合约文档（第一版）

合约已部署到开发网，地址`EaHoDFV3PCwUEFjU6b5U4Y76dW5oP7Bu1ndga8WgksFU`，可通过`client.ts`调用合约方法,该文件需要安装`nodejs`和`typescipt`语言相关库，环境配置完成后，执行以下命令

```
cd app
npm install
tsc
node ./client.js
```

注意：合约中写死的钱包地址需要修改为测试者的钱包地址，在client.ts中搜索Keypair.fromSecretKey()修改其中的数据

目前有3个接口可供调试

`createToken`：创建代币的所有信息

`mintToken`：向指定账户铸币

`burnToken`：账户所有者销毁自己的代币

`createPool`: 创建流动池和账户，初始化池子保有的Token和Sol
注意：此处创建去中心化的池子，非raydium

`buyTokenBaseSol`: 基于Sol，从pool购买Token
注意：此处在去中心化的池子交易，非raydium
注意：当前购买逻辑固定为1`Sol` = 100`Token`

`buyTokenBaseMeme`: 基于购买的meme数量购买Token
注意：此处在去中心化的池子交易，非raydium
注意：当前购买逻辑固定为1`Sol` = 100`Token`

另有4个接口暂时不能调试

`proxy_initialize`: 初始化raydium池子

`proxy_deposit`：向池子添加流动性

`proxy_swap_base_input`：基于付款数量的swap

`proxy_swap_base_output`：基于购买数量的swap

1. `createToken`

1.1 传入参数

```rust
#[derive(Accounts)]
#[instruction(params: CreateTokenParams)]
pub struct CreateToken<'info> {
    #[account(mut)]
    /// CHECK: UncheckedAccount
    pub metadata: UncheckedAccount<'info>,          // 元数据的账户地址
    #[account(
        init,
        seeds = [b"mint", params.id.as_bytes()],
        bump,
        payer = payer,
        mint::decimals = params.decimals,
        mint::authority = payer,
    )]
    pub mint: Account<'info, Mint>,                 // 代币的Mint地址
    #[account(mut)]
    pub payer: Signer<'info>,                      // 代币创建者+gas付费者
    pub rent: Sysvar<'info, Rent>,                // 四个系统固定程序地址
    pub system_program: Program<'info, System>,
    pub token_program: Program<'info, Token>,
    pub token_metadata_program: Program<'info, Metaplex>,
}

// Token的独有信息
#[derive(AnchorSerialize, AnchorDeserialize, Debug, Clone)]
pub struct CreateTokenParams {
    pub name: String,
    pub symbol: String,
    pub uri: String,                           // Token信息的打包json
    pub decimals: u8,                          // Token精度
    pub id: String,                            // Tokenid唯一标志，建议随机生成
}
```

1.2 调用

参考client.ts

1.3 链上的消息事件

```rust
#[event]
pub struct EVENTCreateToken {
    pub name: String,
    pub symbol: String,
    pub uri: String,
    pub decimals: u8,
    pub mint: Pubkey,                // Mint地址
    pub metadata_account: Pubkey,    // 元数据地址
    pub token_id: String,            // TokenId
}
```

链下监听器

```
// 创建Create函数监听器
  const listenerCreateToken = program.addEventListener(
    "EVENTCreateToken",
    (event, slot) => {
      console.log(
        `EVENTCreateToken: id = ${event.tokenId}, name =
${event.name},symbol = ${event.symbol}`
      );
    }
  );
```

1.4 创建结果

创建成功可在 https://explorer.solana.com/address/Cbb3yypZ21pEgua8UC5Z5TB9Egw5UhkYH9CQfFw3p4Vu/tokens?cluster=devnet查看Token信息，其中 Cbb3yypZ21pEgua8UC5Z5TB9Egw5UhkYH9CQfFw3p4Vu是新Token的Mint地址，该地址可在client.js的输出日志中搜索metadatamint address: xxxxxxxxx找到

2. mintToken

2.1 传入参数

```
#[derive(Accounts)]
#[instruction(params: MintTokenParams)]
pub struct MintTokens<'info> {
    #[account(
        mut,
        seeds = [b"mint", params.id.as_bytes()],
        bump,
        mint::authority = payer, // ToDo: payer
    )]
    pub mint: Account<'info, Mint>,             // Token的地址
    #[account(mut)]
    pub destination: Account<'info, TokenAccount>, // 目标的tokenaccount,可
以是创建者的、也可以是其他人的
    #[account(mut)]
    pub payer: Signer<'info>,             // token创建者+gas付费
    pub rent: Sysvar<'info, Rent>,
    pub system_program: Program<'info, System>,
    pub token_program: Program<'info, Token>,
    pub associated_token_program: Program<'info, AssociatedToken>,
}

#[derive(AnchorSerialize, AnchorDeserialize, Debug, Clone)]
```

```
pub struct MintTokenParams {
    pub id: String,
    pub quantity: u64,      // 铸币数量
}
```

2.2 调用

参考client.ts

2.3 链上的消息事件

```
#[event]
pub struct EVENTMintToken {
    pub mint: Pubkey,
    pub token_account: Pubkey,
    pub amount: u64,
    pub token_id: String,
}
```

链下监听器

```
// 创建Mint函数监听器
  const listenerMintToken = program.addEventListener(
    "EVENTMintToken",
    (event, slot) => {
      console.log(
        `EVENTMintToken: id= ${event.tokenId}, dest_token_account =
${event.tokenAccount.toBase58()},amount = ${
          event.amount
        }`
      );
    }
  );
```

2.4 Mint结果

铸币成功可在
`https://explorer.solana.com/address/8xU46cXdK7hc27qj8DEDuBdy8dAoTqwafAddgJCTmZcr/tokens?cluster=devnet`查看拥有的信息，其中`8xU46cXdK7hc27qj8DEDuBdy8dAoTqwafAddgJCTmZcr`是TokenAccount的所属者的钱包地址，该地址可在`client.js`的输出日志中搜索`My address: xxxxxxxxx`或者`UserPair.publickey: PublicKey [PublicKey(`找到

注意：是在钱包地址的页面查看，而不是钱包对应的TokenAccount页面查看

3. burnToken

3.1 传入参数

```
#[derive(Accounts)]
#[instruction(params: BurnTokenParams)]
pub struct BurnTokens<'info> {
    #[account(
        mut,
        seeds = [b"mint", params.id.as_bytes()],
        bump,
    )]
    pub mint: Account<'info, Mint>,     // 代币的mint地址
    #[account(
        mut,
        associated_token::mint = mint,
        associated_token::authority = payer
    )]
    pub token_account: Account<'info, TokenAccount>,    // 想要burn的
tokenaccount地址
    #[account(mut)]
    pub payer: Signer<'info>,       // 签名者+gas付费，注意这个账户必须拥有
token_account的所有权，即只能burn自己拥有的代币
    pub rent: Sysvar<'info, Rent>,
    pub system_program: Program<'info, System>,
    pub token_program: Program<'info, Token>,
    pub associated_token_program: Program<'info, AssociatedToken>,
}

#[derive(AnchorSerialize, AnchorDeserialize, Debug, Clone)]
pub struct BurnTokenParams {
    pub id: String,
    pub quantity: u64,
}
```

3.2 调用

参考client.ts

3.3 链上的消息事件

```
#[event]
pub struct EVENTBurnToken {
    pub mint: Pubkey,
    pub token_account: Pubkey,
    pub amount: u64,
```

```
    pub token_id: String,
}
```

链下监听器

```
// 创建burn函数监听器
  const listenerBurnToken = program.addEventListener(
    "EVENTBurnToken",
    (event, slot) => {
      console.log(
        `EVENTBurnToken: id= ${event.tokenId},token_account =
${event.tokenAccount.toBase58()},amount = ${
          event.amount
        }`
      );
    }
  );
```

3.4 burn结果

同2.4 Mint结果

4. createPool

4.1 传入参数

```
#[derive(Accounts)]
#[instruction(params: CreatePoolParams)]
pub struct CreateLiquidityPool<'info> {
    #[account(
        init,
        // Discriminator (8) + Pubkey (32) + Pubkey (32) + totalsupply (8)
        // + reserve one (8) + reserve two (8) + Bump (1)
        space = 8 + 32 + 32 + 8 + 8 + 8 + 1,
        payer = payer,
        seeds = [b"pool", mint.key().as_ref()],
        bump,
    )]
    pub pool: Account<'info, LiquidityPool>, // 流动性池的账户，在第一次调用接口
时创建

    #[account(
        mut,
        seeds = [b"mint", params.id.as_bytes()],
        bump,
        mint::authority = payer,
    )]
```

```
    pub mint: Box<Account<'info, Mint>>,    // mint账户

    #[account(
        mut,
        associated_token::mint = mint,
        associated_token::authority = pool
    )]
    pub pool_token_account: Account<'info, TokenAccount>, // 流动性池的Token
账户,存储pool中的Token

    #[account(mut)]
    pub payer: Signer<'info>,
    pub rent: Sysvar<'info, Rent>,
    pub system_program: Program<'info, System>,
    pub token_program: Program<'info, Token>,
    pub associated_token_program: Program<'info, AssociatedToken>,
}

#[derive(AnchorSerialize, AnchorDeserialize, Debug, Clone)]
pub struct CreatePoolParams {
    pub id: String,
    pub txid: String,
    pub initial_sol: u64,          // 流动性池中初始的Sol数量
    pub initial_meme: u64,         // 初始Token数量
}
```

4.2 调用

参考client.ts

4.3 链上的消息事件

```
#[event]
pub struct EVENTCreatePool {
    pub init_sol: u64,
    pub init_meme: u64,
    pub token_id: String,
    pub mint: Pubkey,
    pub pool: Pubkey,
    pub pool_token_account: Pubkey,
    pub txid: String,
}
```

链下监听器

```
  // 创建createPool函数监听器
  const listenerCreatePool = program.addEventListener(
```

```
    "EVENTCreatePool",
    (event, slot) => {
      console.log(
        `CreatePool: txid= ${event.txid}, token_id = ${event.tokenId}, pool
= ${event.pool.toBase58()},
        pooltokenaccount = ${event.poolTokenAccount.toBase58()}, initSol:
${event.initSol}, initMeme: ${event.initMeme}`,
      );
    }
  );
```

4.4 结果

同2.4 Mint结果

5. buyTokenBaseSol & buyTokenBaseMeme

5.1 传入参数

```
#[derive(Accounts)]
#[instruction(params: BuyTokenParams)]
pub struct BuyTokens<'info> {
    #[account(
        mut,
        associated_token::mint = mint,
        associated_token::authority = pool
    )]
    pub pool_token_account: Account<'info, TokenAccount>,
    #[account(mut)]
    pub destination: Account<'info, TokenAccount>,

    #[account(
        mut,
        seeds = [b"mint", params.id.as_bytes()],
        bump,
        mint::authority = payer,
    )]
    pub mint: Box<Account<'info, Mint>>,
    #[account(
        mut,
        seeds = [b"pool", mint.key().as_ref()],
        bump,
    )]
    pub pool: Account<'info, LiquidityPool>,
    #[account(mut)]
    pub payer: Signer<'info>,      // gai支付者 && 创建Token和pool的账户
    #[account(mut)]
    pub buyer: Signer<'info>,      // 购买Token的代付账户
    pub rent: Sysvar<'info, Rent>,
```

```
    pub system_program: Program<'info, System>,
    pub token_program: Program<'info, Token>,
    pub associated_token_program: Program<'info, AssociatedToken>,
}

#[derive(AnchorSerialize, AnchorDeserialize, Debug, Clone)]
pub struct BuyTokenParams {
    pub id: String,
    pub amount: u64,      // 2个接口唯一区别是，此处代表Sol还是Token的数量
    pub txid: String,
}
```

5.2 调用

参考client.ts

5.3 链上的消息事件

```
#[event]
pub struct EVENTBuyToken {
    pub token_account: Pubkey,
    pub sol_amount: u64,
    pub meme_amount: u64,
    pub token_id: String,
    pub txid: String,
}
```

5.4 结果

同2.4 Mint结果

注意：以下接口暂时不可调试,

---

10. proxy_initialize

4.1 传入参数

```
#[derive(Accounts)]
pub struct ProxyInitialize<'info> {
    pub cp_swap_program: Program<'info, RaydiumCpSwap>,
    /// Address paying to create the pool. Can be anyone
    #[account(mut)]
    pub creator: Signer<'info>,
```

```
    /// Which config the pool belongs to.
    pub amm_config: Box<Account<'info, AmmConfig>>,

    /// CHECK: pool vault and lp mint authority
    #[account(
        seeds = [
            raydium_cp_swap::AUTH_SEED.as_bytes(),
        ],
        seeds::program = cp_swap_program,
        bump,
    )]
    pub authority: UncheckedAccount<'info>,

    /// CHECK: Initialize an account to store the pool state, init by cp-
swap
    #[account(
        mut,
        seeds = [
            POOL_SEED.as_bytes(),
            amm_config.key().as_ref(),
            token_0_mint.key().as_ref(),
            token_1_mint.key().as_ref(),
        ],
        seeds::program = cp_swap_program,
        bump,
    )]
    pub pool_state: UncheckedAccount<'info>,

    /// Token_0 mint, the key must smaller then token_1 mint.
    #[account(
        constraint = token_0_mint.key() < token_1_mint.key(),
        mint::token_program = token_0_program,
    )]
    pub token_0_mint: Box<InterfaceAccount<'info, Mint>>,

    /// Token_1 mint, the key must grater then token_0 mint.
    #[account(
        mint::token_program = token_1_program,
    )]
    pub token_1_mint: Box<InterfaceAccount<'info, Mint>>,

    /// CHECK: pool lp mint, init by cp-swap
    #[account(
        mut,
        seeds = [
            POOL_LP_MINT_SEED.as_bytes(),
            pool_state.key().as_ref(),
        ],
        seeds::program = cp_swap_program,
        bump,
    )]
    pub lp_mint: UncheckedAccount<'info>,
```

```
    /// payer token0 account
    #[account(
        mut,
        token::mint = token_0_mint,
        token::authority = creator,
    )]
    pub creator_token_0: Box<InterfaceAccount<'info, TokenAccount>>,

    /// creator token1 account
    #[account(
        mut,
        token::mint = token_1_mint,
        token::authority = creator,
    )]
    pub creator_token_1: Box<InterfaceAccount<'info, TokenAccount>>,

    /// CHECK: creator lp ATA token account, init by cp-swap
    #[account(mut)]
    pub creator_lp_token: UncheckedAccount<'info>,

    /// CHECK: Token_0 vault for the pool, init by cp-swap
    #[account(
        mut,
        seeds = [
            POOL_VAULT_SEED.as_bytes(),
            pool_state.key().as_ref(),
            token_0_mint.key().as_ref()
        ],
        seeds::program = cp_swap_program,
        bump,
    )]
    pub token_0_vault: UncheckedAccount<'info>,

    /// CHECK: Token_1 vault for the pool, init by cp-swap
    #[account(
        mut,
        seeds = [
            POOL_VAULT_SEED.as_bytes(),
            pool_state.key().as_ref(),
            token_1_mint.key().as_ref()
        ],
        seeds::program = cp_swap_program,
        bump,
    )]
    pub token_1_vault: UncheckedAccount<'info>,

    /// create pool fee account
    #[account(
        mut,
        address= raydium_cp_swap::create_pool_fee_reveiver::id(),
    )]
    pub create_pool_fee: Box<InterfaceAccount<'info, TokenAccount>>,

    /// CHECK: an account to store oracle observations, init by cp-swap
```

```
    #[account(
        mut,
        seeds = [
            OBSERVATION_SEED.as_bytes(),
            pool_state.key().as_ref(),
        ],
        seeds::program = cp_swap_program,
        bump,
    )]
    pub observation_state: UncheckedAccount<'info>,

    /// Program to create mint account and mint tokens
    pub token_program: Program<'info, Token>,
    /// Spl token program or token program 2022
    pub token_0_program: Interface<'info, TokenInterface>,
    /// Spl token program or token program 2022
    pub token_1_program: Interface<'info, TokenInterface>,
    /// Program to create an ATA for receiving position NFT
    pub associated_token_program: Program<'info, AssociatedToken>,
    /// To create a new program account
    pub system_program: Program<'info, System>,
    /// Sysvar for program account
    pub rent: Sysvar<'info, Rent>,
}
```

11. proxy_deposit

5.1 传入参数

```
#[derive(Accounts)]
pub struct ProxyDeposit<'info> {
    pub cp_swap_program: Program<'info, RaydiumCpSwap>,

    /// Pays to mint the position
    pub owner: Signer<'info>,

    /// CHECK: pool vault and lp mint authority
    #[account(
        seeds = [
            raydium_cp_swap::AUTH_SEED.as_bytes(),
        ],
        seeds::program = cp_swap_program,
        bump,
    )]
    pub authority: UncheckedAccount<'info>,

    #[account(mut)]
    pub pool_state: AccountLoader<'info, PoolState>,
```

```
    /// Owner lp tokan account
    #[account(mut,  token::authority = owner)]
    pub owner_lp_token: Box<InterfaceAccount<'info, TokenAccount>>,

    /// The payer's token account for token_0
    #[account(
        mut,
        token::mint = token_0_vault.mint,
        token::authority = owner
    )]
    pub token_0_account: Box<InterfaceAccount<'info, TokenAccount>>,

    /// The payer's token account for token_1
    #[account(
        mut,
        token::mint = token_1_vault.mint,
        token::authority = owner
    )]
    pub token_1_account: Box<InterfaceAccount<'info, TokenAccount>>,

    /// The address that holds pool tokens for token_0
    #[account(
        mut,
        constraint = token_0_vault.key() ==
pool_state.load()?.token_0_vault
    )]
    pub token_0_vault: Box<InterfaceAccount<'info, TokenAccount>>,

    /// The address that holds pool tokens for token_1
    #[account(
        mut,
        constraint = token_1_vault.key() ==
pool_state.load()?.token_1_vault
    )]
    pub token_1_vault: Box<InterfaceAccount<'info, TokenAccount>>,

    /// token Program
    pub token_program: Program<'info, Token>,

    /// Token program 2022
    pub token_program_2022: Program<'info, Token2022>,

    /// The mint of token_0 vault
    #[account(
        address = token_0_vault.mint
    )]
    pub vault_0_mint: Box<InterfaceAccount<'info, Mint>>,

    /// The mint of token_1 vault
    #[account(
        address = token_1_vault.mint
    )]
    pub vault_1_mint: Box<InterfaceAccount<'info, Mint>>,
```

```
    /// Lp token mint
    #[account(
        mut,
        address = pool_state.load()?.lp_mint)
    ]
    pub lp_mint: Box<InterfaceAccount<'info, Mint>>,
}
```

12. proxy_swap_base_input

6.1 传入参数

```
#[derive(Accounts)]
pub struct ProxySwapBaseInput<'info> {
    pub cp_swap_program: Program<'info, RaydiumCpSwap>,
    /// The user performing the swap
    pub payer: Signer<'info>,

    /// CHECK: pool vault and lp mint authority
    #[account(
      seeds = [
        raydium_cp_swap::AUTH_SEED.as_bytes(),
      ],
      seeds::program = cp_swap_program,
      bump,
  )]
    pub authority: UncheckedAccount<'info>,

    /// The factory state to read protocol fees
    #[account(address = pool_state.load()?.amm_config)]
    pub amm_config: Box<Account<'info, AmmConfig>>,

    /// The program account of the pool in which the swap will be performed
    #[account(mut)]
    pub pool_state: AccountLoader<'info, PoolState>,

    /// The user token account for input token
    #[account(mut)]
    pub input_token_account: Box<InterfaceAccount<'info, TokenAccount>>,

    /// The user token account for output token
    #[account(mut)]
    pub output_token_account: Box<InterfaceAccount<'info, TokenAccount>>,

    /// The vault token account for input token
    #[account(
      mut,
      constraint = input_vault.key() == pool_state.load()?.token_0_vault ||
input_vault.key() == pool_state.load()?.token_1_vault
```

```
    )]
      pub input_vault: Box<InterfaceAccount<'info, TokenAccount>>,

      /// The vault token account for output token
      #[account(
        mut,
        constraint = output_vault.key() == pool_state.load()?.token_0_vault
|| output_vault.key() == pool_state.load()?.token_1_vault
    )]
      pub output_vault: Box<InterfaceAccount<'info, TokenAccount>>,

      /// SPL program for input token transfers
      pub input_token_program: Interface<'info, TokenInterface>,

      /// SPL program for output token transfers
      pub output_token_program: Interface<'info, TokenInterface>,

      /// The mint of input token
      #[account(
        address = input_vault.mint
    )]
      pub input_token_mint: Box<InterfaceAccount<'info, Mint>>,

      /// The mint of output token
      #[account(
        address = output_vault.mint
    )]
      pub output_token_mint: Box<InterfaceAccount<'info, Mint>>,
      /// The program account for the most recent oracle observation
      #[account(mut, address = pool_state.load()?.observation_key)]
      pub observation_state: AccountLoader<'info, ObservationState>,
}
```

13. proxy_swap_base_output

7.1 传入参数

```
#[derive(Accounts)]
pub struct ProxySwapBaseOutput<'info> {
    pub cp_swap_program: Program<'info, RaydiumCpSwap>,
    /// The user performing the swap
    pub payer: Signer<'info>,

    /// CHECK: pool vault and lp mint authority
    #[account(
      seeds = [
        raydium_cp_swap::AUTH_SEED.as_bytes(),
      ],
      seeds::program = cp_swap_program,
```

```
        bump,
    )]
    pub authority: UncheckedAccount<'info>,

    /// The factory state to read protocol fees
    #[account(address = pool_state.load()?.amm_config)]
    pub amm_config: Box<Account<'info, AmmConfig>>,

    /// The program account of the pool in which the swap will be performed
    #[account(mut)]
    pub pool_state: AccountLoader<'info, PoolState>,

    /// The user token account for input token
    #[account(mut)]
    pub input_token_account: Box<InterfaceAccount<'info, TokenAccount>>,

    /// The user token account for output token
    #[account(mut)]
    pub output_token_account: Box<InterfaceAccount<'info, TokenAccount>>,

    /// The vault token account for input token
    #[account(
        mut,
        constraint = input_vault.key() == pool_state.load()?.token_0_vault ||
input_vault.key() == pool_state.load()?.token_1_vault
    )]
    pub input_vault: Box<InterfaceAccount<'info, TokenAccount>>,

    /// The vault token account for output token
    #[account(
        mut,
        constraint = output_vault.key() == pool_state.load()?.token_0_vault
|| output_vault.key() == pool_state.load()?.token_1_vault
    )]
    pub output_vault: Box<InterfaceAccount<'info, TokenAccount>>,

    /// SPL program for input token transfers
    pub input_token_program: Interface<'info, TokenInterface>,

    /// SPL program for output token transfers
    pub output_token_program: Interface<'info, TokenInterface>,

    /// The mint of input token
    #[account(
        address = input_vault.mint
    )]
    pub input_token_mint: Box<InterfaceAccount<'info, Mint>>,

    /// The mint of output token
    #[account(
        address = output_vault.mint
    )]
    pub output_token_mint: Box<InterfaceAccount<'info, Mint>>,
    /// The program account for the most recent oracle observation
```

```
    #[account(mut, address = pool_state.load()?.observation_key)]
    pub observation_state: AccountLoader<'info, ObservationState>,
}

pub fn proxy_swap_base_output(
    ctx: Context<ProxySwapBaseOutput>,
    max_amount_in: u64,
    amount_out: u64,
) -> Result<()> {
    let cpi_accounts = cpi::accounts::Swap {
        payer: ctx.accounts.payer.to_account_info(),
        authority: ctx.accounts.authority.to_account_info(),
        amm_config: ctx.accounts.amm_config.to_account_info(),
        pool_state: ctx.accounts.pool_state.to_account_info(),
        input_token_account:
ctx.accounts.input_token_account.to_account_info(),
        output_token_account:
ctx.accounts.output_token_account.to_account_info(),
        input_vault: ctx.accounts.input_vault.to_account_info(),
        output_vault: ctx.accounts.output_vault.to_account_info(),
        input_token_program:
ctx.accounts.input_token_program.to_account_info(),
        output_token_program:
ctx.accounts.output_token_program.to_account_info(),
        input_token_mint: ctx.accounts.input_token_mint.to_account_info(),
        output_token_mint:
ctx.accounts.output_token_mint.to_account_info(),
        observation_state:
ctx.accounts.observation_state.to_account_info(),
    };
    let cpi_context =
CpiContext::new(ctx.accounts.cp_swap_program.to_account_info(),
cpi_accounts);
    cpi::swap_base_output(cpi_context, max_amount_in, amount_out)
}
```