

# 実践:Ethereumを利用したアプリケーション開発

技術選択・アーキテクチャ・地雷

Masashi Salvador Mitsuzawa, Lead Engineer @ LayerX



## 自己紹介

**Masashi Salvador Mitsuzawa**  
(@masashisalvador)

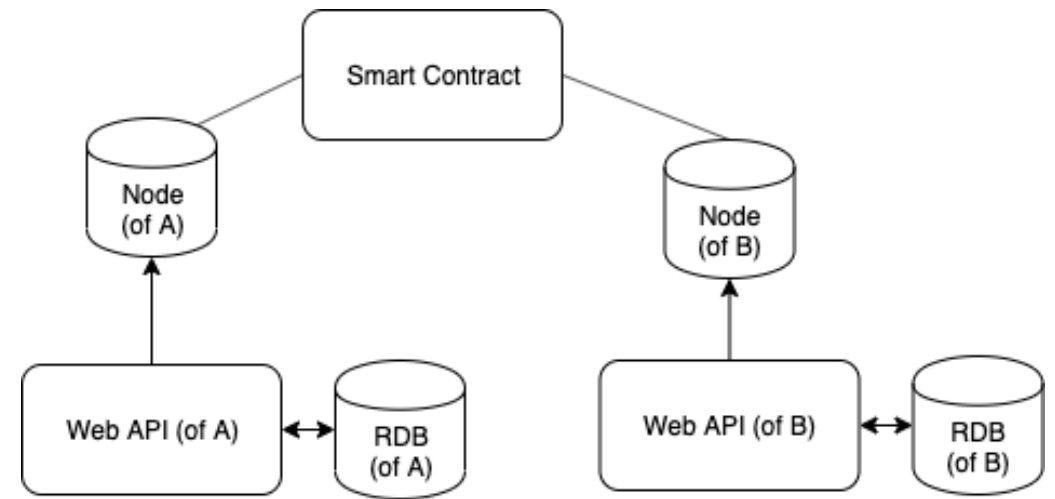
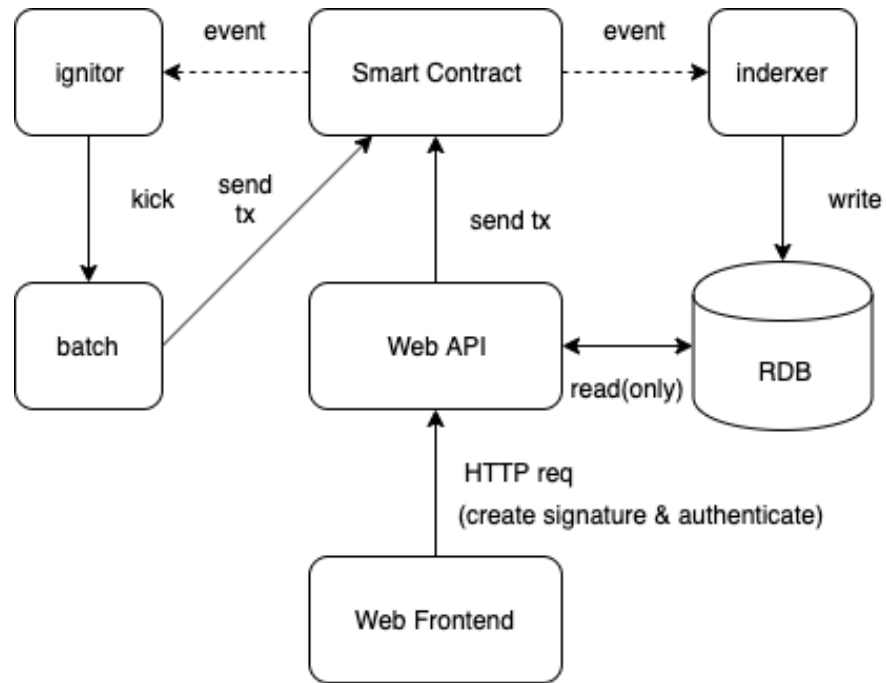
- Lead Engineer @ LayerX
- 好きな言語: Go, JavaScript, Haskell, Kotlin
- 軸足はバックエンドエンジニア
- 趣味: 山・茶道・計算論的神経科学 etc

# 話すこと

- 現実的なアーキテクチャ
  - Web API, indexDB, indexer(ignitor)
- 言語選択・実装・便利ツールチェーン
  - Go?
  - さよならweb3.js....
  - 様々な便利ツールたち
- 骨の折れる実装
  - 署名検証@コントラクト
  - メタトランザクション（的なもの）
  - etc...
- スマートコントラクト開発特有の地雷と対応策

# 現実的なアーキテクチャのために考えるべきこと

- エンドユーザがWalletを持っている仮定しない
  - サーバサイドで鍵を持つ
  - 鍵を使った認証機構
- EthereumのコントラクトはReadの機能が貧弱
  - 基本 key-value的にしか引けない
  - 一覧取得・検索・ページングetc
  - コントラクトをマスターとして別DBにindex
- 自動執行（？）＝イベント駆動
  - イベントは吐けるがバッチ的な機能はコントラクトにはない
  - イベントを見る -> 処理をkick



# 現実的なアーキテクチャと技術選択

- どこまで疎結合にするか
  - コンポーネントが多いと作るものが無駄に増える
- 何で作るか・何を使って作るか
  - Web API
  - Frontend
  - indexDB何にしよう？(RDB? KVS?)
  - indexer, ignitorどうしよう
  - ノードは何が良いだろう？コンセンサスアルゴリズムは？

**さしあたってWeb APIはGoで作ることに**

**@とあるプロジェクト**

# GoでEthereumのコントラクトを叩く

- Contractのbindingsを生成する or RPCリクエストを頑張って作る
- gethのツールチェーン `abigen`

```
truffle compile # truffleのアーティファクト生成
# jqでアーティファクトからabiだけ抽出
for json in `ls contracts/build/contracts/*.json`; do cat $json | jq .abi > $json.abi; done;
```

abiからbindingsを生成

```
abigen --abi /home/contracts/build/contracts/TxRelay.json.abi --pkg contract --type TxRelay --out /home/tx_relay.go mv tx_relay.go api/contracts/
```

適切にMakefileを書いて効率化します

```
make extract_abi
make abigen_all
```



# 生成されるbindings(読み出し/tx実行)

```
func (_SignValidator *SignValidatorCaller) Nonce(opts *bind.CallOpts, arg0 common.Address) (*big.Int, error) {  
    var (  
        ret0 = new(*big.Int)  
    )  
    out := ret0  
    err := _SignValidator.contract.Call(opts, out, "nonce", arg0)  
    return *ret0, err  
}  
  
/* 略 */  
func (_SignValidator *SignValidatorTransactor) ValidateSignature(opts *bind.TransactOpts, _sigV uint8, _sigR [32]byte, _sigS [32]byte, _data []byte, _originalSigner common.Address) (*types.Transaction, error) {  
    return _SignValidator.contract.Transact(opts, "validateSignature", _sigV, _sigR, _sigS, _data, _originalSigner)  
}
```

# 生成されるbindings(イベント監視)

```
// イベントを監視する関数
func (_TxRelay *TxRelayFilterer) WatchNonPayableTxRelayed(opts *bind.WatchOpts, sink chan<-
*TxDelayNonPayableTxRelayed) (event.Subscription, error) {

    logs, sub, err := _TxRelay.contract.WatchLogs(opts, "NonPayableTxRelayed")
    if err != nil {
        return nil, err
    }
    return event.NewSubscription(func(quit <-chan struct{}) error {
```

# いけそうに見えるが...

- buildパイプラインが若干長い
  - contract変更 -> compile -> abigen -> go build
- ライブラリとしてのgethは小回りが効かない
  - estimateGasとかですら結構やるの大変
- バイト列の扱いが各所で異なり辛い
  - "0x1234...." @ Frontend, JavaScript
  - [0x11, 0xab,...] @ Web API, Go[32]byte (prefixの0xは不要)
  - "0x1234...." @ Contract, Solidity
  - Front -> web api -> contractでやり取りする度に変換が必要 (つらすぎる)
- 何かやろうとするとgethを読まないといけない
  - 簡単なドキュメント：[Ethereum Development with Go](#)

**みなさんweb3.js使ってますか？**

**みなさんweb3.jsでハマったことないですか？**

# web3.jsのつらみ

- カジュアルに破壊される後方互換性
  - かゆい所に手が届かないドキュメント (& 突然の404)
- 普通にバグってて動かない(1.0.34 - 1.0.37くらい)
  - イベントのsubscribe周り
  - コントラクトデプロイ -> イベントが履かれているの確認 -> web3.js側が無反応 -> 原因探る -> 無限に時間が溶ける
- TypeScriptと相性が悪い...(型定義が微妙...)
- 他にもいろいろ...

web3.jsを捨てて😓ethers.jsを使うことに😍

# ethers.js

---

npm package 4.0.40

Complete Ethereum wallet implementation and utilities in JavaScript (and TypeScript).

## Features:

- Keep your private keys in your client, **safe** and sound
- Import and export **JSON wallets** (Geth, Parity and crowdsale)
- Import and export BIP 39 **mnemonic phrases** (12 word backup phrases) and **HD Wallets** (English, French, Italian, Japanese, Korean, Simplified Chinese, Spanish, Traditional Chinese)
- Meta-classes create JavaScript objects from any contract ABI, including **ABIV2** and **Human-Readable ABI**
- Connect to Ethereum nodes over [JSON-RPC](#), [INFURA](#), [Etherscan](#), or [MetaMask](#)
- **ENS names** are first-class citizens; they can be used anywhere an Ethereum addresses can be used
- **Tiny** (~84kb compressed; 270kb uncompressed)
- **Complete** functionality for all your Ethereum needs
- Extensive [documentation](#)
- Large collection of **test cases** which are maintained and added to
- Fully **TypeScript** ready, with definition files and full TypeScript source



```
import { ethers } from "ethers";  
// RLP(ないしABIの) デコーダ  
import { defaultAbiCoder } from 'ethers/utils/abi-coder';  
  
const provider = new ethers.providers.JsonRpcProvider(URL);  
// イベントの取得開始点を設定  
provider.resetEventsBlock(fromBlockNumber)  
  
// RLP -> jsonに変換するための対応関係を表したデータ構造  
const eventInterface = contract.interface.events[eventName]  
  
contract.on(filter, (...args) => {  
  /* 略 */  
  const eventRLPData = event.data // RLPエンコードされたイベントデータ  
  const decodedEvent = defaultAbiCoder.decode(eventInterface.inputs, eventRLPData)  
})
```



# ethers.jsの良いところ

- そもそもTypeScriptで開発されている&型定義がイケているの
- かゆい所に手が届くドキュメント
- やりたいことが自然にできる
  - eventを取ってきてRLP decodeしたい
  - データをsolidityの `keccak` と同じ方式でハッシュ化して署名したい
  - コントラクトの関数を動的に呼び替えたい
    - `eth.send(HogeContractFunction.Create, ...hoge.toArgs())`

## ethers.jsを使う前提でWeb APIもGo -> TypeScriptへ


- Ethereumは全体的にjsのライブラリが整っているので、NodeJSを使うのは悪い選択肢ではなさそう
  - Goさん頑張って欲しい
  - *Rust* ツカイタイナ～

## 色々なプロジェクトで採用しているstack

- WebAPI ... TypeScript(express) + TypeORM + ethers.js
- Frontend ... Nuxt.js
- Tools ... dbmate (migration), Ganache CLI, Truffle

## ethers.jsを使い倒すためのツール: buidler.dev

- `@nomiclabs/buidler`
- consoleからもdeployスクリプトからもethersを使いたい～
- ethers.jsが使ってTypeScriptフレンドリーなタスクランナー(Yet Another Truffle)
- コントラクトのエラーの詳細なstack trace出力
- configがTruffleに比べると柔軟
- `yarn add --dev @nomiclabs/buidler`
  - global installだとTypeScriptでconfigが書けない



```
module.exports = {
  /* 略 */
  buidlerevm: {
    gas: 0,
    gasPrice: 0,
    accounts: [ // 指定しないと20個のアカウントに10000EHがデフォルトで付与されて用意される
      {
        privateKey: "0xB3BB3A1EB4EBAAE568DB332251B77F5A029E44DB3C1CA9F39056B29C86ADC62C",
        balance: "0x" + `${55 * (10**10)}`,
      }
    ],
    blockGasLimit: 99 * (10**5),
    hardfork: "constantinople",
  },
};
```

```
1) Contract: ERC721
   like an ERC721
     transfers
       via safeTransferFrom
         to a contract that does not implement the required function
           reverts:
Error: Transaction reverted: function selector was not recognized and there's no fallback function
at ERC721Mock.<unrecognized-selector> (contracts/mocks/ERC721Mock.sol:9)
at ERC721Mock._checkOnERC721Received (contracts/token/ERC721/ERC721.sol:334)
at ERC721Mock._safeTransferFrom (contracts/token/ERC721/ERC721.sol:196)
at ERC721Mock.safeTransferFrom (contracts/token/ERC721/ERC721.sol:179)
at ERC721Mock.safeTransferFrom (contracts/token/ERC721/ERC721.sol:162)
at TruffleContract.safeTransferFrom (node_modules/@nomiclabs/truffle-
contract/lib/execute.js:157:24)
at Context.<anonymous> (test/token/ERC721/ERC721.behavior.js:262:30)
```

```
1) Contract: ERC721
   like an ERC721
     transfers
       via safeTransferFrom
         to a contract that does not implement the required function
           reverts:
Error: Returned error: VM Exception while processing transaction: revert
at PromiEvent (node_modules/truffle/build/webpack:/packages/contract/lib/promievent.js:9:1)
at TruffleContract.safeTransferFrom
(node_modules/truffle/build/webpack:/packages/contract/lib/execute.js:169:1)
at Context.<anonymous> (test/token/ERC721/ERC721.behavior.js:262:30)
```

## その他機能もいろいろ

- pluginでよしなに拡張できる
- builder consoleからethers.jsでノードとインタラクション

# その他便利ツール

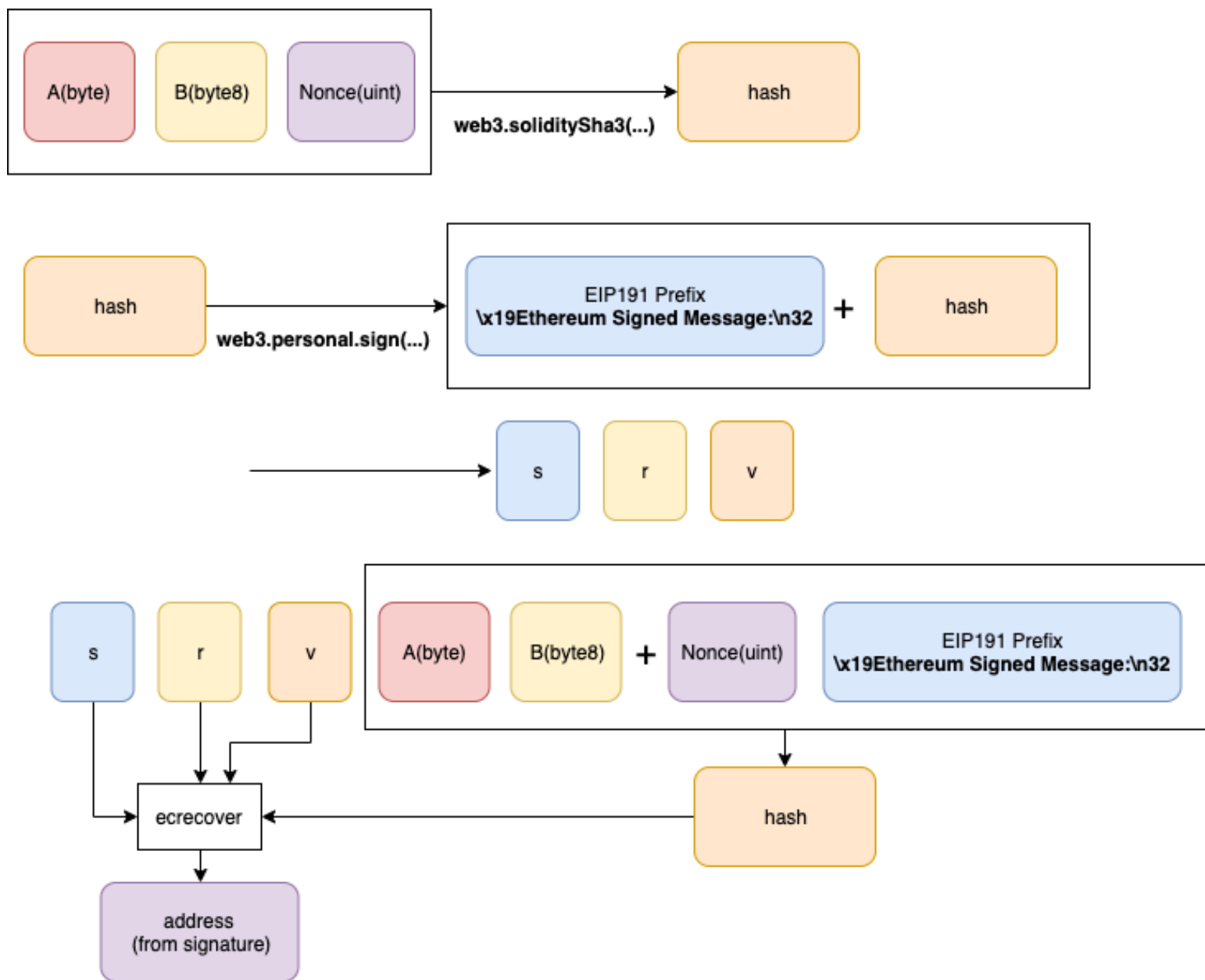
- indexer
  - Consensysの出してるやつ...
  - Eventeum (by Consensys)
  - the Graph
  - Eth.events
  - [https://scrapbox.io/layerx/Ethereum\\_Indexer比較](https://scrapbox.io/layerx/Ethereum_Indexer比較) に詳しい
- BaaS
  - Kaleidoの出来がくっそ良い
    - Zether(秘匿送金)
    - Block explorer
- static analysis tools
  - MythX
  - etc...

骨の折れる（楽しい）実装たち👻



# 署名検証を用いた認証

- 基本の流れ
  - サーバorコントラクトからメッセージを送る
  - メッセージを含んだハッシュ値に秘密鍵で署名
    - EIP191
    - EIP1271
  - サーバorコントラクトで検証(= ecrecover)
  - 実装前のお気持ち：「こんな一瞬やる」
  - 実際：結構大変
    - 少しでも入力が異なるとハッシュが変わる
    - Solidityのkeccakに対応するクライアント側の関数isどれ...??
    - `abi.encodePacked` なにやってるの ...
    - 型の指定が雑だとRLPエンコードの違いでハッシュが変わる





```
    async signForAuth(message: string, signValidatorAddress: string, nonce: number, signerAddr:
string): Promise<any> {
        const prefix = "0x19"
        const version = "0x00"
        const web3 = this.web3
        const hash = web3.utils.soliditySha3(
            { t: "bytes", v: prefix },
            { t: "bytes", v: version },
            { t: "address", v: signValidatorAddress },
            { t: "bytes", v: message },
            { t: "uint", v: nonce },
            { t: "address", v: signerAddr },
        )
        return await web3.eth.personal.sign(hash, signerAddr, "")
    }
```

```
contract SignValidator {
    using SafeMath for uint256;
    mapping(address => uint) public nonce;
    // ハッシュ値への署名をオンチェーン検証 (sigV, _sigR, _sigSと_dataは所与)
    function validateSignature(
        uint8 _sigV,
        bytes32 _sigR,
        bytes32 _sigS,
        bytes memory _data, // original message
        address _originalSigner
    ) public returns (address) {
        require(_originalSigner != address(0), "sender should not be zero address");
        // EIP191 compatible.
        // web3.personal.signに対応するメソッドで署名すると下記のprefixが生データについて署名される
        bytes memory sigPrefix = "\x19Ethereum Signed Message:\n32";
        bytes32 hashBody = keccak256(
            abi.encodePacked(
                byte(0x19), byte(0x00), address(this), _data, nonce[_originalSigner], _originalSigner
            )
        );
        bytes32 h = keccak256(abi.encodePacked(sigPrefix, hashBody));
        address addressFromSig = ecrecover(h, _sigV, _sigR, _sigS);
        require(addressFromSig == _originalSigner, "originalSender should sign");

        // 署名の使い回しを防ぐためにnonceを置いておく
        nonce[addressFromSig] = nonce[addressFromSig].add(1);
        return addressFromSig;
    }
}
```

## 参考

- MetaMaskからsignするときは `web3.personal.sign` でないと処理が止まります（安全性の観点から、任意のメッセージに署名できる `web3.eth.sign` に非対応）
- 署名検証 -> WebAPIへのアクセストークン発給とかやることが多そう
- コントラクトエンドで検証するにしろ、サーバサイドでやるにしろ `nonce` は必須
- `web3.soliditySha3` は型が指定できるので安心（型が違うとRLPエンコードが変わって死にます..）
- `ethers.js` の場合

```
let sig = ethers.utils.splitSignature(flatSig);
let recovered = await contract.verifyHash(messageHash, sig.v, sig.r, sig.s);
utils.solidityKeccak256(types, values)
utils.soliditySha256(types, values)
utils.solidityPack(types, values)
```

# メタトランザクション(#とは)

- Gas代を誰かに代りに払ってもらう
- 仕様は色々提案されている
  - ERC1776([m0t0k1ch1さんのブログ](#)が詳しい)
  - [uPortのやつ](#)
  - ERC865(ERC20でガスを払う)
  - ERC1077
- とあるプロジェクトでやりたかったこと
  - モノを売り買いさせる (ETH払い)
  - ガスはエンドユーザに払わせない
  - 署名だけをクライアントで行わせる
- できることを勘違いしたこともあり辛かった...

# 実装を始めたくらいのイメージ

1. txの雛形を作る=関数, 引数をRLPエンコードする
2. クライアントで署名（！？）
3. 一旦サーバ(Web API)に投げる
4. サーバ側でネットワークにブロードキャスト（署名する！？ & GASを払う）
  - 基本的には署名した人（アドレス）に該当するアカウントからガスが引かれるのでこの仕組みは無理...
  - = `msg.sender` に該当する人がかならずガスを払う

## 最終的な実装

1. クライアント側で実行したい関数と引数を指定アクセストークンと一緒にサーバへ投げつける
2. サーバ側は誰が実行元(originalな署名者)かを明記して `tx` を作ってネットワークに送信、コントラクトでexternal callでtx実行



```
API async createHoge(listing: Listing) {  
    const sender = await this.web3Wrapper.getDefaultAccount()  
    // txProxyAPI = コントラクト名と引数を渡すとRLP encodedなデータを返す  
    const rawTx = await this.txProxyAPI.issueRawTx(  
        bookManager,  
        "createHoge",  
        sender,  
        listing  
    )  
    // 生データをどのコントラクトに投げるか指定  
    return this.sendNonPayableMetaTransaction(rawTx, bookManager)  
}
```



```
function executeNonPayableTx(
    uint8 _sigV,
    bytes32 _sigR,
    bytes32 _sigS,
    address _destination,
    bytes memory _data, // original message
    address _originalSigner
) public {
    // meta-txで読んでいいコントラクトか検証
    require(destinationWhitelist[_destination], "the destination should be registered in
whitelist");

    bytes memory sigPrefix = "\x19Ethereum Signed Message:\n32";
    bytes32 hashBody = keccak256(abi.encodePacked(byte(0x19), byte(0x00), address(this),
_originalSigner, nonce[_originalSigner], _data));
    bytes32 h = keccak256(abi.encodePacked(sigPrefix, hashBody));
    address addressFromSig = ecrecover(h, _sigV, _sigR, _sigS);

    // 元の送信者がtxに署名しているか検証
    require(addressFromSig == _originalSigner, "the signer should be the original signer");
    // meta-txを使えるユーザかバリデーション
    require(registeredAddresses[_originalSigner], "original sender should be registered in
whitelist");

    // external call with data(data is abi encoded function call)
    nonce[addressFromSig] = nonce[addressFromSig].add(1);
    (bool result, ) = _destination.call(_data);

    emit NonPayableTxRelayed(result, _originalSigner, _destination, _data);
}
```

**勘のいいみなさまはお気づきでしょう**



```
function executeNonPayableTx(
    uint8 _sigV,
    bytes32 _sigR,
    bytes32 _sigS,
    address _destination,
    bytes memory _data, // original message
    address _originalSigner
) public {

function executePayableTx(
    uint8 _sigV,
    bytes32 _sigR,
    bytes32 _sigS,
    address _destination,
    bytes memory _data, // original message
    uint256 _value, // the original value signer wants to send
    address _originalSigner
) {
    /* 略 */
    userBalance[_originalSigner] = userBalance[_originalSigner].sub(_value);
    msg.sender.transfer(_value);

    (bool result, ) = _destination.call.value(msg.value)(_data);
```

閑話休題



Andreas M. Antonopoulos、  
Gavin Wood 著、宇野 雅晴、鳩  
貝 淳一郎 監訳、中城 元臣、落  
合 渉悟 技術監修、落合 庸介、  
小林 泰男、土屋 春樹、祢津 誠  
晃、平山 翔、三津澤 サルバドール  
将司、山口 和輝、宇野 雅晴、  
鳩貝 淳一郎 訳

448ページ

定価4,400円 (税込)

ISBN978-4-87311-896-3

**畏・そして畏**

# 事件簿

- Truffleバグってる事件
- ブロックチェーン調子悪い事件
  - 何故か開発ノードが動かなくなる
- Ganacheのautomineの罠
  - 偶然動いてた事件
- 環境変数複雑過ぎ問題
  - サーバとかノードとか多すぎ...
- EVM殿...
  - stack too deep
  - コントラクトサイズが上限超えてデプロイ不能にに
  - mappingはkeys取れない問題
- Solidityむずい
  - transferFromむずい