EthereumのAccount Abstraction

Ethereumのアドレスには、単独鍵から作るEOA(Externally Owned Address)とコントラクトアドレスの2種類がある。
https://ethereum.org/ja/developers/docs/accounts/

コントラクトから別のコントラクトを呼び出すということはできるが、最初のコントラクト呼び出しはEOAからしかできない。なのでユーザは例えばトークンを持っていて誰かに送りたい場合にネイティブトークンのETHからGas代を払わないとトークン送金するコントラクトが呼び出せない。代わりにトークンから支払うということができないのだ。
また個人の秘密鍵を使うのはどこかしらセキュリティ的に心配もあるので、スマートコントラクトで最初から扱えればよいのではないかという考えを持ったようだ。
アカウント抽象化 - ethereum.org

ただスマートコントラクト呼び出しをネイティブレベルで実行できるようにするのは大幅な改造が必要になるので、じゃあ現状でできるのはどこだろうかというのがERC-4337である。
https://eips.ethereum.org/EIPS/eip-4337

全然読んでいないので、OpenZeppelinを使ってERC-4337を使おうとしているという観点での見方をする。

UserOperation構造体を使ったコントラクト呼び出し

おそらくこれがポイントだと思う。
大きくは関係者として「ユーザー」「bundler」「paymaster」がいる。EntryPointは公式スマートコントラクトと思って良い。
Ethereumのコントラクトを呼び出すにはトランザクションを作らねばならず、そのGas代はコントラクトを展開したEOAが支払う。このルールは変えられないので、トランザクションの内容を作る「ユーザー」とトランザクションを展開する「bundler」という役割に分ける。
それだけだとbundlerが支払ったガス代を回収したい場合はどうするのかという問題がある。
ERC4337では例えばDEXがトークン交換のために送金されたのであればガス代は戻すよ、というようなことができるようにしている。このDEXに相当するのが「paymaster」である。
「paymaster」がいない場合はユーザーが支払う。

https://eips.ethereum.org/EIPS/eip-4337#bundler-behavior-upon-receiving-a-useroperation

flow

```
ユーザー → UserOperation作成
        ↓
bundler → mempool的に収集
        ↓
bundler → handleOps() を送信（ガス前払い）
        ↓
EntryPoint → 各UserOpを検証・実行
        ↓
EntryPoint → ガス精算
        ↓
bundler → ガス代を回収
```

ガス代の回収は、EntryPointへ行ったデポジットから行われる。単なるEntryPointのアドレスにETHを払うのではなく、コントラクト呼び出しで「このアカウントに対してデポジットする」という形で行っておけば良い。

depositTo

```
function depositTo(address account) external payable;

# 実際の呼び出しはこういう感じになる。
entryPoint.depositTo{value: 1 ether}(target);
```

`PackedUserOperation` にある `sender` はAccountコントラクトのアドレスが入る。bundlerではない。

https://github.com/eth-infinitism/account-abstraction/blob/v0.8.0/contracts/interfaces/PackedUserOperation.sol

PackedUserOperation v0.8.0

```
struct PackedUserOperation {
    address sender; // ★Accountコントラクトのアドレス
    uint256 nonce;
    bytes initCode;
    bytes callData;
    bytes32 accountGasLimits;
    uint256 preVerificationGas;
    bytes32 gasFees;
    bytes paymasterAndData;
```

```
        bytes signature;
    }
```

EntryPoint

MultiSigで参照しているERC-4337は EntryPoint という考え方があり、特定のアドレスに公式実装がデプロイされるようである。
これは固定のコントラクトアドレスとしてハードコーディングされて OpenZeppelin などでも `constant` 値として参照でき、そのアドレス経由でアクセスするのが望ましい。
バージョン管理もされていて、例えばopenzeppelin-contract v5.5.0では `ENTRYPOINT_V08` でopenzeppelin-contract v5.6.0では `ENTRYPOINT_V09` になっている。

- v0.9.0： `0x433709009B8330FDa32311DF1C2AFA402eD8D009`
- v0.8.0： `0x4337084d9e255ff0702461cf8895ce9e3b5ff108`

デプロイ元はeth-infinitismらしい。バージョンはタグ管理されていて、例えば `struct PackedUserOperation` もv0.8版とv0.9版がある(同じ構造だが)。

EntryPointのコントラクトはどこかの公式なるところが行うため、anvilなどのローカルチェーンにはそれがない。 `anvil --fork-url` で別のチェーンからコピーしてもよいが、anvil のデフォルトアカウントなどが使用できなくなって面倒なので自分でデプロイしたほうが良いだろう。
EntryPointのコントラクトアドレスは `CREATE2` を使うためmainnetなどと同じアドレスになる。 `--network <net>` はデプロイするネットワークらしいが `127.0.0.1` や `127.0.0.1:8545` のようにIPアドレスを指定するものではなさそうだった。 `yarn deploy` が `hardhat deploy` コマンドの薄いラッパーなのでHardHat方面から調べるとなにかわかるかもしれない。

```
deploy
    git clone https://github.com/eth-infinitism/account-abstraction.git
    cd account-abstraction
    # デプロイしたいバージョンをcheckout
    yarn install
    yarn deploy --network localhost
```

EntryPointはバージョンごとにデプロイされていると書いたが、OpenZeppelinは比較的最新バージョンに追従していてv5.5.0はEntryPoint v0.8.0 で v5.6.0は EntryPoint v0.9.0になっている。
しかしRustライブラリのalloyはそうでもなく、今の最新版v1.8.3ではv0.6.0とv0.7.0のstruct定義しかない。といってもalloyが積極的にEntryPointをサポートしているわけではないので自分で構造体を定義するのがよいだろう。

EntryPointのUserOperation実行

`PackedUserOperation` を通してEntryPoint経由でしか実行できないコントラクトを呼び出すために以下の3つの関数を使うことになるだろう。

`handleOps`

```
function getNonce(address sender, uint192 key) external view returns (uint256
nonce);
function getUserOpHash(PackedUserOperation calldata userOp) external view returns
(bytes32);
function handleOps(PackedUserOperation[] calldata ops, address payable beneficiary)
external;
```

なお、いまのところ(2026/04/13) EntryPoint v0.8.0 でしか成功していない。ほぼ同じことをやっているのだが v0.9.0ではrevertしてしまうのだ。revertのメッセージはv0.8.0とは異なるようで同じ感覚でデバッグしたためうまくいっていないのかもしれない。

中心となるアカウントは `sender` である。

`getNonce()` や `PackedUserOperation` の `sender` がEntryPointを使うというイメージだろう。nonceは `handleOps()` が成功するとインクリメントされるが、それは `sender` のnonceである。どのアカウントがデポジットしたETHから払い戻すかというのも `sender` になる。払い戻し先は `handleOps()` の `beneficiary` になるがこれは `sender` である必要はないがコントラクトであればETHを受け取れるようにしておこう(revertする)。