Motoko 语言特性

Motoko Canister 容器

和用户 call canister 一样，canister call canister，也是异步。所以需要 await，但是这个 await 包含以下步骤：

1. 从消息进来，执行到 await，第一个原子性的 call 已经结束，并且底层系统预留好 callback 相关逻辑和空间。
2. 第二个原子性的 call 在被调 canister 里面执行。并将返回值返回给上一个 canister。
3. 第一个 canister 收到消息，从 callback 开始执行接下来的逻辑，也就是第三个原子性 call。

在 canister 升级的时候，如果有 canister 发出去的消息还没返回的情况（有 callback 没有执行），容易导致数据损坏。因为升级后调用栈和之前的不一样。平台通过保证 canister 发出去的消息一定有回复，并在 canister 收到所有的回复后才允许停止 canister 来防止这个问题。

actor 之间的可组合性，通过上面说的和其他 actor 的异步调用来实现。

程序的复用，通过模块化处理 Module 实现。 Module 里面不允许有 stable memory。 通过通信来共享状态，而不是通过共享状态来通信

Module import

- base-library
- 本地自己实现的
- actor class
- Canister

还可以用 vessel 导入其他人实现的库。

Object 和 Class

补充：Object 或者 Class 实例无法直接申明为 stable 变量，所以需要实现 pre_upgrade 和 post_upgrade 做相关处理。

Actor Class

await 创建的 canister 受创建者 canister 控制。

创建后能够得到一个 actor 类型，可以从 actor 类型里面知道新创建的 canister id 以及对外接口，也可以去调用。

子类型关系 subtype

B <= A B 是 A 的子类型 所有接受 A 类型的地方都可以用 B 类型的值 A 更宽泛（general), B 更具体（specific）

使用 vessel 管理程序库

单元测试

```
$(dfx cache show)/moc $(vessel sources) -wasi-system-api -r Test.mo
```

Logger 演示

作业相关

判断下述子类型关系哪些是真的?

1. [ ] 1. {a: Bool} ≤ {a: Bool; b: Nat}
2. [x] 2. {a: Bool} ≤ {}
3. [x] 3. {#red; #blue} ≤ {#red; #yellow; #blue}
4. [x] 4. Nat ≤ Int
5. [ ] 5. Int ≤ Int32
6. [x] 6. () -> () ≤ (Text) -> ()
7. [x] 7. () -> (Text) ≤ () -> ()
8. [x] 8. () -> ({#male; #female}) ≤ () -> ()
9. [x] 9. (Int) -> (Nat) ≤ (Nat) -> (Int)
10. [x] 10. (Int16, Nat8) ≤ (Int32, Nat32)

- 关于枚举类型：https://smartcontracts.org/docs/current/developer-docs/build/languages/motoko/motoko-at-a-glance/#type-system
- 关于函数类型：https://smartcontracts.org/docs/current/developer-docs/build/languages/motoko/language-manual/#subtyping

无限扩容的 Logger

- Devon Sun: https://github.com/0xdzs/elastic-ic-logger/blob/main/ElasticIcLogger.mo 代码清晰简洁。
- 龙君昱: https://github.com/dragon-raja/IC-logger/blob/main/example/MutiLogger.mo 功能比较丰富，包括给新创建的 canister 更改 controller。
- Flanker: https://github.com/dizzy27/my_logger/blob/main/src/my_logger/main.mo 功能比较丰富，包括给新创建的 canister 更改 controller。