Move入門

目標

- ✓堅牢な基盤と設計パターンに基づいて
  Sui上にスマートコントラクトを構築する
- ✓Move言語とSui特有のオブジェクト中心モデルを学ぶ

アジェンダ

1. Moveとは？
2. ツールチェーンと環境構築
3. 変数・データ型・可変性
4. Suiのスマートコントラクト設計パターン
5. SuiにおけるCapability（権限）
6. エラー処理とセキュリティ実践
7. **ハンズオン（今回は既存コードをデプロイ & CLIでMint）**

学習目標（再確認）

- ✓Moveの**安全性**とSuiの**オブジェクトモデル**の勘所を掴む
- ✓既存リポの**Moveパッケージを自分でpublish**してみる
- ✓**CLIでNFTをmint**して確認できる

> 次回：**自作のNFTコントラクト**を作る体験をする

Moveとは？

- ✓**安全**：コピー/破棄の可否を型（Abilities）で制御
- ✓**リソース指向**：資産の二重発行や紛失を型で抑制
- ✓**Sui**：リソースを**オブジェクト（UID付き）**として保存

ツールチェーンと環境構築（概要）

- ✓Sui CLI / Sui Wallet
- ✓Testnet切替 & Faucet
- ✓ `sui move build / test` でローカル検証
- ✓エクスプローラ：Sui Vision（Testnet）

ツールチェーン（よく使うCLI）

```
sui --version
sui client envs
sui client switch --env testnet
sui client active-address
sui client faucet
sui keytool import "<your 12-24 word mnemonic here>" ed25519
```

変数・データ型・可変性①：基本の型とリテラル

- ✓**整数**：`u8, u16, u32, u64, u128, u256`
- ✓**真偽**：`bool`（`true/false`）
- ✓**アドレス**：`address`（例 `@0x42`）
- ✓**配列**：`vector<T>`（長さ可変・0始まり）

```
let n: u64 = 42;
let ok: bool = true;
let who: address = @0x42;
let bytes: vector<u8> = vector[1, 2, 3];
```

> 文字列型は**無い**：文字列は `vector<u8>` で表現するか、標準ライブラリ `std::string` を使用する。

変数・データ型・可変性②：可変性と参照

- ✓ `let` は**不変**、`let mut` で**可変**
- ✓値を直接持つ以外に、**参照**も使える
  - ✓読み取り：`&T` / 変更：`&mut T`

```
let mut count: u64 = 0;
count = count + 1; // OK（可変）

fun bump(x: &mut u64) { *x = *x + 1; }
bump(&mut count); // 参照で渡して更新
```

> 参照は**一時的に借りる**イメージ。所有権は元の変数が持ち続けます。

> `*` は参照先の値に触る合図（デリファレンス）。
> 右辺の `*x` は読む、左辺の `*x = …` は書く。

変数・データ型・可変性③：文字列と“よくある落とし穴”

- ✓文字列は `vector<u8>`（UTF-8推奨）で表現
  - ✓CLIの `--args "hello"` は、期待型が `vector<u8>` なら自動変換される
- ✓**落とし穴**
  - ✓ベクタは可変：サイズ無制限に増やさない（ガス対策）
  - ✓数値の範囲外はエラー：`u8` など**型に合う範囲**を使う
  - ✓参照の寿命：借用中の同時書き換えは不可（安全のため）

```
// 例：文字列（UTF-8）を受け取るAPI
public fun set_name(name: vector<u8>) { /* ... */ }
```

Suiのスマートコントラクト設計①

オブジェクト設計の最初の一歩

- ✓**オブジェクト = UID + データ + 所有者**
- ✓まずは **単一所有** で設計（個人のNFT/プロフィール等）
- ✓**共有** は「複数人が更新する必要」がある時のみ
- ✓**不変共有** は読み取り専用の配布に最適（コスト安）

> 迷ったら**単一所有**。共有は同期やコストが増えるため慎重に。

Suiのスマートコントラクト設計②

`entry` とAPI境界

- ✓ `entry` = **トランザクション入口**：外部から直接呼ばれる場所
- ✓**小さく、薄く** 作る（入力検証＋内部関数呼び出しに専念）
- ✓内部ロジックは `public` / `public(package)` 関数へ分離
- ✓将来の変更に備えて、**公開APIを最小限に維持**

```
entry fun create(ctx: &mut TxContext) { internal_create(ctx); }
fun internal_create(ctx: &mut TxContext) { /* 実処理 */ }
```

> 入口が少ないほど、アップグレードや監査が楽になります。

Suiのスマートコントラクト設計③

状態更新・ガス・テストの基礎

- ✓**上限を決める**：ループ回数・ `vector` 長さ・文字列サイズ
- ✓**イベント/可観測性**：重要な変更はイベントで可視化（必要に応じて
- ✓**テスト**：正常系＋失敗（権限なし・上限超えなど）を**両方**用意
- ✓**誤りの早期発見**： `assert!(cond, E_CODE)` で速やかに中断

```
const E_TOO_LONG: u64 = 1;
assert!(vector::length(name) <= 64, E_TOO_LONG);
```

> 「安全に落ちる」設計がDoSや資産破壊のリスクを下げます。

Capability（権限トークン）の考え方

- ✓Capを持つ人だけが**ミント/設定**などを実行可
- ✓誤配布防止・保管戦略（マルチシグ等）が重要

エラー処理とセキュリティ（実践）

- ✓ `assert!(cond, E_CODE)` で**早期失敗**
- ✓共有オブジェクトは慎重に（競合/コスト）
- ✓ループ/ベクタに**上限**（ガスDoS回避）

ここからハンズオン（今回の方針）

✓**今回はNFTのコントラクトは作りません**

✓既存リポ：**SuiJapan/nft-mint-sample** のコードを**自分でpublish**

✓そのパッケージに **CLIから** `move_call` してNFTをmint

✓ `sui move new` は**別で**実行し、**ファイル構成の観察のみ**

0. 事前準備（共通）

```
# Testnetへ
sui client switch --env testnet

# アドレス&残高確認（必要ならFaucet）
sui client active-address
sui client faucet
```

> Testnet SUIが足りないとpublish/mintに失敗します

1-A. 「構成だけ見る」ための `sui move new`

> これは**デプロイしません**。構成を知るための見学用。

```
mkdir -p ~/tmp && cd ~/tmp
sui move new sample_pkg && tree -a sample_pkg
```

- ✓ `Move.toml` / `sources/` / `tests/` の最小構成を確認
- ✓用が済んだら削除OK

1-B. 今回使うリポの確認

```
# 既にクローン済みを前提
cd <あなたの>/nft-mint-sample
ls -1
# .devcontainer/ app/ contracts/ README.md ... が見えるはず

# Moveパッケージは contracts/ 以下
cd contracts
ls -1
# Move.toml / sources/ ...
```

> コントラクト本体は `contracts/sources/nft.move`

2. 既存コードをビルド

```
cd <あなたの>/nft-mint-sample/contracts

# ビルド（テストは任意）
sui move build
# エラーが出たら Move.toml の edition/依存を確認
```

3. 自分のアドレスで publish（Testnet）

```
cd <あなたの>/nft-mint-sample/contracts

sui client publish --gas-budget 100000000
```

✓出力JSONの `packageId` を控える（**今回の主役**）

✓以降の `--package` に**あなたの** `packageId` を使う

4. モジュール/関数の位置（復習）

- ✓モジュール名： `nft`
- ✓関数： `mint`（name, description, image_url を受け取り、送信者に転送）
- ✓呼び出しは `entry` 関数を **CLIの** `call` で行う

> 具体的なコードは `contracts/sources/nft.move` を参照

5. CLIでNFTをMint

```
sui client call \
  --package <あなたのPACKAGE_ID> \
  --module nft \
  --function mint \
  --args "SuiJapan NFT" "Hello from CLI" "https://example.com/image.png" \
  --gas-budget 100000000
```

✓3つの **文字列引数**： `name` / `description` / `image_url`

✓成功したら **Tx Digest / Created Object** が返る

6. ミント結果を確認

```
# NFTのオブジェクトIDから検索する
sui client objects {object_id}

# もしくはブラウザでエクスプローラを開いて
# Tx Digest / Object ID を検索（Testnet）
```

✓オブジェクト詳細で `name/description/image_url` を確認

✓ウォレットの **Objects** タブにも表示されます

（任意）フロントを自分のPackageに切替

> 後でアプリからも呼びたい人向け

```
cd <あなたの>/nft-mint-sample/app
cp .env.example .env
# .env の VITE_PACKAGE_ID を自分の packageId に置換
pnpm install
pnpm dev -- --host
```

トラブルシュート（よくある）

- ✓**InsufficientGas**：`--gas-budget` を増やす / Faucetで補充
- ✓**Function not found**：`--package`（誤ID）、`--module nft`、`--function mint` を再確認
- ✓**Different network**：envがtestnetか再確認（CLI/Wallet/Explorer）
- ✓**引数の型エラー**：3つとも**文字列**で渡す

まとめ（今回）

✓既存リポの **Moveパッケージを自分の名義でpublish**

✓**packageId** を使って **CLIからmint**

✓構成理解のために `sui move new` で骨格を確認（**デプロイはしない**）

“ 次回：**自作のNFTコントラクト**を作ってpublish & 呼び出し ”

参考リンク

- ✓Move Book：https://move-book.com/
- ✓Sui Dev Portal：<https://sui.io/developer
- ✓リポ（nft-mint-sample）：https://github.com/SuiJapan/nft-mint-sample
- ✓Sui Vision（Testnet）：https://suivision.xyz/（右上でTestnet選択）