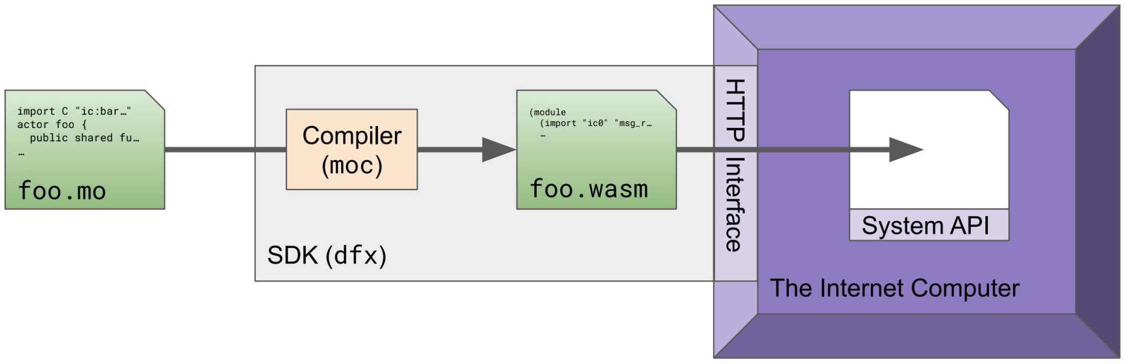
微信扫码打卡

Canister 开发进阶 I 课程回顾

Canister 与系统之间的关系



使用 Motoko 语言编写.mo 源文件

---经过编译器编译--->>

生成 WebAssembly 文件

---通过 HTTP 接口--->>

与 IC 的系统 API 进行交互。

系统对 Canister 的调用

canister_init ：() –> ()

canister_pre_upgrade ：() –> ()

canister_post_upgrade ：() –> ()

canister_inspect_message ：() –> ()

canister_heartbeat ：() –> ()

canister_update <name> ：() –> ()

canister_query <name> ：() –> ()

回调函数，必须符合类型（env ：i32）–> ()

Canister 对系统的调用

```
ic0.msg_arg_data_size : () -> i32;                                        // I U Q Ry F
ic0.msg_arg_data_copy : (dst : i32, offset : i32, size : i32) -> ();      // I U Q Ry F
ic0.msg_caller_size : () -> i32;                                          // I G U Q F
ic0.msg_caller_copy : (dst : i32, offset: i32, size : i32) -> ();         // I G U Q F
ic0.msg_reject_code : () -> i32;                                          // Ry Rt
ic0.msg_reject_msg_size : () -> i32;                                      // Rt
ic0.msg_reject_msg_copy : (dst : i32, offset : i32, size : i32) -> ();    // Rt

ic0.msg_reply_data_append : (src : i32, size : i32) -> ();                // U Q Ry Rt
ic0.msg_reply : () -> ();                                                 // U Q Ry Rt
ic0.msg_reject : (src : i32, size : i32) -> ();                           // U Q Ry Rt

ic0.msg_cycles_available : () -> i64;                                     // U Rt Ry
ic0.msg_cycles_available128 : (dst : i32) -> ();                          // U Rt Ry
ic0.msg_cycles_refunded : () -> i64;                                      // Rt Ry
ic0.msg_cycles_refunded128 : (dst : i32) -> ();                           // Rt Ry
ic0.msg_cycles_accept : (max_amount : i64) -> ( amount : i64 );           // U Rt Ry
ic0.msg_cycles_accept128 : (max_amount_high : i64, max_amount_low: i64, dst : i32)
                    -> ();                                                // U Rt Ry

ic0.canister_self_size : () -> i32;                                       // *
ic0.canister_self_copy : (dst : i32, offset : i32, size : i32) -> ();     // *
ic0.canister_cycle_balance : () -> i64;                                   // *
ic0.canister_cycle_balance128 : (dst : i32) -> ();                        // *
ic0.canister_status : () -> i32;                                          // *

ic0.msg_method_name_size : () -> i32                                      // F
ic0.msg_method_name_copy : (dst : i32, offset : i32, size : i32) -> ();   // F
ic0.accept_message : () -> ();                                            // F

ic0.time : () -> (timestamp : i64);                                       // *
```

```
ic0.call_new :                                                            // U Ry Rt H
  ( callee_src  : i32,
          callee_size : i32,
          name_src : i32,
          name_size : i32,
          reply_fun : i32,
          reply_env : i32,
          reject_fun : i32,
          reject_env : i32
  ) -> ();
ic0.call_on_cleanup : (fun : i32, env : i32) -> ();                       // U Ry Rt H
ic0.call_data_append : (src : i32, size : i32) -> ();                     // U Ry Rt H
ic0.call_cycles_add : (amount : i64) -> ();                               // U Ry Rt H
ic0.call_cycles_add128 : (amount_high : i64, amount_low: i64) -> ();      // U Ry Rt H
ic0.call_perform : () -> ( err_code : i32 );                              // U Ry Rt H

ic0.stable_size : () -> (page_count : i32);                               // *
ic0.stable_grow : (new_pages : i32) -> (old_page_count : i32);            // *
ic0.stable_write : (offset : i32, src : i32, size : i32) -> ();           // *
ic0.stable_read : (dst : i32, offset : i32, size : i32) -> ();            // *
ic0.stable64_size : () -> (page_count : i64);                             // *
ic0.stable64_grow : (new_pages : i64) -> (old_page_count : i64);          // *
ic0.stable64_write : (offset : i64, src : i64, size : i64) -> ();         // *
ic0.stable64_read : (dst : i64, offset : i64, size : i64) -> ();          // *

ic0.certified_data_set : (src: i32, size: i32) -> ()                      // I G U Ry Rt H
ic0.data_certificate_present : () -> i32                                  // *
ic0.data_certificate_size : () -> i32                                     // *
ic0.data_certificate_copy : (dst: i32, offset: i32, size: i32) -> ()      // *

ic0.debug_print : (src : i32, size : i32) -> ();                          // * s
ic0.trap : (src : i32, size : i32) -> ();                                 // * s
```

IC Management Canister

```
service ic : {
        create_canister : (record {
                settings : opt canister_settings
        }) -> (record {canister_id : canister_id});
        update_settings : (record {
                canister_id : principal;
                settings : canister_settings
        }) -> ();
        install_code : (record {
                mode : variant {install; reinstall; upgrade};
                canister_id : canister_id;
                wasm_module : wasm_module;
                arg : blob;
        }) -> ();
        uninstall_code : (record {canister_id : canister_id}) -> ();
        start_canister : (record {canister_id : canister_id}) -> ();
        stop_canister : (record {canister_id : canister_id}) -> ();
        canister_status : (record {canister_id : canister_id}) -> (record {
    status : variant { running; stopping; stopped };
    settings: definite_canister_settings;
```

微信扫码打卡

```
    module_hash: opt blob;
    memory_size: nat;
    cycles: nat;
      });
      delete_canister : (record {canister_id : canister_id}) -> ();
      deposit_cycles : (record {canister_id : canister_id}) -> ();
      raw_rand : () -> (blob);
}

type canister_id = principal;
type user_id = principal;
type wasm_module = blob;
type canister_settings = record {
      controllers : opt vec principal;
      compute_allocation : opt nat;
      memory_allocation : opt nat;
      freezing_threshold : opt nat;
};
type definite_canister_settings = record {
      controllers : vec principal;
      compute_allocation : nat;
      memory_allocation : nat;
      freezing_threshold : nat;
};
```

通过 Ledger Canister 动态创建 Canister

假如我们想在 A.mo 的 actor 中动态创建 B.mo 中的 actor 类，则：

一、先创建 Types.mo 模块，在其中写入以下代码：

```
module IC{
    public type canister_settings = {
        freezing_threshold : ?Nat;
```

微信扫码打卡

```
    controllers : ?[Principal];
    memory_allocation : ?Nat;
    compute_allocation : ?Nat;
};

public type definite_canister_settings = {
    freezing_threshold : Nat;
    controllers : [Principal];
    memory_allocation : Nat;
    compute_allocation : Nat;
};
public type user_id = Principal;

public type wasm_module = [Nat8];
public type canister_id = Principal;
public type ICActor = actor {
    canister_status : shared { canister_id : canister_id } -> async {
        status : { #stopped; #stopping; #running };
        memory_size : Nat;
        cycles : Nat;
        settings : definite_canister_settings;
        module_hash : ?[Nat8];
    };

    create_canister : shared { settings : ?canister_settings } -> async {
        canister_id : canister_id;
    };

    delete_canister : shared { canister_id : canister_id } -> async ();

    deposit_cycles : shared { canister_id : canister_id } -> async ();
```

```
        install_code : shared {
            arg : [Nat8];
            wasm_module : wasm_module;
            mode : { #reinstall; #upgrade; #install };
            canister_id : canister_id;
            } -> async ();

        provisional_create_canister_with_cycles : shared {
            settings : ?canister_settings;
            amount : ?Nat;
            } -> async { canister_id : canister_id };

        provisional_top_up_canister : shared {
            canister_id : canister_id;
            amount : Nat;
            } -> async ();

        raw_rand : shared () -> async [Nat8];
        start_canister : shared { canister_id : canister_id } -> async ();
        stop_canister : shared { canister_id : canister_id } -> async ();
        uninstall_code : shared { canister_id : canister_id } -> async ();

        update_settings : shared {
            canister_id : Principal;
            settings : canister_settings;
            } -> async ();
    };
}
```

然后在 A.mo 中引入 Types：

```
import Types "./Types";
```

二、在 A.mo 和 B.mo 中都引入以下 base 库文件：

```
import Cycles "mo:base/ExperimentalCycles"; //目前为实验版，后续会更新
```

并添加以下代码：

```
public query({caller}) func cycleBalance() : async Nat{
    Cycles.balance()
};

public shared({caller}) func wallet_receive() : async Nat {
    Cycles.accept(Cycles.available())
};
```

三、对 ledger actor 进行调用，以创建 B canister：

假设 B.mo 大致为以下样式：

```
//import ...

shared({caller}) actor class B(installer : Principal) = this{
    //...
};
```

在 A.mo 中添加以下代码：（仅摘取部分代码）

```
import RBT "mo:base/RBTree"; //RB 树的库文件
import Nat "mo:base/Nat";
import Principal "mo:base/Principal";
import B "./B.mo";

private stable var b_index : Nat = 0;
private let Bs = RBT.RBTree<Nat, Principal>(Nat.compare); //存储创建的 B canist
er
```

```
private let IC : Types.IC.ICActor = actor "aaaaa-aa"; //ledger actor 的 ID
private let CYCLE_LIMIT = 1_000_000_000_000; //根据需要进行分配

//动态创建 canister 的函数
public shared({caller}) func createB() : async Result.Result<Principal, Text>{
        Cycles.add(CYCLE_LIMIT);
        let b = await B.B(caller);
        let principal = Principal.fromActor(b);
        await IC.update_settings({
            canister_id = principal;
            settings = {
                freezing_threshold = ?2592000;
                controllers = ?[caller]; //A 不作为 B 的控制者的写法
                memory_allocation = ?0;
                compute_allocation = ?0;
            }
        });
        Bs.put(b_index, principal);
        b_index += 1;
        #ok(principal)
    };
```

完毕。

详见：https://qiuyedx.com/?p=865

Candid 接口规范

- 与 Protobuf, CBOR 这一类数据序列化协议的差别
    - 可以描述更多数据类型，包括函数类型，递归类型
    - 函数类型可以用于描述服务接口和方法

微信扫码打卡

    - 升级过程中的类型适配
- 多语言支持（包括）
    - Javascript, Motoko, Rust
    - Python, Go, Haskell, AssemblyScript
    - 生成 Candid 类型规范，编码解码，从 Candid 规范导入数据类型

IC 双向消息传递的保证(bi-directional messaging)

- 但凡发出的消息，必然会收到回答
- 升级的时候，需要先 stop 再 upgrade
- 每个消息最多被处理一次（没有被处理的，会返还错误给发送方）

Motoko 异常处理

Motoko try/catch 仅用于对异步的异常处理

```
public func dec(v: Nat) : async Nat {
        assert(v > 0);
        v - 1
};
public func test(n: Nat) : async Nat {
        try {
                await dec(n)
        } catch(e) {
                Debug.print("Caught error: " # Error.message(e));
                0
        }
};
```

微信扫码打卡

课程作业

作业要求

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；
3. 暂时不需要考虑权限控制，所有人都可操作该 Canister。

要求至少实现的函数：

- create_canister
- install_code
- start_canister
- stop_canister
- delete_canister

作业展示

李锦华同学：https://github.com/jinhuaio/icp-study/tree/main/part2/course2/dynamic_canister 待完善

```
import IC "./ic";
import Principal "mo:base/Principal";
import Option "mo:base/Option";
import Buffer "./ExtBuffer";
import Map "mo:base/HashMap";

actor class () = self{

  //多签名 Canister 的数据结构
```

微信扫码打卡

```
  type MultiSignatureCanister = {
      canister_id: IC.canister_id;
      controllers : [Principal];
      describe : ?Text;
  };

  //多签名 Canister 的缓存数据
  private var multiSignatureCanisters = Map.HashMap<IC.canister_id, MultiSignat
ureCanister>(10, Principal.equal, Principal.hash);

  public func greet(name : Text) : async Text {
    return "Hello, " # name # "!";
  };

  // 创建 canister 需要指定
  // maintainers: 维护者人数, 即后续升级和维护此 canister 最低需要多少人才允许
升级
  // controllers: 控制者清单, 控制者清单人数必须大于等于 maintainers 数量
  // describe: 该 canister 的描述信息
  public shared ({caller}) func create_canister({maintainers : Nat; controllers : ?
[Principal]; describe : ?Text}) : async IC.canister_id{
    let cs : [Principal] = switch (controllers) {
      case null {[caller,Principal.fromActor(self)]};
      case (?c) {
        let buf = Buffer.ExtBuffer<Principal>(c.size() + 2);
        buf.appendArray(c);
        buf.add(caller);
        buf.add(Principal.fromActor(self));
        buf.toArray();
      };
    };
    //确认维护者数量是否超过控制者人数
```

微信扫码打卡

```
    assert(cs.size() >= maintainers);

    let setting = {
      freezing_threshold = null;
      controllers = ?cs;
      memory_allocation = null;
      compute_allocation = null;
    };
    let ic : IC.Self = actor("aaaaa-aa");
    let result = await ic.create_canister({settings = ?setting});
    let canister = {
      canister_id = result.canister_id;
      controllers = cs;
      describe = describe;
    };
    //缓存 canister
    multiSignatureCanisters.put(result.canister_id, canister);
    result.canister_id;
  };

  //安装/升级 Canister 代码
  public func install_code({canister_id : IC.canister_id ;mode : { #reinstall; #upgra
de; #install }; wasm : IC.wasm_module}) : async () {

    assert checkMultiSignature();

    //TODO 此处应先缓存 install_code 的执行内容，待签名人数达到最低要求后，再
执行
    let ic : IC.Self = actor("aaaaa-aa");
    let install = {
      arg = [];
      wasm_module = wasm;
```

```
        mode = mode;
        canister_id = canister_id;
      };
      await ic.install_code(install);
    };

    private func checkMultiSignature() : Bool{
      //TODO 校验签名的人数是否达到最低要求，待实现
      true;
    };

    public func start_canister(canister_id : IC.canister_id) : async (){
      let ic : IC.Self = actor("aaaaa-aa");
      await ic.start_canister({canister_id = canister_id});
    };

    public func stop_canister(canister_id : IC.canister_id) : async (){
      let ic : IC.Self = actor("aaaaa-aa");
      await ic.stop_canister({canister_id = canister_id});
    };

    public func delete_canister(canister_id : IC.canister_id) : async (){
      let ic : IC.Self = actor("aaaaa-aa");
      await ic.delete_canister({canister_id = canister_id});
    };

    public func uninstall_code(canister_id : IC.canister_id) : async (){
      assert checkMultiSignature();
      //TODO 此处应先缓存 install_code 的执行内容，待签名人数达到最低要求后，再
  执行
      let ic : IC.Self = actor("aaaaa-aa");
```

```
        await ic.uninstall_code({canister_id = canister_id});
    };
};
```

韩东昌同学：https://github.com/handcishere/icjj2 完成度比较高

```
import IC "./ic";
import Array "mo:base/Array";
import Buffer "mo:base/Buffer";
import Deque "mo:base/Deque";
import List "mo:base/List";
import Nat "mo:base/Nat";
import Option "mo:base/Option";
import TrieMap "mo:base/TrieMap";
import TrieSet "mo:base/TrieSet";
import Hash "mo:base/Hash";
import Text "mo:base/Text";
import Cycles "mo:base/ExperimentalCycles";
import Result "mo:base/Result";
import Principal "mo:base/Principal";
//如果一个 canister 被限制了则升级代码和关闭必须通过提案
//提案通过后需要 member 去执行 executeProposal 函数
shared(install) actor class icjj2(m : Nat, member : [Principal]) = this {
    private type canister_id = IC.canister_id;
    private type Oprate = {
        #InstallCode;
        #StopCanister;
    };
    private type InstallCodeArgs = {
        wsm : [Nat8];
        canister_id : canister_id;
    };
    private type StopCanisterArgs = {
```

微信扫码打卡

```
        canister_id : canister_id;
    };
    private type ProposalAgrs = {
        oprate : Oprate;
        memo : Text;
        install_code_args : ?InstallCodeArgs;
        stop_canister_args : ?StopCanisterArgs;
    };
    private type Proposal = {
        args : ProposalAgrs;
        done : Bool;
        var agreed : TrieSet.Set<Principal>;
    };
    private type Error = {
        #PermissionDenied;
        #ProposalNotFound;
        #MemberOnly;
        #ArgsError;
        #RestrictedCanister;
    };

    private let ic : IC.Self = actor "aaaaa-aa";
    private stable var canisters_entires : [Principal] = [];
    private var canisters = TrieSet.fromArray<Principal>(canisters_entires, Princip
al.hash, Principal.equal);
    private stable var members : [Principal] = [];
    private var member_set = TrieSet.fromArray<Principal>(member, Principal.has
h, Principal.equal);
    private stable var N = member.size();
    private stable var M = do {
        if (m >= members.size()) {
            members.size()
        }
        else {
            m
        };
    };
    private stable var pId : Nat = 1;
    private stable var proposal_entries : [var (Nat,Proposal)]      = [var];
    private var proposals = TrieMap.fromEntries<Nat, Proposal>(proposal_entries.
vals(), Nat.equal, Hash.hash);
    private stable var restricted_canister_entries : [Principal] = [];
    private var restricted_canisters = TrieSet.fromArray<Principal>(member, Princi
pal.hash, Principal.equal);
    //后续可以改成通过提案执行
    public shared({caller}) func setRestrictedCanister(arg : Bool,id : Principal) : a
sync Result.Result<() , Error> {
        if(caller != Principal.fromActor(this)){
            if(not TrieSet.mem(member_set,caller,Principal.hash(caller),Principal.eq
ual)){
                return #err(#MemberOnly);
            };
        };
        if(arg){
            ignore TrieSet.put<Principal>(restricted_canisters,id,Principal.hash(id),
Principal.equal);
        }else{
            ignore TrieSet.delete<Principal>(restricted_canisters,id,Principal.hash(i
d),Principal.equal);
        };
        return #ok();
    };

    public shared({caller}) func issuedProposal(args : ProposalAgrs) : async Resu
```

```
lt.Result<() , Error> {
        if(not TrieSet.mem(member_set,caller,Principal.hash(caller),Principal.equal))
{
            return #err(#MemberOnly);
        };
        proposals.put(pId,{
            args=args;
            done=false;
            var agreed=TrieSet.empty<Principal>();
        });
        pId+=1;
        return #ok();
    };
    public shared({caller}) func voteProposal(proposalId:Nat,vote : Bool) : async
Result.Result<() , Error> {
         if(not TrieSet.mem(member_set,caller,Principal.hash(caller),Principal.equa
l)){
            return #err(#MemberOnly);
        };
        switch(proposals.get(proposalId)){
            case null{
                return #err(#ProposalNotFound);
            };
            case (?v){
                if(vote){
                    let set =TrieSet.put<Principal>(v.agreed,caller,Principal.hash(c
aller),Principal.equal);
                    v.agreed := set;
                    proposals.put(proposalId,v);
                }
                else{
                    let set =TrieSet.delete<Principal>(v.agreed,caller,Principal.has
```

微信扫码打卡

微信扫码打卡

```
h(caller),Principal.equal);
                    v.agreed := set;
                    proposals.put(proposalId,v);
                };
                return #ok();
            };
        };
    };



    public shared({caller}) func executeProposal(proposalId:Nat) : async Result.Re
sult<() , Error> {
        if(caller != Principal.fromActor(this)){
            if(not TrieSet.mem(member_set,caller,Principal.hash(caller),Principal.eq
ual)){
                return #err(#MemberOnly);
            };
        };
        switch(proposals.get(proposalId)){
            case null{
                return #err(#ProposalNotFound);
            };
            case (?v){
                if(TrieSet.size<Principal>(v.agreed)>=M){ //投票人数足够
                    //do someting for executing the proposal
                    switch(v.args.oprate){
                        case(#InstallCode){
                            switch(v.args.install_code_args){
                                case null{
                                    return #err(#ArgsError);
                                };
                                case (?install_args){
                                   ignore await install_code(install_args.wsm,inst
all_args.canister_id);
                                };
                            };
                        };
                        case(#StopCanister){
                            switch(v.args.stop_canister_args){
                                case null{
                                    return #err(#ArgsError);
                                };
                                case (?stop_canister_args){
                                   ignore await stop_canister(stop_canister_args.
canister_id);
                                };
                            };
                        };
                    };
                };
                return #ok();
            };
        };
        return #ok();
    };
    public shared({caller}) func create_canister(is_restirct : Bool) :  async Result.
Result<canister_id, Error> {
        if(caller != Principal.fromActor(this)){
            if(not TrieSet.mem(member_set,caller,Principal.hash(caller),Principal.eq
ual)){
                return #err(#MemberOnly);
            };
        };
        let settings = {
            freezing_threshold = null;
            controllers = ?[Principal.fromActor(this)];
            memory_allocation = null;
            compute_allocation = null;
        };
        let res = await ic.create_canister({ settings = ?settings;});
        ignore TrieSet.put<Principal>(canisters,res.canister_id,Principal.hash(res.ca
nister_id),Principal.equal);
        if(is_restirct){
            ignore TrieSet.put<Principal>(restricted_canisters,res.canister_id,Princi
pal.hash(res.canister_id),Principal.equal);
        };
        #ok(res.canister_id)
    };

    public shared({caller}) func install_code(wsm : [Nat8], canister_id : canister_i
d) : async Result.Result<Text, Error> {
        if(caller != Principal.fromActor(this)){
            if(TrieSet.mem(restricted_canisters,canister_id,Principal.hash(caller),Pri
ncipal.equal) ){
                return #err(#RestrictedCanister);
            }else
            if(not TrieSet.mem(member_set,caller,Principal.hash(caller),Principal.eq
ual)){
                return #err(#MemberOnly);
            };
        };
        await ic.install_code({
            arg = [];
            wasm_module = wsm;
            mode = #install;
            canister_id = canister_id;
        });
        #ok("ok")
    };

    public shared({caller}) func start_canister(canister_id : canister_id) : async Result.Result<Text, Error> {
        if(caller != Principal.fromActor(this)){
            if(not TrieSet.mem(member_set,caller,Principal.hash(caller),Principal.equal)){
                return #err(#MemberOnly);
            };
        };
        await ic.start_canister({ canister_id = canister_id;});
        #ok("ok")
    };

    public shared({caller}) func stop_canister(canister_id : canister_id) : async Result.Result<Text, Error> {
        if(caller != Principal.fromActor(this)){
            if(TrieSet.mem(restricted_canisters,canister_id,Principal.hash(caller),Principal.equal)){
                return #err(#RestrictedCanister);
            };
            if(not TrieSet.mem(member_set,caller,Principal.hash(caller),Principal.equal)){
                return #err(#MemberOnly);
            };
        };
        await ic.stop_canister({ canister_id = canister_id;});
        #ok("ok")
    };
```

微信扫码打卡

```
    public shared({caller}) func delete_canister(canister_id : canister_id) : async
Result.Result<Text, Error> {
        if(caller != Principal.fromActor(this)){
            if(not TrieSet.mem(member_set,caller,Principal.hash(caller),Principal.eq
ual)){
                return #err(#MemberOnly);
            };
        };
        await ic.delete_canister({ canister_id = canister_id;});
        #ok("ok")
    };

};
```

微信扫码打卡