# 简介

本文所讲内容基于DFINITY Canister SDK，区别于NNS系统，他们的账户系统不一样。

Internet Computer区块链使多台计算机能够像一台非常强大的虚拟机一样运行。组成虚拟机的计算机被组织成子网络，每个子网都是一个区块链，由一定数量的独立机器（称为节点的对等连接的计算机）组成，它们运行 Internet 计算机协议的软件组件。在每个节点上运行的 Internet 计算机软件组件称为副本，它们在子网区块链中的所有节点之间复制状态和计算，副本的核心组件组织成以下逻辑层：

* 一个P2P网络层，它从用户或当前子网的其他节点或其他子网收集和发布消息。网络层接收到的消息被复制到当前子网中的所有节点。
* 一个共识层，通过选择并序列化从用户和不同子网接收的消息来创建输入区块，并且在区块传递到消息路由层之前进行公证和最终确定。
* 一个消息路由层，用于在子网之间路由用户和系统生成的区块消息，管理应用程序的输入和输出队列，并调度消息以供执行。
* 一个执行环境，用于计算执行程序所涉及的确定性计算，并处理从消息路由层接收的消息。

通过Internet Computer，开发人员可以专注于使用智能合约编写代码，而不会受到与环境相关的以下类似的干扰：

* 物理或虚拟网络配置要求
* 负载均衡服务
* 防火墙、网络拓扑或端口管理
* 数据库配置和维护
* 存储卷和设备

# 概念

## actor

Actor 是现代编程语言中的一种特殊对象，它在隔离状态下处理消息，使它们能够被远程和异步处理。

通常，每个canister都包含一个actor对象的编译代码。每个canister还包括一些附加信息，例如界面描述或前端资产。可以创建包含多个canister的项目，但每个canister只能包含一个Actor。

## canister

canister是具有通用唯一标识符的类似于智能合约概念的对象，是定义特定应用程序、服务或微型站点的所有者。

canister封装了所有的编程逻辑、公共入口方法、提供消息类型的接口描述、应用程序或服务或微服务的状态信息。

当通过向canister入口点发送消息来调用函数时，有两种类型的调用：

* Query calls(非提交查询调用)：允许调用查询容器当前状态的函数或调用对容器状态进行操作但不进行更改的函数。有如下特征：
  + 调用是同步并立即响应的。
  + 可以对持有canister的任何节点进行，不需要共识来验证结果。但是存在安全性问题。
  + 不允许对canister的状态进行更改。
  + 不允许被调用的canister调用其他canister的inter-canister类型的函数。
* Update calls(提交更新调用)：允许调用更改canister的状态的函数。有如下特征：
  + 调用是异步响应的。
  + 必须通过共识才能返回结果。由于需要子网中三分之二的副本就结果达成一致，因此更改容器的状态可能需要时间。
  + 允许被调用的canister调用其他canister的inter-canister类型的函数。

## controller

controller是具有特殊权限的身份，该权限用于管理其控制的canister。只有controller身份可用于安装、升级或删除其控制的canister。

可以使用与user或canister关联的principal identifier来指定controller身份。

## ledger

Internet Computer将所有涉及ICP代币的交易在一个被称为ledger canister的管理canister中进行记录。ledger canister中是一个简化的并行区块链，它与其他网络管理canister一起在互联网计算机的子网中运行。

ledger canister实现了一个智能合约，该合约持有账户和余额，并保留影响账户和余额的交易历史。记录交易用以跟踪以下特定事件：

* 铸造 ICP 代币。
* 转移 ICP 代币。
* 燃烧 ICP 代币。

## principal

首次使用 DFINITY Canister SDK 时，dfx 命令行工具会使用 PEM 文件中的公钥/私钥对为您创建默认的开发人员身份。开发人员身份是由派生的principal数据类型和代表principal的文本进行表示，也就是principal identifier(主体身份)。

开发人员身份还可用于派生出account id（帐户标识符,类似于比特币或以太坊地址），用于在ledger canister中持有 ICP 代币。

## wallet

在 Internet Computer上，钱包是一种专门用于cycle的应用程序。钱包应用程序被实现为一个canister，并在互联网计算机上运行。

钱包使您能够管理cycle余额，将ICP代币转换为cycle，将cycle分发到您自己或其他用户的canister中作为访问或提供互联网服务的一种方式。

当需要创建canister或者操作canister时，必须要绑定一个wallet canister到当前主体标识才可以进行操作。

## smart contract

智能合约是一种软件，可以在不需要任何中央机构或法律系统的情况下，在分布式、去中心化的区块链网络上执行可信交易和协议。

通过智能合约，交易或协议的条款直接写入在区块链网络上执行的代码行中。 代码控制执行，交易防篡改、可追溯、不可逆。在互联网计算机上，智能合约被部署到canister中。

## WebAssembly

WebAssembly (Wasm) 是一种底层计算机指令格式。因为 WebAssembly 定义了一种可移植的、开放标准的二进制格式，可以在大多数现代计算机硬件上清晰地抽象出来，所以被在Internet上运行的程序广泛支持。用Motoko编写的程序被编译为 WebAssembly 代码，以便在 Internet 计算机副本上执行。

## replica

在互联网计算机区块链的上下文中，replica是指运行在网络中的物理计算机节点上的Internet Computer进程。

对于 DFINITY Canister SDK，使用 dfx start 和 dfx stop 命令在本地启动和停止replica进程，为开发提供本地网络。

# 账户

在IC中有如下三个账户：

* identity：identity账户是主体标识，其可以与ICP账户(account id)相互映射，并且identity是控制canister的钥匙。
* ledger：ledger账户是ICP账户(account id)，所有和ICP相关的操作都在ledger中进行。
* wallet：wallet账户是部署了cycle wallet程序的canister，identity必须设置了wallet才能部署canister。

在整个IC中账户操作有如下一些：

1. DFX canister SDK第一次使用时就会关联一个主体标识(principal identity)。使用命令：

|  |
| --- |
| dfx identity get-principal |

可以获取当前主体标识。

1. 每个主体标识都会派生出一个在leger canister中唯一标识的account id，类似address。使用命令：

|  |
| --- |
| dfx ledger --network ic account-id |

可以获取当前的ICP地址。

1. 第一个canister需要使用leger canister创建，并指定canister的controller，并会向其中存入指定数量cycle。使用命令：

|  |
| --- |
| dfx ledger --network ic create-canister tsqwz-udeik-5migd-ehrev-pvoqv-szx2g-akh5s-fkyqc-zy6q7-snav6-uqe --amount .25 |

可以创建一个canister，并指定了controller主体以及初始化的cycle。

1. 第一个创建的canister是空的，需要对其部署wallet程序来创建一个wallet canister。使用命令：

|  |
| --- |
| dfx identity --network ic deploy-wallet gastn-uqaaa-aaaae-aaafq-cai |

可以在创建的一个canister中部署wallet程序。该canister称为wallet canister。

1. 默认情况下部署wallet后，该wallet canister会作为当前主体的cycle wallet，也可以使用如下命令设置当前主体的cycle wallet：

|  |
| --- |
| dfx identity --network ic set-wallet --force gastn-uqaaa-aaaae-aaafq-cai |

成功设置后，当前identity使用该cycle wallet进行canister操作。

1. 当前主体存在cycle wallet后，就可以使用canister相关操作。例如命令：

|  |
| --- |
| dfx deploy --network ic |

部署当前project中的所有canister，部署完成后，这些canister的controller为当前主体。

# Quick Start

## Local development

1. 安装nodejs和npm。
2. 安装dfx canister sdk。

|  |
| --- |
| sh -ci "$(curl -fsSL https://sdk.dfinity.org/install.sh)" |

1. 新建项目。

|  |
| --- |
| dfx new hello; cd hello; |

1. 启动local网络。

|  |
| --- |
| dfx start |

1. 进行工程编译。

|  |
| --- |
| npm install |

1. 进行canister部署。

|  |
| --- |
| dfx deploy |

## Network deployment

1. 安装nodejs和npm。
2. 安装dfx canister sdk。

|  |
| --- |
| sh -ci "$(curl -fsSL https://sdk.dfinity.org/install.sh)" |

1. 新建项目。

|  |
| --- |
| dfx new hello; cd hello; |

1. 启动local网络。

|  |
| --- |
| dfx start |

1. 获取当前主体标识。

|  |
| --- |
| dfx identity get-principal |

1. 获取当ICP账户地址，并向其转账。

|  |
| --- |
| dfx ledger account-id |

1. 创建cycle wallet的canister。

|  |
| --- |
| dfx ledger --network ic create-canister <principal-identifier> --amount <icp-tokens> |

1. 部署cycle wallet程序至新创建的canister。

|  |
| --- |
| dfx identity --network ic deploy-wallet <canister-identifer> |

1. 进行工程编译。

|  |
| --- |
| npm install |

1. 进行canister部署。

|  |
| --- |
| dfx deploy |

# 项目结构

当使用命令dfx new hello时，会创建具有如下结构的目录：

|  |
| --- |
| hello/  ├── README.md # default project documentation  ├── dfx.json # project configuration file  ├── node\_modules # libraries for front-end development  ├── package-lock.json  ├── package.json  ├── src # source files directory  │ ├── hello  │ │ └── main.mo  │ └── hello\_assets  │ ├── assets  │ │ ├── logo.png  │ │ ├── main.css  │ │ └── sample-asset.txt  │ └── src  │ ├── index.html  │ └── index.js  └── webpack.config.js |

其中：

* dfx.json是整个工程的配置信息。
* package.json中是前端项目的依赖以及打包信息。
* src目录下存放整个工程的源文件。
* 约定src/xxx/中存放后端项目的源文件。
* 约定src/xxx\_assets/中存放前端项目的源文件。
* 约定src/xxx\_assets/assets/中存放前端项目的资源文件。
* 约定src/xxx\_assets/src/index.html为前端项目的入口html文件。
* 约定src/xxx\_assets/src/index.js为前端项目的会自动载入的且唯一的JavaScript脚本。

# dfx配置文件

当使用命令dfx new hello时，会创建具有如下内容的dfx.json工程配置文件：

|  |
| --- |
| {  "canisters": {  "hello": {  "main": "src/hello/main.mo",  "type": "motoko"  },  "hello\_assets": {  "dependencies": [  "hello"  ],  "frontend": {  "entrypoint": "src/hello\_assets/src/index.html"  },  "source": [  "src/hello\_assets/assets",  "dist/hello\_assets/"  ],  "type": "assets"  }  },  "defaults": {  "build": {  "packtool": ""  }  },  "dfx": "0.7.2",  "networks": {  "local": {  "bind": "127.0.0.1:8000",  "type": "ephemeral"  }  },  "version": 1  } |

其中：

* canisters指定了工程需要部署的canisters。assets为前端类型的canister，motoko为后端类型的canister。
  + 对于后端类型，main指定其编译入口文件。
  + 对于前端类型，frontend.entrypoint指定编译入口文件。该文件及其同名的js文件在编译后会在dist/<canister\_name>/目录下。
  + 对于前端类型，source指定编译后，需要进行部署的所有文件。
  + 对于前端类型，如果存在后端依赖，则需要使用dependencies进行指定。
* networks指定连接的网络信息。默认设置将本地 Internet 计算机网络绑定到本地主机地址 127.0.0.1 和端口 8000。
* defaults指定使用非motoko语言进行编程时的一些设置。