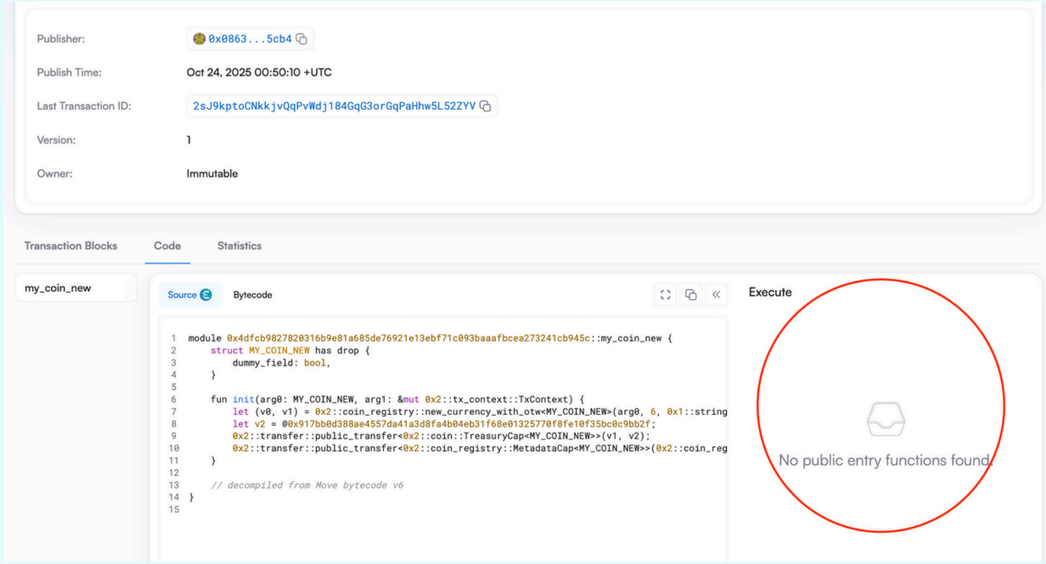
Build on Sui

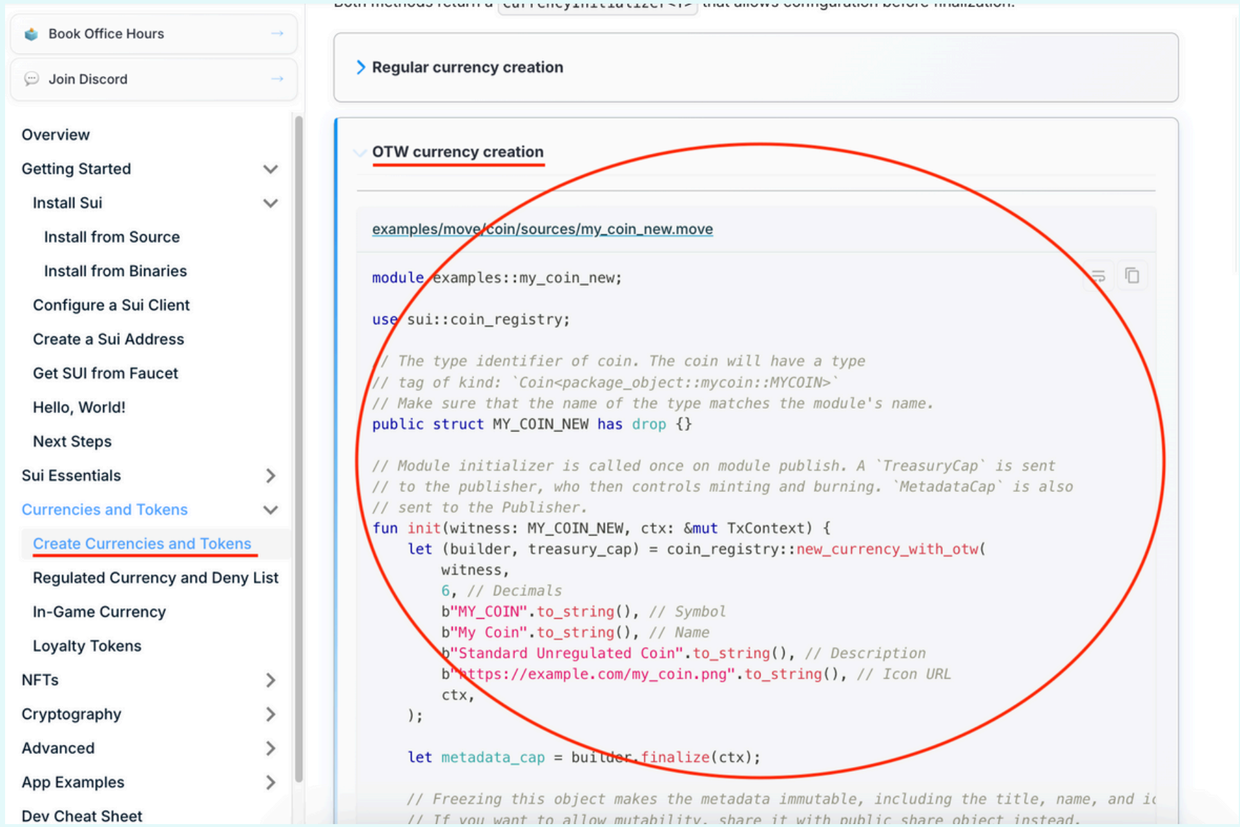
Weekend Move Workshop in Tokyo // Day 2

Powered by

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。

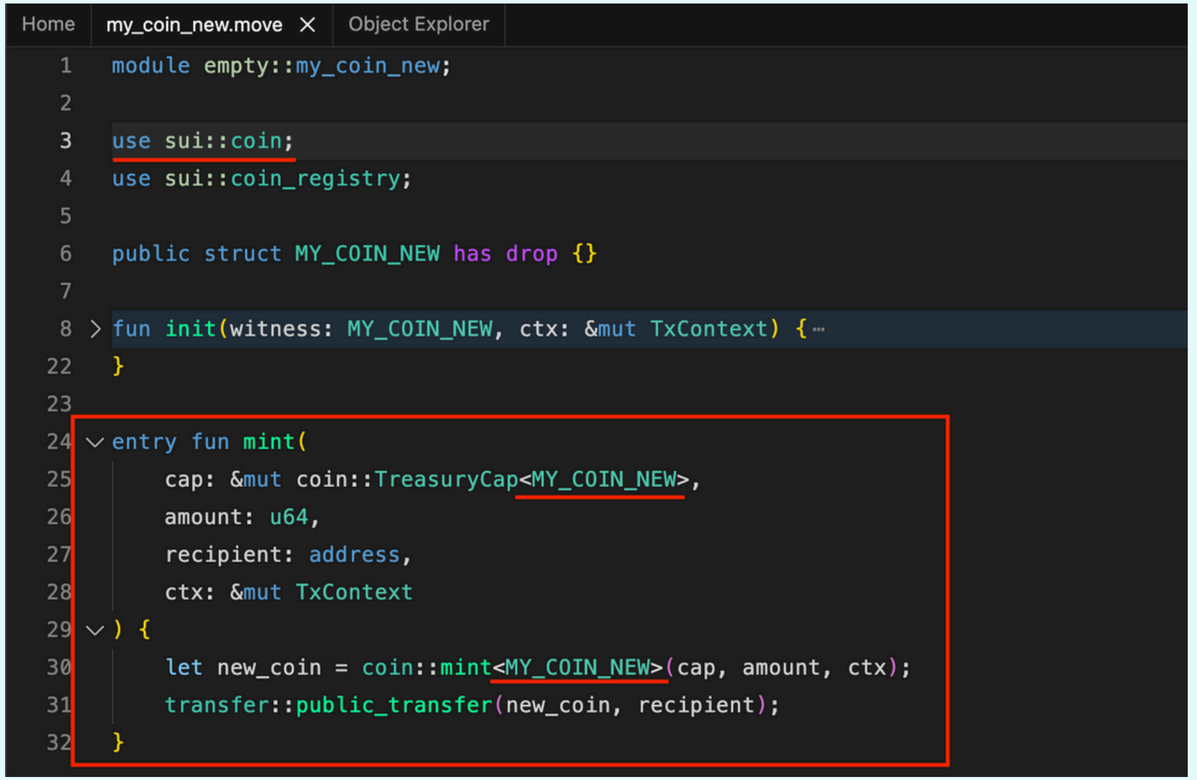




https://docs.sui.io/guides/developer/currency

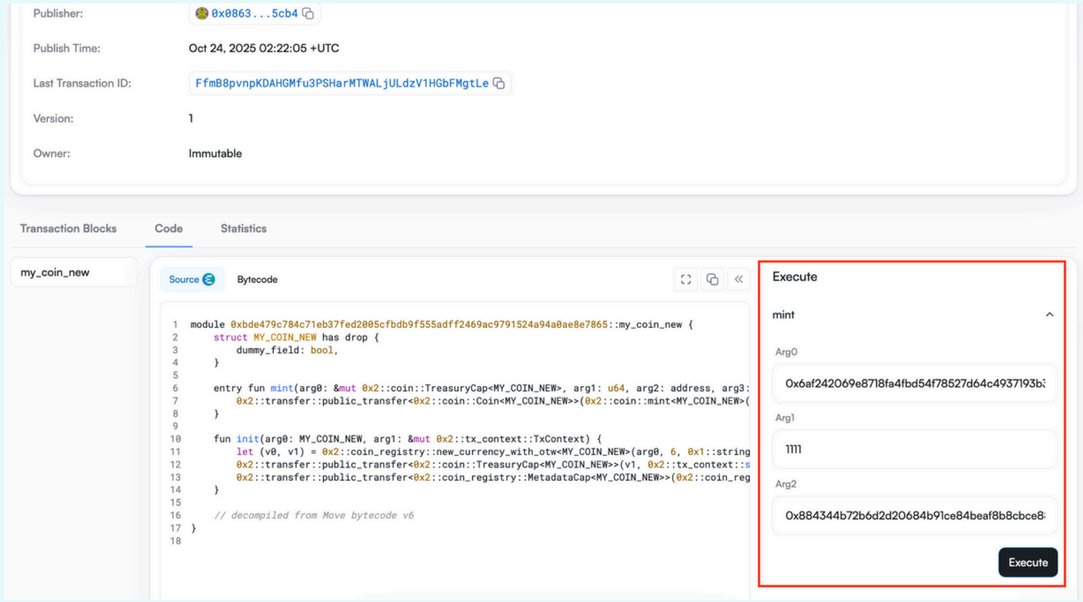
開発26　entry関数の作成

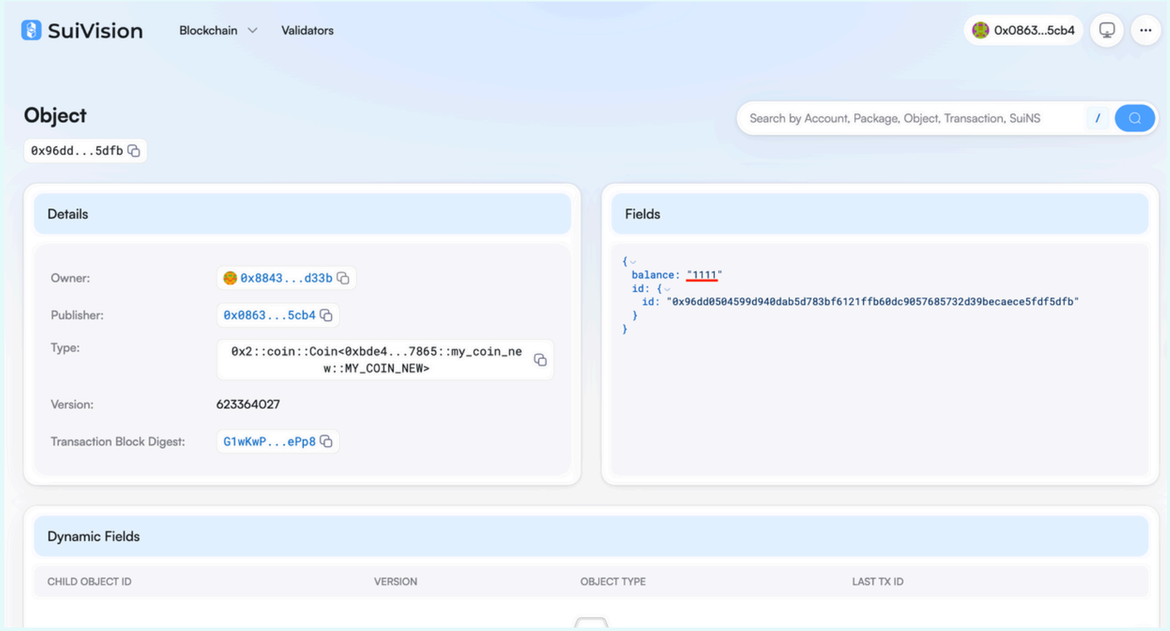
以下のようにmint関数を作ります。
<MY_COIN_NEW>で型を設定しています。



開発27　SuiVisionでの実行

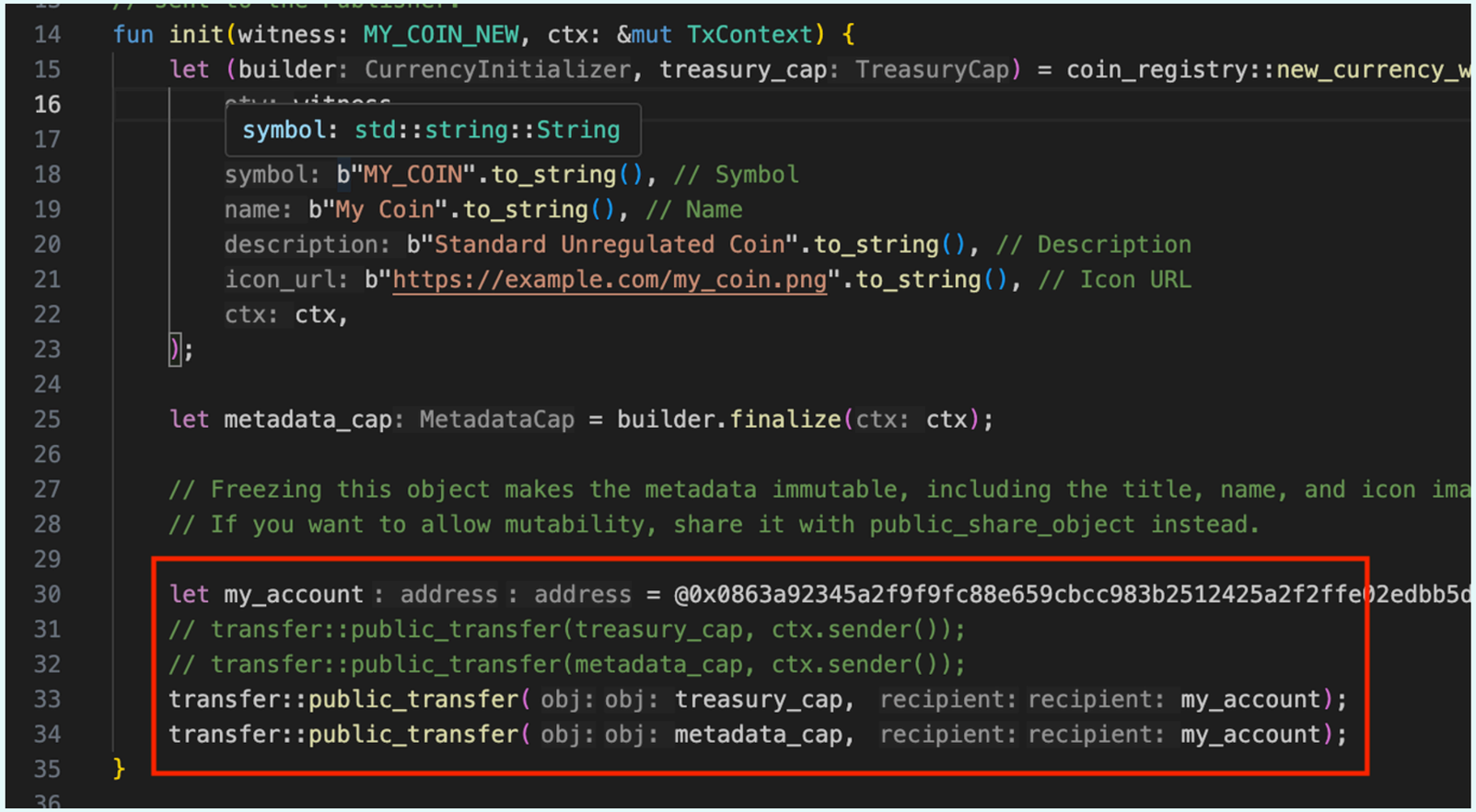
SuiVisionで実行ができることを確認しましょう。
あれ、できない？





開発27　SuiVisionでの実行

実行者が異なっていたのですね。



https://github.com/ytakahashi2020/my_coin/blob/main/sources/my_coin.move

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

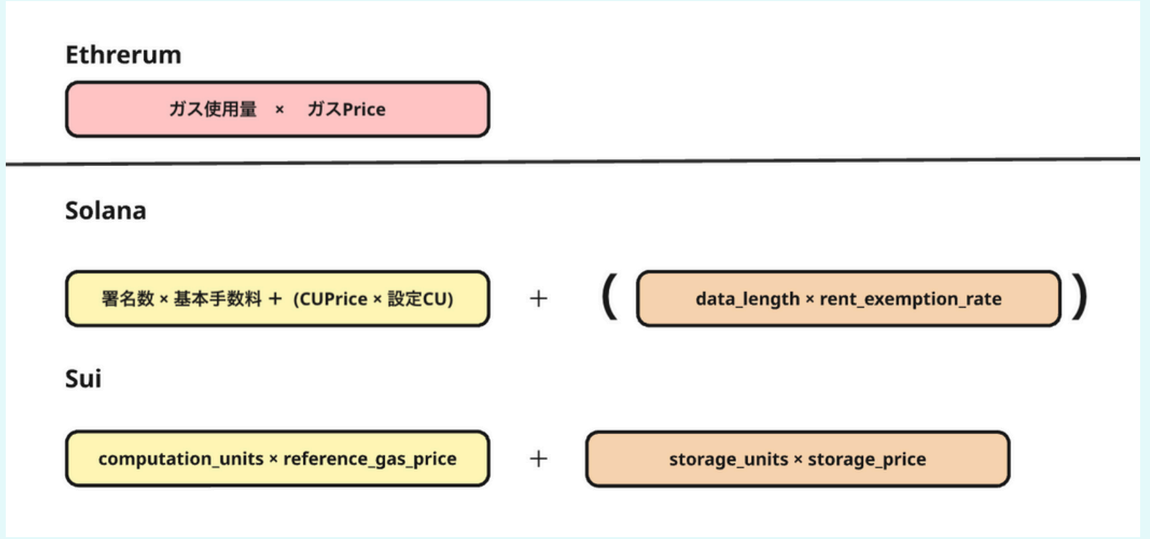
重要なコントラクトロジックの正しさを数学的に証明します。

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。

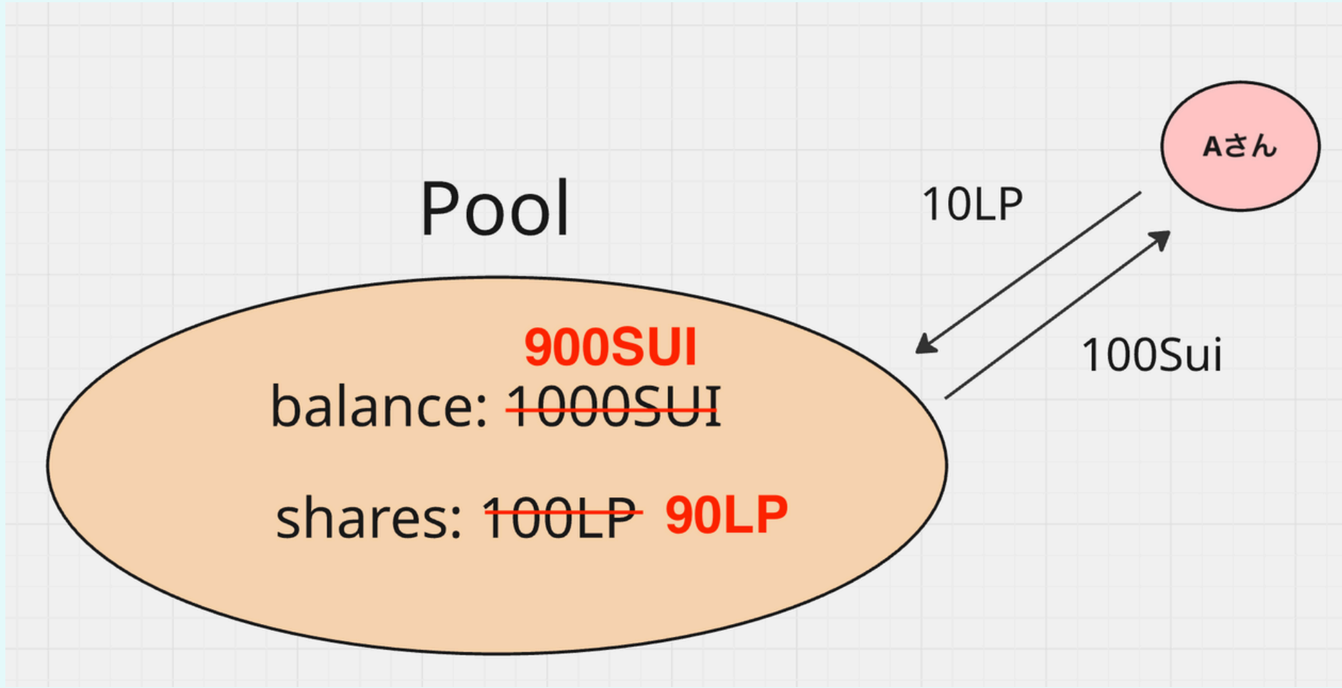


https://docs.sui.io/concepts/tokenomics/gas-in-sui

型からの派生　⑦形式的検証

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。

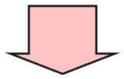


型からの派生　⑦形式的検証

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。

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



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

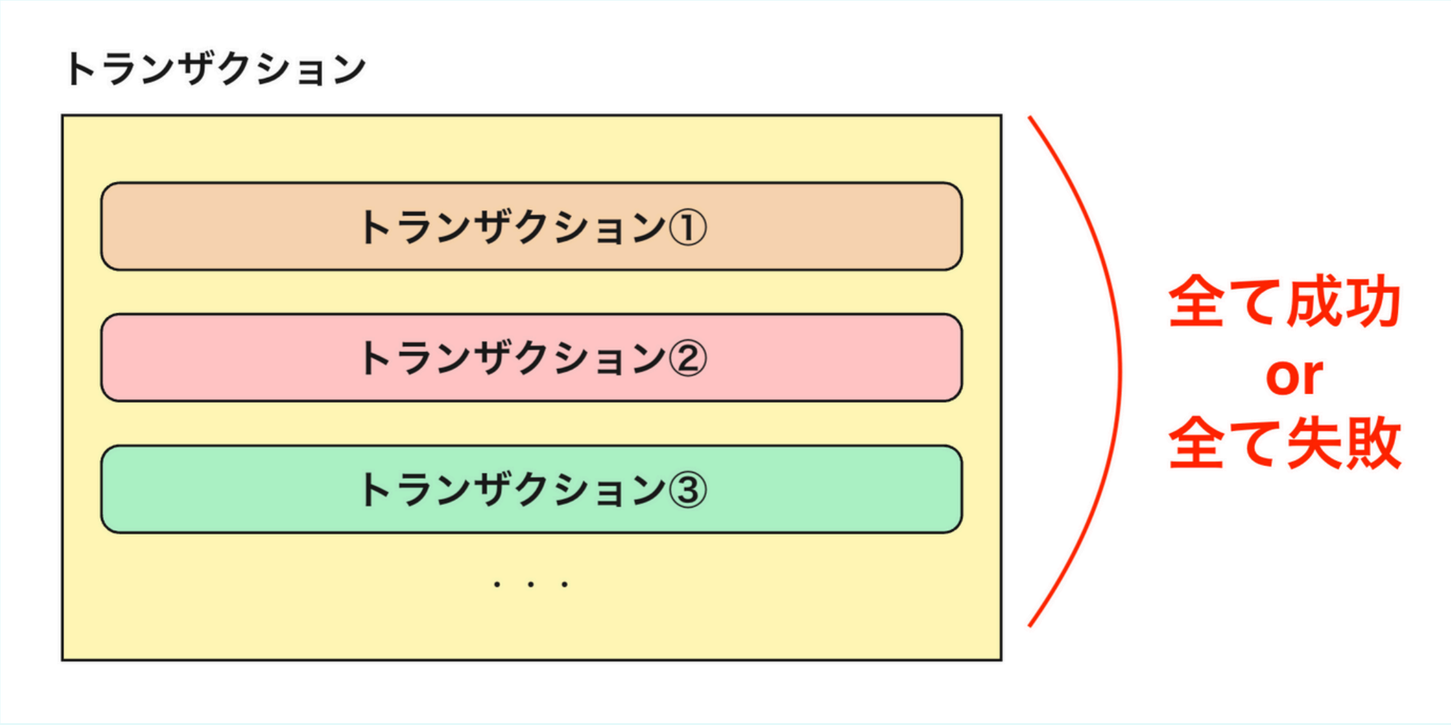
型からの派生　⑦形式的検証

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

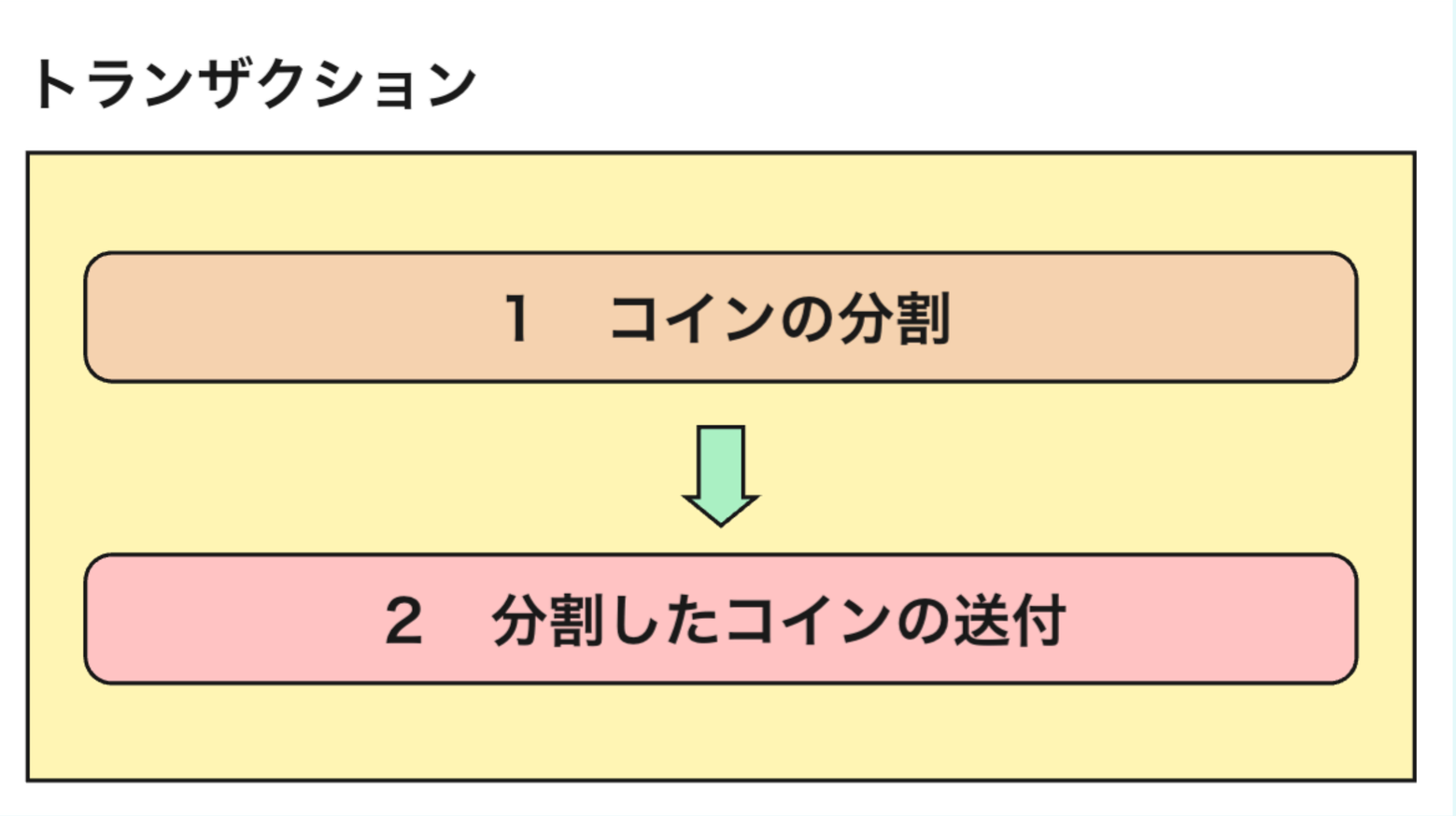
PTBとは

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。



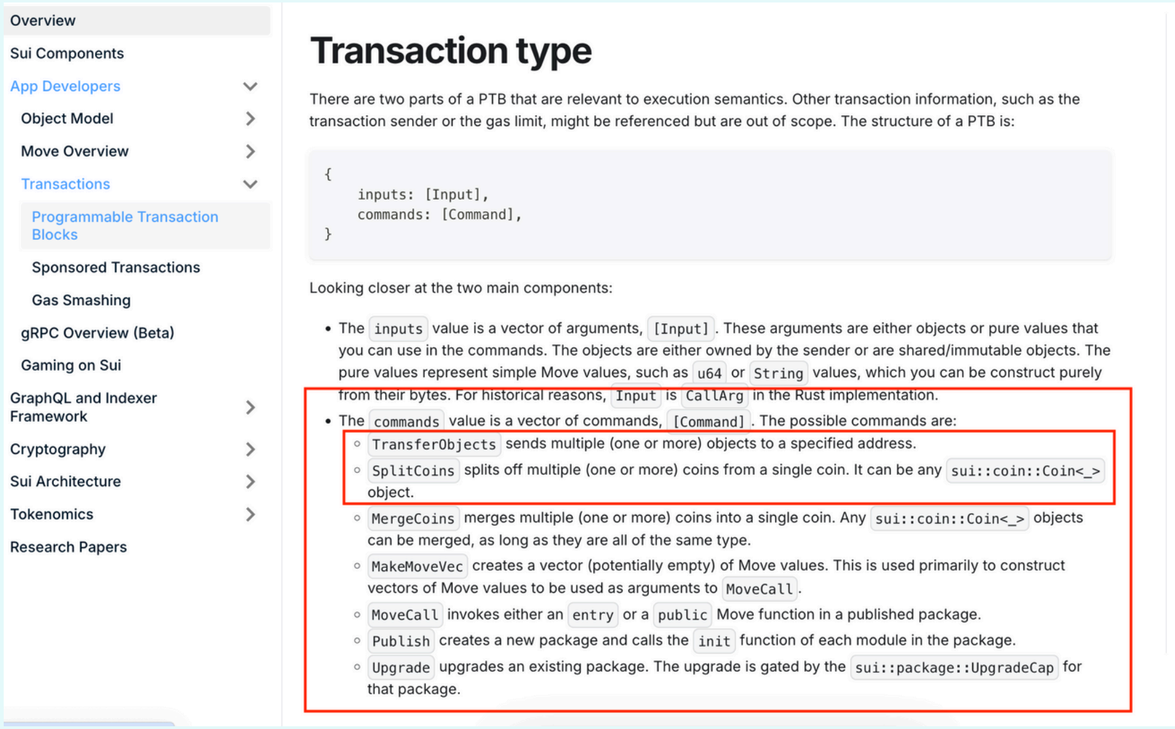
PTBとは

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。



PTBとは

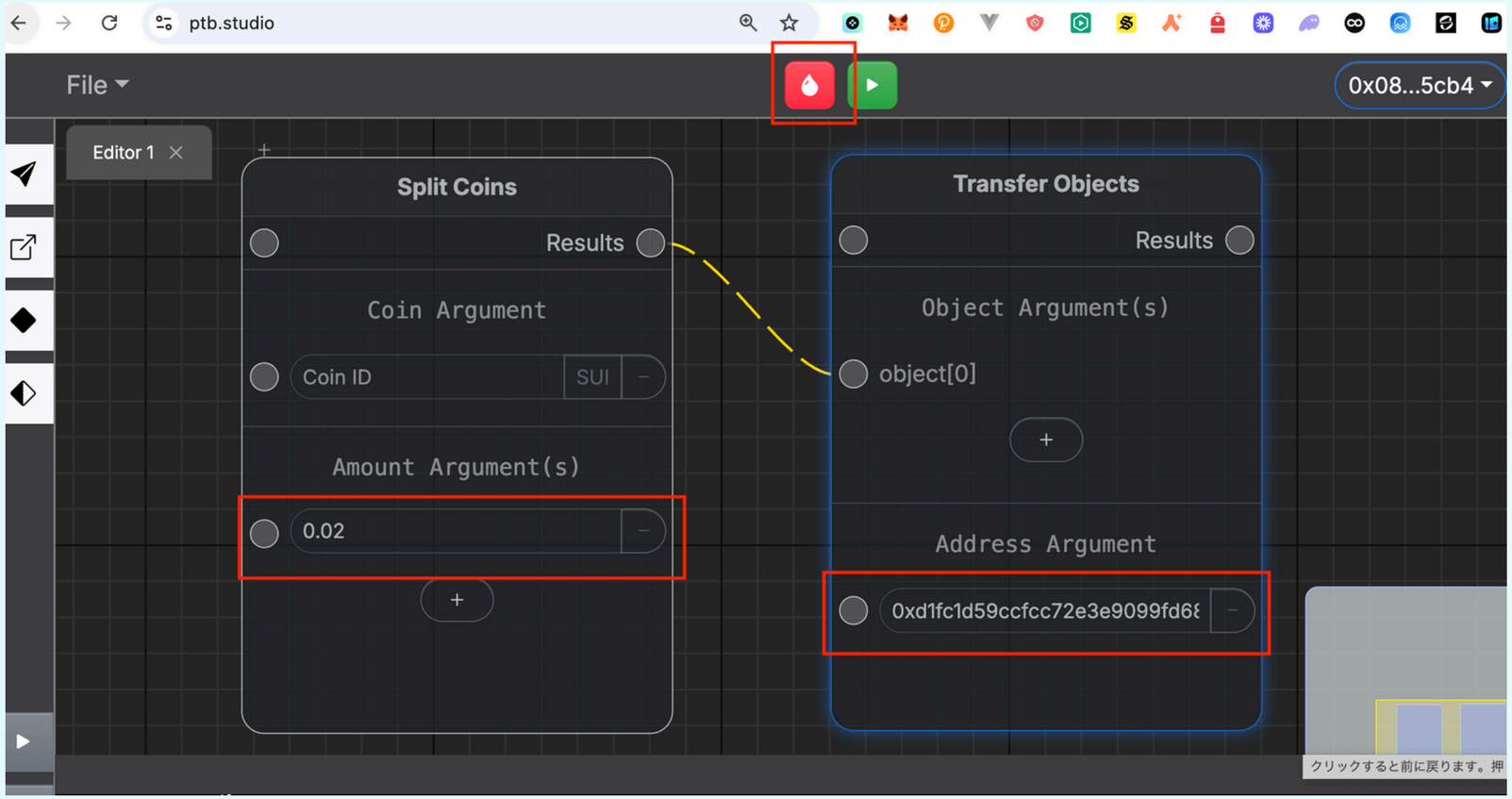
こちらが使用できるコマンドです。
今回は「TransferObjects」と「SplitCoins」を使います。



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

PTBとは

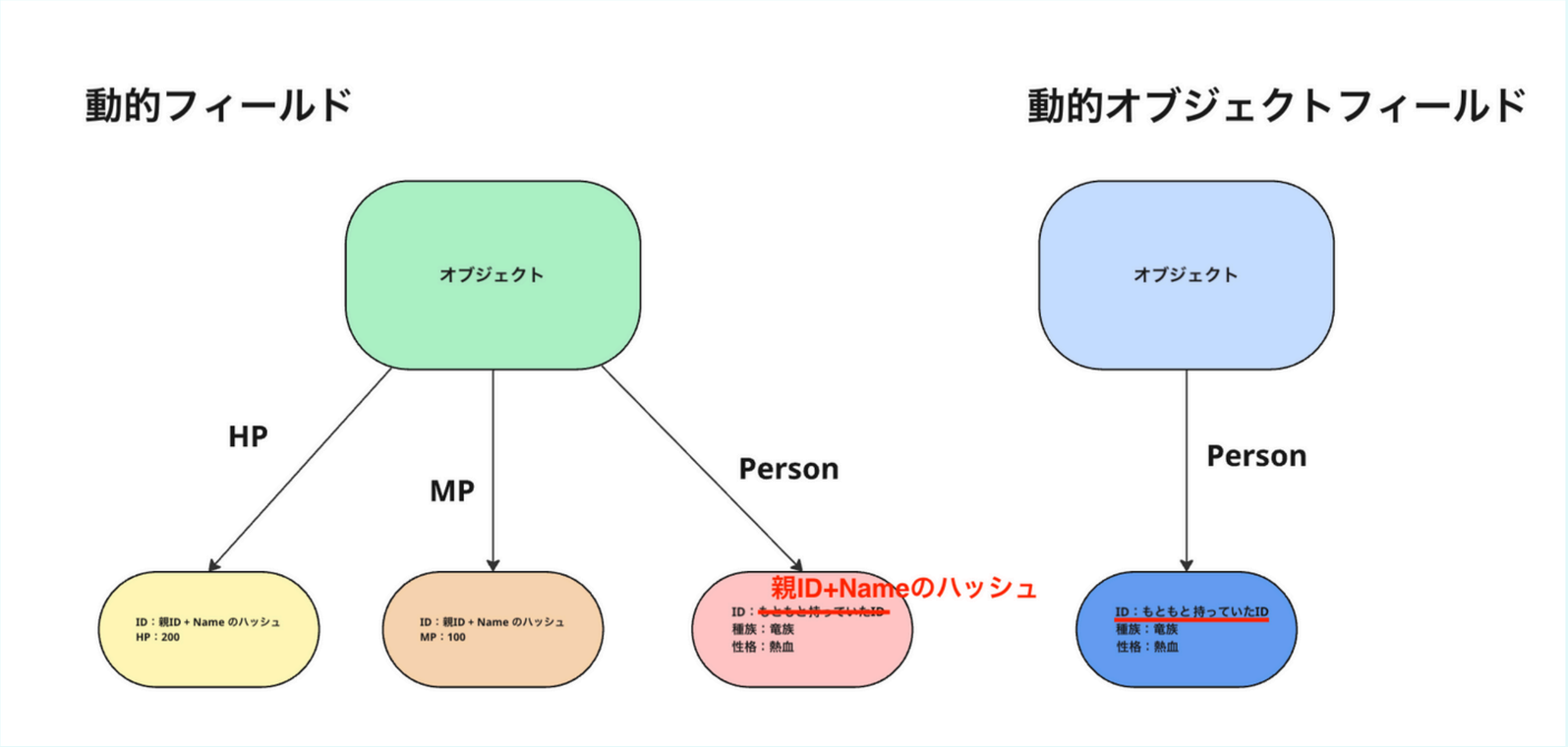
ptb.studioで体験してみましょう。



https://ptb.studio/

動的（オブジェクト）フィールド

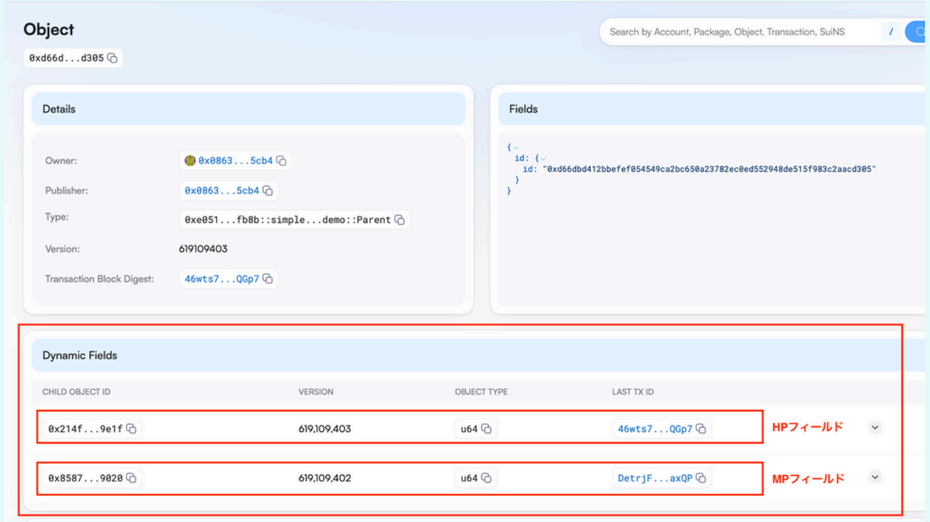
Suiではオブジェクトに動的にフィールドの設定ができます。
２種類がありますが、どちらもオブジェクトとして付属させます。



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

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

ガスレストランザクション

ガススポンサーの役割

✓ユーザー、ガスステーション、スポンサー間で手数料をシームレスに管理します。

サービス例

✓ShinamiのGas Stationを利用することで、ガス代のスポンサーが容易になります。
https://blog.sui.io/shinami-gas-station-tutorial/

ユースケース

✓ゲームdAppが初期トランザクションのガス代を負担し、ユーザー獲得を促進します。

演習

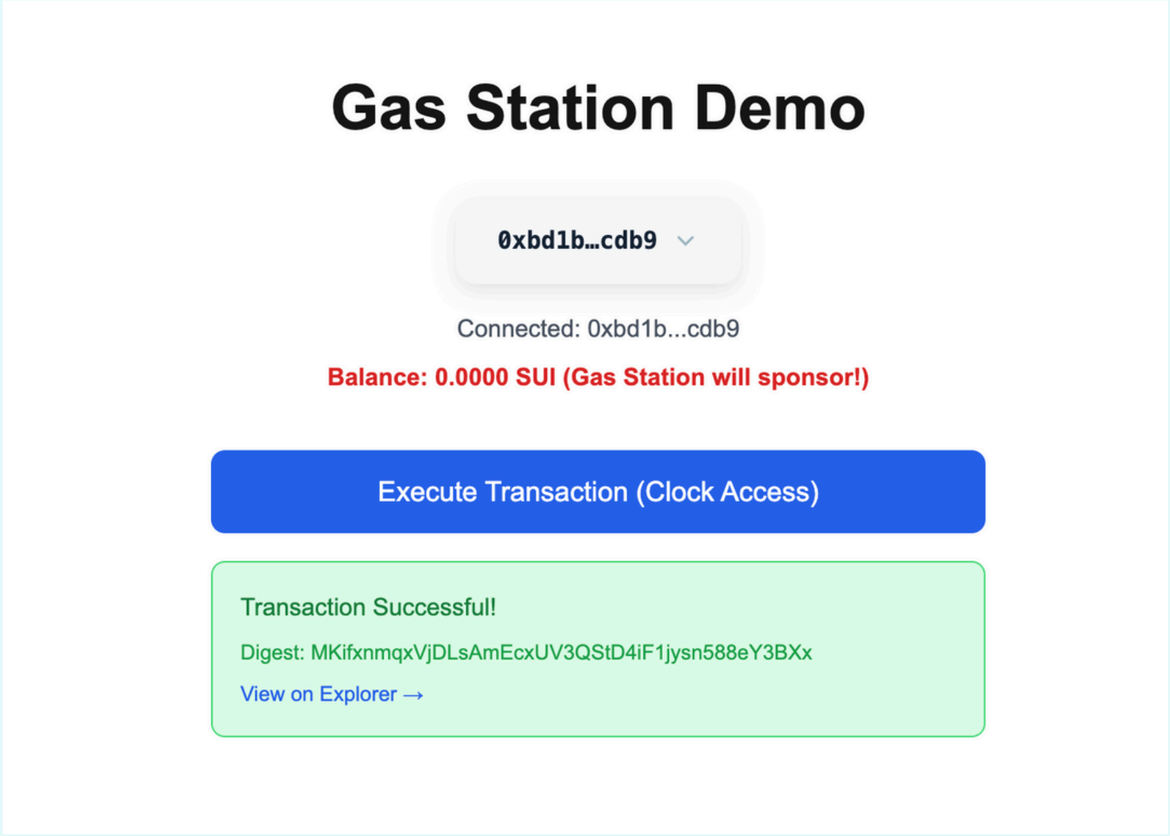
✓スポンサーを設定します。

✓ガスレストランザクションを送信します。

✓レシートとガス支払者をオンチェーンで確認します。

1 ガスレストランザクション

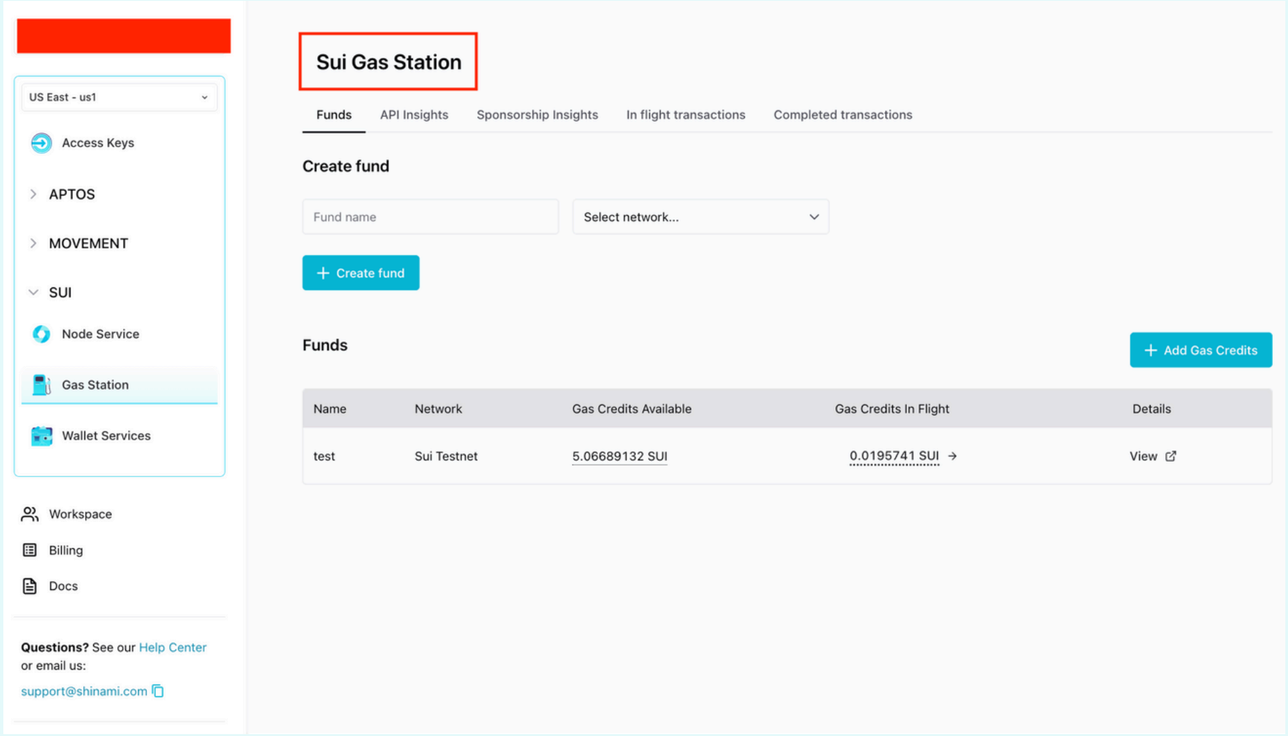
管理者がガス代を負担することで、**利用者はガス代なしで実行**ができます。



https://sui-gas-station-demo.vercel.app/

1 ガスレストランザクション

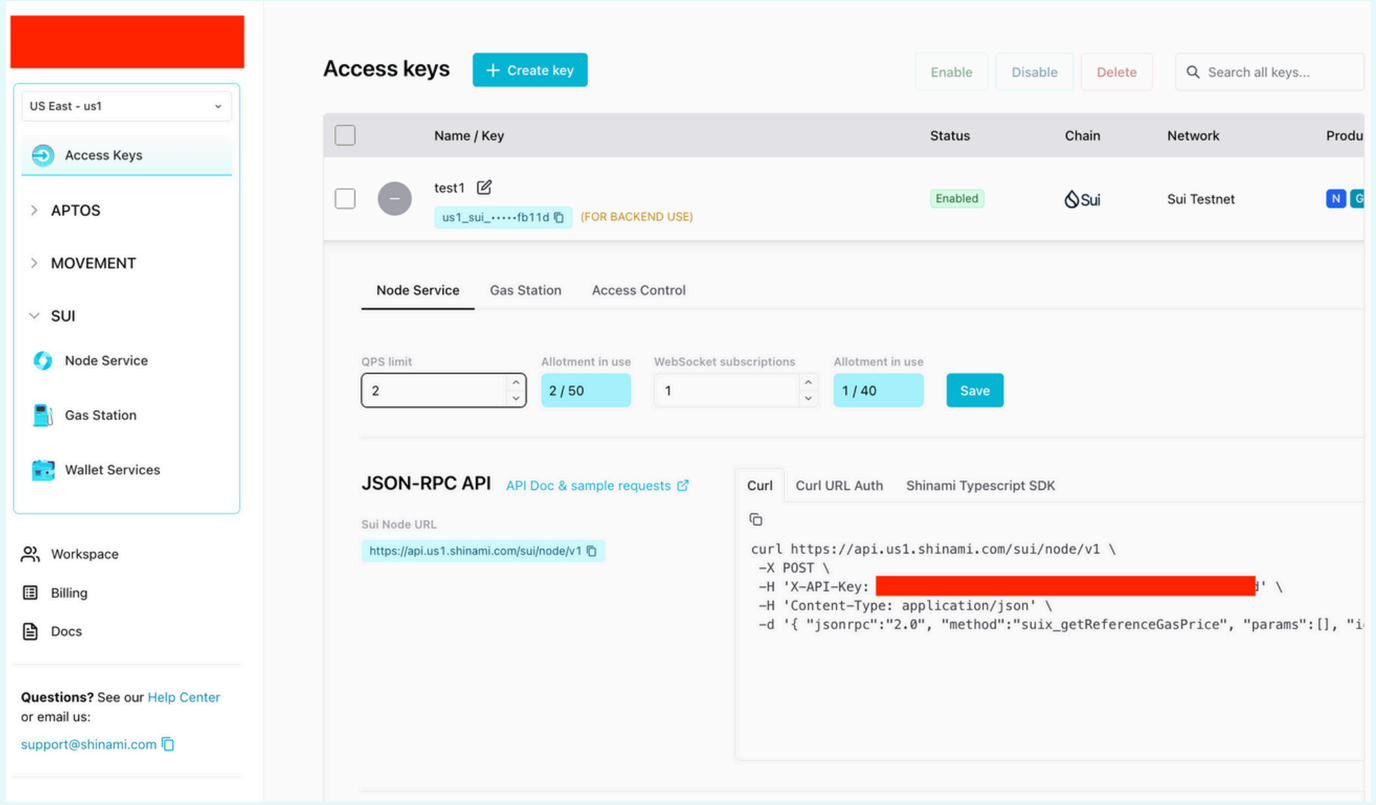
Shinamiを使うことで簡単に実装できます。
Gas Stationにガスを入れます。



https://app.shinami.com/

1 ガスレストランザクション

APIキーを取得することで簡単に実装できます。



1 ガスレストランザクション

そもそもの**スポンサードトランザクション**は**５つのステップ**からなります。

Create a user-initiated sponsored transaction

A user-initiated sponsored transaction involves the following steps:

1. A user initializes a `GasLessTransactionData` transaction. ①ユーザーが「ガスレストランザクションデータ」を作る
2. The user sends `GasLessTransactionData` to the sponsor. ②ユーザーが①をスポンサーに送る
3. The sponsor validates the transaction, constructs `TransactionData` with gas fees, and then signs `TransactionData`. ③スポンサーはガスをつけて「トランザクションデータ」を作り、署名する
4. The sponsor sends the signed `TransactionData` and the sponsor `Signature` back to the user. ④スポンサーは③とその署名をユーザーに送る
5. The user verifies and then signs `TransactionData` and sends the dual-signed transaction to Sui network through a full node or the sponsor. ⑤ユーザーは③を署名し、スポンサーの署名とともにSuiネットワークに送る

https://docs.sui.io/concepts/transactions/sponsored-transactions

1 ガスレストランザクション

コードを読み解いてみましょう。

```ts
async function main() {
  const node = createSuiClient(ACCESS_KEY); // Node Service
  const gas = new GasStationClient(ACCESS_KEY); // Gas Station
  const keypair = Ed25519Keypair.fromSecretKey(SENDER_SECRET);

  // 1) ガス無しTxKindを作る（時計アクセス）
  const tx = new Transaction();
  tx.moveCall({
    target:
      "0xfa0e78030bd16672174c2d6cc4cd5d1d1423d03c28a74909b2a148eda8bcca16::clock::access",
    arguments: [tx.object("0x6")],
  });
  const txKindBytes = await tx.build({
    client: node,
    onlyTransactionKind: true,
  });
  const txKindB64 = Buffer.from(txKindBytes).toString("base64");

  // 2) スポンサー取得（Auto-budget）
  const sponsorship = await gas.sponsorTransaction({
    txKind: txKindB64,
    sender: keypair.toSuiAddress(),
  });

  // 3) 送信者署名
  const signedBySender = await Transaction.from(sponsorship.txBytes).sign({
    signer: keypair,
  });

  // 4) 送信（スポンサー署名 + 送信者署名）
  const exec = await node.executeTransactionBlock({
    transactionBlock: sponsorship.txBytes,
    signature: [signedBySender.signature, sponsorship.signature],
  });

  console.log("txDigest:", exec.digest);
}
```

https://github.com/SuiJapan/core-workshop/blob/main/chapter5/shinami/gas_station_min.ts

1 ガスレストランザクション

- npm init -yでpackage.json作成
- touch index.jsでファイル作成
- APIキー、プライベートキーを設定
- npm i @shinami/clients @mysten/sui/keypairs

https://github.com/SuiJapan/core-workshop/blob/main/chapter5/shinami/gas_station_min.ts

1 ガスレストランザクション

①ガスレストランザクションデータの作成

ポイントは**onlyTransactionKind：true**です。

```
  // 1) ガス無しTxKindを作る（時計アクセス）
  const tx = new Transaction();
  tx.moveCall({
    target:
      "0xfa0e78030bd16672174c2d6cc4cd5d1d1423d03c28a74909b2a148eda8bcca16::clock::access",
    arguments: [tx.object("0x6")],
  });
  const txKindBytes = await tx.build({
    client: node,
    onlyTransactionKind: true,
  });
  const txKindB64 = Buffer.from(txKindBytes).toString("base64");

  // 2) スポンサー取得（Auto-budget）
  const sponsorship = await gas.sponsorTransaction({
    txKind: txKindB64,
    sender: keypair.toSuiAddress(),
  });

  // 3) 送信者署名
  const signedBySender = await Transaction.from(sponsorship.txBytes).sign({
    signer: keypair,
  });

  // 4) 送信（スポンサー署名 + 送信者署名）
  const exec = await node.executeTransactionBlock({
    transactionBlock: sponsorship.txBytes,
    signature: [signedBySender.signature, sponsorship.signature],
  });

  console.log("txDigest:", exec.digest);
}
```

https://sdk.mystenlabs.com/typescript/transaction-building/sponsored-transactions

1 ガスレストランザクション

②③スポンサーによる署名

ShinamiのSDKを使い、ガスの設定～署名を行なっています。

```
// 1) ガス無しTxKindを作る（時計アクセス）
const tx = new Transaction();
tx.moveCall({
  target:
    "0xfa0e78030bd16672174c2d6cc4cd5d1d1423d03c28a74909b2a148eda8bcca16::clock::access",
  arguments: [tx.object("0x6")],
});
const txKindBytes = await tx.build({
  client: node,
  onlyTransactionKind: true,
});
const txKindB64 = Buffer.from(txKindBytes).toString("base64");

// 2) スポンサー取得（Auto-budget）
const sponsorship = await gas.sponsorTransaction({
  txKind: txKindB64,
  sender: keypair.toSuiAddress(),
});

// 3) 送信者署名
const signedBySender = await Transaction.from(sponsorship.txBytes).sign({
  signer: keypair,
});

// 4) 送信（スポンサー署名 + 送信者署名）
const exec = await node.executeTransactionBlock({
  transactionBlock: sponsorship.txBytes,
  signature: [signedBySender.signature, sponsorship.signature],
});

console.log("txDigest:", exec.digest);
}
```

1 ガスレストランザクション

④⑤ユーザーによる署名

スポンサーによりガスが設定された取引データに対して、署名しています。

```
// 1) ガス無しTxKindを作る（時計アクセス）
const tx = new Transaction();
tx.moveCall({
  target:
    "0xfa0e78030bd16672174c2d6cc4cd5d1d1423d03c28a74909b2a148eda8bcca16::clock::access",
  arguments: [tx.object("0x6")],
});
const txKindBytes = await tx.build({
  client: node,
  onlyTransactionKind: true,
});
const txKindB64 = Buffer.from(txKindBytes).toString("base64");

// 2) スポンサー取得（Auto-budget）
const sponsorship = await gas.sponsorTransaction({
  txKind: txKindB64,
  sender: keypair.toSuiAddress(),
});

// 3) 送信者署名
const signedBySender = await Transaction.from(sponsorship.txBytes).sign({
  signer: keypair,
});

// 4) 送信（スポンサー署名 + 送信者署名）
const exec = await node.executeTransactionBlock({
  transactionBlock: sponsorship.txBytes,
  signature: [signedBySender.signature, sponsorship.signature],
});

console.log("txDigest:", exec.digest);
}
```

1 ガスレストランザクション

④⑤ユーザーによる署名

スポンサーによりガスが設定された取引データに対して、署名しています。

```
// 1) ガス無しTxKindを作る（時計アクセス）
const tx = new Transaction();
tx.moveCall({
  target:
    "0xfa0e78030bd16672174c2d6cc4cd5d1d1423d03c28a74909b2a148eda8bcca16::clock::access",
  arguments: [tx.object("0x6")],
});
const txKindBytes = await tx.build({
  client: node,
  onlyTransactionKind: true,
});
const txKindB64 = Buffer.from(txKindBytes).toString("base64");

// 2) スポンサー取得（Auto-budget）
const sponsorship = await gas.sponsorTransaction({
  txKind: txKindB64,
  sender: keypair.toSuiAddress(),
});

// 3) 送信者署名
const signedBySender = await Transaction.from(sponsorship.txBytes).sign({
  signer: keypair,
});

// 4) 送信（スポンサー署名 + 送信者署名）
const exec = await node.executeTransactionBlock({
  transactionBlock: sponsorship.txBytes,
  signature: [signedBySender.signature, sponsorship.signature],
});

console.log("txDigest:", exec.digest);
}
```

1 ガスレストランザクション

⑤トランザクションの実行

ユーザーとスポンサーの署名をつけて実行しています。

```
// 1) ガス無しTxKindを作る（時計アクセス）
const tx = new Transaction();
tx.moveCall({
  target:
    "0xfa0e78030bd16672174c2d6cc4cd5d1d1423d03c28a74909b2a148eda8bcca16::clock::access",
  arguments: [tx.object("0x6")],
});
const txKindBytes = await tx.build({
  client: node,
  onlyTransactionKind: true,
});
const txKindB64 = Buffer.from(txKindBytes).toString("base64");

// 2) スポンサー取得（Auto-budget）
const sponsorship = await gas.sponsorTransaction({
  txKind: txKindB64,
  sender: keypair.toSuiAddress(),
});

// 3) 送信者署名
const signedBySender = await Transaction.from(sponsorship.txBytes).sign({
  signer: keypair,
});

// 4) 送信（スポンサー署名 + 送信者署名）
const exec = await node.executeTransactionBlock({
  transactionBlock: sponsorship.txBytes,
  signature: [signedBySender.signature, sponsorship.signature],
});

console.log("txDigest:", exec.digest);
}
```

Suiでのオラクル利用

利用可能なオラクル

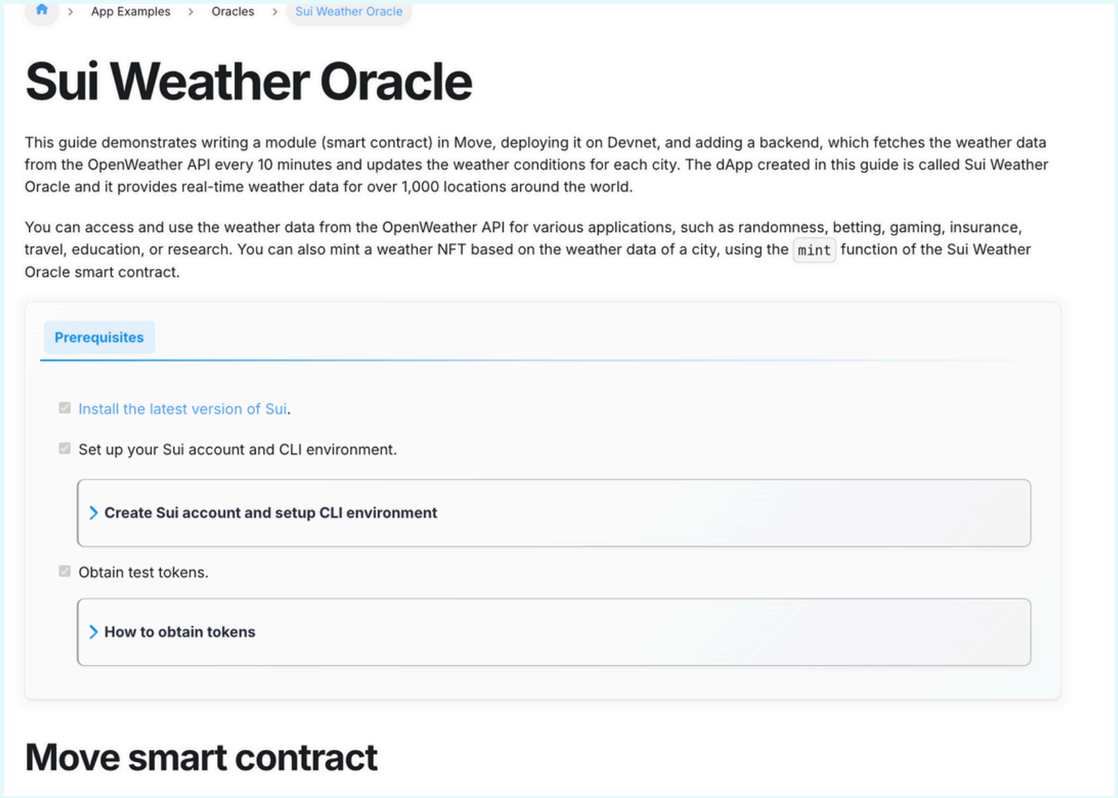
- Chainlink
- Band Protocol
- Mysten Labs製のsimple oracleとmeta oracle

開発者向けガイド

- 天気オラクルの実装例
  https://docs.sui.io/guides/developer/app-examples/weather-oracle#initialize-the-project
- Moveオラクルのサンプルコード
  https://github.com/pentagonxyz/move-oracles

2 オラクル

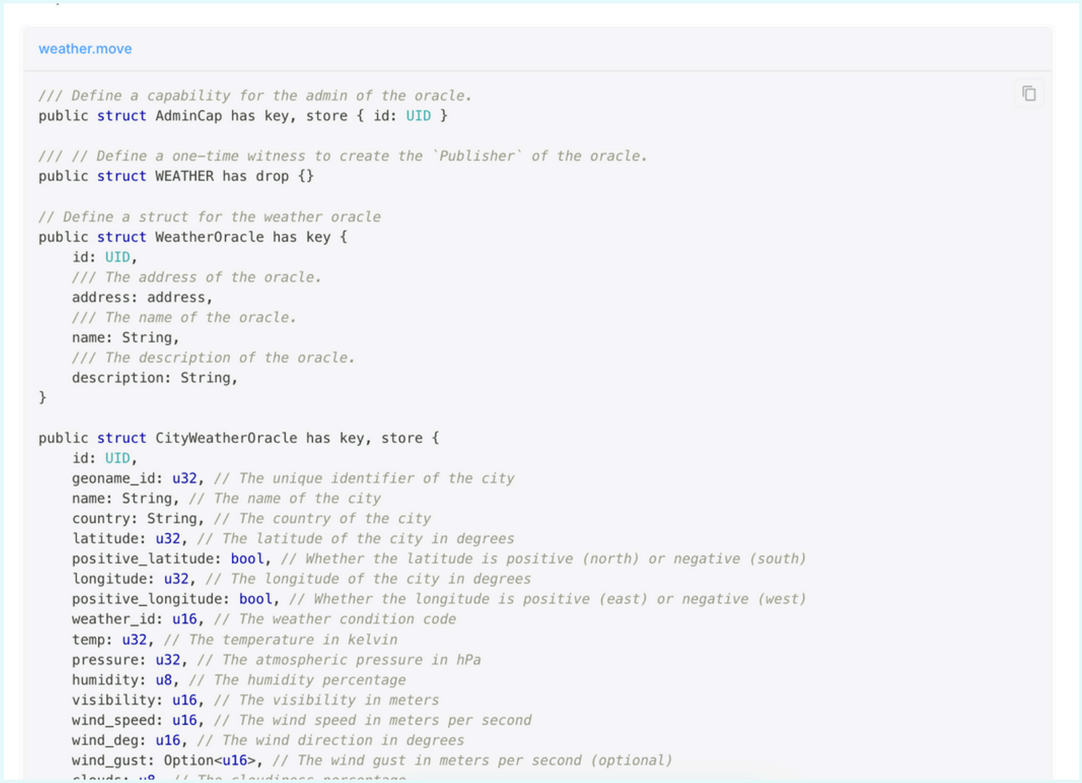
Sui Weather Oracleは、かなりの文量です。。



https://docs.sui.io/guides/developer/app-examples/weather-oracle

2 オラクル

Sui Weather Oracleは、かなりの文量です。。



weather.move

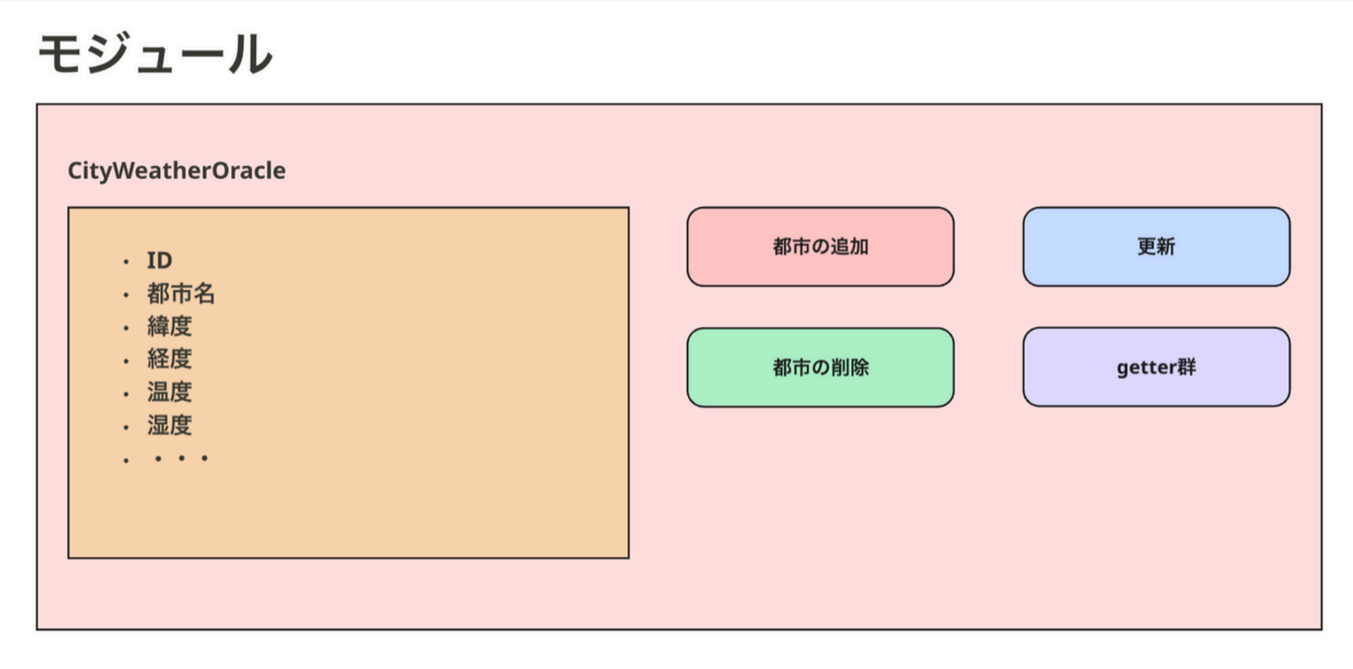
```
/// Define a capability for the admin of the oracle.
public struct AdminCap has key, store { id: UID }

/// // Define a one-time witness to create the `Publisher` of the oracle.
public struct WEATHER has drop {}

// Define a struct for the weather oracle
public struct WeatherOracle has key {
    id: UID,
    /// The address of the oracle.
    address: address,
    /// The name of the oracle.
    name: String,
    /// The description of the oracle.
    description: String,
}

public struct CityWeatherOracle has key, store {
    id: UID,
    geoname_id: u32, // The unique identifier of the city
    name: String, // The name of the city
    country: String, // The country of the city
    latitude: u32, // The latitude of the city in degrees
    positive_latitude: bool, // Whether the latitude is positive (north) or negative (south)
    longitude: u32, // The longitude of the city in degrees
    positive_longitude: bool, // Whether the longitude is positive (east) or negative (west)
    weather_id: u16, // The weather condition code
    temp: u32, // The temperature in kelvin
    pressure: u32, // The atmospheric pressure in hPa
    humidity: u8, // The humidity percentage
    visibility: u16, // The visibility in meters
    wind_speed: u16, // The wind speed in meters per second
    wind_deg: u16, // The wind direction in degrees
    wind_gust: Option<u16>, // The wind gust in meters per second (optional)
    clouds: u8, // The cloudiness percentage
```

https://docs.sui.io/guides/developer/app-examples/weather-oracle

2 オラクル

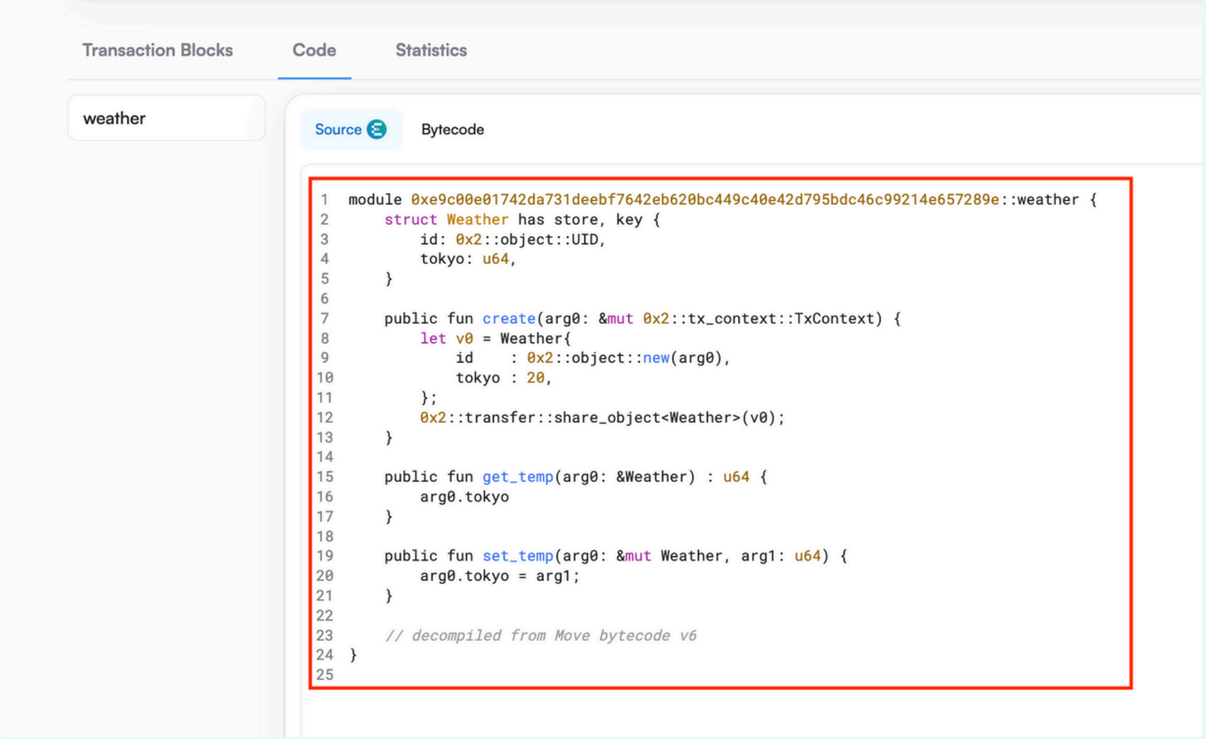
結局はただのモジュールです。**バックエンドで定期的に値を更新**します。

もちろんオラクルなので、**更新者が1人なのか分散した平均か**、また**権限**も重要にはなります。



2 オラクル

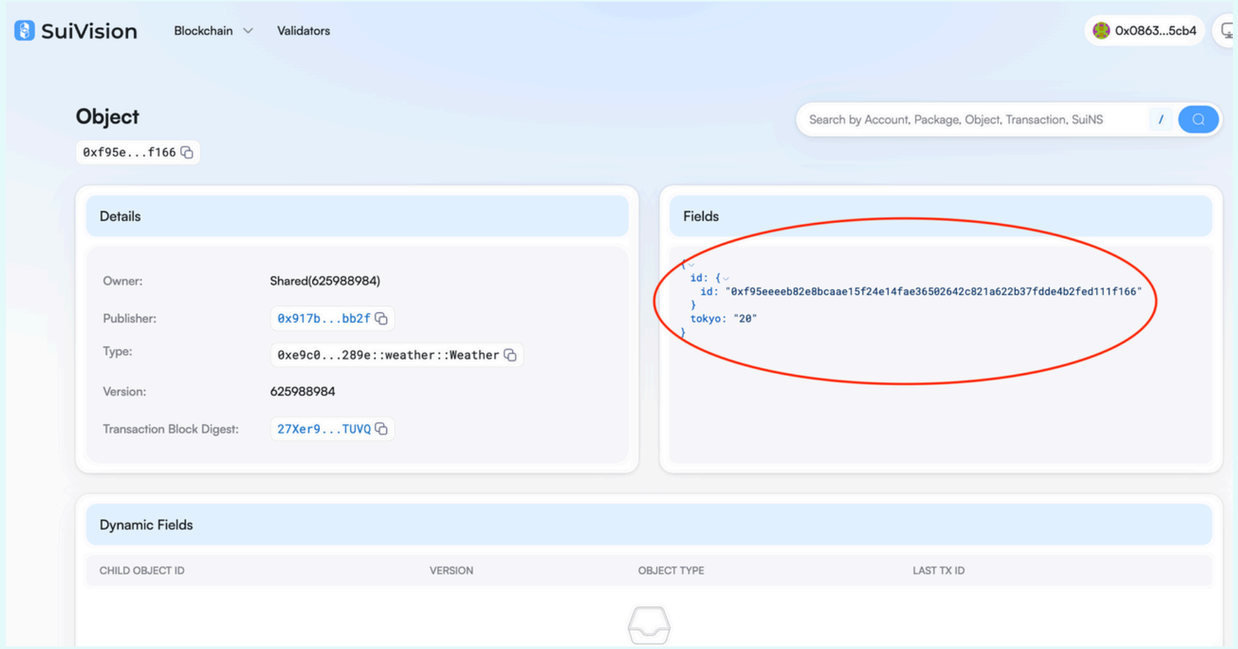
そのため、今回は、外部ですでに作られているパッケージの値を読み取り処理を行う方法を学びましょう。



https://testnet.suivision.xyz/package/0xe9c00e01742da731deebf7642eb620bc449c40e42d795bdc46c99214e657289e?tab=Code

2 オラクル

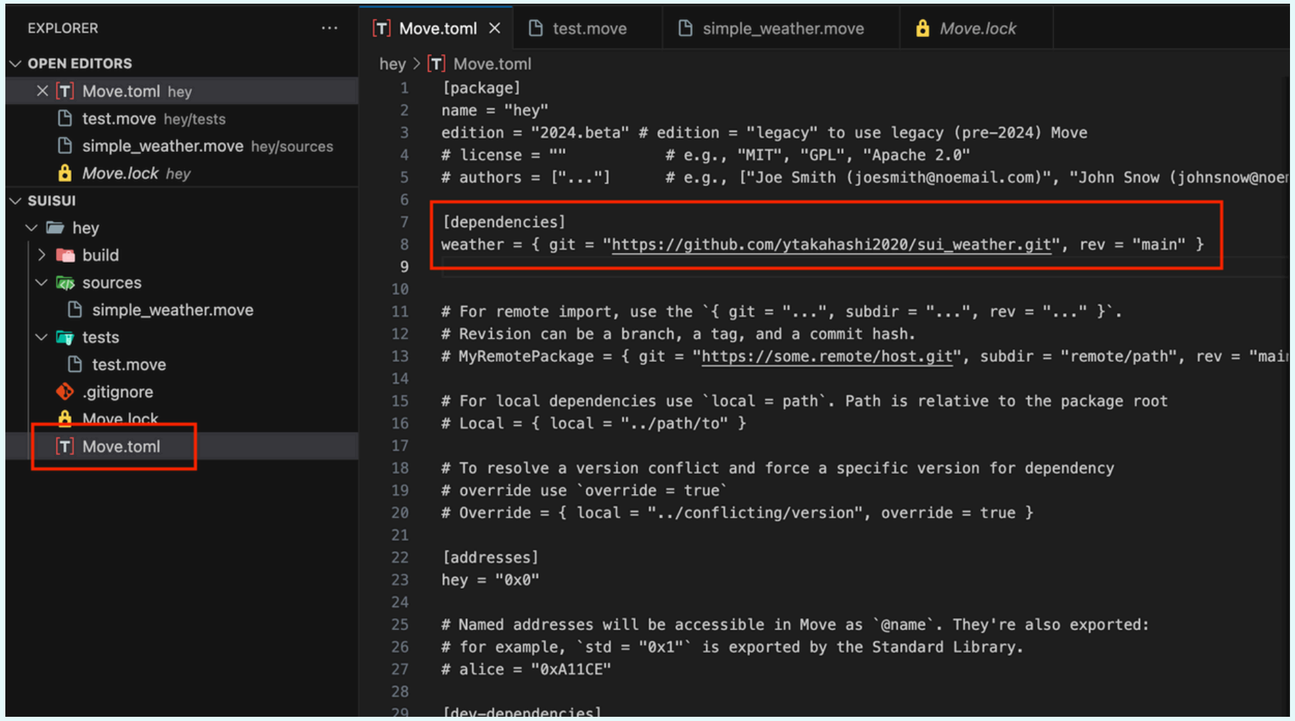
こちらがパッケージから作られたオブジェクトです。
東京が20度であることがわかります。
これを取得し、**20度以上なら◯◯する**という処理を作りましょう。



https://testnet.suivision.xyz/object/0xf95eeeeb82e8bcaae15f24e14fae36502642c821a622b37fdde4b2fed111f166

2 オラクル

最も大事なのが、こちらです。
Move.tomlでパッケージのコードが格納されているGitを指定します。
https://github.com/ytakahashi2020/sui_weather



https://github.com/ytakahashi2020/sui_oracle

2 オラクル

では、作ってみましょう。
ここでは、20度以上の場合、**イベントを発火するモジュール**を作ります。
まずは構造体を作ります。

hey > sources > simple_weather copy.move > ...

```
1   module hey::weather_alert;
2
3   use std::string;
4   use sui::event;
5
6   /// イベント構造体
7   public struct HotEvent has copy, drop, store {
8       message: string::String,
9       temperature: u64,
10  }
11
12
```

https://github.com/ytakahashi2020/sui_oracle

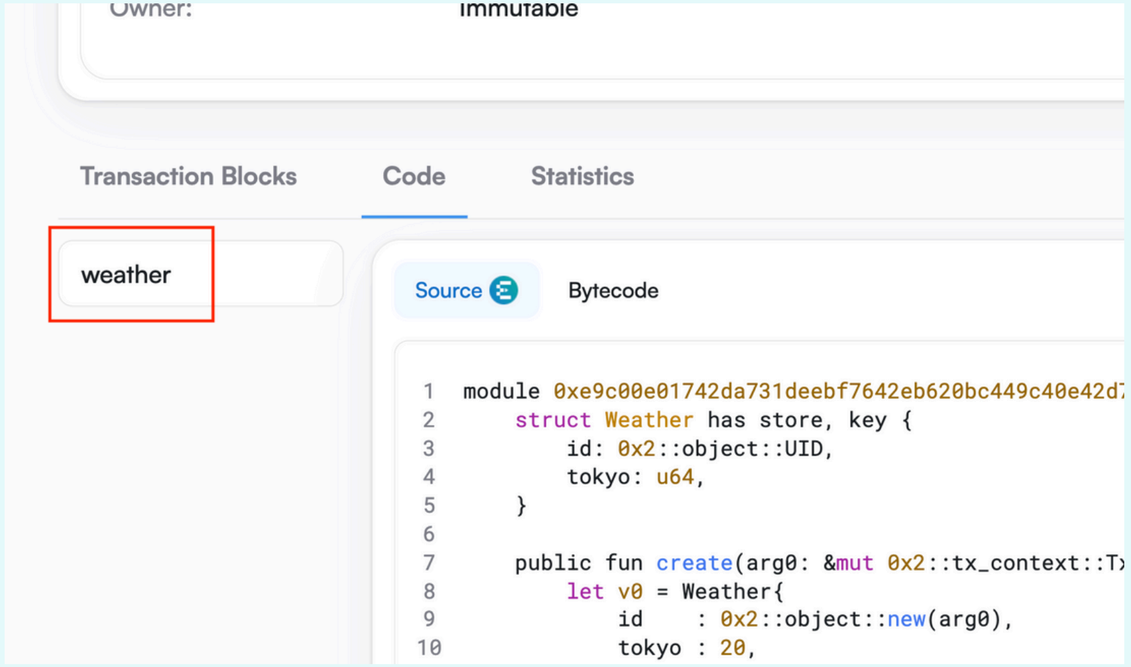
2 オラクル

次に、weatherパッケージのweatherモジュールの使用を宣言します。

e.toml .gitignore test.move simple_weather.mov

hey > sources > simple_weather copy.move > ...

```
1   module hey::weather_alert;
2
3   use std::string;
4   use sui::event;
5   use weather::weather;
6
7   /// イベント構造体
8   public struct HotEvent has copy, drop, store {
9       message: string::String,
10      temperature: u64,
11  }
12
```



https://github.com/ytakahashi2020/sui_oracle

2 オラクル

次に、weatherモジュールのget_tempから値を取得します。

hey > sources > simple_weather.move > { } simple_weather > check_and_emit

```
1  module hey::simple_weather;
2
3  use std::string;
4  use sui::event;
5  use weather::weather;
6
7  /// イベント構造体
8  public struct HotEvent has copy, drop, store {
9      message: string::String,
10     temperature: u64,
11 }
12
13 /// Weatherオブジェクトを読み取って条件チェック
14 public fun check_and_emit(weather_obj: &weather::Weather, _ctx: &mut TxContext) {
15     let temp = weather::get_temp(weather_obj);
16
17
18 }
19
```

Transaction Blocks | Code | Statistics

weather

Source | Bytecode

```
1  module 0xe9c00e01742da731deebf7642eb620bc449c40e42d795bdc46c99214e657289e::weather {
2      struct Weather has store, key {
3          id: 0x2::object::UID,
4          tokyo: u64,
5      }
6
7      public fun create(arg0: &mut 0x2::tx_context::TxContext) {
8          let v0 = Weather{
9              id    : 0x2::object::new(arg0),
10             tokyo : 20,
11         };
12         0x2::transfer::share_object<Weather>(v0);
13     }
14
15     public fun get_temp(arg0: &Weather) : u64 {
16         arg0.tokyo
17     }
18
19     public fun set_temp(arg0: &mut Weather, arg1: u64) {
20         arg0.tokyo = arg1;
21     }
22
23     // decompiled from Move bytecode v6
24 }
25
```

https://github.com/ytakahashi2020/sui_oracle

2 オラクル

最後にこのようにイベントを発火しています。

Move.toml | simple_weather.move M

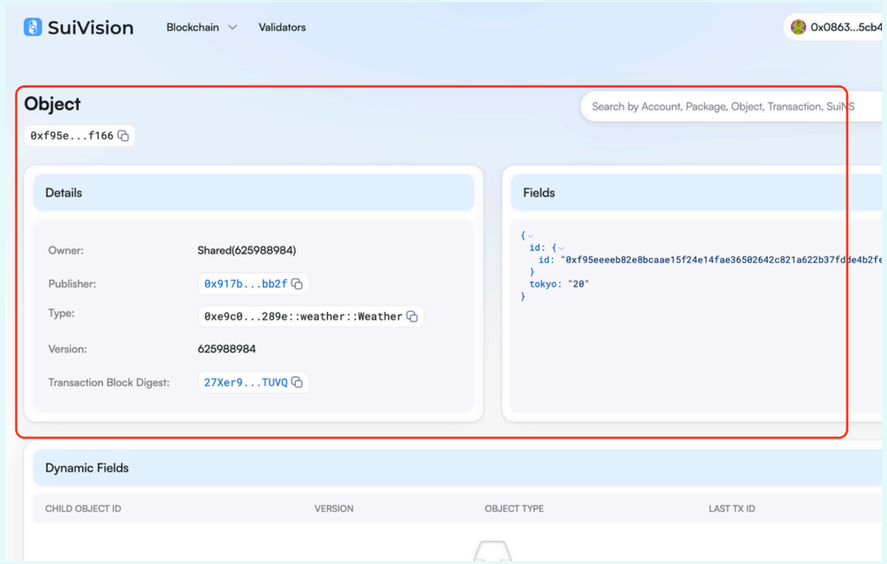
hey > sources > simple_weather.move > sui-move-analyzer > { } simple_weather > check_and_emit

```
1  module hey::simple_weather;
2
3  use std::string;
4  use sui::event;
5  use weather::weather;
6
7  /// イベント構造体
8  public struct HotEvent has copy, drop, store {
9      message: string::String,
10     temperature: u64,
11 }
12
13 /// Weatherオブジェクトを読み取って条件チェック
14 public fun check_and_emit(weather_obj: &weather::Weather, _ctx: &mut TxContext) {
15     let temp: u64 = weather::get_temp(weather: weather_obj);
16
17     if (temp >= 20) {
18         // 🍡 20度以上ならイベント発火
19         let message: String = b"It's hot today".to_string();
20         let event_data : HotEvent : HotEvent = HotEvent {
21             message : String ,
22             temperature : u64 : temp,
23         };
24         event::emit( event: event: event_data);
25     };
26 }
27
```

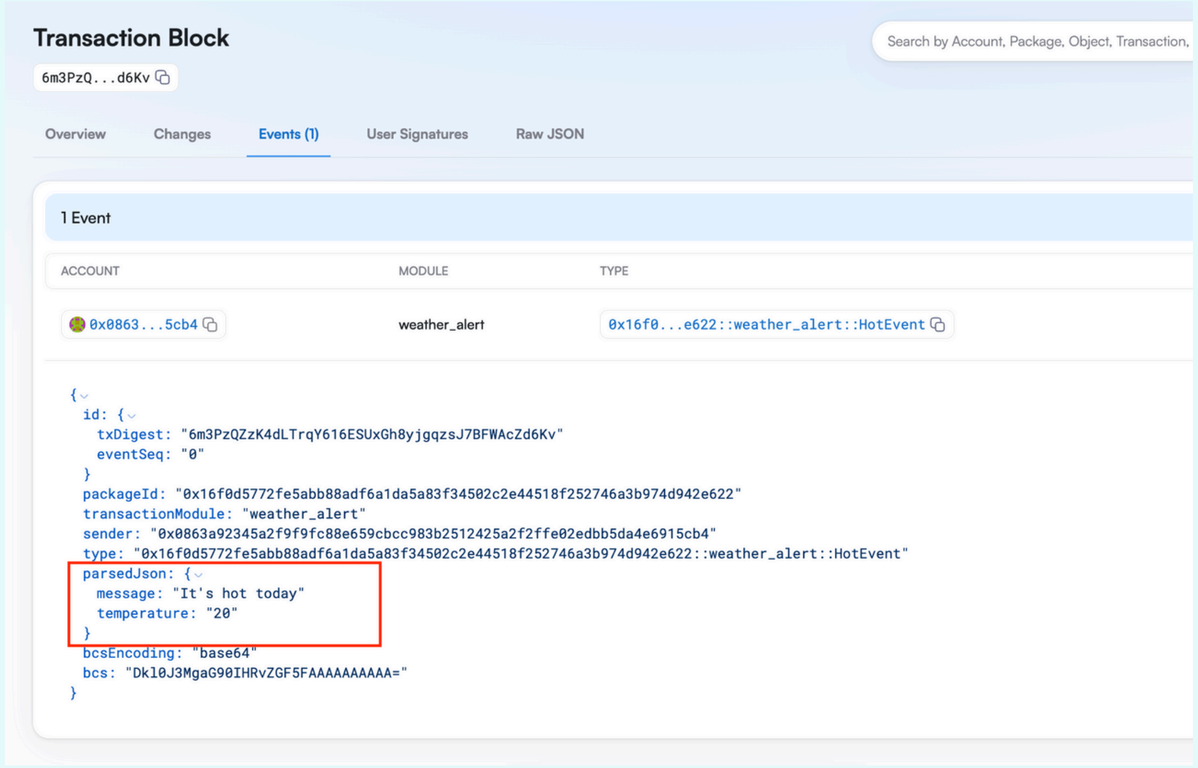
https://github.com/ytakahashi2020/sui_oracle

2 オラクル

できたパッケージの関数を実行すると、イベントが発火していることがわかります。



https://testnet.suivision.xyz/object/0xf95eeeeb82e8bcaae15f24e14fae36502642c821a622b37fdde4b2fed111f166



https://testnet.suivision.xyz/txblock/6m3PzQZzK4dLTrqY616ESUxGh8yjgqzsJ7BFWAcZd6Kv?tab=Events

SuiでのAIエージェント構築

エージェントロジック

- オンチェーンでのアクションはMoveスマートコントラクトで定義します。

オラクルと自動化

- データフィードを接続し、スポンサー付きトランザクションでプロセスを自動化します。

外部AIモデル

- 意思決定にオフチェーンのAIモデルを統合します。

セキュリティ

- Moveのケイパビリティとアクセス制御を活用します。

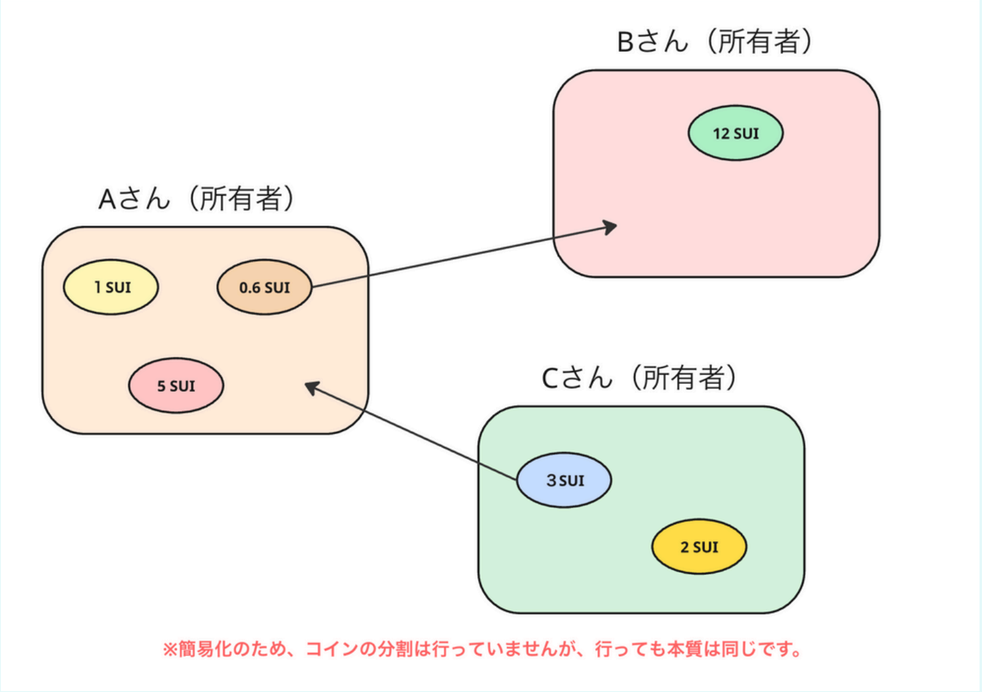
https://cloud.google.com/blog/ja/products/ai-machine-learning/announcing-agents-to-payments-ap2-protocol

(再掲)型からの派生②オブジェクトと所有権

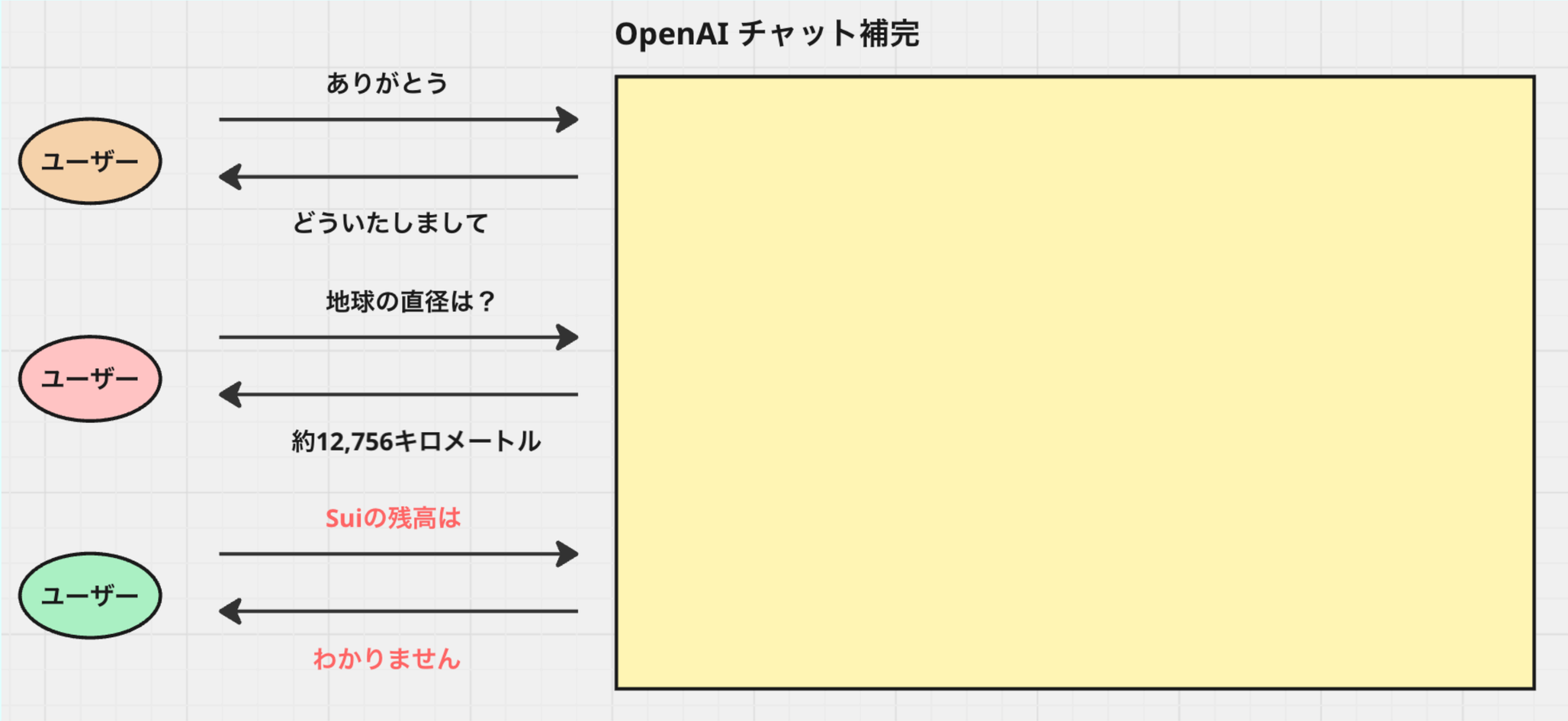
Suiの場合はオブジェクトを元にした所有権の設計です。

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。**

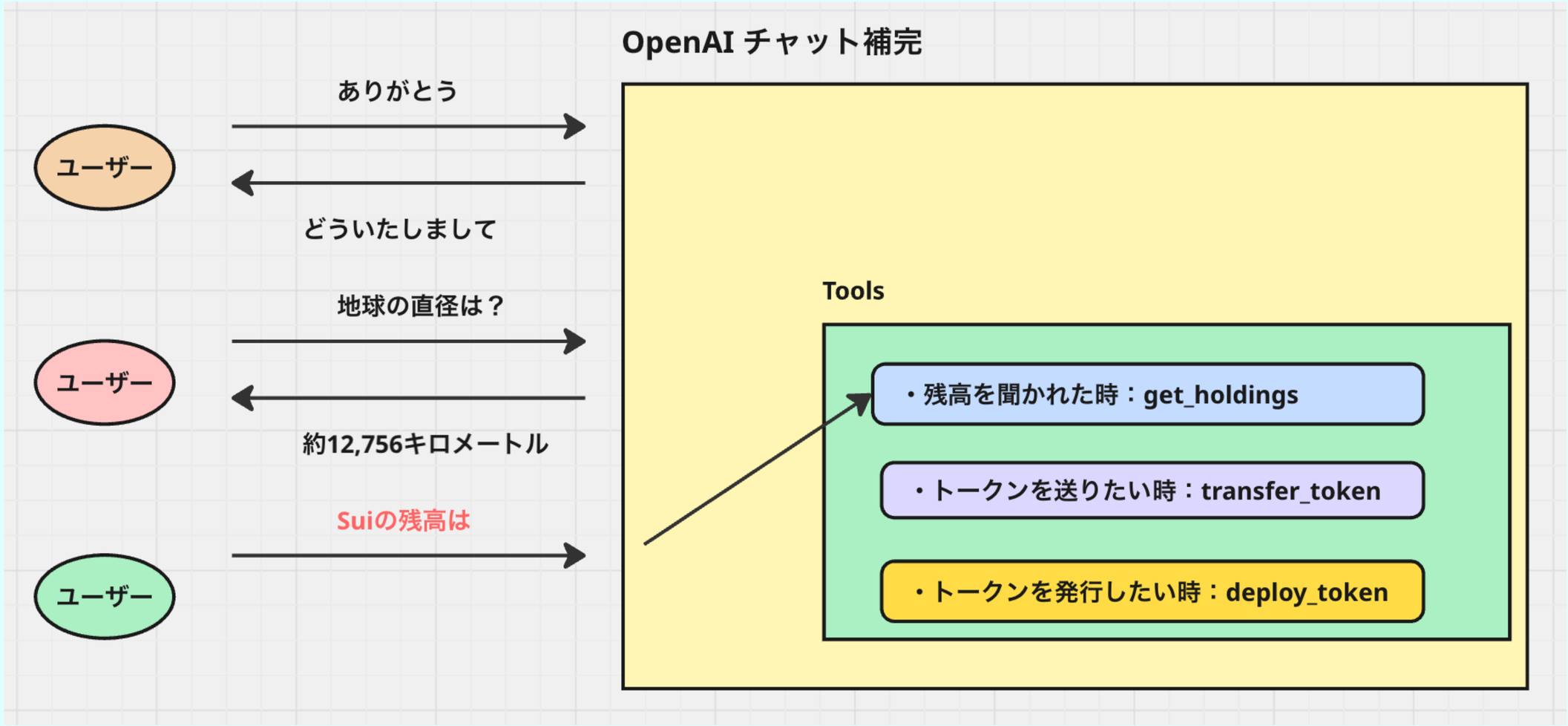
そのため**機械的に並列処理**を行うことができます。



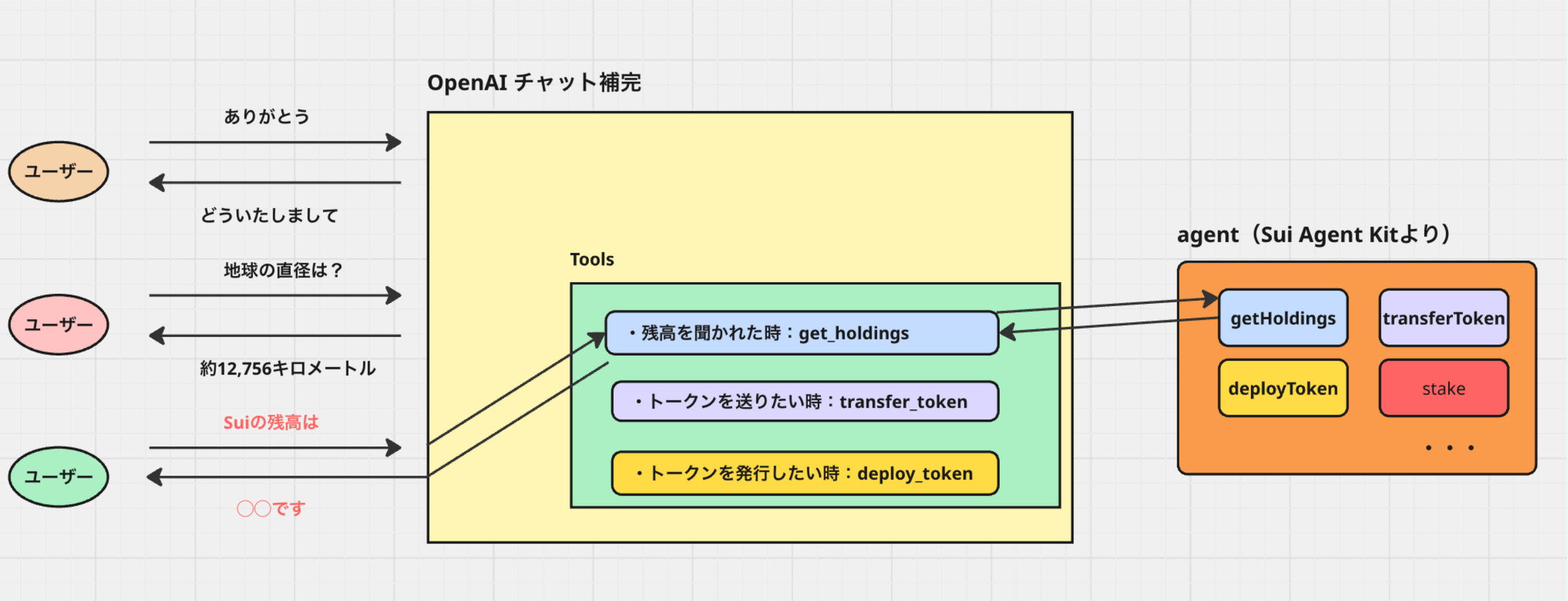
3 AIエージェント


OpenAI チャット補完
ありがとう
ユーザー
どういたしまして
地球の直径は？
ユーザー
約12,756キロメートル
Suiの残高は
ユーザー
わかりません

3 AIエージェント


OpenAI チャット補完
ありがとう
ユーザー
どういたしまして
地球の直径は？
ユーザー
約12,756キロメートル
Suiの残高は
ユーザー
Tools
・残高を聞かれた時：get_holdings
・トークンを送りたい時：transfer_token
・トークンを発行したい時：deploy_token

3 AIエージェント


OpenAI チャット補完
ありがとう
ユーザー
どういたしまして
地球の直径は？
ユーザー
約12,756キロメートル
Suiの残高は
ユーザー
〇〇です
Tools
・残高を聞かれた時：get_holdings
・トークンを送りたい時：transfer_token
・トークンを発行したい時：deploy_token
agent（Sui Agent Kitより）
getHoldings
transferToken
deployToken
stake
・・・

3 AIエージェント

まずはreadlineを用いて、おうむ返しするコードを書いてみましょう。
npm init -yのあと、index.jsでファイルを作成しましょう。

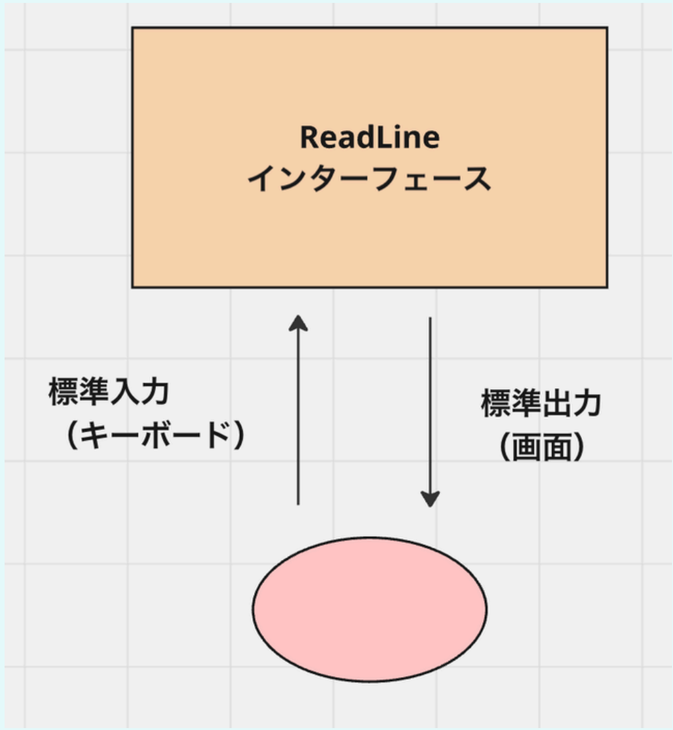
index.js > main

```
1  import * as readline from "readline";
2
3  async function main() {
4    const rl = readline.createInterface({
5      input: process.stdin,
6      output: process.stdout,
7    });
8
9    while (true) {
10     const input = await new Promise((resolve) => {
11       rl.question("\nPrompt: ", resolve);
12     });
13     console.log(`${input}!!`);
14   }
15 }
16
17 main();
18
```

PROBLEMS OUTPUT DEBUG CONSOLE

TERMINAL node

```
(base) ytakahashi@yuki sui_agent % node index.js

Prompt: Hello
Hello!!
```



https://github.com/ytakahashi2020/sui_agent

3 AIエージェント

dotenvを利用します。
npm i dotenv

index.js > main

```
1  import * as readline from "readline";
2  import * as dotenv from "dotenv";
3
4  dotenv.config();
5
6  async function main() {
7    const rl = readline.createInterface({
8      input: process.stdin,
9      output: process.stdout,
10   });
11
12   while (true) {
13     const input = await new Promise((resolve) => {
14       rl.question("\nPrompt: ", resolve);
15     });
16     console.log(`${input}!!`);
17   }
18 }
```

ytakahashi2020 Update .env_example

Code | Blame | 3 lines (3 loc) · 77 Bytes

```
1  OPENAI_API_KEY=
2  SUI_PRIVATE_KEY=
3  RPC_URL=https://fullnode.testnet.sui.io:443
```

https://github.com/ytakahashi2020/sui_agent/blob/main/.env_example

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

3 AIエージェント

openAIのインスタンスを作成します。
npm i openai

~/Desktop/sui_event/agent/index.js

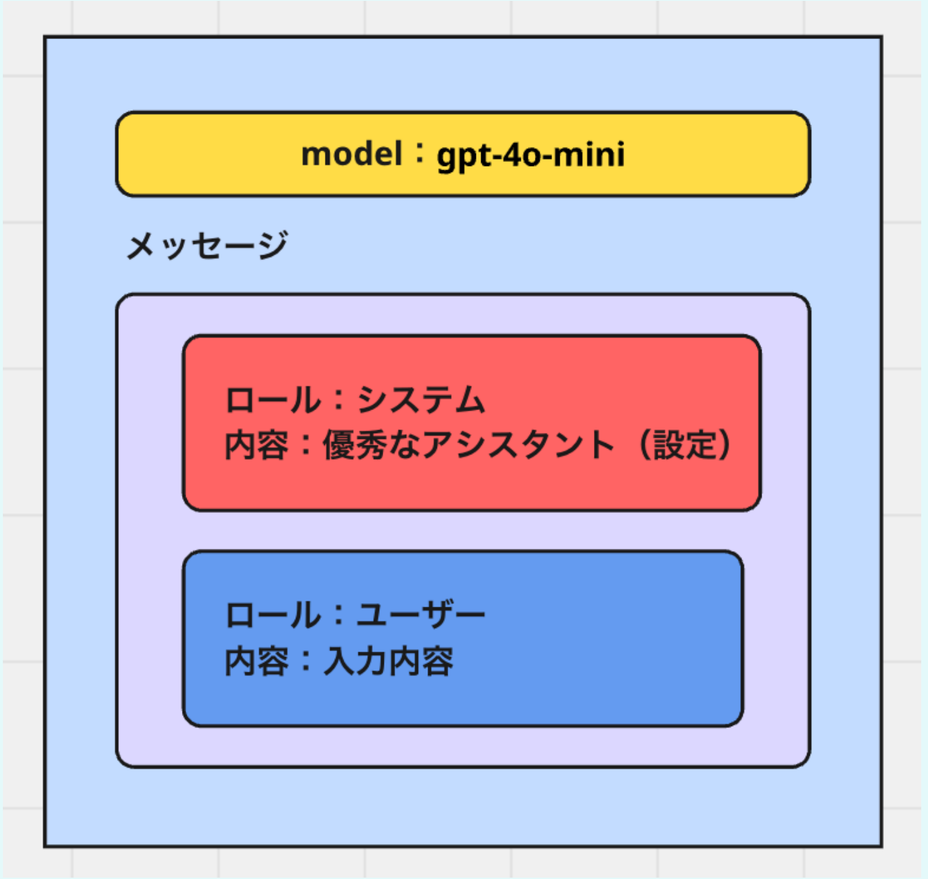
```
1  import * as readline from "readline";
2  import * as dotenv from "dotenv";
3  import { OpenAI } from "openai/client.js";
4
5  dotenv.config();
6
7  const openai = new OpenAI({
8    apiKey: process.env.OpenAI_API_KEY,
9  });
10
11 const f_name = "get_holdings";
12 const tools = [
```

3 AIエージェント

inputを元にチャットを作りましょう。

```
index.js > ...
11  async function main() {
17    while (true) {
18      const input = await new Promise((resolve) => {
19        rl.question("\nPrompt: ", resolve);
20      });
21      const response = await openai.chat.completions.create({
22        model: "gpt-4o-mini",
23        messages: [
24          {
25            role: "system",
26            content: "You are helpful assistant",
27          },
28          {
29            role: "user",
30            content: input,
31          },
32        ],
33      });
34      console.log(response.choices[0].message.content);
35    }
36  }
37
38  main();
39
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE
TERMINAL
(base) ytakahashi@yuki sui_agent % node index.js

Prompt: hi
Hello! How can I assist you today?

Prompt:
```



3 AIエージェント

inputを元にチャットを作りましょう。

```
Prompt: hello
{
  id: 'chatcmpl-CHky0topf4M4Xktbnw9qzWXyqZqnf',
  object: 'chat.completion',
  created: 1758348152,
  model: 'gpt-4o-mini-2024-07-18',
  choices: [
    {
      index: 0,
      message: [Object],
      logprobs: null,
      finish_reason: 'stop'
    }
  ],
  usage: {
    prompt_tokens: 37,
    completion_tokens: 9,
    total_tokens: 46,
    prompt_tokens_details: { cached_tokens: 0, audio_tokens: 0 },
    completion_tokens_details: {
      reasoning_tokens: 0,
      audio_tokens: 0,
      accepted_prediction_tokens: 0,
      rejected_prediction_tokens: 0
    }
```

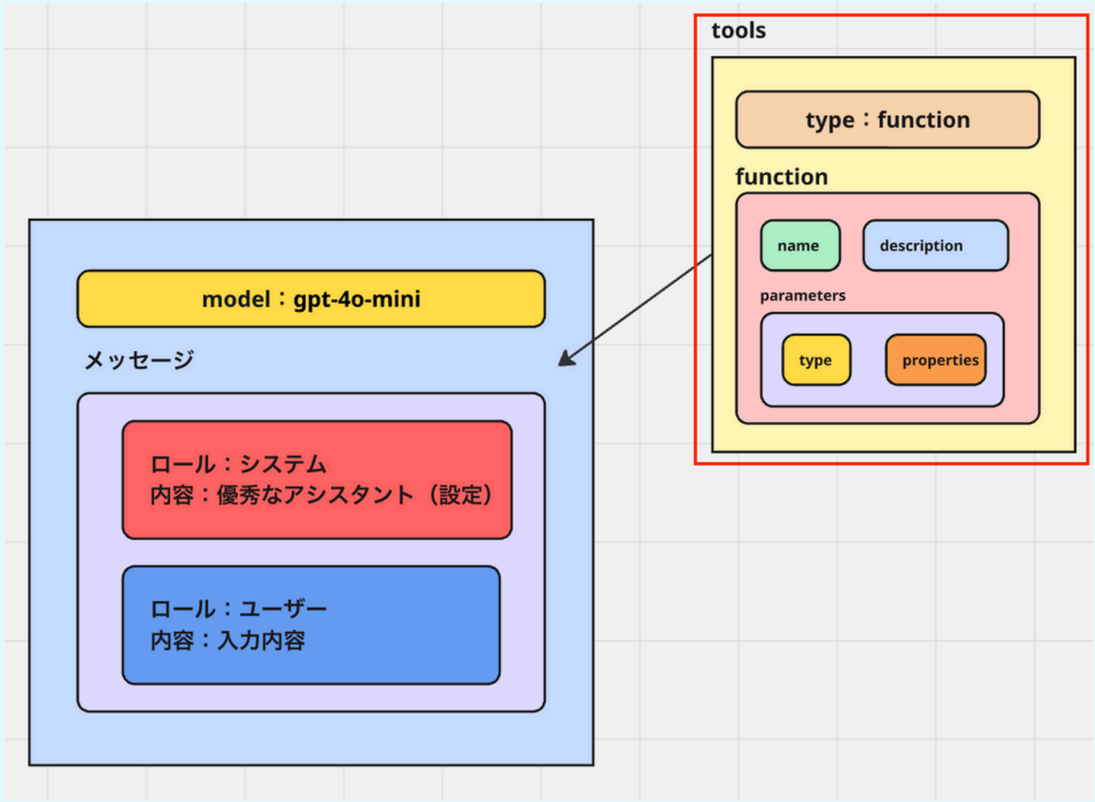
```
Prompt: hello
{
  index: 0,
  message: {
    role: 'assistant',
    content: 'Hello! How can I assist you today?',
    refusal: null,
    annotations: []
  },
  logprobs: null,
  finish_reason: 'stop'
}
```

3 AIエージェント

inputを元にチャットを作りましょう。

index.js > main > response

```
const tools = [
  {
    type: "function",
    function: {
      name: "get_holdings",
      description: "Get all token balances in the wallet",
      parameters: {
        type: "object",
        properties: {},
      },
    },
  },
];

async function main() {
  const rl = readline.createInterface({…
  });

  while (true) {
    const input = await new Promise((resolve) => {
      rl.question("\nPrompt: ", resolve);
    });
    const response = await openai.chat.completions.create({
      model: "gpt-4o-mini",
      messages: [
        {
          role: "system",
          content: "You are helpful assistant",
        },
        {
          role: "user",
          content: input,
        },
      ],
      tools: tools,
      tool_choice: "auto",
    });
```



3 AIエージェント

SuiAgentKitを作成します。
npm i @getnimbus/sui-agent-kit

```
import OpenAI from "openai";
import { SuiAgentKit } from "@getnimbus/sui-agent-kit";

dotenv.config();

const openai = new OpenAI({
  apiKey: process.env.OPENAI_API_KEY,
});

const tools = [
  {
    type: "function",
    function: {
      name: "get_holdings",
      description: "Get all token balances in the wallet",
      parameters: {
        type: "object",
        properties: {},
      },
    },
  },
];

const agent = new SuiAgentKit(
  process.env.SUI_PRIVATE_KEY,
  process.env.RPC_URL,
  {
    OPENAI_API_KEY: process.env.OPENAI_API_KEY,
  }
);

async function main() {
```

Token Operations

Check Holding Asset

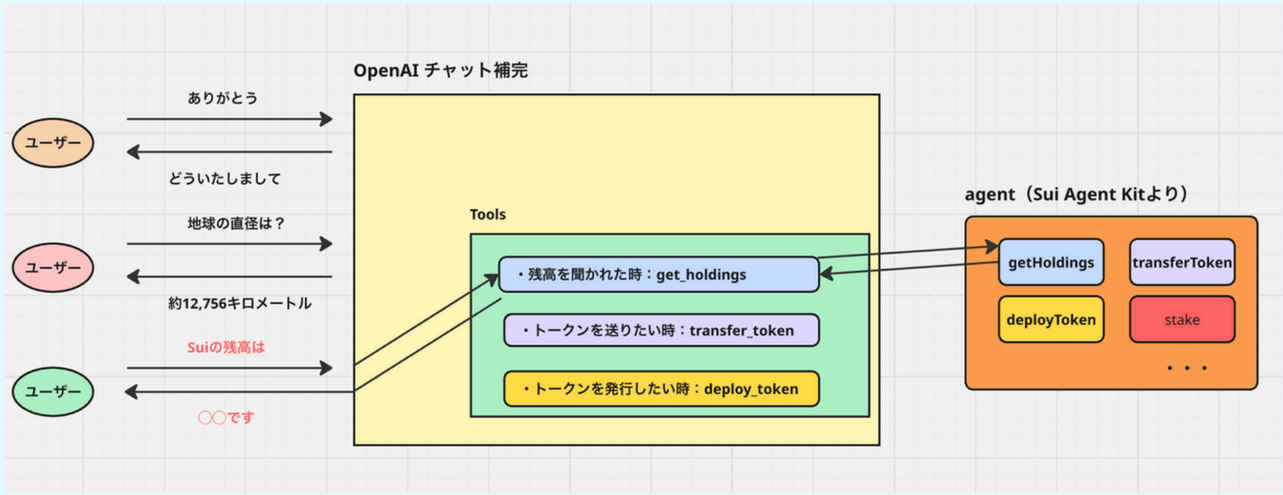
Learn how to check your wallet

The toolkit provides method:

- **getWalletHolding**: Check balances your own wallet

Usage

```
// Check your wallet's asset
const balance = await agent.getWalletHolding();
```



3 AIエージェント

最後に分岐させましょう。