ICP区块链开发进阶课程

3. Canister 开发进阶 II

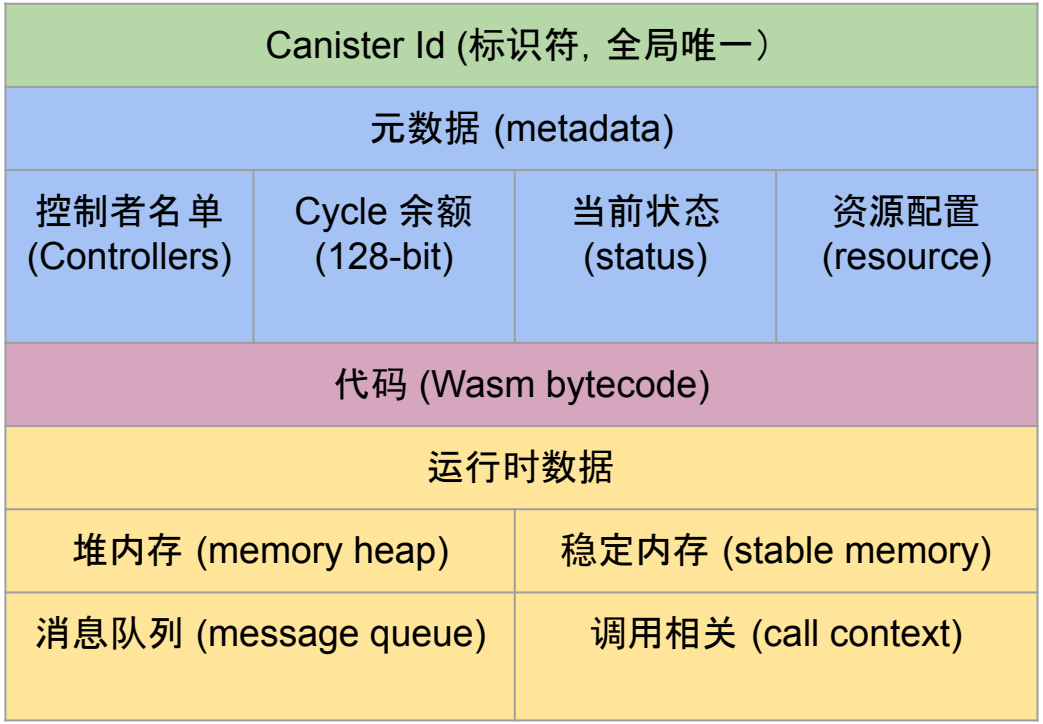
主讲：Paul Liu - DFINITY 工程师

课程大纲

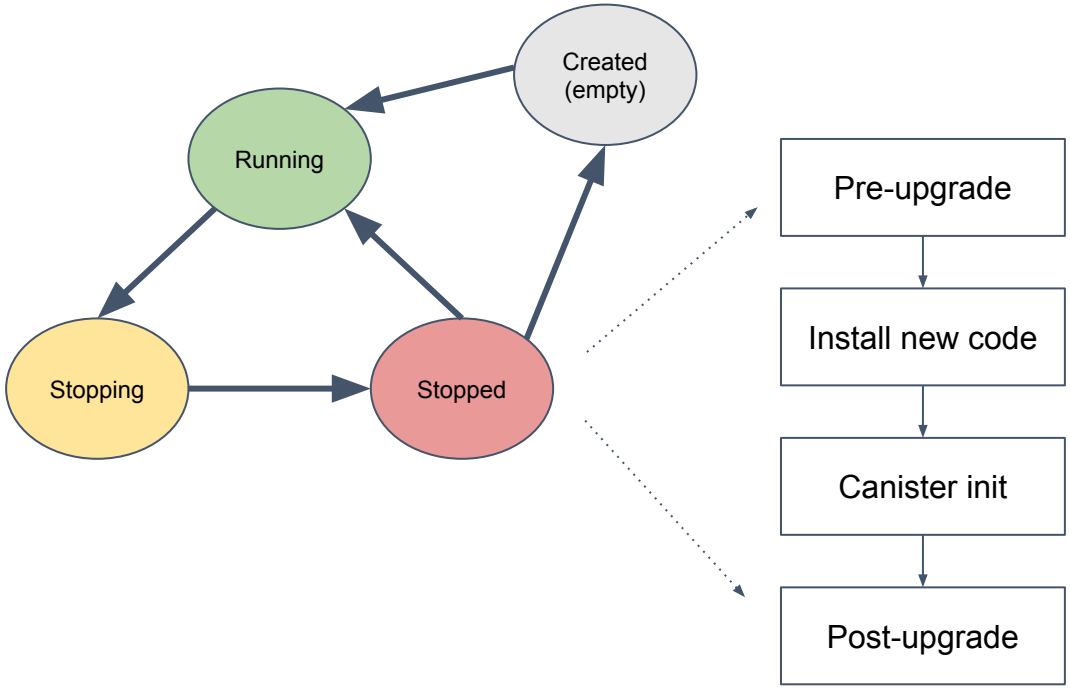
1. Motoko 语言进阶
2. Canister 开发进阶 I
3. Canister 开发进阶 II
4. 整合 ICP 系统服务
5. 项目实例分析

正交持久化 (Orthogonal Persistence)

- 方法调用**成功**完成，被修改的状态会自动持久化
- 即便方法调用失败，计费相关的 Meta 信息仍然被持久化
- Heap 内存在升级 Canister 代码时被清空，而 Stable 内存则永久保留
- Stable 内存不仅仅是用于升级，随时可以使用



升级 Canister 代码



- 运行 pre-upgrade 的是旧代码
- 运行 post-upgrade 的是新代码
- 期间发生任何错误，则升级中断回滚
- Stopped 期间不能处理消息，发送方会收到错误返回

在 Motoko 中使用 Stable Var

- 语言层面上支持把(满足规则的)stable 变量保留到升级后使用

```
actor {
  // A stable data structure that holds player data.
  stable var accounts : [(PlayerId, PlayerState)] = [];

  let players : Players = {
    id_map   = ... // build HashMap<PlayerId, PlayerState> from accounts
    name_map = ... // build HashMap<PlayerName, PlayerId> from accounts
  };

  // before upgrade, convert data from players to accounts.
  system func preupgrade() {
    accounts := Iter.toArray(players.id_map.entries());
  };
  ...
}
```

- 如果升级前后类型发生变化，必须满足子类型的关系(之前是之后的子类型)
- 向记录结构中添加新字段，是 option 类型可以吗？

概念对比：stable vs. shared

Stable	Shared
用于在升级过程中保留数据，升级后可用。 支持的数据类型和 shared 类似，并包括带有 mutable 字段的类型	用于在 Actor 之间交换数据。 包括不可变 (immutable) 数据类型，actor 指针，公共方法指针。 不包括本地的私有方法， 或者可变 (mutable) 数据类型。

	Stable
基础类型 (Int, Nat, Text, etc)	✓
本地函数引用 (local function references)	✘
Actor 或者共享函数引用 (reference to actors or shared methods)	✓
简单的对象或者记录结构 (结构中不包含本地函数引用)	✓
广义上的对象类型 (general object)	✘

代码演示

Canister 使用 Cycles 的场景

- 终端用户不直接为计算付费
- Canister 为自己付费：
  1. 从 cycle 余额中支付 (update call, 存储）
  2. 以发消息的方式支付 (跨 canister 通信，调用 IC Management Canister)

在消息传递中发送 Cycles

- 随着下一个消息发送 cycle
  `add : Nat -> ();`
- 接收本次消息发来的 cycle
  `accept: Nat -> ();`
- 查看本次消息收到的 cycle 数量
  `available: () -> Nat;`
- 查看被上一个消息退回 cycle 数量
  `refunded: () -> Nat;`
- 查看 Canister cycle 余额
  `balance: () -> Nat;`

```
import IC "./ic";
import Cycles "mo:base/ExperimentalCycles";

actor class () = self {
    public func create_canister() : async IC.canister_id {
        let settings = {
            ...
        };
        let ic : IC.Self = actor("aaaaa-aa");
        Cycles.add(1_000_000_000_000);
        let result = await ic.create_canister({
            settings = ?settings;
        });
        result.canister_id
    }
};
```

课程作业

实现一个简单的多人 Cycle 钱包：

1. 团队 N 个成员，每个人都可以用它控制和安装 canister。
2. 升级代码需要 M/N 成员同意。

要求：

- 用 actor class 参数初始化最初的 M, N, 以及小组成员。
- 简单的提案功能和流程（发起，投票，执行）。

下一节：整合 ICP 系统服务

- 使用 Internet Identity
- 使用 Ledger Canister