Canister合约基础概念：

Cycle： 价值稳定，价值锚定联合国的SDR稳定币（一揽子法币加权求和所得），用作合约运行消耗。与ethereum不同的是，cycle由项目方支付。

1T(1 x 10^12) Cycle 约为 1.42 $

Canister ：

合约容器

Canister可以理解为主网上的一个“占位”， 这个占位向主网申请一定的资源，比如计算，存储空间并支付Cycle（同ETH的gas）等。如果没有申请资源，主网节点会动态分配。

Canister 包含一些基础属性：

最小计算能力分配 compute-allocation

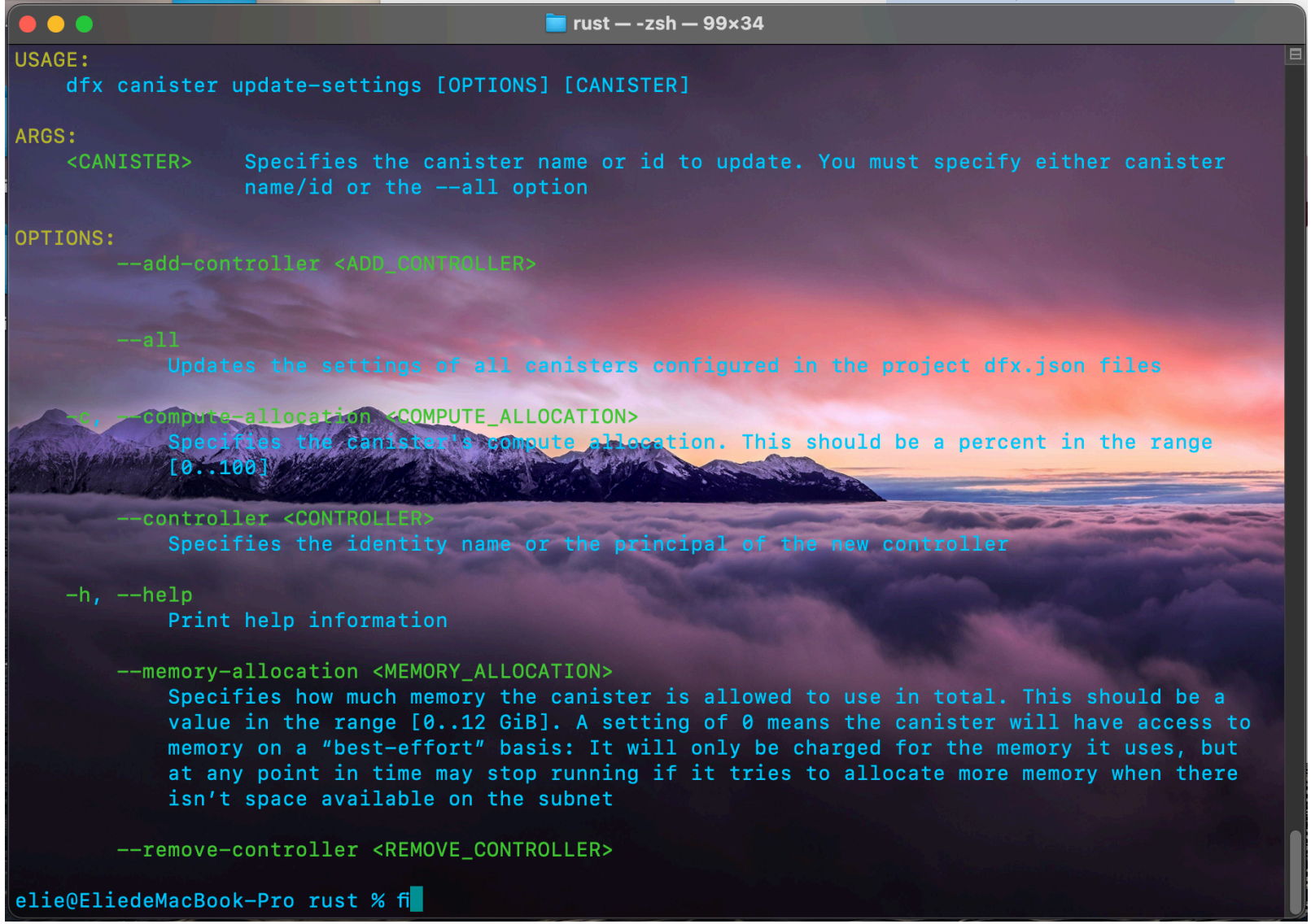
最小内存分配 memory-allocation

合约控制者 controllers（一个数组）

WebAssembly： 智能合约运行字节码

Memory：RAM

运行时状态均存在RAM中


rust — -zsh — 99×34
USAGE:
dfx canister update-settings [OPTIONS] [CANISTER]
ARGS:
<CANISTER> Specifies the canister name or id to update. You must specify either canister name/id or the --all option
OPTIONS:
--add-controller <ADD_CONTROLLER>
--all
Updates the settings of all canisters configured in the project dfx.json files
-c, --compute-allocation <COMPUTE_ALLOCATION>
Specifies the canister's compute allocation. This should be a percent in the range [0..100]
--controller <CONTROLLER>
Specifies the identity name or the principal of the new controller
-h, --help
Print help information
--memory-allocation <MEMORY_ALLOCATION>
Specifies how much memory the canister is allowed to use in total. This should be a value in the range [0..12 GiB]. A setting of 0 means the canister will have access to memory on a "best-effort" basis: It will only be charged for the memory it uses, but at any point in time may stop running if it tries to allocate more memory when there isn't space available on the subnet
--remove-controller <REMOVE_CONTROLLER>
elie@EliedeMacBook-Pro rust % fi

创建一个Canister属于在主网分配或者说预先申请了一部分资源， wasm字节码是合约运行的程序字节码， 在IC上开发合约首先要创建Canister，然后将代码编译为wasm字节码，最后，将wasm下载到canister中，canister就可以向外提供服务了。

语言：
Motoko： InternetComputer原生合约语言，语法类似js
Rust： 可以用Rust配合ic_cdk crate（CDK：Canister Development Kit）写IC的合约。将源码编译为wasm32格式后即可部署到主网的Canister中。
Candid： 接口描述语言，方便不同语言写的wasm之间通信，比如前后端canister通信。

下载SDK： dfx
sh –ci "$(curl –fsSL https://sdk.dfinity.org/install.sh)”
生成一个项目：
dfx new hello
目录结构介绍：
dfx.json
后端 .mo文件
前端 assets & src：css, html, index.js ···
Canister可以托管前端和后端代码，前端托管在IC上可以得到主网的密码学保护。

本地部署：
dfx start —background —emulator
dfx deploy
deploy = create canister + build code + install wasm to canister

Motoko语言快速入门：

demo解释：

函数类型分为两种，一种是update，一种是query。

update消息可以对状态进行持久更改，query消息不更改状态。

update消息单线程，非阻塞，顺序执行。query可以达到万级的并行执行。

Actor：Actor是并行计算的数学模型， 在Canister中，每个Canister当前都是一个Actor，通过Actor模型中规定的Mailbox模型进行Actor间通信，以及并行处理query信息。即Canister之间通过Actor模型实现Actor（Canister）间的通信。

变量：

let & var

let 静态变量，不可变

```
let a = 1;
a := 2; // error
```

var 动态变量， 可变’

```
var a = 1;
a := 3; // successful
```

基础类型

字符串： Text ，

```
var a : Text = “”;
a := “hello world ! ”;
```

数据类型： Nat, Nat8(16,32,64)， Int(Int8,···), Float(0.02), Bool(true, false)

数组： [T] T : Type

```
var a : [Nat] = [];
var b = [1,2,3,4,5];
```

结构体：

```
type temp = {
    a : Text;
    b : Bool;
};
let t : temp. = {
    a = “A”;
    b = false;
};
```

枚举类型：

```
type error = { #A, #B };
let a : error = #A;
```

Option : ?T, ?Text = ?Text or null ;

使用switch解开或者do ? { a! }

```
let a : ?Text = ?“value”;
switch(a){
    case null {};
    case(?t){ assert(t == “value”) }
}
```

地址： Principal(用户地址，canister合约地址)

用户 principal qsvan-3ak2y-g5nyp-otvmw-ievwo-deewc-yu6tj-ze5wn-qsfeu-oubtv-hae

canister id ： xxx-cai

流程控制

if else

```
if(bool){}else if(){}else{}
```

switch case

```
switch(res){
    case
    case
}
```

for while loop :

```
for I in iter{}
    for e in arr.vals(){
        Debug.print(debug_show(e)) // debug_show : T -> Text
    }

while(a == 1){ // do something }

label l loop{ break l; }
```

函数：

private func : private func temp() : Text{}; // 同步返回，不能async

public func : public query & public (shared):

public query(msg) func get() : async Principal{ msg.caller }; // 返回调用者地址

var a : ?Principal = null;

public shared(msg) func change() : async (){};

async, await

let res = await funcA(); // await –> async

system func : preupgrade, postupgrade

system func preupgrade(){}

system func postupgrade(){}

升级：

stable

system func pre upgrade

Canister想要对外提供服务，就需要在.mo文件中写出actor：

```
Import
actor demo{
        var v = “old variable”;
        public query func read() : async Text{ v };
        public shared func write() : async Text{ v := “new variable” }; // public 方法默认shared，这里不写shared也可以
}
```

如果是想写工具类，可以写：

```
Module tools {
        public func transfer() : Nat{}
        private func _transfer() {};
};
Import tools “tools”;
tools.transfer()

Class tools(){
        public func transfer() : Nat{};
        private func _transfer() : Nat{};
};

let t = tools();
tools.transfer()
```

跨Canister（合约）调用：

```
type Service = actor{
    get : query() -> async Text;
    put : () -> async Text;
}
let a : Service = actor(Principal.fromText(“”))
let res = await a.get();
```