# 基于长安链的数字资产管理

【实验目的】

（1）熟悉长安链 CMNFA 标准接口与事件约定，理解NFA（Non-Fungible Asset）非同质化资产的铸造、授权、转移、查询全流程。

（2）掌握在长安链中基于 Go 合约（standard-nfa） 实现与调用 NFA 资产管理。

（3）能独立完成：部署合约 → Mint → 授权/分类 → TransferFrom → OwnerOf/TokenURI → 统计/销毁。

**【实验环境】**

（1）Ubuntu 22.04，python：3.10。

（2）依赖：SmartIDE（v2.3.0+)、ChainMaker CLI（cmc）。

（3）链：本地多节点或教学用测试链（具备 org 证书与 cmc 调用权限）。

**【实验原理】**

本实验以长安链ChainMaker的 CMNFA标准为抽象接口与交互规范，利用官方的 standard-nfa（Go 合约）作为实现载体，完成对NFA（Non-Fungible Asset）的“确权—授权—转移—查询—销毁”全生命周期管理。NFA 的本质是“以唯一 Token ID 为索引的不可替代数字权属记录”，其核心状态包含三部分：所有权（Owner）、类别信息（Category）与元数据（Metadata）。在联盟链语境中，参与者以证书标识的组织身份接入链网，交易由客户端发起、节点共识排序、合约确定性执行、状态持久化到账本，确保每一次所有权变更都具备可验证的来源、不可篡改的历史与可追溯的路径。CMNFA 标准统一了方法名与事件名，如 Mint、TransferFrom、OwnerOf、TokenURI 以及对应的 mint/transfer/burn 事件，从而保证了不同业务系统、SDK 与工具之间的可互操作性；standard-nfa 合约则遵循该接口对状态与约束进行具体落地，实现“同标准、多实现”的生态兼容。

从数据模型看，NFA 以 tokenId 为主键唯一标识，链上状态至少维护三类映射：tokenId → owner（所有权）、tokenId → metadata/URI（元数据或其指针）、tokenId → categoryName（所属类别）；为满足高频检索与统计，还会维护账户维度与类别维度的派生索引（如 account → [tokenIds]、category → supply），以及全局计数器 totalSupply。元数据可存链上（节省依赖，但受体量限制）或以 URI/哈希 的方式指向链下存储（如对象存储、IPFS），链上仅保存不可抵赖的完整性摘要与定位信息；当类别被定义了 categoryURI 时，TokenURI(tokenId) 常规上表达为 categoryURI + tokenId 的拼接，既统一命名空间，又便于批量管理。上述设计使得链上账本承担“权属与索引”的职责，而大对象的内容分发交给更擅长存储与带宽的系统，这也是联盟链在性能与合规之间取得平衡的常见模式。

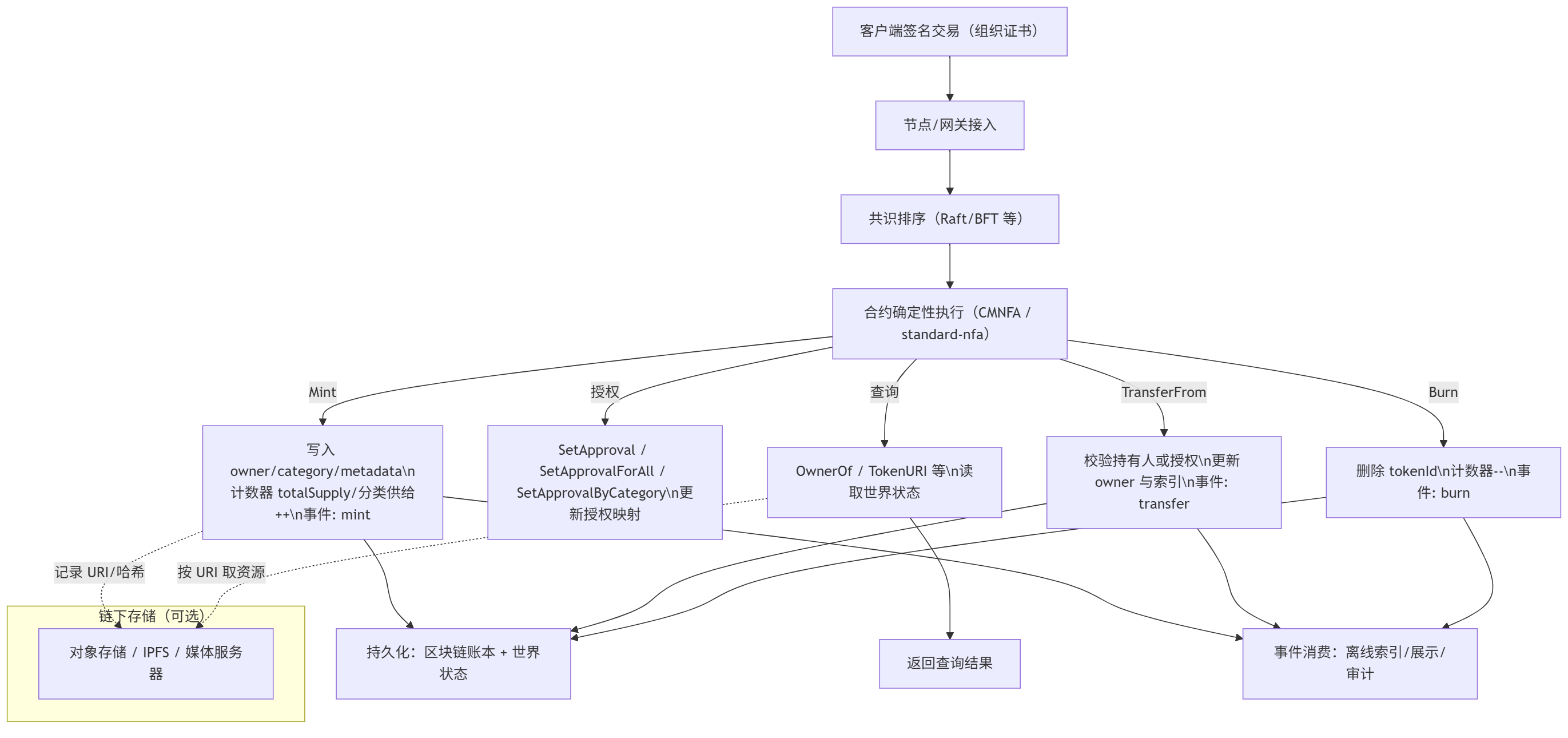
在权限与授权模型上，CMNFA 同时支持“精细到单 Token”与“批量到账号/类别”的授权路径：持有人可以对某一枚 tokenId 授权给经纪账户（SetApproval），也可以对自己的全部 NFA授予代理权（SetApprovalForAll），还可选择对某个类别的 NFA 一次性授权（SetApprovalByCategory）。这三种授权粒度覆盖了从单件流转到整批托管的真实业务场景：例如数字藏品平台需要对同一系列（Category）进行批量上架与转移，就无需逐一签发单 Token 授权。授权只授予“处置权”而非“所有权”——被授权方可代表持有人执行 TransferFrom，但状态变更一旦上链，新的所有权仍明确记录在 owner 字段里；任何转移都必须满足“要么请求方为当前持有人，要么已获得有效授权”的前置条件，合约在执行前进行断言校验以阻断未授权转移。

状态机与不变量保障了 NFA 的唯一性与安全性。不变量包括但不限于：tokenId 在铸造前必须不存在（防止重号）、在转移时必须存在且当前 owner 与来源 from 一致（防止虚构来源或双重花费）、在销毁时必须由持有人或具备相应权限的主体触发；任何会改变所有权或供给量的操作（Mint/Burn/Transfer）都伴随原子化的计数与索引维护（如 totalSupply、categorySupply、账户持有列表的增删），以保证读路径（OwnerOf、BalanceOf、AccountTokens、TotalSupply/OfCategory）始终与写路径严格一致。由于链上执行具有串行线性化的全序视图（由共识排序提供），并行提交在进入合约逻辑前已被确立唯一顺序，使得同一 tokenId 的竞争更新不会产生撕裂或重放成功的空间。

在交易与共识流程中，客户端首先以组织证书对调用请求签名，通过网关或直接与节点交互；节点将交易纳入待共识池，采用联盟链常见的一致性协议（如 Raft/BFT 家族）产生全网一致的区块顺序；随后各背书/执行节点在相同状态根上确定性地执行合约代码，输出相同的读写集与事件序列，最后持久化到账本（区块链）与世界状态（键值数据库）。这一流程确保了可重放验证与跨节点一致性：任一节点在收到同一区块后，都会产出相同的 NFA 状态变更与事件日志。事件作为链上“二级索引”，可被应用层消费，用于驱动离线索引构建、素材分发、审计对账与可视化大屏等，从而形成“链上确权 + 链下服务”的完整闭环。

类别（Category）机制是 CMNFA 在长安链业务中的关键扩展。传统 ERC-721 的集合概念更多存在于应用侧，而 CMNFA 将“类别”前置到合约层：每个类别可具备独立的 categoryName 与 categoryURI，既可实现分组统计与访问控制（如类目级授权与上架），又能为下游媒体/检索提供稳定的根路径。在运营上，这意味着平台可以按系列统一配置版权声明、展示规则与版税策略（若在扩展合约中实现），而在技术上，TotalSupplyOfCategory 与 GetCategoryByTokenId 等接口让“系列维度的可观测性”天然存在于链上状态，避免各系统各自解析元数据带来的口径不一致。

安全与合规方面，联盟链以证书体系和组织管理为根基：谁有权部署合约、谁能调用敏感方法（如 Mint/Burn）、是否需要多签或审批流，均可通过链级 ACL、合约内角色控制或外部治理合约约束；同时，元数据若涉及版权或隐私，可结合链下访问控制（鉴权下载、加密存储）与链上哈希校验实现“开源可证、内容受控”。在性能维度，NFA 操作以小对象键值更新为主，写入成本可控；若存在海量批量场景，可采用 MintBatch/TransferFromBatch 等批量接口降低交易与索引维护开销，并辅以事件驱动的异步读取优化（例如离线构建持有者—Token 列表）。



上述原理映射到实验流程：当调用 Mint 时，合约校验 tokenId 唯一性与调用者铸造权限，写入 owner/metadata/category 并递增供给计数，同时发出 mint 事件；SetApproval/SetApprovalForAll/SetApprovalByCategory 会更新授权映射，为随后 TransferFrom 的合法性提供依据；执行 TransferFrom 时合约再次核对持有人或授权关系，原子性更新 owner 与账户索引，并产出 transfer 事件；OwnerOf/TokenURI 等读接口直接读取世界状态验证结果，而 Burn 则回收 tokenId、递减供给并广播 burn 事件。通过比对每一步的交易哈希、区块高度、事件载荷与读接口返回，即可完整验证 NFA 生命周期在链上的可追溯与可审计性，这正是本实验希望学生切实掌握的“标准约定 + 状态机不变量 + 联盟链交易共识”三者合一的工程化方法论。

**【实验步骤】**

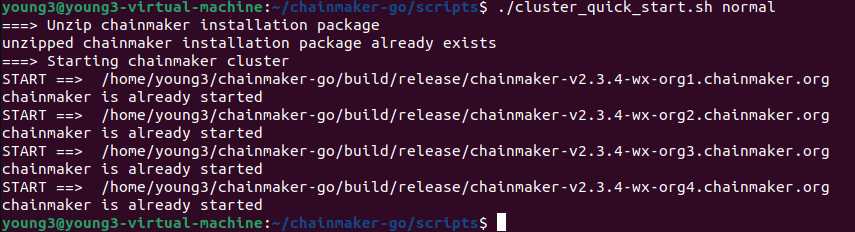
**任务一：环境连通与账号准备（通过命令行使用长安链）**

确认 CLI 已连上链、配置无误，并固定默认配置路径.

通过前面实验确保开启长安链节点：

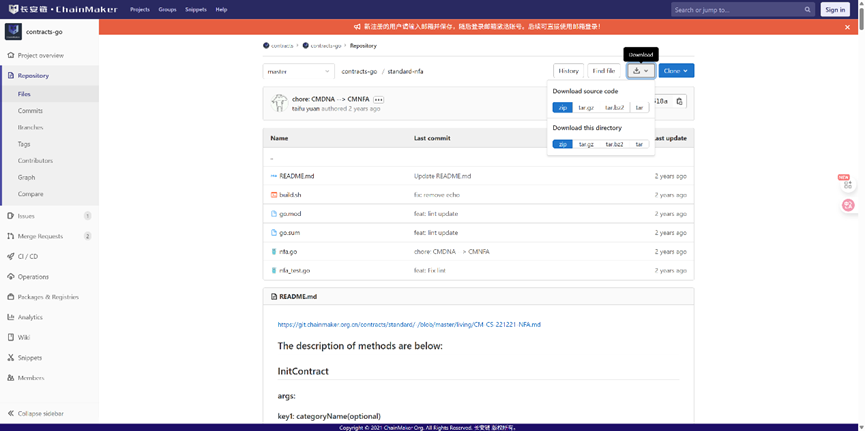
./cluster\_quick\_start.sh normal

出现如图所示结果，未报错即可**。**

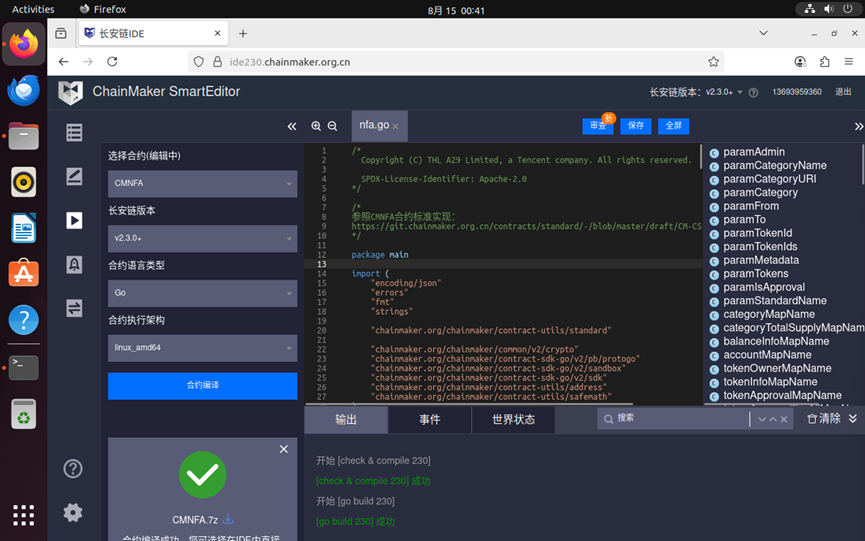


**任务二：部署 standard-nfa（CMNFA 标准 NFA 合约）**

1. 下载源码
2. 前往链接<https://github.com/A1LinLin1/CMNFA> 拷贝CMNFA.go代码，参考官方demo代码为：<https://git.chainmaker.org.cn/contracts/contracts-go/-/tree/master/standard-nfa>，在contracts-go 仓库中 standard-nfa/ 目录下的nfa.go文件代码。



1. 将源码粘贴到SmartIDE中（<https://ide230.chainmaker.org.cn/>），重新将文件编译成功后压缩为CMNFA.7z，方便后续部署。并放到~/chainmaker/chainmaker-go/tools/cmc目录下



进入到cmc工具目录下：

cd ~/chainmaker/chainmaker-go/tools/cmc

1. 设置环境变量

export SDK=./testdata/sdk\_config.yml

export CM=CMNFA

1. 部署1.0版本合约，。

./cmc client contract user create \

--contract-name "$CM" \

--runtime-type DOCKER\_GO \

--byte-code-path ./CMNFA.7z \

--version 1.0 \

--sdk-conf-path ./testdata/sdk\_config.yml \

--admin-org-ids wx-org1.chainmaker.org \ --admin-crt-file-paths ./testdata/crypto-config/wx-org1.chainmaker.org/user/client1/client1.sign.crt \

--admin-key-file-paths ./testdata/crypto-config/wx-org1.chainmaker.org/user/client1/client1.sign.key \

--sync-result=true

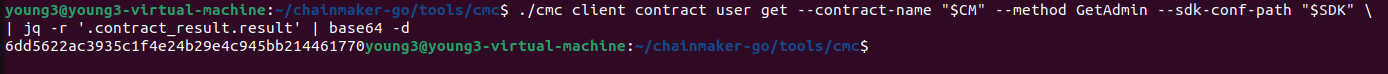
可以看到合约部署成功，地址为：dc2e430c7b155738dc0bf07c69aa2a9f55f81abf



1. 然后查看合约admin地址：

./cmc client contract user get --contract-name "$CM" --method GetAdmin --sdk-conf-path "$SDK" \

| jq -r '.contract\_result.result' | base64 -d



然后将admin地址设置为环境便令，方便后续使用。

export ADMIN\_HEX=**6dd5622ac3935c1f4e24b29e4c945bb214461770** #不同主机不同

1. 进行合约功能验证，读取CMNFA合约里的只读方法TotalSupply，即当前链上已发行的NFA总量。

./cmc client contract user invoke \

--contract-name "$CM" \

--method TotalSupply \

--sdk-conf-path "$SDK" \

--sync-result=true \

--timeout 30



其中 "result" 就是总量，base64是MA==，解码后就是0，表示当前发行量为0

**任务三：NFA的mint**

1. 首先需要创建一个分类，类名设置为team。

./cmc client contract user invoke \

--contract-name "$CM" \

--method CreateOrSetCategory \

--params '{"category":"{\"CategoryName\":\"team\",\"CategoryURI\":\"http://team.uri\"}"}' \

--sdk-conf-path ./testdata/sdk\_config.yml \

--user-signcrt-file-path ./testdata/crypto-config/wx-org1.chainmaker.org/user/client1/client1.sign.crt \

--user-signkey-file-path ./testdata/crypto-config/wx-org1.chainmaker.org/user/client1/client1.sign.key \

--sync-result=true \

--timeout 30



1. tokenId可以设置为队名队伍名称的拼音，如：team001，metadata可以采用自己的学号，如：2025985211等。

export TID=team001

./cmc client contract user invoke \

--contract-name "$CM" \

--method Mint \

--sdk-conf-path "$SDK" \

--params "{\"to\":\"$ADMIN\_HEX\",\"tokenId\":\"$TID\",\"categoryName\":\"team\",\"metadata\":\"2025985211\"}" \

--user-signcrt-file-path ./testdata/crypto-config/wx-org1.chainmaker.org/user/client1/client1.sign.crt \ --user-signkey-file-path ./testdata/crypto-config/wx-org1.chainmaker.org/user/client1/client1.sign.key \

--sync-result=true --timeout 60



可以看到mint成功，"result": "TWludCBzdWNjZXNz" → Base64 解码 = Mint success

1. 再次查看mint总量为1。

./cmc client contract user invoke \

--contract-name "$CM" \

--method TotalSupply \

--sdk-conf-path "$SDK" \

--sync-result=true \

--timeout 30



1. 验证nfa归属：

./cmc client contract user invoke \

--contract-name "$CM" \

--method OwnerOf \

--sdk-conf-path "$SDK" \

--params "{\"tokenId\":\"$TID\"}" \

--sync-result=true --timeout 30



"result": "NmRkNTYyMmFjMzkzNWMxZjRlMjRiMjllNGM5NDViYjIxNDQ2MTc3MA==" 解码就是6dd5622ac3935c1f4e24b29e4c945bb214461770，和 Mint 时传入到的to地址完全一致（就是我们传入的$ADMIN\_HEX）。

**任务四：NFA的转移**

为了快速方便测试，我们采取随机生成的地址，更严谨需要另一个用户证书派生一个真实的地址。

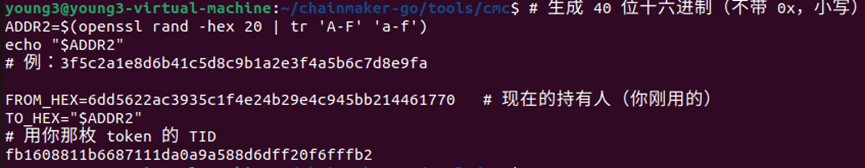
1. 生成一个随机地址：

ADDR\_RANDOM=$(openssl rand -hex 20 | tr 'A-F' 'a-f')

echo "$ADDR\_RANDOM"

FROM\_HEX=$ADMIN\_HEX #当前的TID拥有者地址

TO\_HEX="$ADDR\_RANDOM" #转移到的账户地址



可以看到生成的地址为：fb1608811b6687111da0a9a588d6dff20f6fffb2

1. 进行转账

./cmc client contract user invoke \

--contract-name "$CM" \

--method TransferFrom \

--sdk-conf-path "$SDK" \

--params "{\"from\":\"$FROM\_HEX\",\"to\":\"$TO\_HEX\",\"tokenId\":\"$TID\"}" \

--sync-result=true \

--timeout 30



"result": "VHJhbnNmZXJGcm9tIHN1Y2Nlc3M=" → 解码就是 TransferFrom success。

1. 验证转账的token归属：

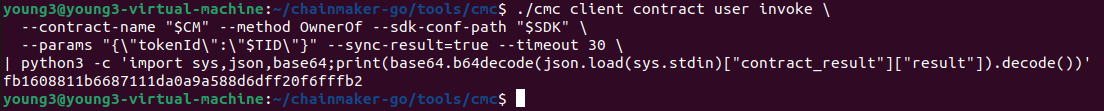
./cmc client contract user invoke \

--contract-name "$CM" --method OwnerOf --sdk-conf-path "$SDK" \

--params "{\"tokenId\":\"$TID\"}" --sync-result=true --timeout 30 \

| python3 -c 'import sys,json,base64;print(base64.b64decode(json.load(sys.stdin)["contract\_result"]["result"]).decode())'

# 预期输出就是 $ADDR\_RANDOW



说明成功转移了NFA。

**任务五：NFA的销毁**

1. 再铸造一枚到管理员地址：

export TID2=team002

./cmc client contract user invoke \

--contract-name "$CM" \

--method Mint \

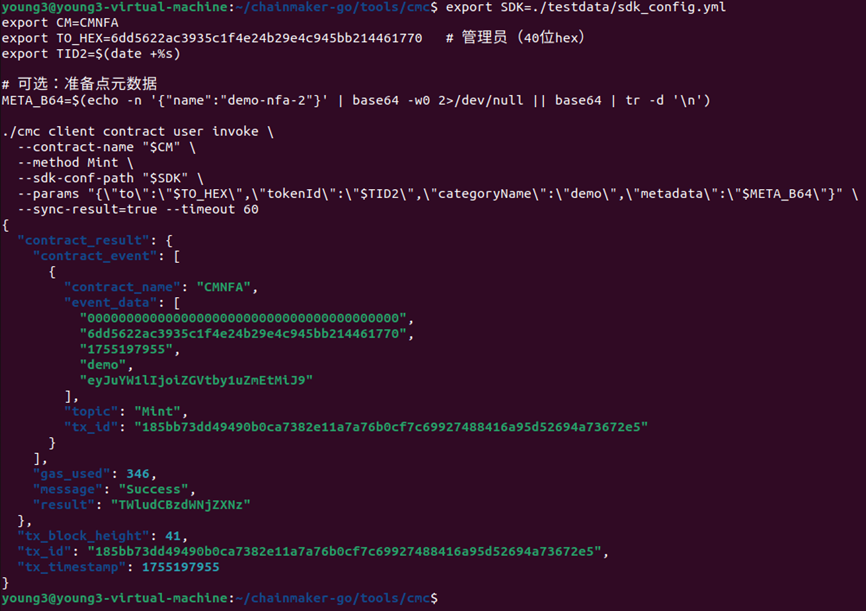
--sdk-conf-path "$SDK" \

--params "{\"to\":\"$ADMIN\_HEX\",\"tokenId\":\"$TID2\",\"categoryName\":\"team\",\"metadata\":\"2025985211\"}" \

--user-signcrt-file-path ./testdata/crypto-config/wx-org1.chainmaker.org/user/client1/client1.sign.crt \

--user-signkey-file-path ./testdata/crypto-config/wx-org1.chainmaker.org/user/client1/client1.sign.key \

--sync-result=true --timeout 60



1. 由管理员进行销毁刚铸造的这枚NFA

./cmc client contract user invoke \

--contract-name "$CM" \

--method Burn \

--sdk-conf-path "$SDK" \

--params "{\"tokenId\":\"$TID2\"}" \

--user-signcrt-file-path ./testdata/crypto-config/wx-org1.chainmaker.org/user/client1/client1.sign.crt \

--user-signkey-file-path ./testdata/crypto-config/wx-org1.chainmaker.org/user/client1/client1.sign.key \

--sync-result=true --timeout 30

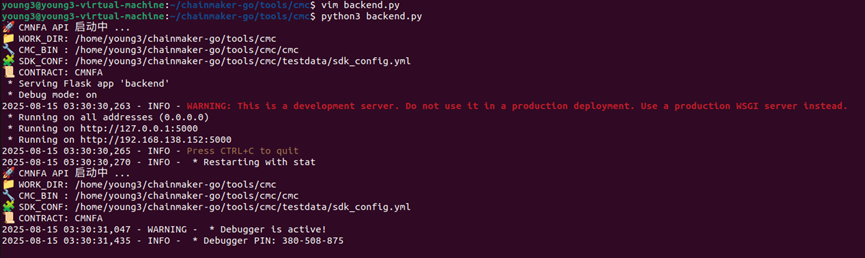


可以看到显示销毁成功了！

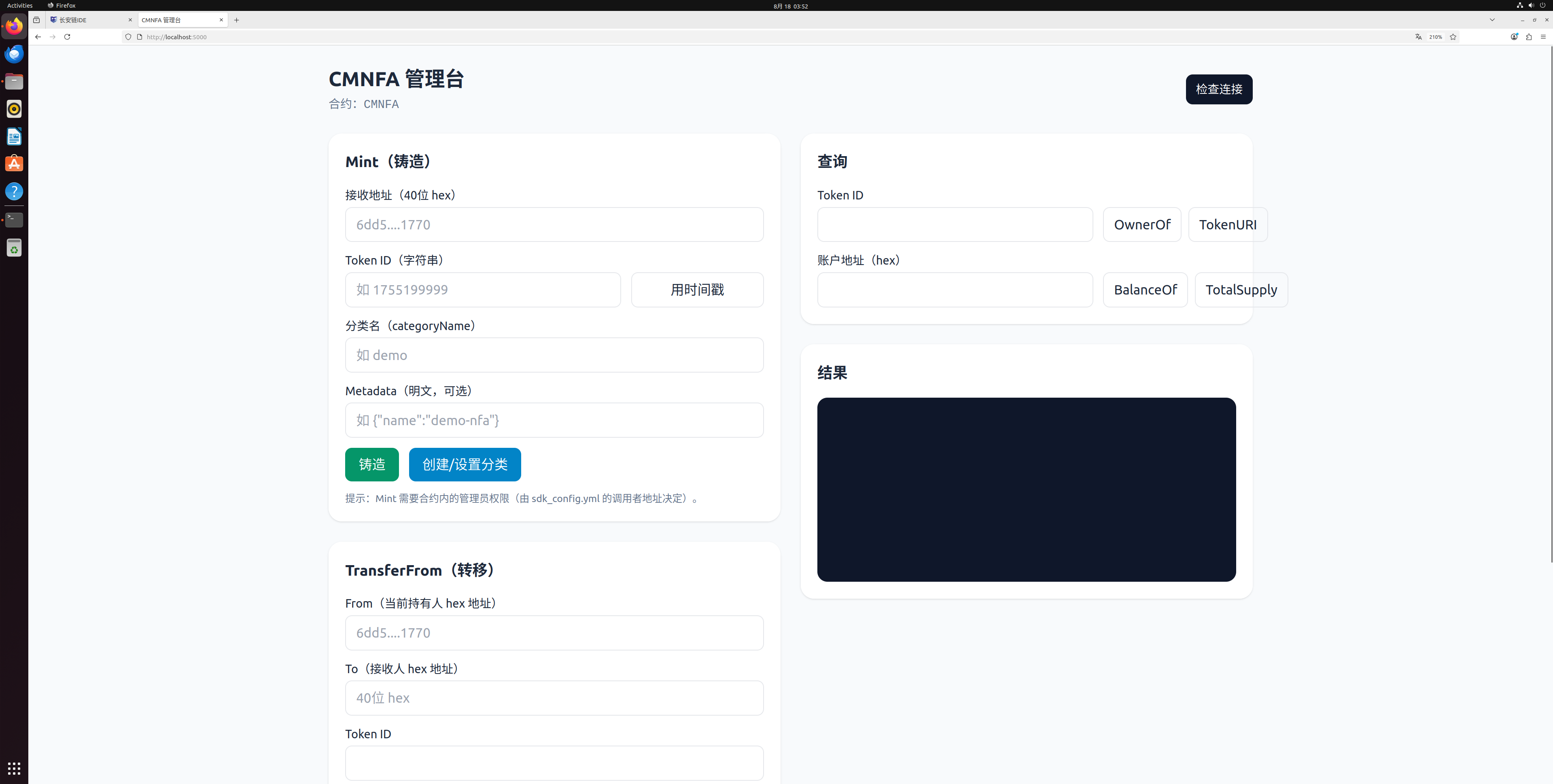
**任务六：图形化管理数字资产**

1. 启动后端服务

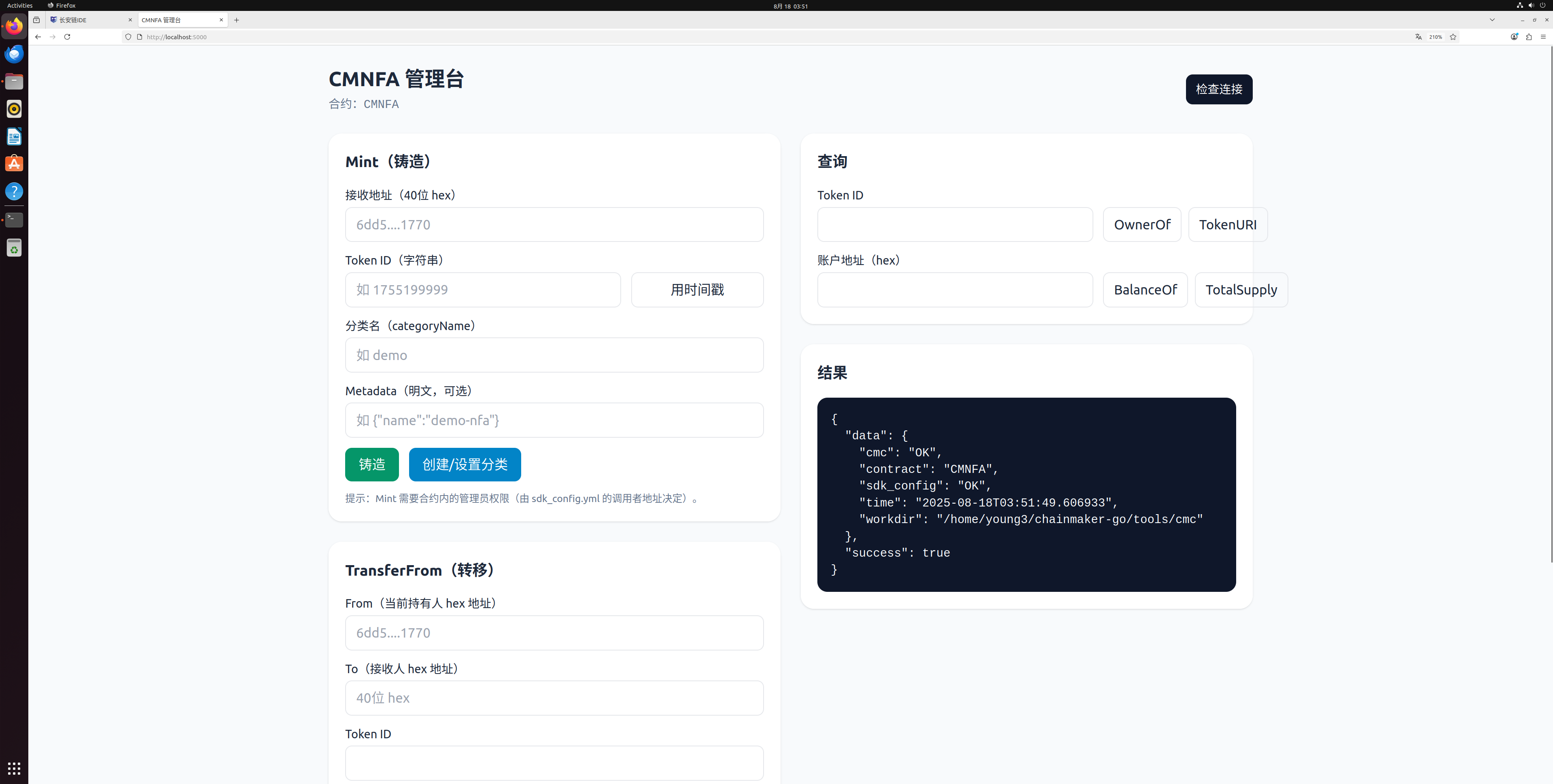
在当前目录下新建backend.py文件，写入附录参考python代码，以及新建templates/index.html代码，写入代码后保存。运行python3 backend.py。



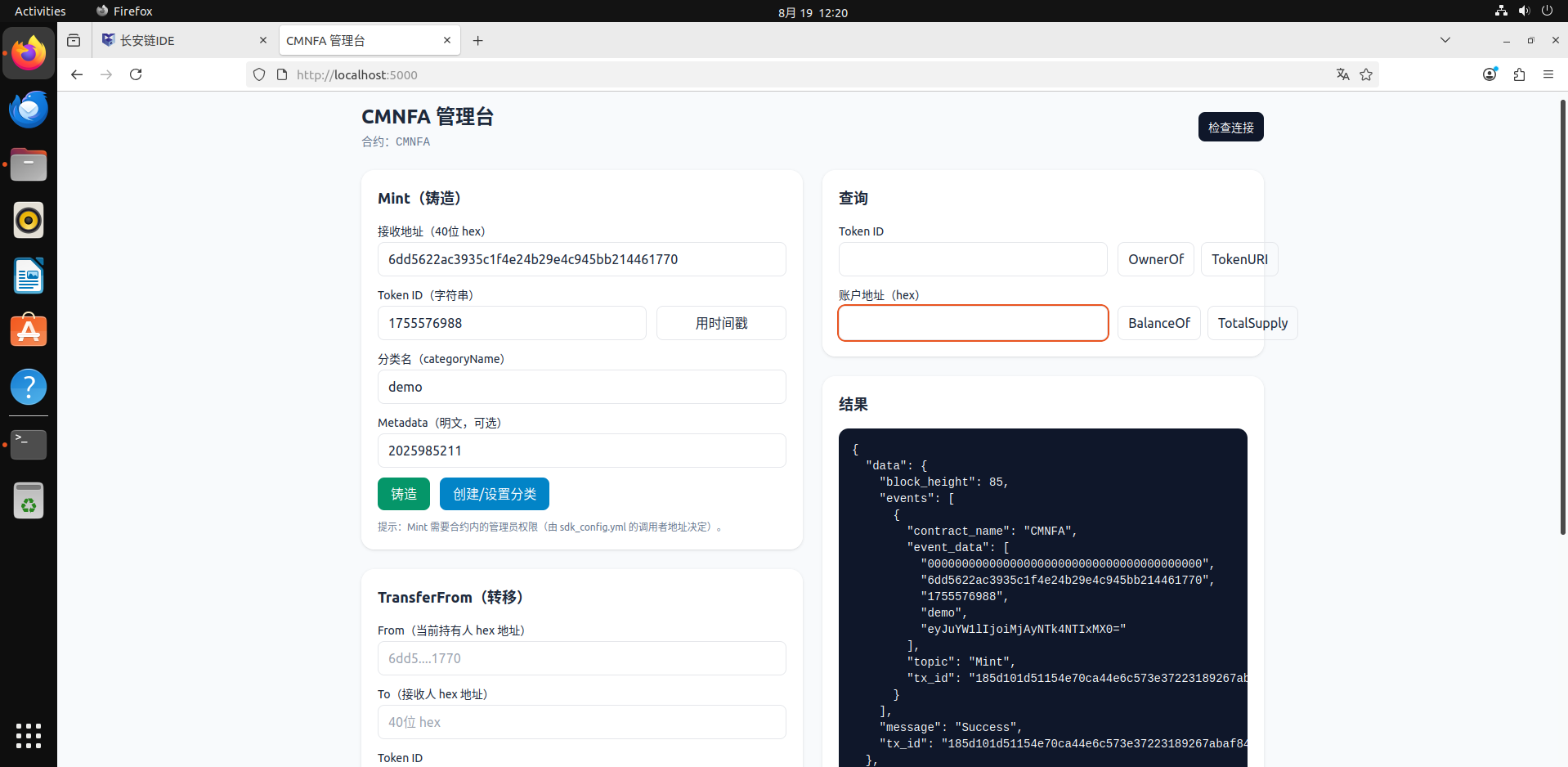
1. 前端管理数字资产
2. 打开浏览器，运行<http://localhost:5000>



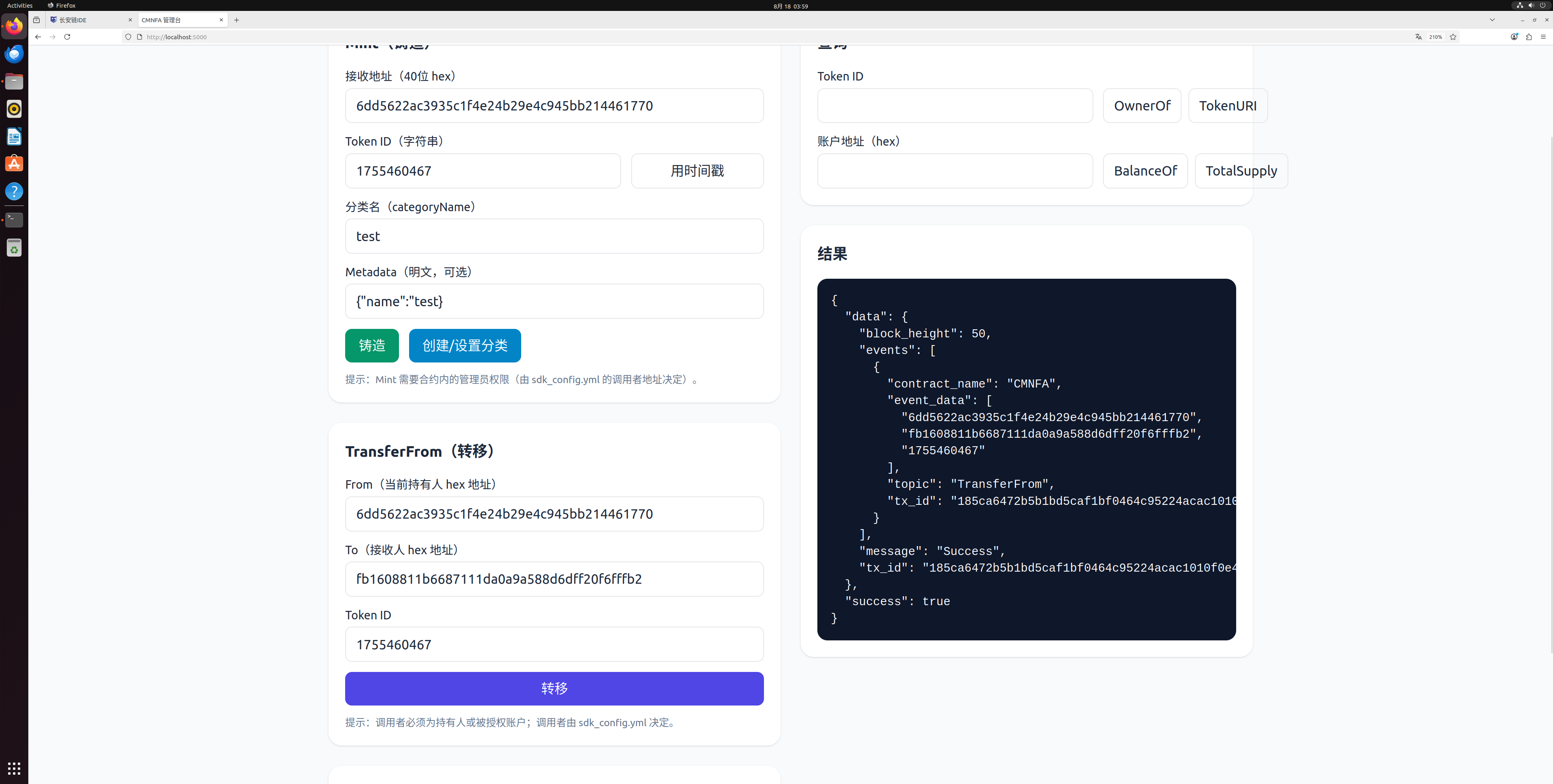
1. 检查链接，看到合约正常运行。



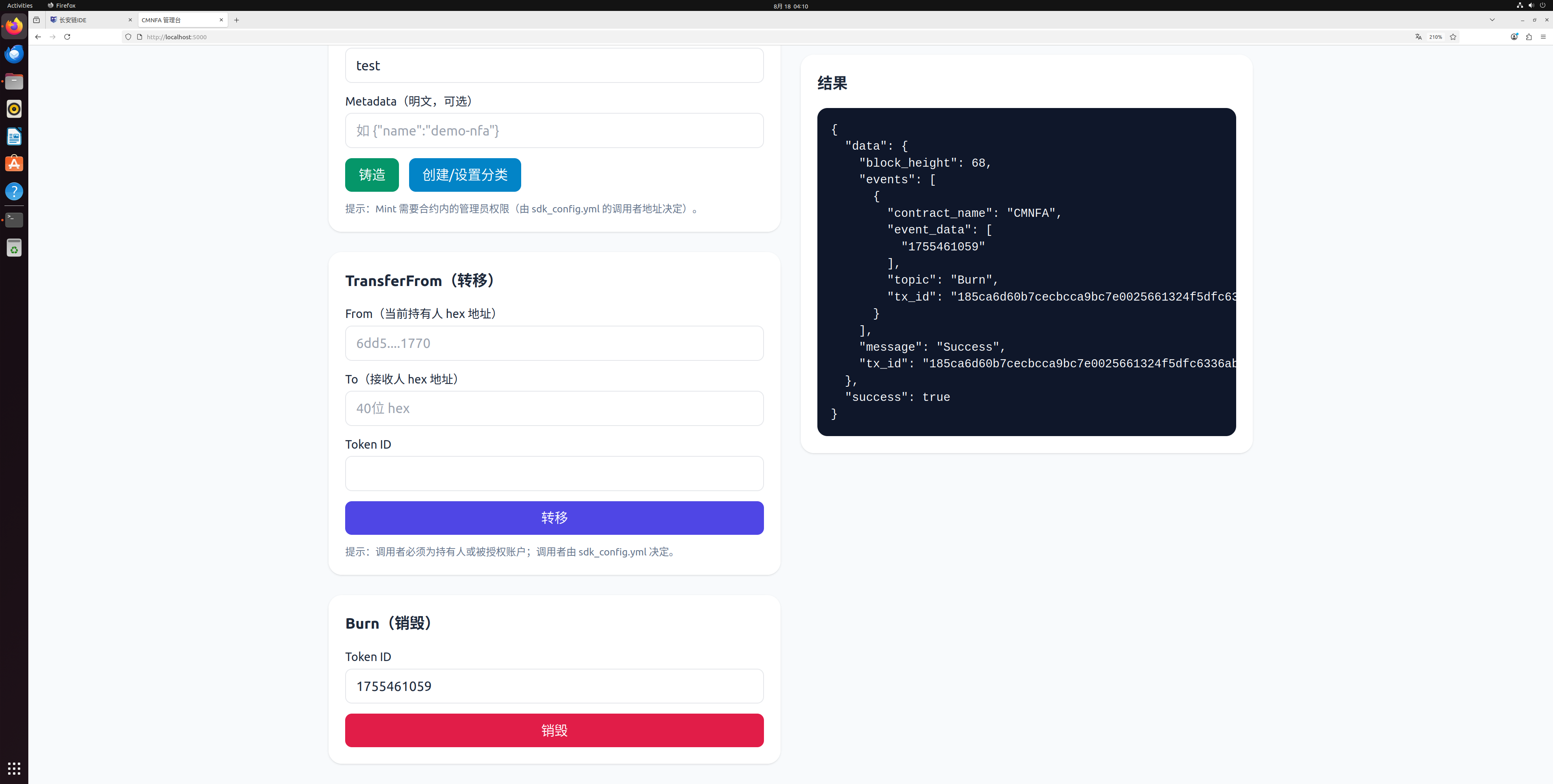
1. 对admin地址进行mint，metadate信息采用学号2025985211，可以看到mint成功。



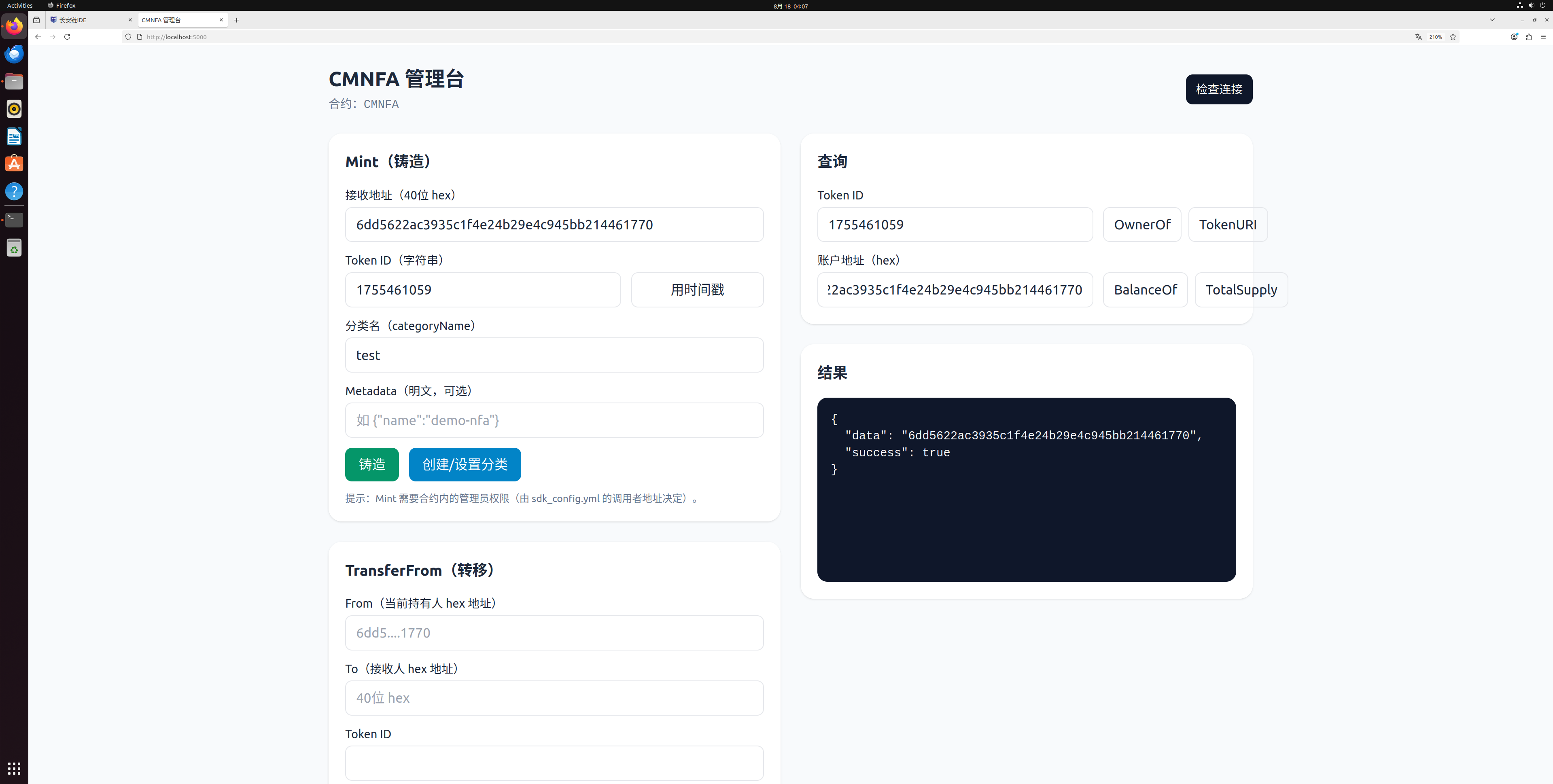
1. 转移NFA到地址：fb1608811b6687111da0a9a588d6dff20f6fffb2，采用刚才的token id，成功进行转移NFA。

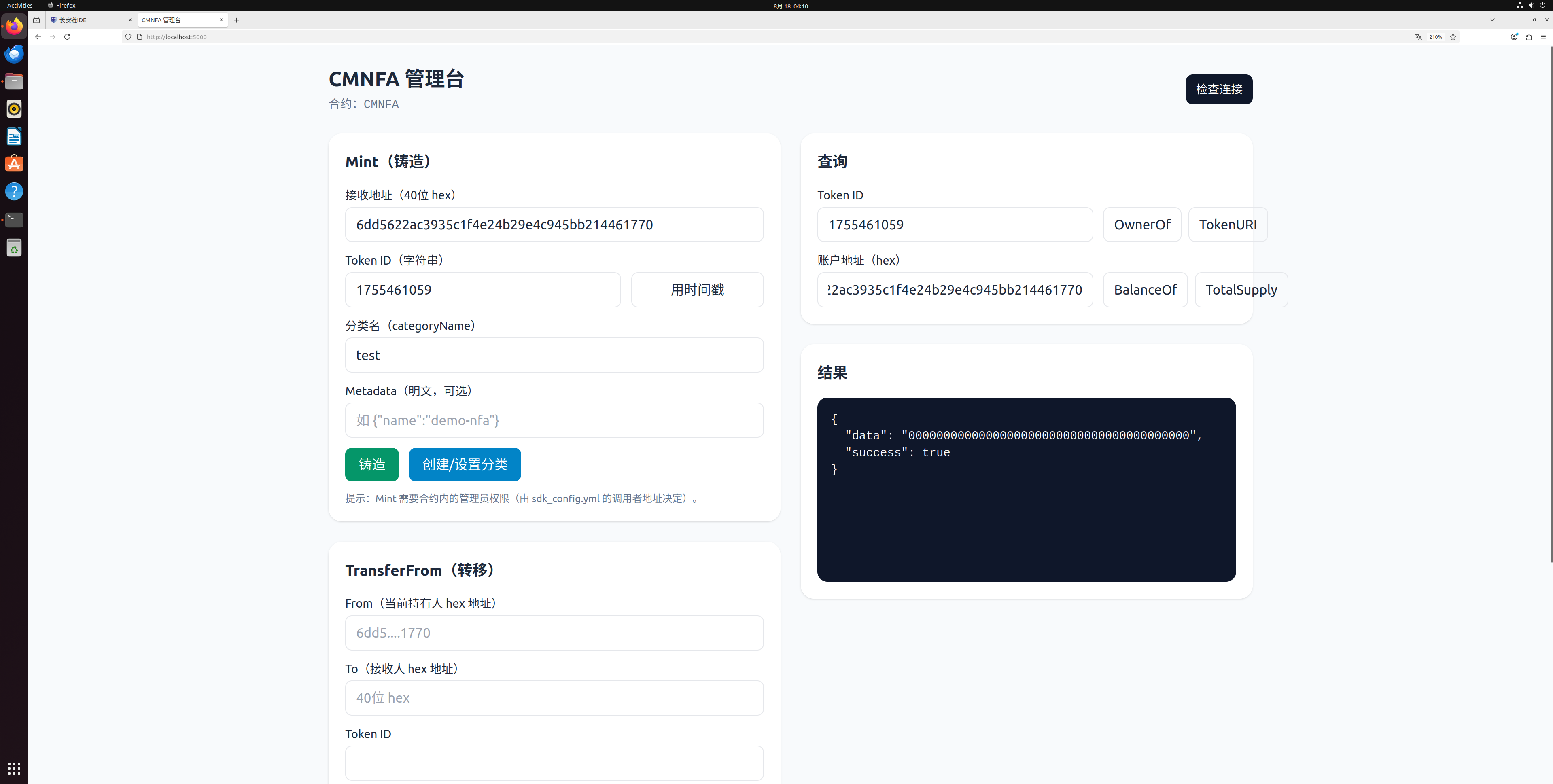


1. 销毁NFA，输入生成的token id即可销毁

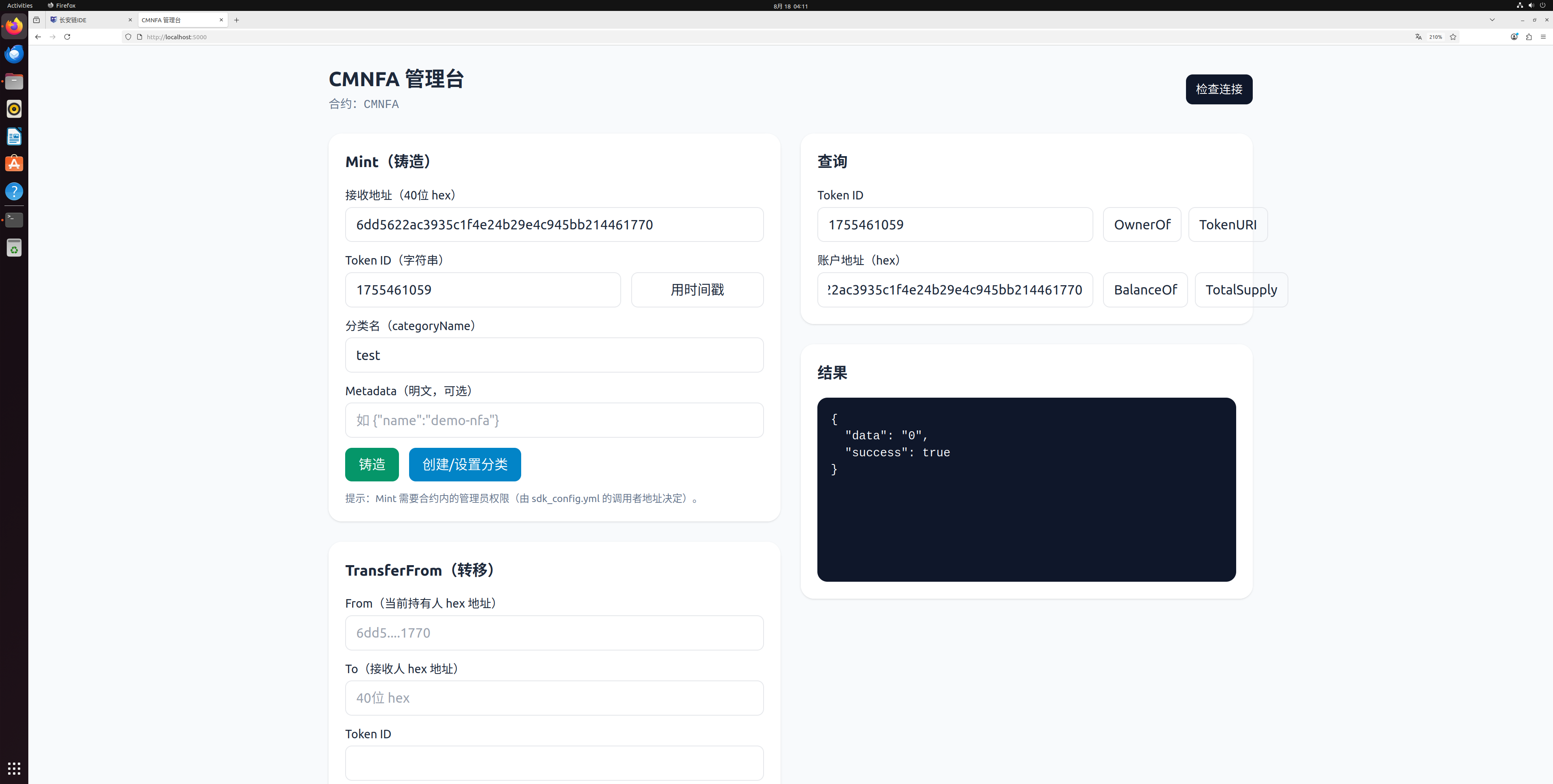


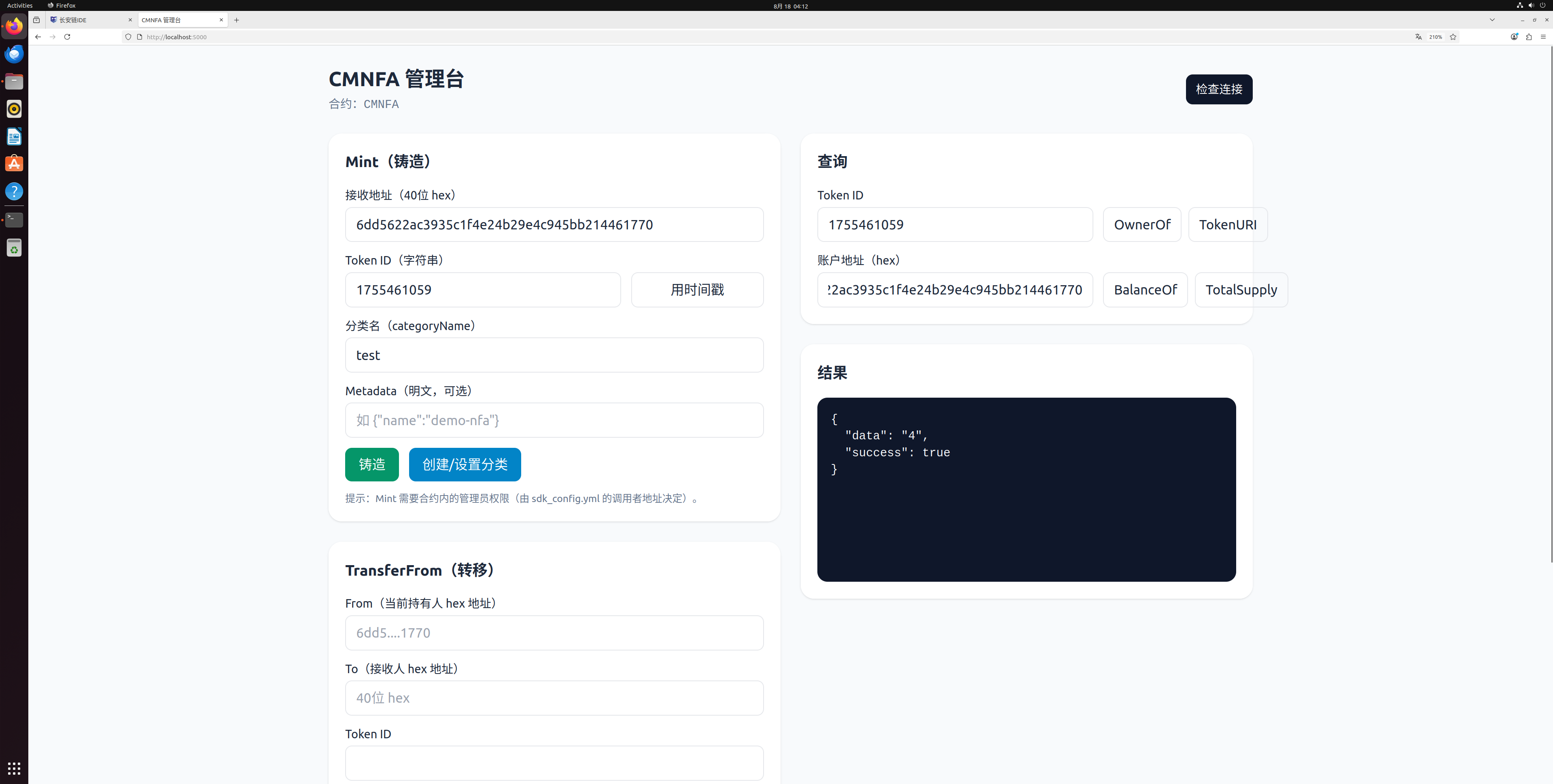
1. 分别查看销毁前后的token id值进行验证，结果如下两图：分别输入admin地址和空。





1. 查看地址的NFA总量，包括当前总量和总铸造量。





**【实验作业】**

（1）参考实验指导书完成实验，并撰写实验报告。

（2）注意实验指导书中要求体现实验完成人个人信息的地方，这些个人信息可以是学号信息，可以是队伍的名称信息等，需在实验报告中插入截图说明。

**【附录】**

**<https://github.com/A1LinLin1/CMNFA>**

1. 后端python代码

#!/usr/bin/env python3

# -\*- coding: utf-8 -\*-

from flask import Flask, request, jsonify, render\_template

from flask\_cors import CORS

import subprocess

import json

import os

import base64

import logging

from datetime import datetime

# =========================

# 基本配置（按需修改）

# =========================

CONTRACT\_NAME = os.getenv("CM\_CONTRACT", "CMNFA")

SDK\_CONF\_PATH = os.getenv("CM\_SDK", "./testdata/sdk\_config.yml")

CMC\_BIN = os.getenv("CM\_CMC\_BIN", "./cmc") # 相对工作目录

WORK\_DIR = os.getenv("CM\_WORKDIR", "/home/young3/chainmaker-go/tools/cmc")

# Flask

app = Flask(\_\_name\_\_, template\_folder="templates", static\_folder="static")

CORS(app)

logging.basicConfig(level=logging.INFO, format="%(asctime)s [%(levelname)s] %(message)s")

log = logging.getLogger("cmnfa")

# =========================

# 工具函数

# =========================

def \_is\_base64(s: str) -> bool:

try:

# 粗略判断：长度与字符集

if not s or any(c not in "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/=\n\r" for c in s):

return False

base64.b64decode(s, validate=True)

return True

except Exception:

return False

def \_decode\_result(maybe\_b64: str) -> str:

if isinstance(maybe\_b64, str) and \_is\_base64(maybe\_b64):

try:

return base64.b64decode(maybe\_b64).decode("utf-8")

except Exception:

return maybe\_b64

return maybe\_b64

def exec\_cmc(method: str, params: dict | None = None, sync: bool = True, timeout=60):

"""

调用：cmc client contract user invoke

说明：本机使用 sdk\_config.yml 指定的默认用户签名发起交易/查询。

"""

cmd = [

CMC\_BIN, "client", "contract", "user", "invoke",

f"--contract-name={CONTRACT\_NAME}",

f"--method={method}",

f"--sdk-conf-path={SDK\_CONF\_PATH}",

f"--sync-result={'true' if sync else 'false'}",

]

if params:

# 用紧凑 JSON，避免 shell 解析问题

cmd.append("--params=" + json.dumps(params, separators=(',', ':')))

if not os.path.exists(os.path.join(WORK\_DIR, os.path.basename(CMC\_BIN))):

return False, f"找不到 CMC 可执行文件：{os.path.join(WORK\_DIR, CMC\_BIN)}"

if not os.path.exists(os.path.join(WORK\_DIR, SDK\_CONF\_PATH)):

return False, f"找不到 SDK 配置：{os.path.join(WORK\_DIR, SDK\_CONF\_PATH)}"

log.info("Exec: %s", " ".join(cmd))

try:

p = subprocess.run(cmd, cwd=WORK\_DIR, capture\_output=True, text=True, timeout=timeout)

stdout = (p.stdout or "").strip()

stderr = (p.stderr or "").strip()

if p.returncode != 0:

return False, stderr or stdout or f"cmc 退出码 {p.returncode}"

# stdout 一般就是 JSON

try:

data = json.loads(stdout)

except Exception:

data = stdout

return True, data

except subprocess.TimeoutExpired:

return False, "执行超时"

except Exception as e:

return False, f"执行异常：{e}"

def ok(data): return jsonify({"success": True, "data": data})

def err(msg): return jsonify({"success": False, "error": msg})

# =========================

# 页面

# =========================

@app.get("/")

def home():

return render\_template("index.html", contract=CONTRACT\_NAME)

# =========================

# 系统与基础查询

# =========================

@app.get("/api/system/status")

def system\_status():

cmc\_ok = os.path.exists(os.path.join(WORK\_DIR, os.path.basename(CMC\_BIN)))

cfg\_ok = os.path.exists(os.path.join(WORK\_DIR, SDK\_CONF\_PATH))

return ok({

"cmc": "OK" if cmc\_ok else "MISSING",

"sdk\_config": "OK" if cfg\_ok else "MISSING",

"contract": CONTRACT\_NAME,

"workdir": WORK\_DIR,

"time": datetime.now().isoformat()

})

@app.get("/api/nfa/total-supply")

def total\_supply():

ok\_, data = exec\_cmc("TotalSupply", params=None, sync=True)

if not ok\_:

return err(data)

# data -> {"contract\_result":{"result": "MA==", "message":"Success",...}}

res = data.get("contract\_result", {}).get("result", "MA==")

return ok(\_decode\_result(res))

@app.post("/api/nfa/owner")

def owner\_of():

body = request.get\_json(force=True)

token\_id = body.get("tokenId", "").strip()

if not token\_id:

return err("tokenId 不能为空")

ok\_, data = exec\_cmc("OwnerOf", {"tokenId": token\_id})

if not ok\_:

return err(data)

res = data.get("contract\_result", {}).get("result", "")

return ok(\_decode\_result(res))

@app.post("/api/nfa/token-uri")

def token\_uri():

body = request.get\_json(force=True)

token\_id = body.get("tokenId", "").strip()

if not token\_id:

return err("tokenId 不能为空")

ok\_, data = exec\_cmc("TokenURI", {"tokenId": token\_id})

if not ok\_:

return err(data)

res = data.get("contract\_result", {}).get("result", "")

return ok(\_decode\_result(res))

@app.post("/api/nfa/balance-of")

def balance\_of():

body = request.get\_json(force=True)

account = body.get("account", "").strip()

if not account:

return err("account 不能为空")

ok\_, data = exec\_cmc("BalanceOf", {"account": account})

if not ok\_:

return err(data)

res = data.get("contract\_result", {}).get("result", "MA==")

return ok(\_decode\_result(res))

# =========================

# 业务操作：Mint / TransferFrom / Burn / 类别

# =========================

@app.post("/api/nfa/mint")

def mint():

"""

仅合约内置管理员地址（state 中 admin）可操作。

注意：调用者由 sdk\_config.yml 指定的用户决定。

"""

b = request.get\_json(force=True)

to = b.get("to", "").strip()

token\_id = b.get("tokenId", "").strip()

category = b.get("categoryName", "").strip()

metadata\_text = (b.get("metadata\_text") or "").encode("utf-8")

metadata\_b64 = b.get("metadata\_b64")

if not (to and token\_id and category):

return err("to / tokenId / categoryName 不能为空")

if metadata\_b64:

meta = metadata\_b64

else:

# 允许传明文，后端代为 base64

meta = base64.b64encode(metadata\_text).decode("utf-8") if metadata\_text else ""

ok\_, data = exec\_cmc("Mint", {"to": to, "tokenId": token\_id, "categoryName": category, "metadata": meta})

if not ok\_:

return err(data)

# 取事件与提示

cr = data.get("contract\_result", {})

msg = cr.get("message", "")

evts = cr.get("contract\_event", [])

pretty = {

"message": msg,

"events": evts,

"tx\_id": data.get("tx\_id"),

"block\_height": data.get("tx\_block\_height")

}

return ok(pretty)

@app.post("/api/nfa/transfer-from")

def transfer\_from():

b = request.get\_json(force=True)

from\_addr = b.get("from", "").strip()

to\_addr = b.get("to", "").strip()

token\_id = b.get("tokenId", "").strip()

if not (from\_addr and to\_addr and token\_id):

return err("from / to / tokenId 不能为空")

ok\_, data = exec\_cmc("TransferFrom", {"from": from\_addr, "to": to\_addr, "tokenId": token\_id})

if not ok\_:

return err(data)

cr = data.get("contract\_result", {})

pretty = {

"message": cr.get("message", ""),

"events": cr.get("contract\_event", []),

"tx\_id": data.get("tx\_id"),

"block\_height": data.get("tx\_block\_height")

}

return ok(pretty)

@app.post("/api/nfa/burn")

def burn():

b = request.get\_json(force=True)

token\_id = b.get("tokenId", "").strip()

if not token\_id:

return err("tokenId 不能为空")

ok\_, data = exec\_cmc("Burn", {"tokenId": token\_id})

if not ok\_:

return err(data)

cr = data.get("contract\_result", {})

pretty = {

"message": cr.get("message", ""),

"events": cr.get("contract\_event", []),

"tx\_id": data.get("tx\_id"),

"block\_height": data.get("tx\_block\_height")

}

return ok(pretty)

@app.post("/api/nfa/create-or-set-category")

def create\_or\_set\_category():

"""

给分类设置 URI（或新建分类）。

请求体示例：

{

"categoryName": "demo",

"categoryURI": "https://example.org/nfa"

}

"""

b = request.get\_json(force=True)

name = b.get("categoryName", "").strip()

uri = b.get("categoryURI", "").strip()

if not (name and uri):

return err("categoryName / categoryURI 不能为空")

category\_json = json.dumps({"categoryName": name, "categoryURI": uri})

ok\_, data = exec\_cmc("CreateOrSetCategory", {"category": category\_json})

if not ok\_:

return err(data)

cr = data.get("contract\_result", {})

return ok({

"message": cr.get("message", ""),

"events": cr.get("contract\_event", []),

"tx\_id": data.get("tx\_id"),

"block\_height": data.get("tx\_block\_height")

})

# =========================

# 入口

# =========================

if \_\_name\_\_ == "\_\_main\_\_":

print(f"🚀 CMNFA backend | contract={CONTRACT\_NAME}")

print(f"📁 workdir: {WORK\_DIR}")

print(f"🔧 sdk: {SDK\_CONF\_PATH}")

app.run(host="0.0.0.0", port=5000, debug=True)

1. Flask templates/index.html代码

temlpates/index.html前端代码：

<!doctype html>

<html lang="zh-CN">

<head>

<meta charset="utf-8"/>

<meta name="viewport" content="width=device-width,initial-scale=1"/>

<title>CMNFA 管理台</title>

<script src="https://cdn.tailwindcss.com"></script>

</head>

<body class="bg-slate-50 text-slate-800">

<div class="max-w-6xl mx-auto p-6">

<header class="mb-6 flex items-center justify-between">

<div>

<h1 class="text-2xl font-bold">CMNFA 管理台</h1>

<p class="text-sm text-slate-500 mt-1">合约：<span id="contractName" class="font-mono">{{ contract }}</span></p>

</div>

<button id="btnStatus" class="px-3 py-2 rounded-lg bg-slate-900 text-white text-sm">检查连接</button>

</header>

<div class="grid md:grid-cols-2 gap-6">

<!-- 左侧：操作 -->

<div class="space-y-6">

<!-- Mint -->

<div class="bg-white rounded-2xl shadow p-5">

<h2 class="text-lg font-semibold mb-4">Mint（铸造）</h2>

<div class="grid gap-3">

<div>

<label class="block text-sm mb-1">接收地址（40位 hex）</label>

<input id="mintTo" class="w-full rounded-lg border px-3 py-2" placeholder="6dd5....1770">

</div>

<div class="grid grid-cols-3 gap-3">

<div class="col-span-2">

<label class="block text-sm mb-1">Token ID（字符串）</label>

<input id="mintTokenId" class="w-full rounded-lg border px-3 py-2" placeholder="如 1755199999">

</div>

<div class="flex items-end">

<button id="btnGenTid" class="w-full rounded-lg border px-3 py-2">用时间戳</button>

</div>

</div>

<div>

<label class="block text-sm mb-1">分类名（categoryName）</label>

<input id="mintCategory" class="w-full rounded-lg border px-3 py-2" placeholder="如 demo">

</div>

<div>

<label class="block text-sm mb-1">Metadata（明文，可选）</label>

<input id="mintMeta" class="w-full rounded-lg border px-3 py-2" placeholder='如 {"name":"demo-nfa"}'>

</div>

<div class="flex gap-3">

<button id="btnMint" class="rounded-lg px-4 py-2 bg-emerald-600 text-white">铸造</button>

<button id="btnCreateCate" class="rounded-lg px-4 py-2 bg-sky-600 text-white">创建/设置分类</button>

</div>

<p class="text-xs text-slate-500">提示：Mint 需要合约内的管理员权限（由 sdk\_config.yml 的调用者地址决定）。</p>

</div>

</div>

<!-- TransferFrom -->

<div class="bg-white rounded-2xl shadow p-5">

<h2 class="text-lg font-semibold mb-4">TransferFrom（转移）</h2>

<div class="grid gap-3">

<div>

<label class="block text-sm mb-1">From（当前持有人 hex 地址）</label>

<input id="tfFrom" class="w-full rounded-lg border px-3 py-2" placeholder="6dd5....1770">

</div>

<div>

<label class="block text-sm mb-1">To（接收人 hex 地址）</label>

<input id="tfTo" class="w-full rounded-lg border px-3 py-2" placeholder="40位 hex">

</div>

<div>

<label class="block text-sm mb-1">Token ID</label>

<input id="tfTokenId" class="w-full rounded-lg border px-3 py-2">

</div>

<button id="btnTf" class="rounded-lg px-4 py-2 bg-indigo-600 text-white">转移</button>

<p class="text-xs text-slate-500">提示：调用者必须为持有人或被授权账户；调用者由 sdk\_config.yml 决定。</p>

</div>

</div>

<!-- Burn -->

<div class="bg-white rounded-2xl shadow p-5">

<h2 class="text-lg font-semibold mb-4">Burn（销毁）</h2>

<div class="grid gap-3">

<div>

<label class="block text-sm mb-1">Token ID</label>

<input id="burnTokenId" class="w-full rounded-lg border px-3 py-2">

</div>

<button id="btnBurn" class="rounded-lg px-4 py-2 bg-rose-600 text-white">销毁</button>

</div>

</div>

</div>

<!-- 右侧：查询/结果 -->

<div class="space-y-6">

<div class="bg-white rounded-2xl shadow p-5">

<h2 class="text-lg font-semibold mb-4">查询</h2>

<div class="grid gap-3">

<div class="grid grid-cols-3 gap-3">

<div class="col-span-2">

<label class="block text-sm mb-1">Token ID</label>

<input id="qTokenId" class="w-full rounded-lg border px-3 py-2">

</div>

<div class="flex items-end gap-2">

<button id="btnOwner" class="flex-1 rounded-lg px-3 py-2 border">OwnerOf</button>

<button id="btnURI" class="flex-1 rounded-lg px-3 py-2 border">TokenURI</button>

</div>

</div>

<div class="grid grid-cols-3 gap-3">

<div class="col-span-2">

<label class="block text-sm mb-1">账户地址（hex）</label>

<input id="qAccount" class="w-full rounded-lg border px-3 py-2">

</div>

<div class="flex items-end gap-2">

<button id="btnBal" class="flex-1 rounded-lg px-3 py-2 border">BalanceOf</button>

<button id="btnTS" class="flex-1 rounded-lg px-3 py-2 border">TotalSupply</button>

</div>

</div>

</div>

</div>

<div class="bg-white rounded-2xl shadow p-5">

<h2 class="text-lg font-semibold mb-4">结果</h2>

<pre id="out" class="bg-slate-900 text-slate-50 rounded-xl p-4 text-sm overflow-auto min-h-[220px]"></pre>

</div>

</div>

</div>

</div>

<script>

const out = document.getElementById('out');

const setOut = (v) => out.textContent = typeof v === 'string' ? v : JSON.stringify(v, null, 2);

const post = async (url, data) => {

const r = await fetch(url, {method:'POST', headers:{'Content-Type':'application/json'}, body:JSON.stringify(data||{})});

return r.json();

}

const get = async (url) => (await fetch(url)).json();

// 状态

document.getElementById('btnStatus').onclick = async () => {

const r = await get('/api/system/status');

setOut(r);

};

// 生成时间戳 tokenId

document.getElementById('btnGenTid').onclick = () => {

document.getElementById('mintTokenId').value = Math.floor(Date.now()/1000);

};

// Mint

document.getElementById('btnMint').onclick = async () => {

const to = document.getElementById('mintTo').value.trim();

const tokenId = document.getElementById('mintTokenId').value.trim();

const categoryName = document.getElementById('mintCategory').value.trim();

const metaText = document.getElementById('mintMeta').value;

const r = await post('/api/nfa/mint', {to, tokenId, categoryName, metadata\_text: metaText});

setOut(r);

};

// 创建/设置分类

document.getElementById('btnCreateCate').onclick = async () => {

const name = document.getElementById('mintCategory').value.trim() || 'demo';

const uri = prompt('请输入该分类的 CategoryURI（例如 https://example.org/nfa）', 'https://example.org/nfa');

if (!uri) return;

const r = await post('/api/nfa/create-or-set-category', {categoryName: name, categoryURI: uri});

setOut(r);

};

// TransferFrom

document.getElementById('btnTf').onclick = async () => {

const from = document.getElementById('tfFrom').value.trim();

const to = document.getElementById('tfTo').value.trim();

const tokenId = document.getElementById('tfTokenId').value.trim();

const r = await post('/api/nfa/transfer-from', {from, to, tokenId});

setOut(r);

};

// Burn

document.getElementById('btnBurn').onclick = async () => {

const tokenId = document.getElementById('burnTokenId').value.trim();

const r = await post('/api/nfa/burn', {tokenId});

setOut(r);

};

// OwnerOf

document.getElementById('btnOwner').onclick = async () => {

const tokenId = document.getElementById('qTokenId').value.trim();

const r = await post('/api/nfa/owner', {tokenId});

setOut(r);

};

// TokenURI

document.getElementById('btnURI').onclick = async () => {

const tokenId = document.getElementById('qTokenId').value.trim();

const r = await post('/api/nfa/token-uri', {tokenId});

setOut(r);

};

// BalanceOf

document.getElementById('btnBal').onclick = async () => {

const account = document.getElementById('qAccount').value.trim();

const r = await post('/api/nfa/balance-of', {account});

setOut(r);

};

// TotalSupply

document.getElementById('btnTS').onclick = async () => {

const r = await get('/api/nfa/total-supply');

setOut(r);

};

</script>

</body>

</html>