スマート・コントラクトのアップデート方法

Ethereum スマートコントラクト環境における
コントラクトのアップデート方法について

一般社団法人フィンテック研究振興協会

2018 年 5 月 30 日

スマート・コントラクトのアップデートについて

◆ アップデートできない問題への対策

イーサリアムの特徴として、一度デプロイしたコードが改竄されない点があります。
しかし、ストレージとロジックを分離することにより、コントラクトも一部アップデートが可能になります。

◆ アップデートのために使用する手法

アップデート可能なコントラクトを作成するために利用される手法は以下があります。

1. Interface
   抽象コントラクトを利用することで、参照先のコントラクトの関数内部を書き換える方法
2. Eternal Storage
   ロジックとデータストレージ用のコントラクトを分けることによって、ロジック部分をアップデート可能にする

Interface

抽象コントラクトを利用することで、関数のロジック部分を置き換えることができるようになります。
通常のコントラクトは以下のようになります。

```
contract SampleContract &amp;quot {
    function get() public returns (bytes32) { return "call get"; }
}
```

一方抽象コントラクトは以下のようになります。

```
contract SampleContract {
    function get() public returns (bytes32);
}
```

以下の例のように、関数の名前やアクセス制限、返り値と引数の型を定義することで、後から関数の中身を変更できるようになります。

```
contract test {
    SampleContract sample;
    constructor(address _felineAddr) public {
        sample = SampleContract(_felineAddr);
    }
    function get() public returns (bytes32) {
        sample.get();
    }
}
```

contract の中で get を利用することで、test contract 自体をアップデートすることなく、get 関数を持ったコントラクトをデプロイし、そのアドレスを設定することで関数の中身を更新することができます。

Eternal Storage

Ethernal Storage は、コントラクトのストレージ部分を切り出す手法です。
Sh3 で hash 化した key と value を保持できるストレージ contract を作成します。

```
contract EternalStorage {
    mapping(bytes32 => uint) UIntStorage;
    function getUIntValue(bytes32 record) public constant returns (uint) {
        return UIntStorage[record];
    }
    function setUIntValue(bytes32 record, uint value) public {
        UIntStorage[record] = value;
    }

    mapping(bytes32 => bytes32) Bytes32Storage;
    function getBytes32Value(bytes32 record) public constant returns (bytes32) {
        return Bytes32Storage[record];
    }
    function setBytes32Value(bytes32 record, bytes32 value) public {
        Bytes32Storage[record] = value;
    }
}
```

このように state 変数と値をセットする external(又は public)アクセスの関数を定義することで、ストレージとして利用するコントラクトを用意します。
作成したコントラクトを以下のように、別コントラクトから呼び出すことで、ロジックとストレージを分離することができます。
ここでは、EternalStorage の Interface を利用することでストレージの関数を呼び出しています。

```
Import “EternalStorage.sol”;
contract Proposals{
    function getProposalCount(address _storageContract) public constant returns(uint256) {
        return EternalStorage(_storageContract).getUIntValue(keccak256("ProposalCount"));
    }
    function addProposal(address _storageContract, bytes32 _name) public {
        uint idx = getProposalCount(_storageContract);
        EternalStorage(_storageContract).setBytes32Value(keccak256("proposal_name", idx), _name);
        EternalStorage(_storageContract).setUIntValue(keccak256("proposal_eth", idx), 0);
        EternalStorage(_storageContract).setUIntValue(keccak256("ProposalCount"), idx + 1);
    }
}
```

このように、EternalStorage コントラクトに値をセットすることで、ロジック部分と、ストレージ部分を完全に分離することができます。