第一个账号 与 SOLANA 程序

通过 Solana CLI 理解 Solana 账户

登链社区 - Tiny熊

要点

- 准备账户：Solana 账号创建
- 三个简单的案例，演示开发 Solana 程序全过程
  - 项目创建 、程序编写、编译、部署、交互
  - 探究程序账户、升级权限
  - 理解程序操作数据账户、PDA 账户

账号 - 创建（CLI）

- 一切皆账号，账号默认的 owner 是系统程序
  - 部署程序时，先生成程序账号

```
> solana-keygen new --outfile <FILE_PATH> keypair 文件
> solana-keygen new -o my.json   # 可直接导入的 Phantom 等钱包

# 靓号
> solana-keygen grind --starts-with tiny:1

查看地址：
> solana address
> solana address  [-k ~/.config/solana/second_id.json]
> solana address -k my.json
```

Solana Cli 文档： https://docs.anza.xyz/cli/usage#usage

账号 - 领水

- Devnet 领水： https://faucet.solana.com/
- 本地测试网：
  - 启动 solana-test-validator ， 默认 rpc url: http://127.0.0.1:8899

```
> solana airdrop 5

> solana airdrop 5 -u http://127.0.0.1:8899
```

账户 - 获取余额

先查看网络设置：

```
> solana config get

> solana config set --url devnet
> solana config set --url $DEVNET_RPC

> solana balance
> solana balance <address>
> solana balance  <—keypair my.json>

> solana balance <address> -u http://127.0.0.1:8899
```

SOLANA 程序

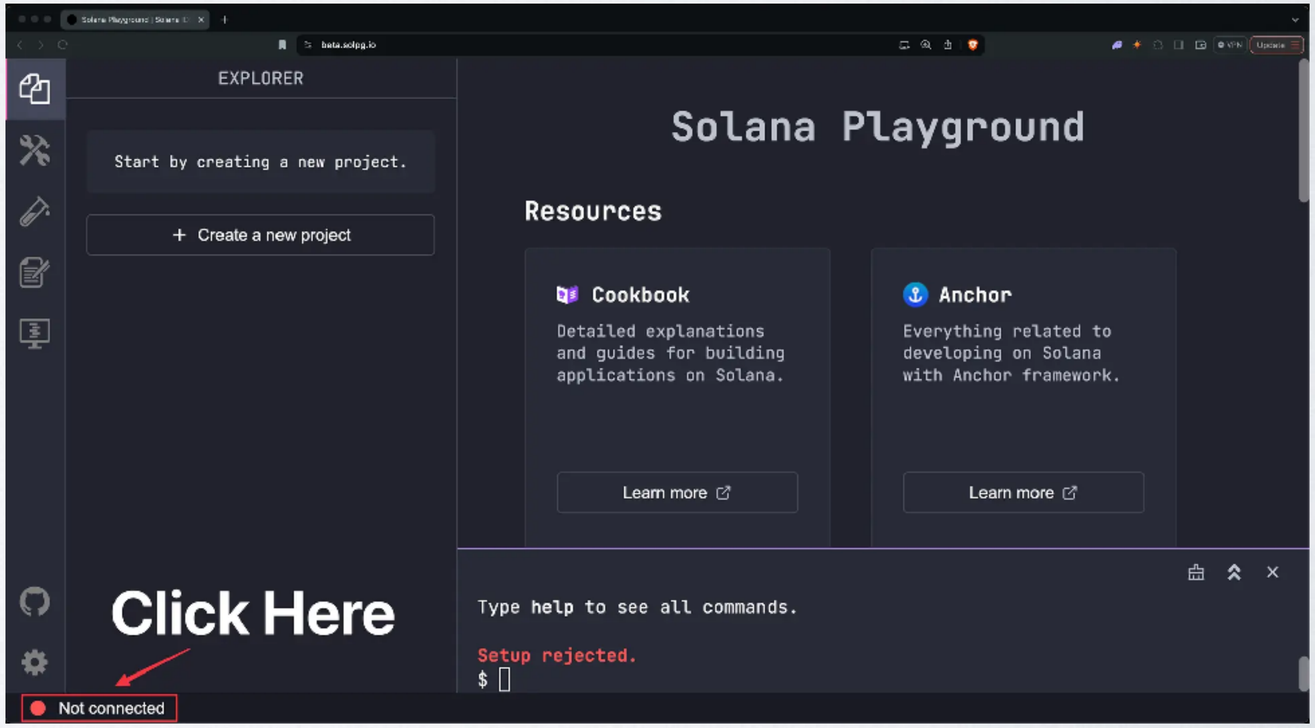
- 程序是包含可执行代码的账户， 处理用户发送的指令
- 程序本身是无状态的（stateless），但通过代码会控制（创建和修改）一些 PDA 数据账户（作为 PDA 账户的 owner ）
- 程序默认是可以升级的

开发 SOLANA 程序

- 原生 Rust
- Anchor：Solana 程序开发框架
  - 内置安全检测、序列化、账户处理等
  - 用**宏**及属性来简化程序编写
  - 完善的命令命令行工具，简化构建、测试、部署流程
- Anchor 组成：
  - anchor-lang：Rust 宏定义框架
  - anchor-cli：部署、测试工具
  - client-sdk：Rust / TypeScript 客户端交互库

SOLANA PLAYGROUND

- https://beta.solpg.io/ 在线 IDE —（类似于 Remix）
- 开发、部署、测试简单的 Solana 程序

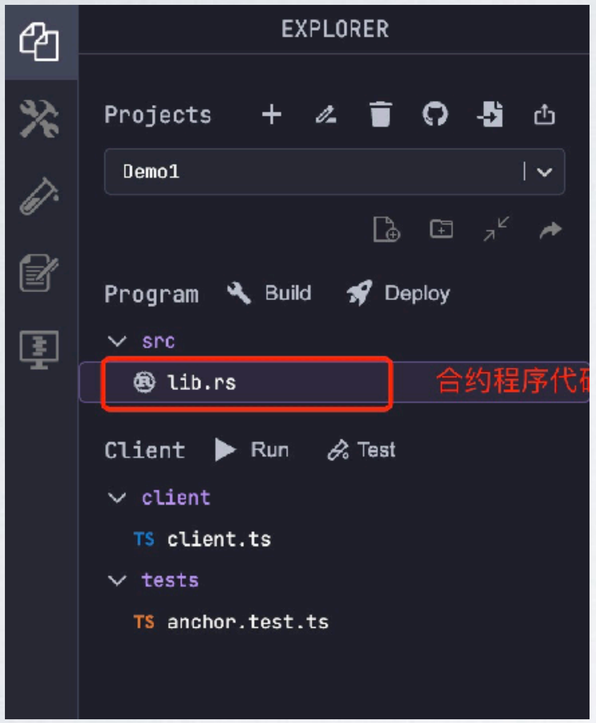


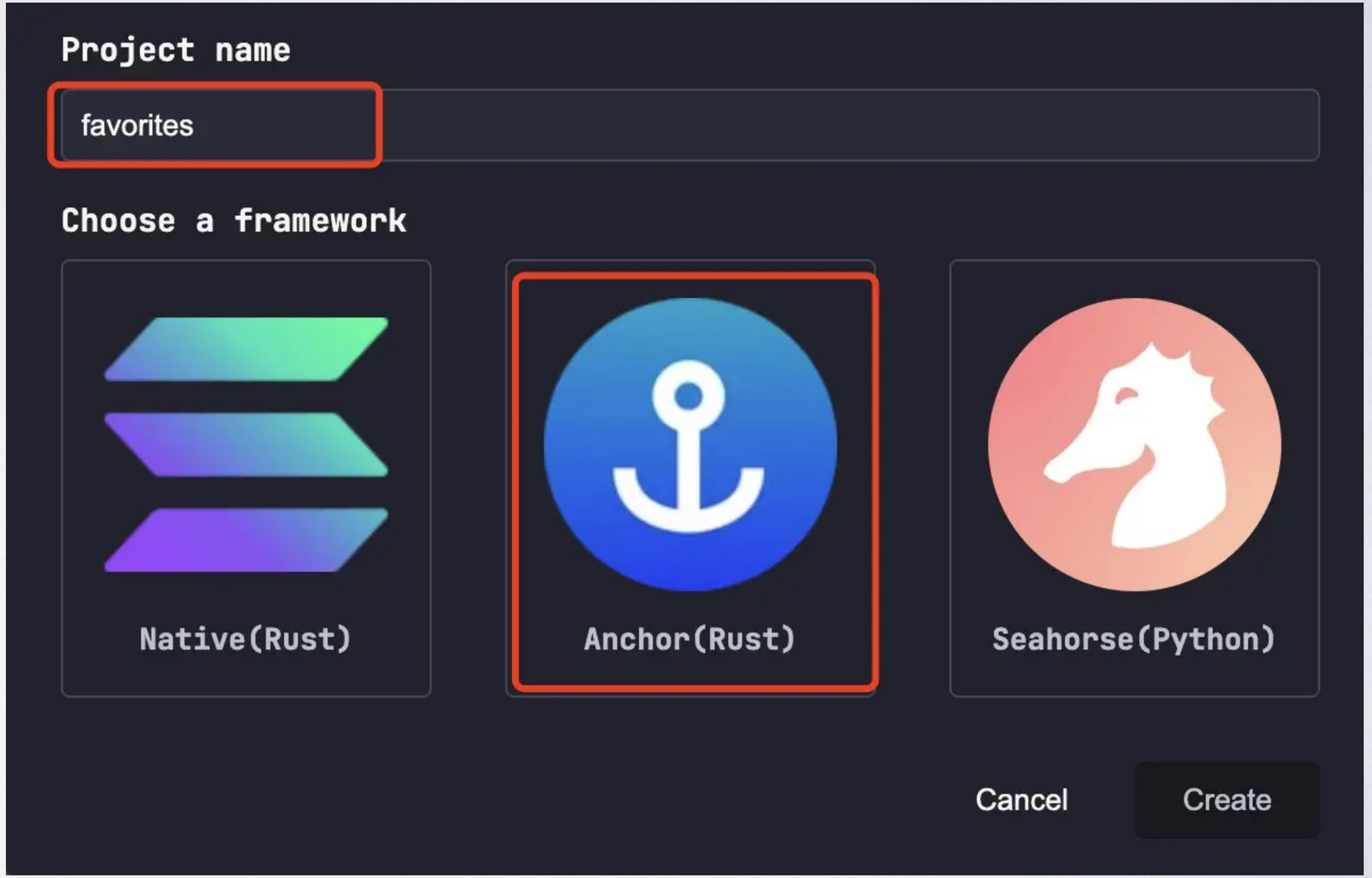
DEMO

- 项目创建
- 合约编写
- 编译
- 部署
- 交互（测试）

案例 1
一个简单的加法

项目创建


EXPLORER
Projects
Demo1
Program
Build
Deploy
src
lib.rs
合约程序代
Client
Run
Test
client
client.ts
tests
anchor.test.ts


Project name
favorites
Choose a framework
Native(Rust)
Anchor(Rust)
Seahorse(Python)
Cancel
Create

合约编写 - ADD

```
use anchor_lang::prelude::*;

declare_id!("GeqvtPd6x9MUjXmZb2panGiw9N2kALnN45AEkUfVSLXy");

#[program]
mod Adder {
    use super::*;
    pub fn add(ctx: Context<Add>, d1: u64, d2: u64) -> Result<()> {
        msg!("Sum is: {}!", d1 + d2);
        Ok(())
    }
}

#[derive(Accounts)]
pub struct Add<> {
}
```

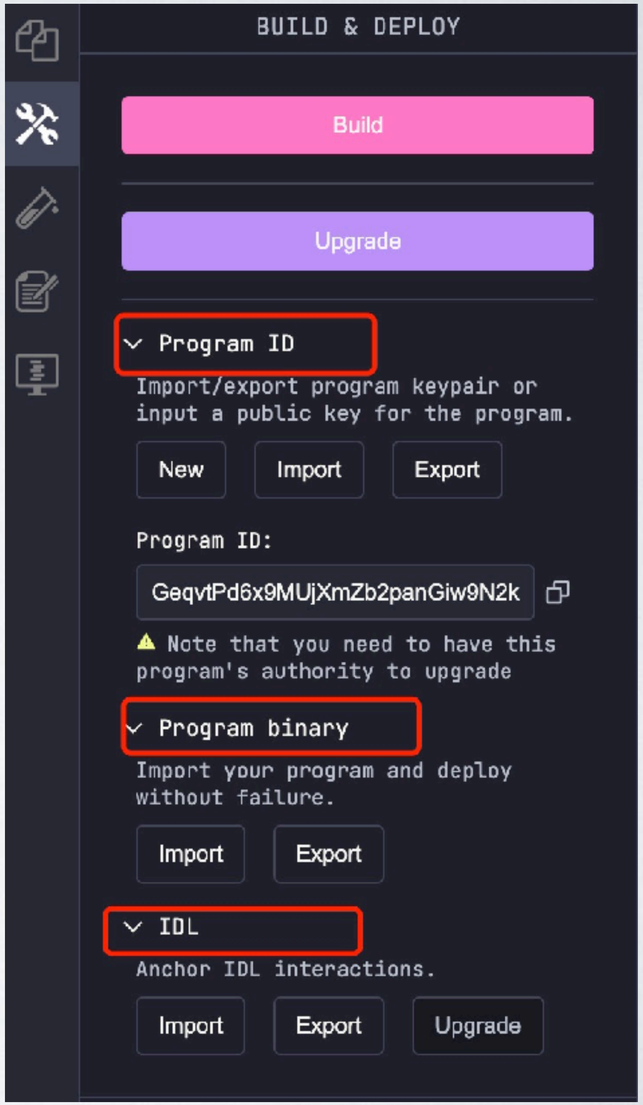
https://github.com/lbc-team/hello_solana/blob/main/solpg_demo/demo1_lib.rs

程序结构

- Anchor 程序使用宏来简化开发:
  - use anchor_lang::prelude::*; // 导入 anchor 实现的各种结构和宏
  - declare_id: 声明程序的链上地址(program ID)
  - #[program]: 处理指令 的Solana 程序模块
  - #[derive(Accounts)]: 指令所需的账户列表，包装在 Context 结构中
  - #[account]: 创建程序的自定义数据账户

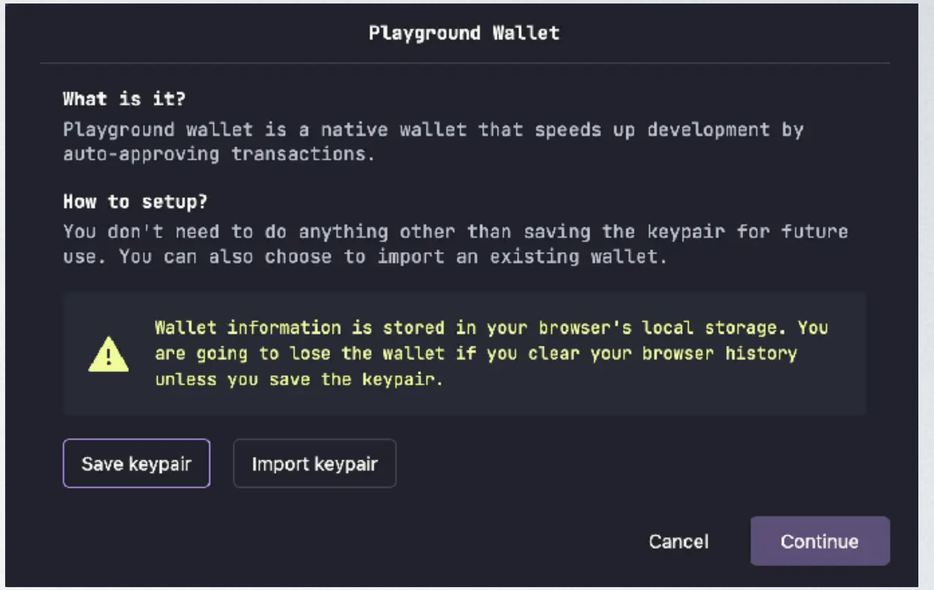
编译 BUILD

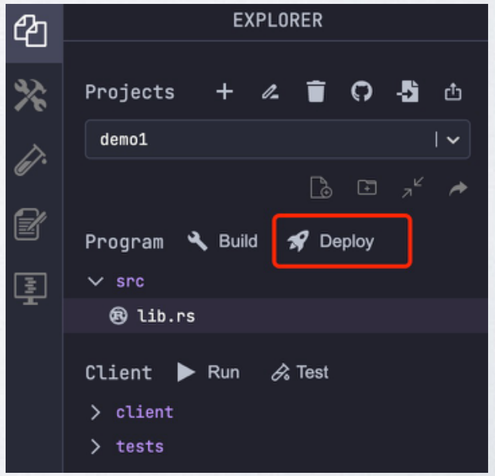
- Anchor 编译完成会生成（也可自己创建程序账号）：
  - Keypair.json (Program ID) : 在 Ed25519 曲线上点
    - 和 declare_id!() 一致， 程序会部署到该地址
  - 程序字节码（.so 文件）
  - IDL： 描述如何与Solana程序交互的JSON文件



部署

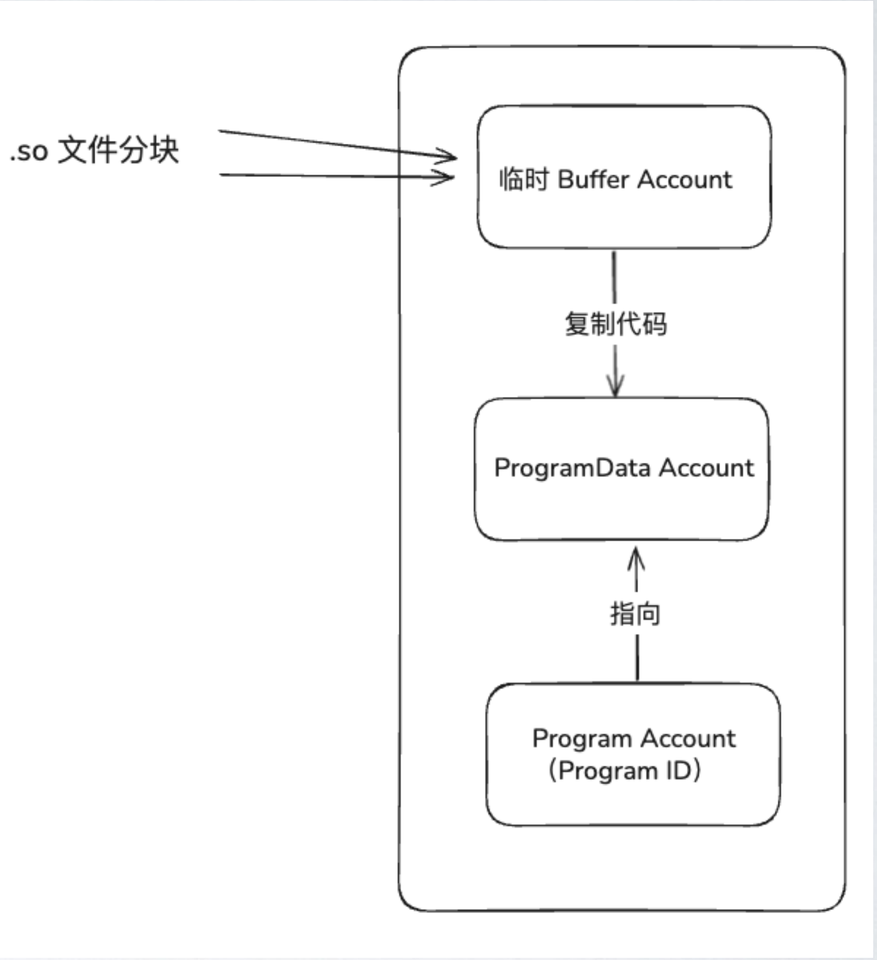
- Account:
  - 账号（创建或使用之前领好水的账号）
- RPC 节点
  - 各网络的节点 RPC： https://solana.com/zh/rpc
    - Helius 节点服务 Solana 不错的节点服务商
  - 本地节点，启动： solana-test-validator





部署

- 部署动作的背后:
  - 自动拆分 .so 、分批上传 Buffer；
  - 创建 Program id / ProgramData （升级时更新）；
  - 设置权限，销毁 Buffer
- ProgramData： 保存字节码、升级权限authority 、部署 slot 时间戳
  - 升级时，替换掉原有的字节码
- ProgramID： Build 阶段生成的账号，与程序交互的地址。



查看程序信息

```
# 查看某个 Program 的状态
> solana program show <PROGRAM_ID>  -u http://127.0.0.1:8899

# 查看账户的基本信息
> solana account <ACCOUNT> -u http://127.0.0.1:8899
```

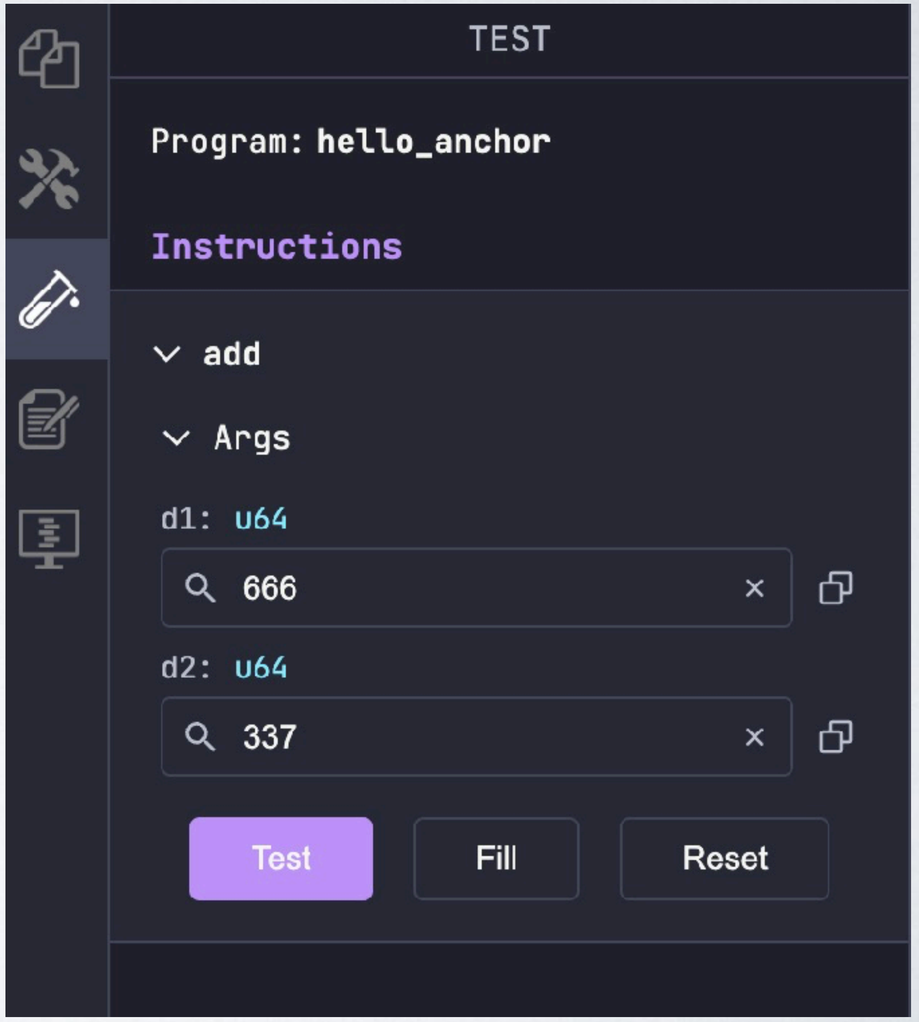
```
➜  hello_solana git:(main) solana program show BqVNxB4bggMbrAkiV6v5cyREqS4VFjD2d8i1ZAV8C9v5 -u http://127.0.0.1:8899

Program Id: BqVNxB4bggMbrAkiV6v5cyREqS4VFjD2d8i1ZAV8C9v5
Owner: BPFLoaderUpgradeab1e11111111111111111111111
ProgramData Address: 21W8FQjPqF2WNtF11x7LtpsRVu8W7YgzwgFYYyM9NYmZ
Authority: F5asSuRbkTfkufTctmM9mZudnUcYuFCTib1LDxdveuN7
Last Deployed In Slot: 599063
Data Length: 366624 (0x59820) bytes
Balance: 2.55290712 SOL
```

交互

- 交互后，在浏览器可以看到交易指令及日志

监听程序的日志:

```
solana logs <programID> -u http://127.0.0.1:8899
```



案例 2
创建账户存数据

```
use anchor_lang::prelude::*;
declare_id!("GeqvtPd6x9MUjXmZb2panGiw9N2kALnN45AEkUfVSLXy");

#[program]
mod hello_anchor {
    use super::*;
    pub fn initialize(ctx: Context<Initialize>, data: u64) -> Result<()> {
        ctx.accounts.new_account.data = data;
        msg!("Changed data to: {}!", data);
        Ok(())
    }
}

#[derive(Accounts)]
pub struct Initialize<'info> {
    #[account(init, payer = signer, space = 8 + 8)]
    pub new_account: Account<'info, NewAccount>,

    #[account(mut)]
    pub signer: Signer<'info>,

    pub system_program: Program<'info, System>,
}

#[account]
pub struct NewAccount {
    data: u64,
}
```

init: 创建这个账号

payer: 支付手续费

space: 账户空间

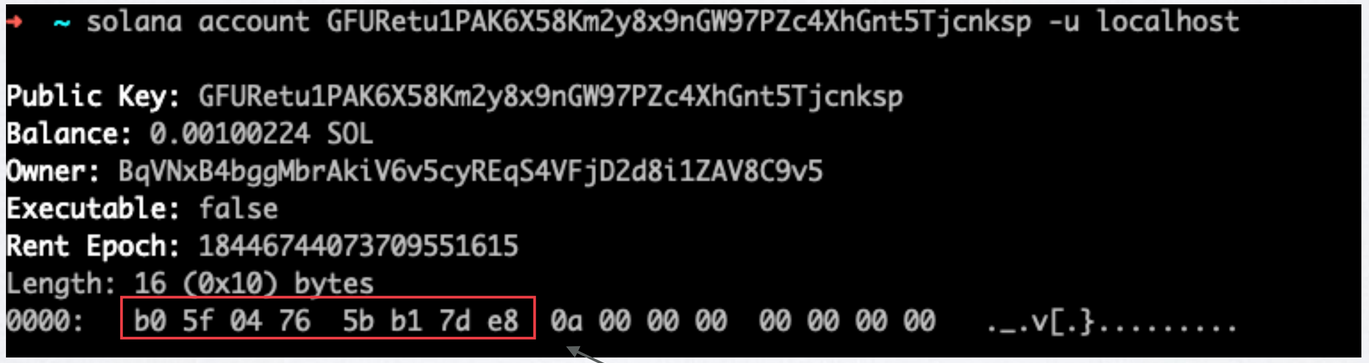
组织数据的存储

决定如何序列化

可查看程序创建的账户信息

- 默认创建账号的 owner 将修改为程序 id , 程序拥有账户控制权（修改数据）

```
# 查看账户的基本信息
> solana account <ACCOUNT> -u http://127.0.0.1:8899
```



Discriminator表示是一个 NewAccount 类型数据

案例 3
为每个用户各自的存储账户（PDA）
保存用户喜欢的 数字和颜色

solidity

```
struct Favorites {
    uint number;
    string color;
}

mapping(address => Favorites) public userFavorites;
```

Solana

Seeds

每个用户关联一个 PDA

```
pub struct Favorites {
    pub number: u64,
    pub color: String,
}
```

PDA 账户

```rust
use anchor_lang::prelude::*;
declare_id!("ww9C83noARSQVBnqmCUmaVdbJjmiwcV9j2LkXYMoUCV");

pub const ANCHOR_DISCRIMINATOR_SIZE: usize = 8;

#[program]
pub mod favorites {
    use super::*;

    pub fn set_favorites(
        context: Context<SetFavorites>, number: u64,  color: String,
    ) -> Result<()> {
        msg!("Greetings from {}", context.program_id);
        let user_public_key = context.accounts.user.key();
        msg!(
            "User {user_public_key}'s favorite number is {number}, favorite color is: {color}",
        );

        context.accounts.favorites.set_inner(Favorites {
            number,
            color,
        });
        Ok(())
    }
}

#[derive(Accounts)]
pub struct SetFavorites<'info> {
    #[account(mut)]
    pub user: Signer<'info>,

    #[account(
        init_if_needed,
        payer = user,
        space = ANCHOR_DISCRIMINATOR_SIZE + Favorites::INIT_SPACE,
        seeds=[b"favorites", user.key().as_ref()],
        bump
    )]
    pub favorites: Account<'info, Favorites>,

    pub system_program: Program<'info, System>,
}

// What we will put inside the Favorites PDA
#[account]
#[derive(InitSpace)]
pub struct Favorites {
    pub number: u64,

    #[max_len(50)]
    pub color: String,

}
```

代码：hello_solana/blob/main/solpg_demo/demo3_libs.rs

作业

- 编写一个简单的计数器程序，包含两个指令：
  - initialize(ctx)： 用 seed 派生出账户，初始化 count = 0
  - increment(ctx)： 将账户中的 count 加 1

https://decert.me/challenge/90c331f2-6a0e-4a68-bc32-a50e1879a4bb