tags
Substrate 进阶课, batch-02

到这里看线上版 md：https://hackmd.io/lrHZ5pCGQtywDfF2kflf5Q?view

Substrate 进阶课第 3 讲 - Substrate Kitties 教程（二）| Jimmy 部份

大纲

- Polkadot-js API
- FRAME Sudo 模块讲解
- FRAME treasury 模块讲解

讲在开始之前

- 授人以鱼，不如授之以渔
- 会讲如何在各文档之间穿梭，这是学用 Substrate 最重要的技能

Substrate/Polkadot-JS 文档：

- 主要：substrate.dev
    - 教程 tutorials
    - 基础知识 knowledge base
    - 进阶菜谱 Recipes
    - API 文档 Rustdocs
- Polkadot wiki
    - 纪录着 Polkadot 及 Kusama 网络的基础知识及网络行为
- Polkadot JS 文档

Polkadot-js API

一些基本的，我假设你们都应该懂了，如果未搞清楚，可看回基础课：

- `api.tx.<pallet>.<call>` 来发送外部交易 (extrinsics)
- `api.consts.<pallet>.<const>` 来拿取 pallet 常数
- `api.query.<pallet>.<name>` 来读取 pallet 存储

读取链上元数据 (metadata)

JavaScript

```
const { magicNumber, metadata } = await api.rpc.state.getMetadata();

console.log("Magic number: " + magicNumber);
console.log("Metadata: " + metadata.raw);
```

为什么这个重要？因为你能知道整个 链提供了什么外部交易给客户端使用

JavaScript

```
{
  magicNumber: 1635018093,
  metadata: {
    V12: {
      modules: [
        // { ... }
        {
          name: TemplateModule,
          storage: {
            prefix: TemplateModule,
            items: [
              {
                name: Something,
                modifier: Optional,
                type: {
                  Plain: u32
                },
                fallback: 0x00,
                documentation: []
              }
            ]
          },
          calls: [
            {
              name: do_something,
              args: [
                {
                  name: something,
                  type: u32
                }
              ],
              documentation: [
```

```
                An example dispatchable that takes a singles value as a parame
                storage and emits an event. This function must be dispatched b
              ]
            },
            {
              name: cause_error,
              args: [],
              documentation: [
                An example dispatchable that may throw a custom error.
              ]
            }
          ],
          events: [
            {
              name: SomethingStored,
              args: [
                u32,
                AccountId
              ],
              documentation: [
                Event documentation should end with an array that provides des
                parameters. [something, who]
              ]
            }
          ],
          constants: [],
          errors: [
            {
              name: NoneValue,
              documentation: [
                Error names should be descriptive.
              ]
            },
            {
              name: StorageOverflow,
              documentation: [
                Errors should have helpful documentation associated with them.
              ]
            }
          ],
          index: 8
        }
      ],
      extrinsic: {
        version: 4,
        signedExtensions: [
          CheckSpecVersion,
          CheckTxVersion,
```

```
            CheckGenesis,
            CheckMortality,
            CheckNonce,
            CheckWeight,
            ChargeTransactionPayment
          ]
        }
      }
    }
}
```

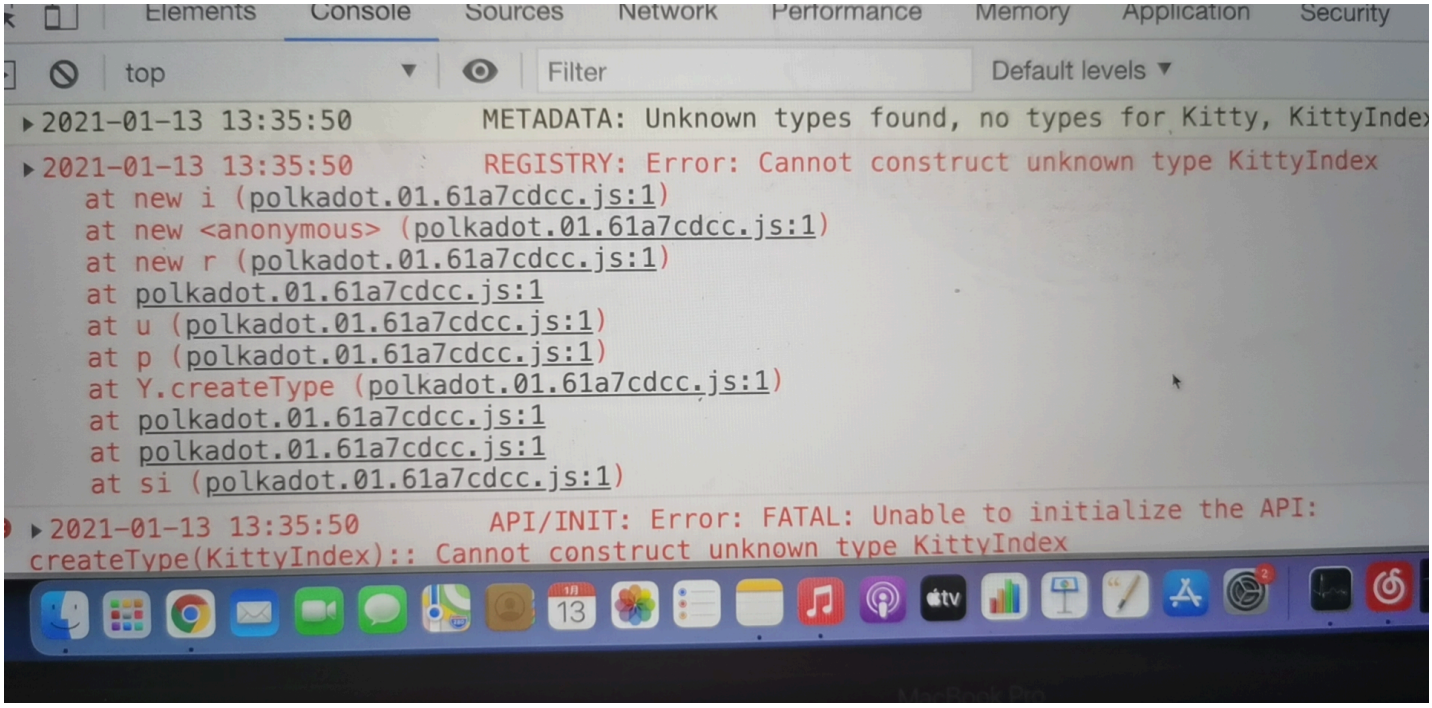
- 这里提到有多少个 pallets (又叫 module)，每个 module 的名字，有什么 storage, calls, events, constants, errors
- 它的 index 数。出现错误信息时，用来追踪是那个 pallet 发出的

设定自订义类型

如果你看到有以下错误信息，

```
Cannot construct unknown type ...
```

比如：我直接拿了一位学生截图来用 🙏



那就说明你的链有一自定义类型，但 polakdot-js API 不知道怎么解释它。需要做的是：

- 参看：https://polkadot.js.org/docs/api/start/types.extend

```JavaScript
const api = await ApiPromise.create({
  provider: wsProvider,
  types: {
    KittyIndex: 'u64'
  }
});
```

增加 `types` 这个参数

批量查询及订阅

1. **同时发多个查询**

   可同时发多个查询，而不是一条一条发

```JavaScript
// Subscribe to balance changes for 2 accounts, ADDR1 & ADDR2 (already defi
const unsub = await api.query.system.account.multi([ADDR1, ADDR2], (balance
 const [{ data: balance1 }, { data: balance2 }] = balances;

 console.log(`The balances are ${balance1.free} and ${balance2.free}`);
});
```

   也可同时发多个不同类型查询

```JavaScript
// Subscribe to the timestamp, our index and balance
const unsub = await api.queryMulti([
 api.query.timestamp.now,
 [api.query.system.account, ADDR]
], ([now, { nonce, data: balance }]) => {
 console.log(`${now}: balance of ${balance.free} and a nonce of ${nonce}`);
});
```

   以上的开发模式有两点要注意：

   - 作查询时，传入一个 回调函数 (callback) / 订阅函数。你在这里更新你 react 的 state 的话，就不会出现为什么链上数据改了，而前端没有更新数据的问题。
   - `unsub` ：这个 `unsub` 是一个函数，用来取消这个订阅的。如果是 react/前端开发，你在 `ComponentWillUnmount()` ，或 `useEffect()` 里，就会 call 这个取消订阅函数。整个模式类似以下：

```JavaScript
useEffect(() => {
 let unsub = null;

 const asyncFetch = async () => {
   unsub = await api.query.pallet.storage(
     param,
     result => console.log(`Result: ${result}`)
   );
 };

 asyncFetch();

 return () => {
   unsub && unsub()
 }
}, [api, keyring]);
```

2. **交易并订阅事件**

```JavaScript
// Create alice (carry-over from the keyring section)
const alice = keyring.addFromUri('//Alice');

// Make a transfer from Alice to BOB, waiting for inclusion
const unsub = await api.tx.balances
 .transfer(BOB, 12345)
 .signAndSend(alice, (result) => {
   console.log(`Current status is ${result.status}`);

   if (result.status.isInBlock) {
     console.log(`Transaction included at blockHash ${result.status.asInBlo
   } else if (result.status.isFinalized) {
     console.log(`Transaction finalized at blockHash ${result.status.asFina
     unsub();
   }
 });
```

keyring 钥匙圈

```JavaScript
// Import
const { Keyring } = require('@polkadot/keyring');
```

- 这里有几个概念要讲，首先 keypair 钥匙对。一个帐号背后是一对公钥和私钥。
- 这个钥匙对用来你所作的交易签名的。

- 你用你的私钥对一个交易 (可理解为一组信息，一堆 bytes) 作签名。其他人可用你的公钥来验证这个交易为你用私钥签署的
- 签名的方法 polkadot-js API 支持：
  - ed25519
  - sr25519
  - ecdsa
  - 及 ethereum

- 而同一对钥匙对，会因应不同的网络，生成出不同的帐号 (AccountID)。也就是说同一对钥匙对，在 Substrate 网络是一个 AccountID, 在 Polkadot 网络则显示为另一组 AccountID, 而在 Kusama 又是另一个。

```JavaScript
import { Keyring } from '@polkadot/keyring';
// create a keyring with some non-default values specified
const keyring = new Keyring();
```

> 小窍门： 你可访问 polkadot-js App, Developer > Javascript 内，可再加 debugger 与里面的对象物件互动。

这样，默认生成出来是用 `ed25519` 签名法，及为 Substrate 网络的帐号。

```JavaScript
const keyring = new Keyring({ type: 'sr25519', ss58Format: 2 });
```

这样，默认生成的出来是用 `sr25519` 签名法，及为 Kusama 网络的帐号。

ss58Format:

- `0` : Polkadot 网络
- `2` : Kusama 网络
- `42` : 一般 Substrate 网络

然后，就可这样加一个帐号：

```JavaScript
const mnemonic = mnemonicGenerate();

// create & add the pair to the keyring with the type and some additional
// metadata specified
const pair = keyring.addFromUri(mnemonic, { name: 'first pair' });
```

最后，拿着这个帐号，你就可对一个交易作签名：

```javascript
const txHash = await api.tx.balances
  .transfer(BOB, 12345)
  .signAndSend(alice);
```

参考:

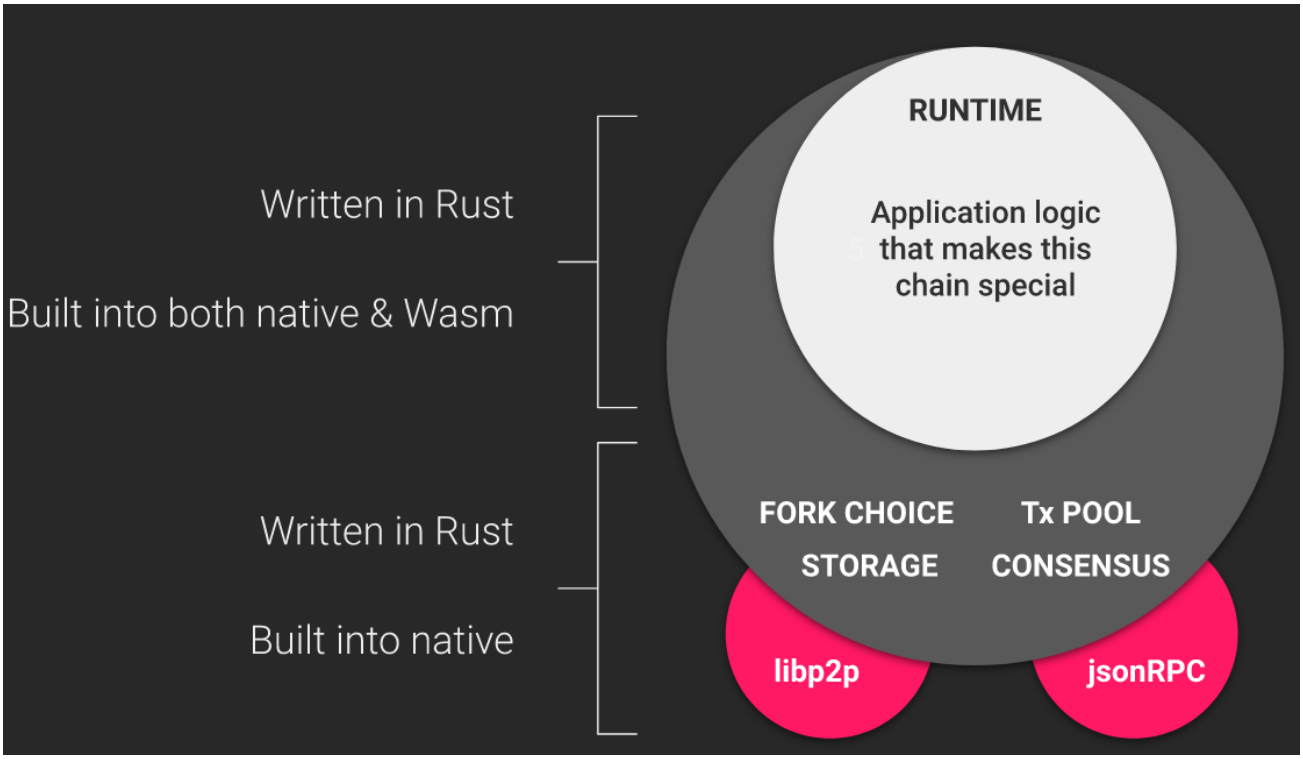
- SS58 地址格式
- Polkadot Accounts
- ecdsa 签名法

FRAME Sudo 模块讲解

> 这部份大家听的时候：
>
> - git clone 这个版本：https://github.com/paritytech/substrate/tree/v2.0.1
> - 打开代码
> - 也打开 Rustdocs 相应页

- sudo rustdocs 的相应页
- rustdoc 及 代码之间穿梭
- Call
  - `sudo`
  - `sudo_unchecked_weight`
  - `sudo_as`
  - 特别的是 `T::Lookup` 是哪里来的？

    还记得这结构图



- 回到 runtime 里，runtime 来定义。而 `bin/node/runtime/src/lib.rs#170`：

```rust
type Lookup = Indices
```

而 `Indices` 就是定义在 `bin/node/runtime/src/lib.rs#901`：

```rust
Indices: pallet_indices::{Module, Call, Storage, Config<T>, Event<T>}
```

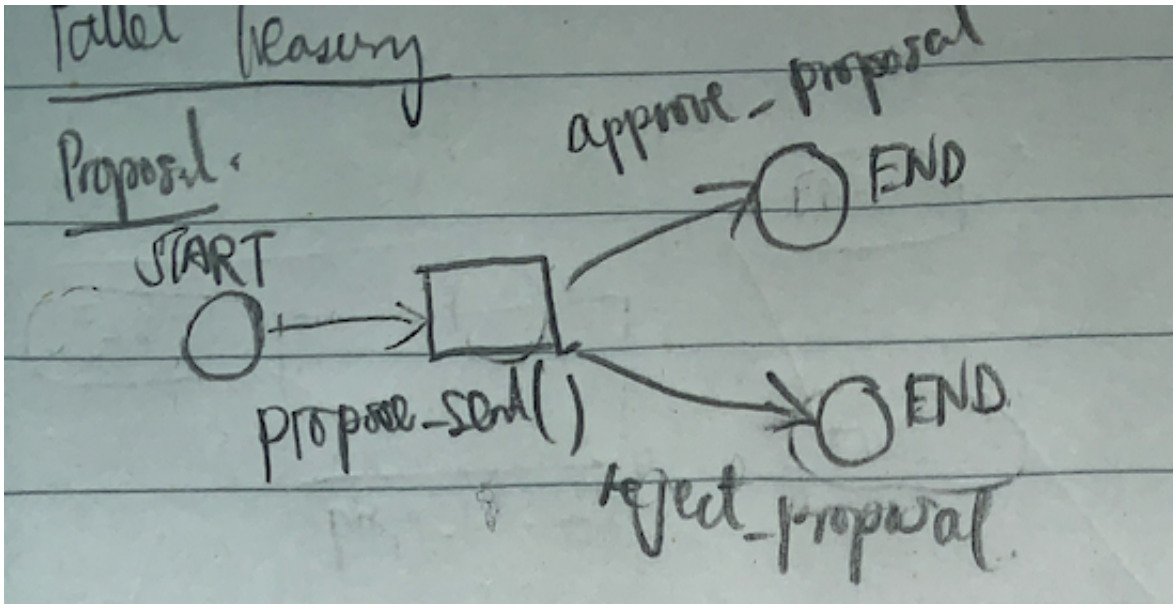
- 这里展示了一个开发模式在一个 pallet 调用另一个 pallet 的函数。

FRAME Treasury 模块讲解

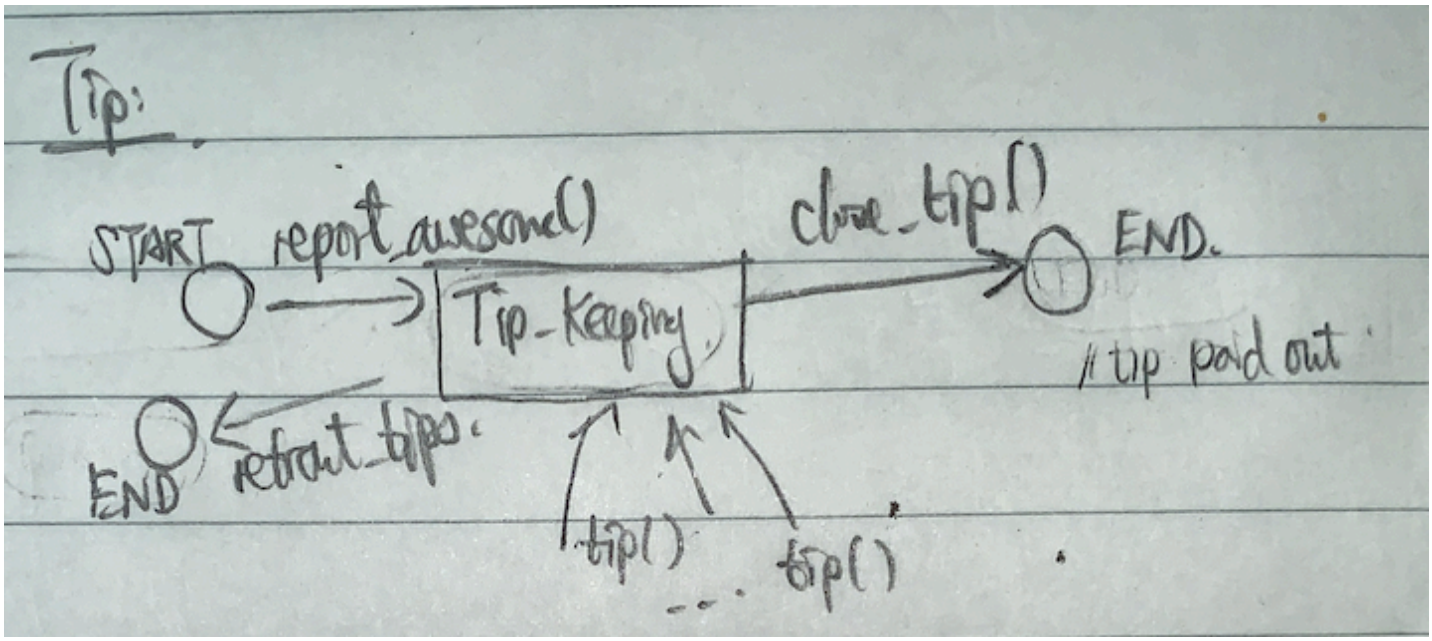
- treasury rustdocs 的相应页
- `Structs` 部份。有很多 `Instance0`, `Instance1`. 这个是 Instantiable Pallet。在一个 runtime 里，这个模块 (pallet) 能有多个 instance. 看 Substrate Recipe.
- `Call` 部份有不同的 extrinsics.

Treasury 讲的就是三个东西，Proposal，Tip，和 Bounty。

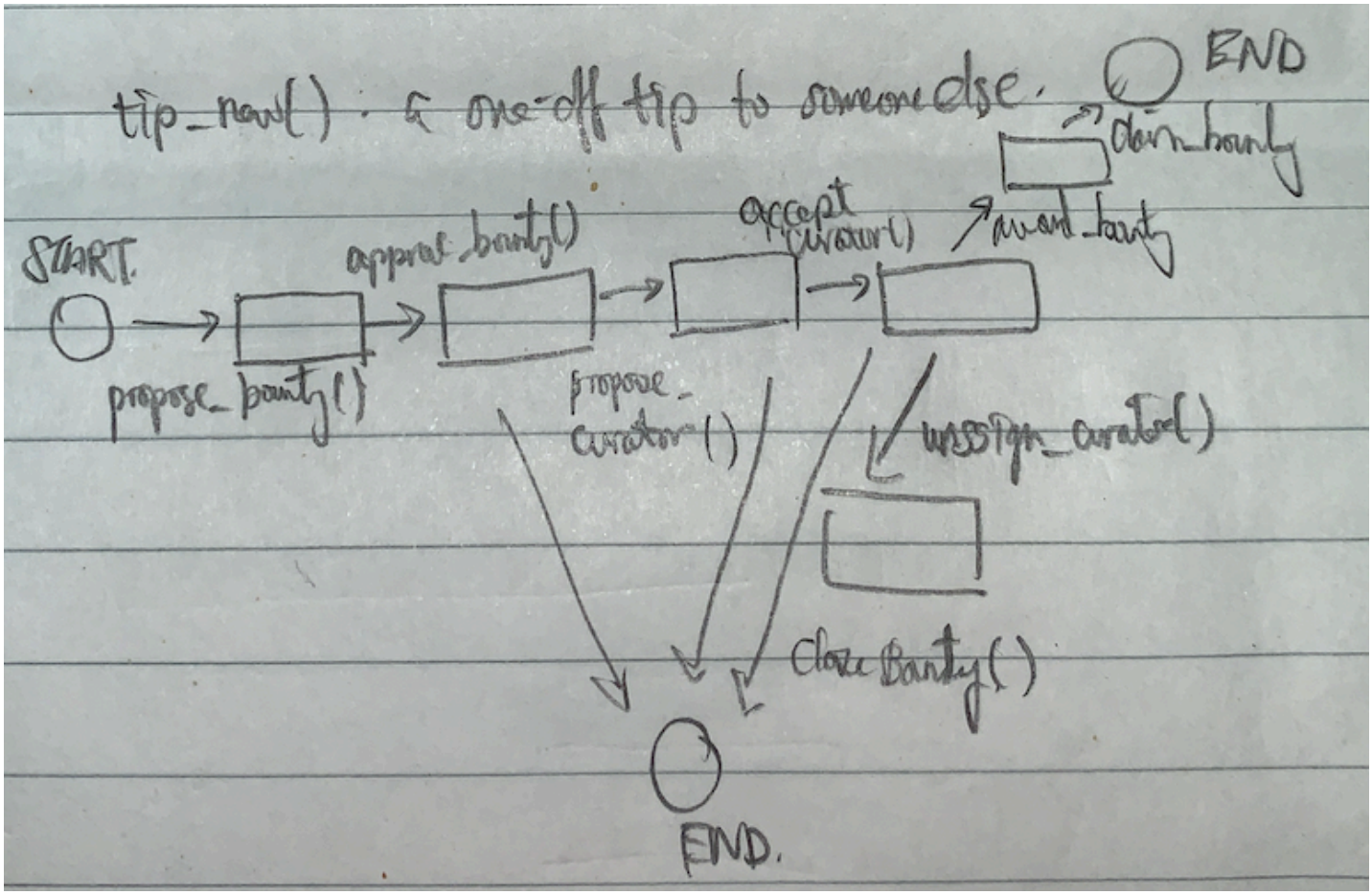
Proposal 的状态转移



Tip 的状态转移



Bounty 的状态转移



作业

前端基于 kitties-course 已有前端 加以下 UX 及功能。这部份共 10 分:

1. 能创建一个毛孩 (**3 分**)
2. 每一个毛孩展示成一张卡片，並显示是不是属于你的 (**4 分**)
3. 可以转让毛孩给另一位用户 (**3 分**)

现在前端展示：