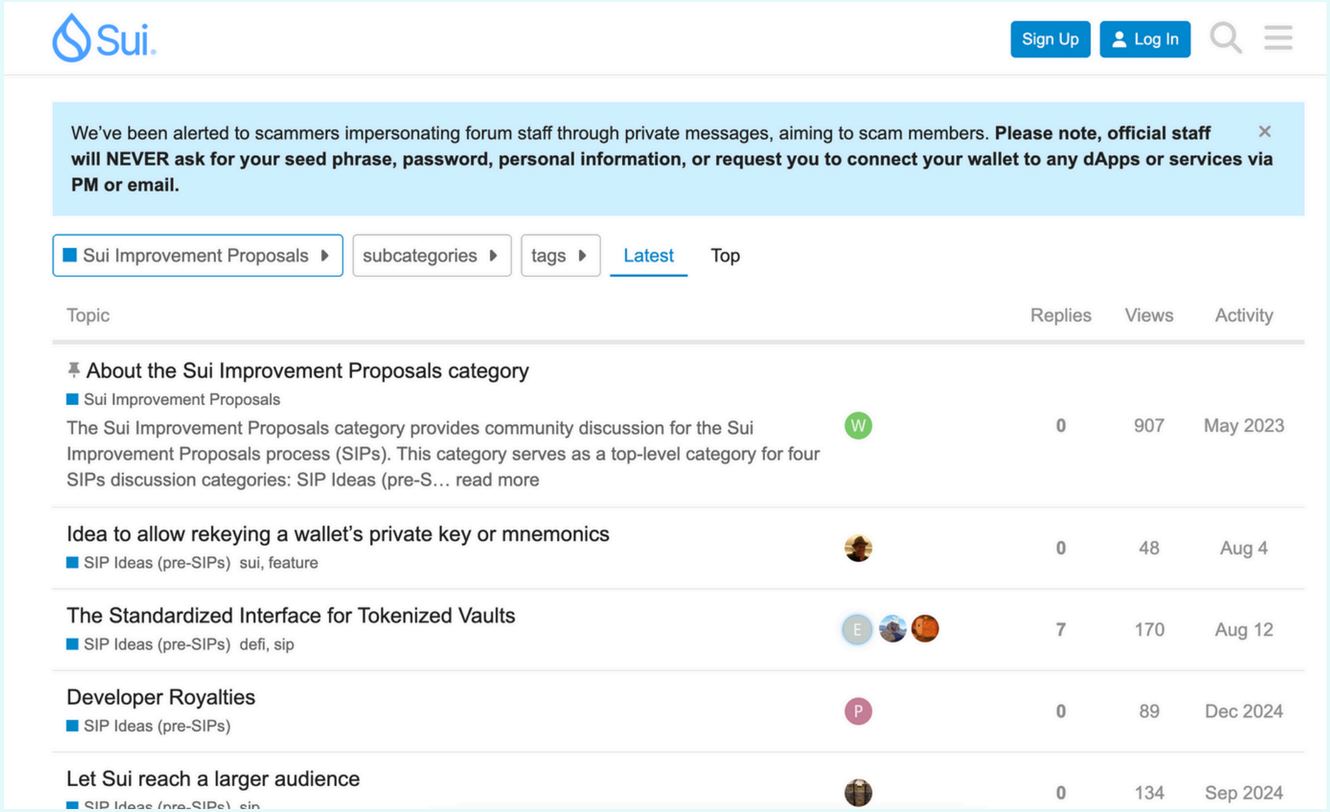
Sui Improvement Proposalの略で、Suiについての新しい提案を行うことができます。



https://github.com/sui-foundation/sips/blob/main/sips/sip-1.md

5 SIPとは

Sui Improvement Proposalの略で、Suiについての新しい提案を行うことができます。

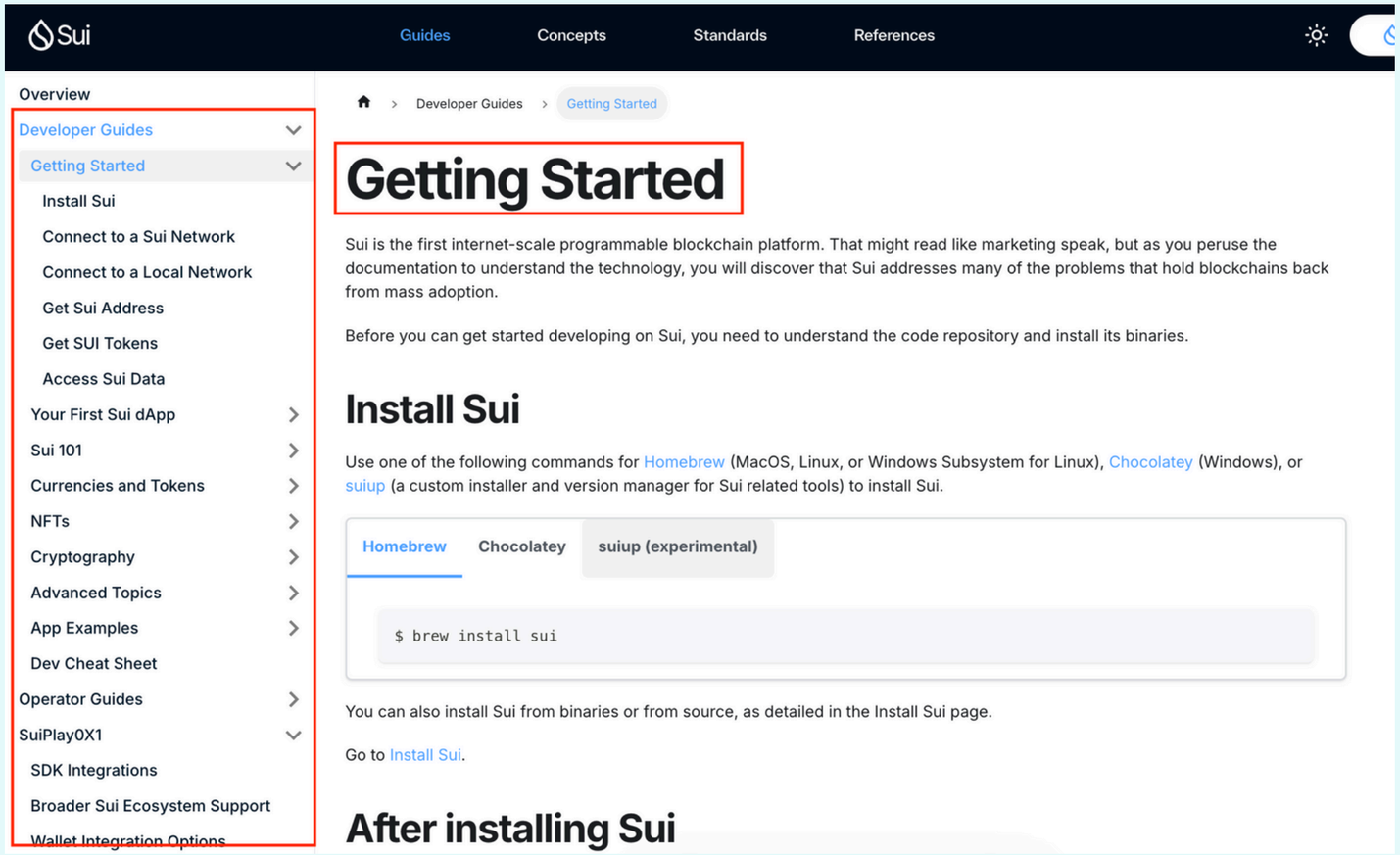


https://github.com/sui-foundation/sips/blob/main/sips/sip-1.md

開発について　①開発コミュニティ

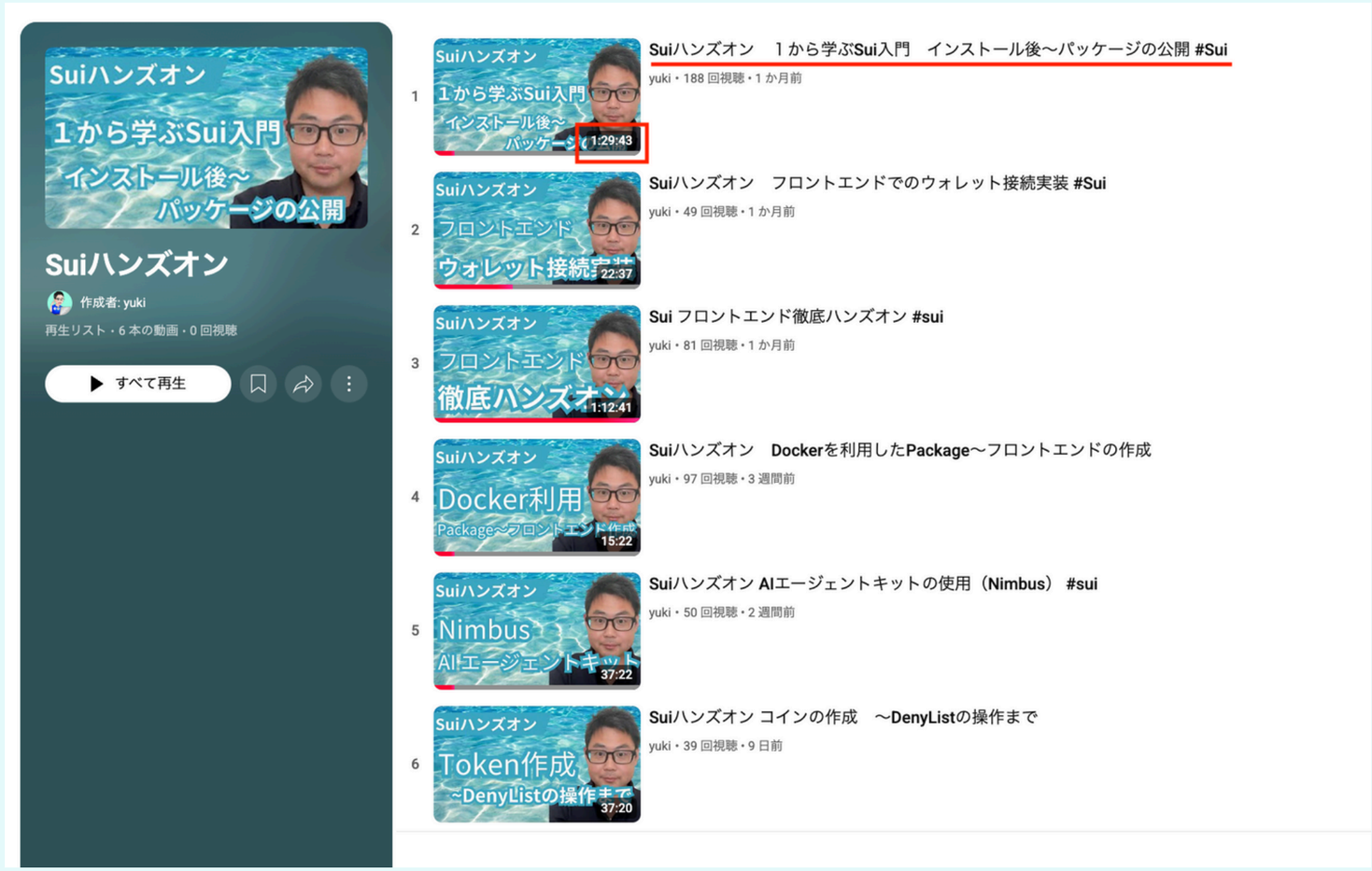
https://discord.com/invite/NM8jWvmQ6V

開発について　②公式ドキュメント



https://docs.sui.io/guides/developer/getting-started

開発について　③ハンズオン動画



https://www.youtube.com/playlist?list=PL1rRUVBEqXMnlkv4B5FZAllt76kwe6Q_7

免責事項

本資料は、ブロックチェーン技術の学習・情報共有を目的として作成したものであり、正確性や完全性を保証するものではありません。内容には誤りや不十分な点が含まれる可能性があります。

また、本資料は投資を勧誘・推奨する意図は一切なく、投資判断に利用することを目的としたものではありません。

実際に投資や取引を行う場合は、ご自身の責任と判断において行ってください。