スマートコントラクト開発環境構築

安野 裕貴 *

2023 年 11 月 21 日

概要

Hardhat と Ganache を用いたスマートコントラクトの開発環境の構築方法を説明する。まずスマートコントラクトの簡単な説明を行う。次に開発環境と利用したモジュールの紹介をする。利用したモジュールは Hardhat と ethers.js、VSCode、Ganache、Remix IDE である。Hardhat では実際に ERC721 を利用した NFT 発行プログラムを作成し、JavaScript でテストコードによるスマートコントラクトのテストを行った。Ganache ではスマートコントラクトのデプロイとフロントエンドからの呼び出しまでを行った。

目次

* 東京工科大学コンピュータサイエンス学部

1　序章

1.1　スマートコントラクトとは

スマートコントラクトとはブロックチェーン上で実施されるプログラムである。スマートコントラクトにより、第 3 者の仲介者がいない状態でも一連の契約を正しく履行できるようになる。スマートコントラクトはブロックチェーンに記載されるコンピュータプログラムのため改ざんをすることができない。またパブリックなブロックチェーンに記載されたスマートコントラクトのプログラムは誰でも参照することができるので透明性が高い。これにより中央集権的に管理されていたアプリケーションを分散型へ移行することが望まれている。

1.2　Hardhat と Ganache の比較

今回はスマートコントラクトの開発を行う Hardhat と Ganache の 2 つのツールについて紹介する。2 つのツールの違いは以下の表の通りである。また Hardhat の特徴としてコンパイル・デプロイ・テストまでの一

表 1　開発開発

	Hardhat	Ganache
UI	CUI	GUI
拡張機能	豊富	少ない
実行方法	テスト実行とテストネットの起動	テストネット起動

連の作業が自動で行われる。逆に Ganache は GUI で動作ができるので操作が理解しやすいといった特徴がある。

1.3　開発環境

開発環境は以下の通りである。

表 2　開発開発

ソフトウェア名	バージョン
OS	ubuntu20.04
node.js	18.16.1
npm	9.5.1
hardhat	2.17.3
@openzeppelin	4.9.3
Solidity	0.8.19
Ganache	2.7.1

ディレクトリは以下の通りである。

```
project
 |−package.json          // インストールされるモジュールや設定が記載
 |−package−lock.json     // インストールされるモジュールの詳細が記載
 |−hardhat.config.json   // hardhatの設定が記載されている
 |−artifacts             // コンパイル後の出力結果が生成される
 | |−contracts
 | |−build−info
 | └−@openzeppelin
 |−contracts             // スマートコントラクトのプログラムを作成する
 | └−Medal.sol
 |−scripts               // スクリプトを作成する
 | └−deploy.js
 └test                    // テストコードを作成する
   └−Medal.js
```

2　Hardhat を用いたスマートコントラクトの開発

2.1　主なモジュール

スマートコントラクト開発に利用されている Hardhat、ethers.js について紹介する。今回の環境構築ではスマートコントラクトで ERC721 を活用した NFT 発行プログラムの作成とコンパイル、JavaScript によるスマートコントラクトのデプロイと NFT 発行のテストを行う。

2.1.1　HardHat

Hardhat[1] とはローカルにブロックチェーンのテスト環境を作成することができる。Ethereum のブロックチェーン (EVM) の環境を開発できる。他のローカルにテストネット環境を構築する方法として Ganache

図 1　Hardhat

や Go-Ethereum などが挙げられる。主な特徴として以下の 4 つがある。

1. コマンドライン (CUI) で操作でき
2. 豊富なプラグインを設定ファイルで管理ができる
3. 初期化や再起動が容易であり開発を効率化できる
4. 直接テストネットにデプロイすることができる

HardHat は 3 つのツール群から構成される

1. Hardhat Runner：スマートコントラクトをコンパイルしてバイナリファイルに変換する
2. Harthat Ignition：スマートコントラクトをテストネットやメインネットにデプロイ
3. Hardhat Network：スマートコントラクトをローカルのネットワークを構築してデバッグすることができる

Hardhat は 2 つのモードで用いることができる。1 つ目はブロックチェーンを起動して、テストコードを元にスマートコントラクトをテストする方法である。2 つ目は HardHat でローカルのブロックチェーンとブロックチェーンへ接続するための Web サーバーを起動させる方法である。

2.1.2 VSCode

スマートコントラクトのプログラムの作成は VSCode を利用する。しかしスマートコントラクトの作成には Remix IDE や他のテキストエディタなども活用可能である。VSCode の拡張機能として図 n の「Solidity」を利用する。これによりプログラムの補間が行われる。

図 2　VSCode の Solidity 拡張機能

2.1.3 ethers.js

ethers.js[2] でスマートコントラクトのデプロイとテストコードの作成を行う。ethers.js は HardHat-ToolBox に内包されている。ethers.js とは JavaScript で Dapps のフロントエンド側のプログラムを作成する npm ライブラリの一種である。他のライブラリとして web3.js などがあるが ethers.js は後発のライブラリである。ethers.js は 4 つのモジュールによって構成されている。

1. ethers.provider：Ethereum のネットワークに接続を容易にすることができる
2. ethers.contract：Ethereum のスマートコントラクトの呼び出しやイベントの受け取り、デプロイなどを行う
3. ethers.utils：アドレスや Eth(Ethereum 内の通貨) の単位を変換したりする
4. ethers.wallets：ウォレットとの接続を容易にすることができる

2.2 スマートコントラクトの作成とテスト

2.2.1 HardHat のセットアップ

まず npm の初期化を行いプロジェクトの作成を行う。package.json などの npm ライブラリの管理を行うファイルを作成する。

```
$ npm init
```

次に HardHat の npm ライブラリのインストールを行う。

```
$ npm install -D hardhat
```

プロジェクトの新規作成を行う。

```
$ npx hardhat
```

以下の 3 つの選択肢から Create JavaScirpt Project を選択。

- Create a JavaScript Project：JavaScript を利用したプロジェクトファイルを作成する
- Create a TypeScript Project：TypeScript を利用したプロジェクトファイルを作成する
- Create an empty hardhat.config.js：hardhat.config.js のみを作成する

以下の設定ファイル (hardhat.config.js) が生成される。

Listing 1　hardhat.config.js

```
require("@nomicfoundation/hardhat-toolbox");

/** @type import('hardhat/config').HardhatUserConfig */
module.exports = {
  solidity: "0.8.19",
};
```

2.2.2　スマートコントラクトの作成

ERC721 を活用して NFT を発行するスマートコントラクトの作成を行う。全体の流れは以下の通りである。

1. モジュールのインストール
2. プログラムの作成
3. コンパイル

まず openzeppelin の以下のコマンドで ERC721 を含むモジュールのインストールを行う。

```
$ npm install @openzeppelin/contracts
```

次に contracts ディレクトリ内に solidity のプログラムを作成する。

Listing 2　Medal.sol

```
// SPDX-License-Identifier: MIT
pragma solidity >=0.8.0 <0.9.0;

import "@openzeppelin/contracts/token/ERC721/extensions/ERC721URIStorage.sol";
import "@openzeppelin/contracts/access/Ownable.sol";

contract Medal is ERC721URIStorage, Ownable {

    constructor() ERC721("Medal", "TamaMedal") {}

```

```
11      mapping(uint256 => uint256) private level;
12      mapping(uint256 => address) private minterAddress;
13      mapping(uint256 => string)  private minterName;
14      mapping(uint256 => string) private ownerName;
15
16      /*
17        * @titlle メダルミント
18        * @notice NFTを発行するメソッド
19        * @param id 発行するメダル番号
20        * @param name 発行者の名前
21      */
22      function mint(uint256 id, string calldata name) public returns (uint256) {
23          _mint(msg.sender, id);
24          _setTokenURI(id, "<外部URL>");
25
26          minterAddress[id] = msg.sender;
27          level[id] = 1;
28          ownerName[id] = name;
29
30          return id;
31      }
32
33      // メダル番号から現在のレベルを確認する関数
34      function getLevel(uint256 tokenId) public view returns (uint256) {
35          return level[tokenId];
36      }
37
38      // メダル番号から所持者のアドレスを確認する関数
39      function getMinterAddress(uint256 tokenId) public view returns (address) {
40          return minterAddress[tokenId];
41      }
42
43      // メダル番号から所持者の名前を確認する関数
44      function getMinterName(uint256 tokenId) public view returns (string memory) {
45          return minterName[tokenId];
46      }
47
48      // メダルの所有者を変更
49      function changeMinter(
50          uint256 tokenId,
```

```
51          address to,
52          string calldata name
53          ) public onlyOwner {
54
55          minterAddress[tokenId] = to;
56          minterName[tokenId] = name;
57          level[tokenId] += 1;
58      }
59  }
```

最後に Hardhat を用いてプログラムをコンパイルする。コンパイル後は artifacts のディレクトリ内に ABI ファイルと JSON ファイルが生成される。ABI にはプログラムのインターフェースが定義されており、フロントエンドからスマートコントラクトを呼び出す際に利用される。JSON ファイルにはコンパイル後のバイナリファイルが記載されている。以下がコンパイルのコマンドである。

```
$ npx hardhat compile
```

2.2.3 スマートコントラクトのテスト

スマートコントラクトが正しく利用できることを確認する。今回行うテストは以下である。

1. スマートコントラクトのデプロイ
2. NFT 発行
3. レベルの確認

テストコードは test ディレクトリ内に作成する。利用するモデルは JS でテストコードを作成する chai とスマートコントラクトを呼び出す ethers である。テストコードは以下の通りである。

Listing 3 Medal.js

```
1   const { expect } = require("chai");
2   const { ethers } = require("hardhat");
3
4   describe("medal test", () => {
5
6     it("mint and check level", async () => {
7       // スマートコントラクトのデプロイ
8       const Medal = await ethers.getContractFactory("Medal")
9       const medal = await Medal.deploy();
10
11      // メダルの発行
12      const mintId = 1;
13      await medal.mint(mintId, "test")
14
```

```
15          // 結果の検証
16          expect(await medal.getLevel(mintId)).to.equal(mintId);
17        })
18
19    })
```

テストは以下のコマンドで実行する。

```
$ npx hardhat test
```

実行結果は以下のようになる。

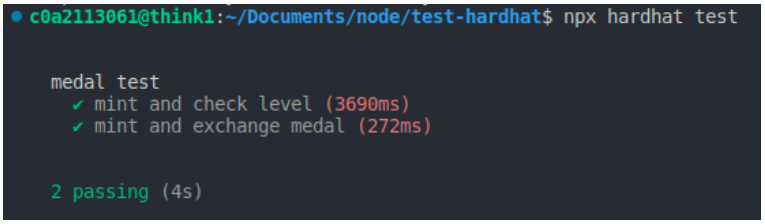

図 3　test コマンドの実行結果

3　Ganache を用いたスマートコントラクトの開発

3.1　主なモジュール

3.1.1　Ganache

Ganache とは Ethereum ブロックチェーンのテストネットをローカルで構築できるツールである。Ganache の特徴として 2 つが挙げられる。1 つ目は GUI で複数のアカウントやトランザクションの状態を確認することができる。確認できる内容は主に 3 つある。

1. アカウントのアドレスと秘密鍵
2. アカウントごとの残高
3. スマートコントラクトのコントラクトアドレス

2 つ目は Ganache ではワークスペース単位でブロックチェーンのネットワークが管理されることである。各ワークスペースに現在のネットワークの状態が保持され切り替えて利用することができる。Ganache を用いてテストネットの保存と ethers.js の接続を行う。

3.1.2　Remix IDE

Remix IDE とはスマートコントラクトの開発を行うためのツールが一つにまとまったツールである。

- フォルダ管理
- Solidity のソースコードをバイナリにコンパイル
- バイナリをネットワークにデプロイ
- スマートコントラクトの実行

図 4　Ganache

ブラウザで利用することができるので手軽である半面、スマートコントラクトを実行するネットワークは状態が引き継がれないため注意する必要がある。今回は Remix IDE と Ganache を接続することでテストネットのデータを保存する。

3.1.3　ethers.js

ethers.js でスマートコントラクトのデプロイとテストコードの作成を行う。ethers.js は HardHat-ToolBox に内包されている。ethers.js とは JavaScript で Dapps のフロントエンド側のプログラムを作成する npm ライブラリの一種である。他のライブラリとして web3.js などがあるが ethers.js は後発のライブラリである。ethers.js は 4 つのモジュールによって構成されている。

1. ethers.provider：Ethereum のネットワークに接続を容易にすることができる
2. ethers.contract：Ethereum のスマートコントラクトの呼び出しやイベントの受け取り、デプロイなどを行う
3. ethers.utils：アドレスや Eth(Ethereum 内の通貨) の単位を変換したりする
4. ethers.wallets：ウォレットとの接続を容易にすることができる

3.2　スマートコントラクトの開発

3.2.1　開発環境のセットアップとスマートコントラクトのデプロイ

開発環境のセットアップは以下の手順で行う。

1. Ganache のインストール
2. RemixIDE を Ganache に接続
3. フロントエンドのライブラリのインストール
4. スマートコントラクトのデプロイ

まず Ganache のインストールを `https://trufflesuite.com/ganache/` から行う。もし OS が Ubuntu の場合は権限を変更する。

```
$ chmod 764 ganache-2.7.1-linux-x86_64.AppImage
```

起動は以下のコマンドで行う。

```
$ ./ganache-2.7.1-linux-x86_64.AppImage
```

次に Remix IDE と Ganache の接続を行う。

まず `https://remix.ethereum.org/` にアクセスして Remix IDE を起動する。次に「deploy & Run Transaction」を左のメニューから選択する。ENVIROMENT から「Dev - Ganace provider」を選択する。

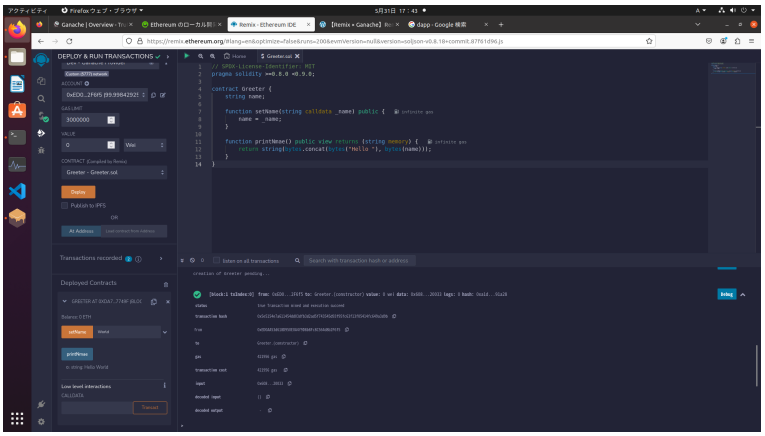

図 5　Remix IDE の起動

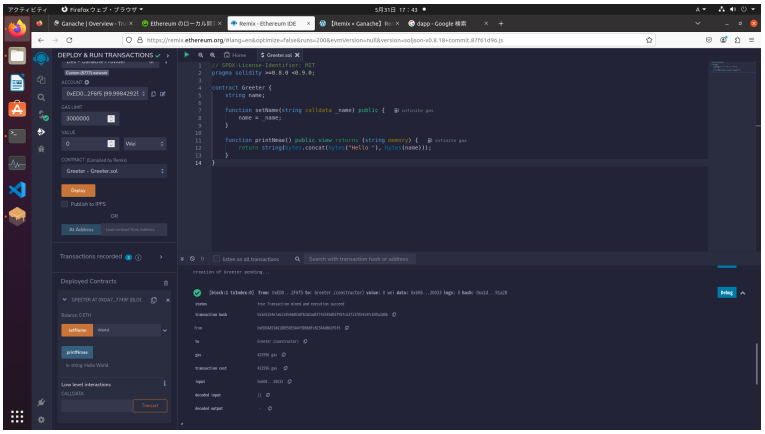

図 6　Remix IDE の起動

URL を入力する欄が表れるので `http://127.0.0.1:7545` を入力する。これで Remix IDE と Ganache の接続が完了する。

最後にフロントエンドのライブラリ (ethers.js) をインストールする。まず node.js のインストールする

```
$ sudo apt install nodejs
```

次に Vite + React.js の環境を用意する

```
$ npm init vite@latest
```

プロジェクト名とフレームワークとして React.js、言語のを選択する。ethers.js のインストールをする。

```
$ npm install ethers
```

最後に作成した Solidity のプログラムを Gnache のテストネットにデプロイする。まずスマートコントラクトのプログラムをコーディングする。次に左のメニューから「Solidity compiler」を選択する。コンパイラーのバージョンを 0.8.9(Solidity のソースコードで指定したバージョン) に揃えてから「Compile」ボタンをクリックしてコンパイルする。その後「ENVIROMENT」で Dev - Ganache が選択され、CONTRACT で自

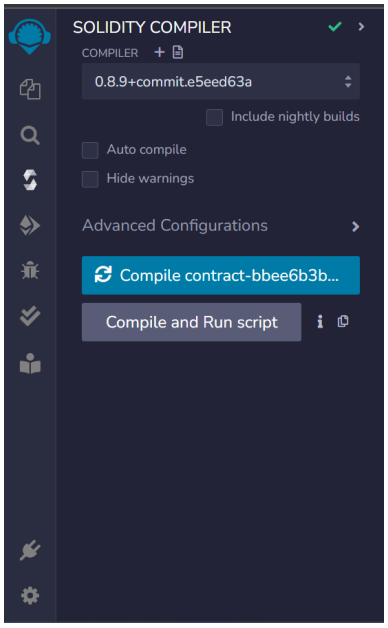

図 7　スマートコントラクトのコンパイル

身のコンパイルしたプログラムが選択されていることを確認してから「Deploy」ボタンをクリックする。これでスマートコントラクトのデプロイが完了する。

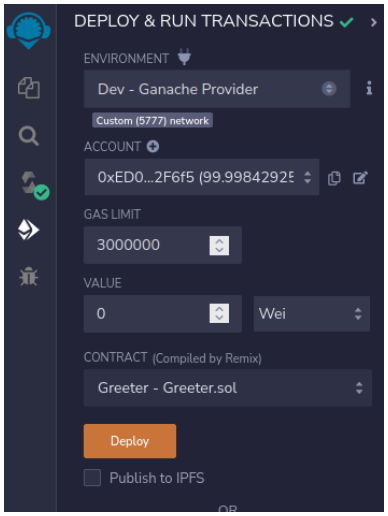

図 8　スマートコントラクトのデプロイ

3.2.2 スマートコントラクトの呼び出し

フロントエンドライブラリ ethers.js を用いてスマートコントラクトの呼び出しを行う。先ほど作成した src/App.tex をテキストエディタで開き以下のコードを追加する。

Listing 4 src/App.tex

```
1  import { ethersm Contract } from "ethers";
2  import { abi } from "./Abi.ts";
3
4  const URL = "http://127.0.0.1:7545"
5  const ADDRESS = "<自身のアドレスを入力する>";
6
7  function App() {
8    const [symbol, setSymbol] = useState("");
9
10   const getSymobol = async () => {
11     // ブロックチェーンとの接続
12     const provider = new ethers.JsonRpcProvider(URL);
13     // スマートコントラクトの初期化
14     const contract = new Contract(CONTRACT_ADDRESS, abi, provider);
15
16     // スマートコントラクトの呼び出し
17     const value = contract.symbol();
18     setSymbol(value);
19   }
20
21   useEffect(() , () => {
22     getSymobol();
23   }, [])
24
25   return(
26     <div>{symbol}</div>
27   )
28 }
```

ソースコードの COTRACT_ADDRESS 定数には Ganache にデプロイされたスマートコントラクトのアドレスを入力する。まず Ganache のメニューから「TRANSACTIONS」を選択し「CONTRACT CREATION」をクリックする。スマートコントラクトをデプロイする度に「CONTRACT CREATION」は増えるので注意する。

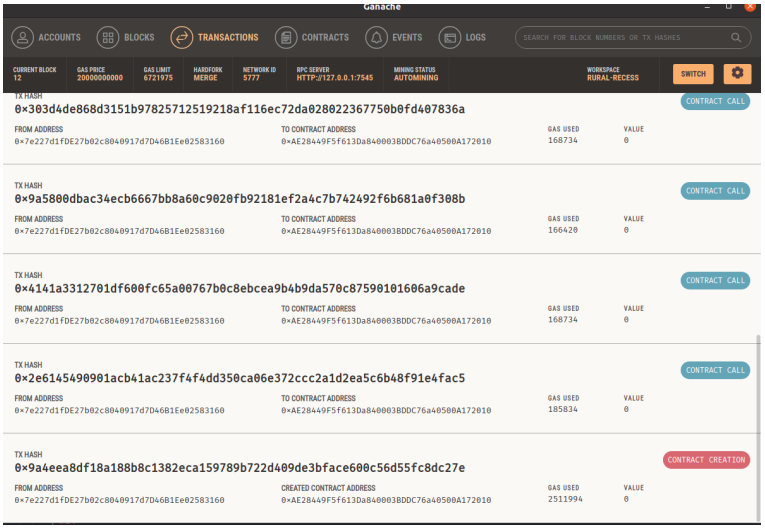

図 9　デプロイされたスマートコントラクトの確認

その後「CREATED CONTRACT ADDRESS」に記されているアドレスをコピーし、先ほどの App.tsx の COTRACT_ADDRESS 定数に入力する。

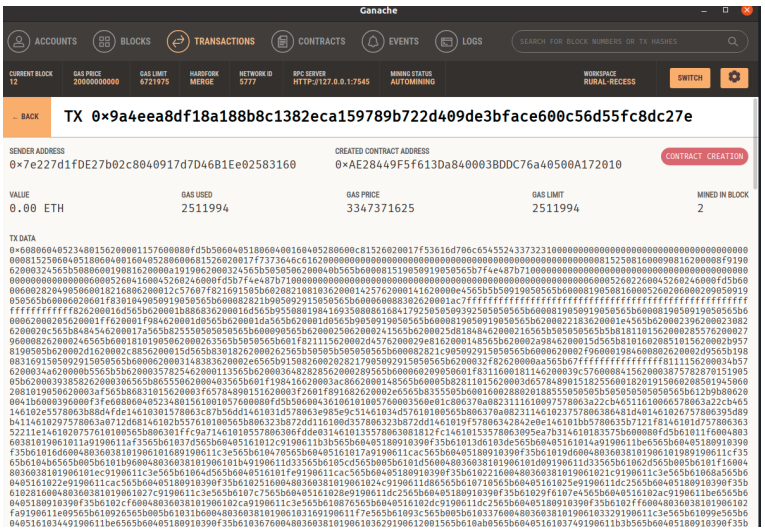

図 10　コントラクトアドレスの確認

これによりスマートコントラクトの呼び出しができるようになる。ブラウザで `http://localhost:5173` を確認すると NFT のシンボルが表示されていることが分かる。

ssdlab

図 11　結果 (symbol の値) の確認

参考文献

[1] NOMIC FUNDATION,「Hardhat」, https://hardhat.org/

[2] ethers.org,「Documentation」, https://docs.ethers.org/v6/

[3]「Hardhat の使い方: 初心者向けの Solidity 開発入門」, https://smacon.dev/posts/hardhat/

[4] Developers IO,「Hardhat で始めるスマートコントラクト開発」, https://dev.c