

Atomicals dmint 进阶指南



XXXXXXXXXXXXXXXXXXXXXXXXXXXX

ATOMICALS

XXXXXXXXXXXXXXXXXXXXXXXXXXXX



DMINT

Blockchain
Architecture

Blockchain architecture is a distributed ledger technology that allows for secure and transparent transactions without the need for a central authority.

XXXXXXXXXXXX

Blockchain
Architecture

Blockchain architecture is a distributed ledger technology that allows for secure and transparent transactions without the need for a central authority.

XXXXXXXXXXXX

Blockchain
Architecture

Blockchain architecture is a distributed ledger technology that allows for secure and transparent transactions without the need for a central authority.

XXXXXXXXXXXX

Blockchain
Architecture

Blockchain architecture is a distributed ledger technology that allows for secure and transparent transactions without the need for a central authority.

XXXXXXXXXXXX

XXXXXXXXXXXXXXXXXXXXXXXXXXXX

XXXXXXXXXXXXXXXXXXXXXXXXXXXX

XXXXXXXXXXXX

一、写在前面

鉴于 Atomicals 生态还十分早期，并且前端 mint 也要收服务费，掌握使用官方脚本进行 mint 的方法还是十分必要的，本文将从安装 atomicals-js 脚本开始到如何使用脚本 dmint，再到如何查看 container 元数据与 item 查重进行详细介绍。

不了解 dmint？请先参考这篇文章：[Atomicals dmint 简易指南 — wusimpl \(mirror.xyz\)](#)

本文略长，请勿紧张，可直接跳至感兴趣处阅读。

请注意本篇涉及知识较多，读者需具备加密货币基本素养和进阶的计算机知识，熟悉 Ordinals 或 Atomicals 生态，否则阅读起来可能十分吃力。

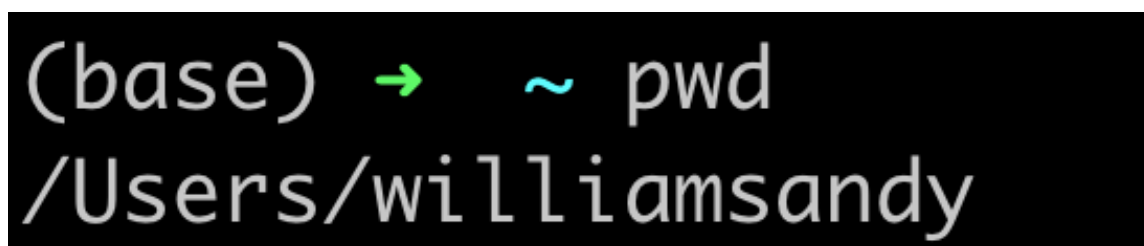
二、命令行的基本知识（熟悉者可以跳过此节）

命令行终端是操控电脑运行各种程序的字符界面程序，相比可视化界面来说更加简洁，快速，但是上手难度也比较大。

首先了解第一个概念：**当前工作目录**，这表明你的所有指令都相对于你的当前工作目录来寻找其他文件或文件夹。查看当前工作目录的命令是：

```
pwd
```

执行结果：

A terminal window with a black background. The prompt is '(base)'. A green arrow points from the prompt to a tilde '~'. The command 'pwd' is entered. The output is '/Users/williamsandy'.

可以看到我当前的目录是/Users/williamsandy

现在我想要跳转到/Users/williamsandy/code目录，可以使用命令：

```
cd code
```

或者

```
cd ./code
```

.表示当前目录，可以省略。

执行结果：

```
(base) → ~ cd code  
(base) → code pwd  
/Users/williamsandy/code
```

然后我想查看这个目录有什么文件，可以使用命令：

ls

```
(base) → code ls  
abc.txt      atomicals-js
```

可以看到我的 code 目录有一个文本文件 **abc.txt** 和一个文件夹 **atomicals-js**。

三、安装 atomicals-js 脚本

1. 安装 node.js

首先请确保你已经安装了 node.js，且可以在命令行中成功查看到 node 版本。

查看 node 版本的命令：

node -v

执行结果：

```
(base) → ~ node -v  
v16.20.2
```

我电脑上的版本是 v16，大家安装的版本可能有所不同。

2. 安装 git

node.js 和 git 工具网上教程多如牛毛，请大家自行安装。

```
(base) → ~ git -v  
git version 2.41.0
```

3. 下载 atomicals-js 代码

现在我想将 atomicals-js 代码下载到/Users/williamsandy/code/目录下，所以我先执行：

```
cd ./code
```

```
(base) → ~ pwd  
/Users/williamsandy  
(base) → ~ cd code  
(base) → code pwd  
/Users/williamsandy/code
```

然后执行 clone 命令：

```
git clone https://github.com/atomicals/atomicals-js
```

```
(base) → code git clone https://github.com/atomicals/atomicals-js  
正克隆到 'atomicals-js'...  
remote: Enumerating objects: 1343, done.  
remote: Counting objects: 100% (1343/1343), done.  
remote: Compressing objects: 100% (458/458), done.  
remote: Total 1343 (delta 978), reused 1174 (delta 867), pack-reused 0  
接收对象中：100% (1343/1343), 3.68 MiB | 2.59 MiB/s, 完成。  
处理 delta 中：100% (978/978), 完成。
```

然后执行：

```
ls
```



```
(base) → code ls
atomicals-js
```

可以看到代码已经被我们下载到了 code 目录，然后我进入 atomicals-js 目录。

```
cd ./atomicals-js
```

4. 安装依赖项

使用 npm 包管理工具安装 atomicals-js 需要的依赖：

```
npm install
```

执行结果：

```
(base) → atomicals-js git:(master) npm install
npm ERR! Invalid Version: 3001.0001.0000-dev-harmony-fb
npm ERR! A complete log of this run can be found in:
npm ERR! /Users/williamsandy/.npm/_logs/2023-12-05T09_13_34_667Z-debu
```

可以看到执行失败了，ERR就是error，失败的意思，原因也写得很清楚了：invalid version。从 stackoverflow ([Npm ERR! Invalid version on npm install – Stack Overflow](#)) 的回答可以得知，可能是 yarn.lock 或者 pnpm-lock.yaml 文件的问题，让我们来把这两个碍事的文件删掉：

```
rm yarn.lock pnpm-lock.yaml
```

然后再执行：

```
npm install
```

```
E binaries. Upgrade to safe fsevents v2
npm WARN deprecated stable@0.1.8: Modern JS already guarantees Array#sort()
  stable sort, so this library is deprecated. See the compatibility table on
  https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_O
  s/Array/sort#browser_compatibility
npm WARN deprecated chokidar@2.1.8: Chokidar 2 does not receive security up
  since 2019. Upgrade to chokidar 3 with 15x fewer dependencies
npm WARN deprecated svgo@0.7.2: This SVG0 version is no longer supported. U
  e to v2.x.x.
npm WARN deprecated browserslist@1.7.7: Browserslist 2 could fail on readin
  wrowserslist >3.0 config used in other tools.
npm WARN deprecated browserslist@1.7.7: Browserslist 2 could fail on readin
  wrowserslist >3.0 config used in other tools.
npm WARN deprecated uuid@3.4.0: Please upgrade to version 7 or higher. 0l
  versions may use Math.random() in certain circumstances, which is known to b
  blematic. See https://v8.dev/blog/math-random for details.
npm WARN deprecated react-tools@0.13.3: react-tools is deprecated. For more
  rmation, visit https://fb.me/react-tools-deprecated

added 1870 packages in 3m
(base) → atomicals-js git:(master) x
```

这下没有报错了，虽然有很多烦人的 WARNING 警告信息，但这不影响运行。

5. 编译

使用命令：

```
npm run build
```

```
(base) → atomicals-js git:(master) x npm run build

> atomicals-js@0.1.56 build
> tsc && gulp build

[17:32:46] Using gulpfile ~/materials/code/unix/atomicals-js/gulpfile.js
[17:32:46] Starting 'build'...
[17:32:49] Finished 'build' after 3.53 s

> atomicals-js@0.1.56 postbuild
> echo '#!/usr/bin/env node
> ' | cat - dist/cli.js > temp && mv temp dist/cli.js
```

可以看到我已经成功编译，没有显示异常的日志信息。

7. 使用 atomicals-js 工具

运行命令：

```
node dist/cli.js -h
```

```
(base) → atomicals-js git:(master) × node dist/cli.js -h
Usage: Atomicals CLI Utility [options] [command]

Command line utility for interacting with Atomicals

Options:
  -V, --version            output the version number
  -h, --help               display help for command

Commands:
  server-version           Get electrumx server version info
  wallet-create            Creates and displays new 12-word secret mnemonic phrase
  wallet-decode [options] <phrase> Decode secret mnemonic phrase to display derived address
  wallet-init [options]    Initializes a new wallet at wallet.json
  wallet-import <wif> <alias> Import a wallet by WIF and assign it to provided alias
  address-script <addressOrAlias> Encodes an address or wallet alias as the hex output
  script-address <script>       Decodes a script as an address
  outpoint-compact <hex>       Decodes hex outpoint to compact location id form
  compact-outpoint <compactId> Encodes the compact id to outpoint hex format
  address [options] <address>  Get balances and Atomicals stored at an address
  wallets [options]           Get balances and atomicals stored at internal wallet
  balances [options]          Get balances and atomicals stored at internal wallet
  address-utxos <address>     List all utxos owned by an address
```

可以看到 cli.js 打印出了 Atomicals CLI 工具的使用方法，大家可以把这些输出喂给 GPT 或者翻译就知道这些命令分别是什么意思了。

所以 Atomicals CLI 工具的基本用法就是：

```
node dist/cli.js <command> <options>
```

<command> 和 **<options>** 可以填什么东西命令行都给你打印出来了。

对于 **<options>**，可以填 **-v** 和 **-h**，表示输出版本或者帮助的意思，比如 `node dist/cli.js -h` 中的 **-h** 就是输出 cli.js 的帮助文本。

8. 初始化钱包

使用命令：

```
node dist/cli.js wallet-init
```

来初始化钱包。初始化的本质是创建一个 wallet.json 文件来保存钱包的私钥。


```
(base) → atomicals-js git:(master) × node dist/cli.js wallet-init
Wallet created at wallet.json
phrase: merry hungry rich suspect culture sibling woman split regular glimpse dog screen
Primary address (P2TR): bc1p3025cneck2vc9cxgptep55mh6dz6lqrmvz2nekwe3hj00trwdxrs5wq8tl
Primary address WIF: L5FCGqD9GbT9wf8rsJ75TpYvXTdVq6DZ9Sh7gFB5RdYwiefzkauv
Primary address path: m/86'/0'/0'/0/0
Funding address (P2TR): bc1pps7ds367xlnn7a3kptnfhtjq3eqn5g0mv2uvfdy4mwhkqyxck2qkp0usq
Funding address WIF: L22cv3mjUQunZsQivCxQbHiYbgbRk3FGsDXodHf3JUWnpZ3HgQxQ
Funding address path: m/86'/0'/0'/1/0
Full Data: {
  "phrase": "merry hungry rich suspect culture sibling woman split regular glimpse dog screen",
  "primary": {
    "address": "bc1p3025cneck2vc9cxgptep55mh6dz6lqrmvz2nekwe3hj00trwdxrs5wq8tl",
    "path": "m/86'/0'/0'/0/0",
    "WIF": "L5FCGqD9GbT9wf8rsJ75TpYvXTdVq6DZ9Sh7gFB5RdYwiefzkauv"
  },
  "funding": {
    "address": "bc1pps7ds367xlnn7a3kptnfhtjq3eqn5g0mv2uvfdy4mwhkqyxck2qkp0usq",
    "path": "m/86'/0'/0'/1/0",
    "WIF": "L22cv3mjUQunZsQivCxQbHiYbgbRk3FGsDXodHf3JUWnpZ3HgQxQ"
  },
  "imported": {}
}
```

可以看到 wallet.json 中有钱包的助记词，通过助记词生成了两个私钥，一个私钥地址叫做**primary**，一个叫做**funding**，funding wallet 默认作为花费 btc 的钱包，primary wallet 默认作为接收 atomicals 的钱包。

atomicals-js工具最近的升级将 wallet.json 放到了 **./wallets/wallet.json**下面。其实并不是非要执行 **node dist/cli.js wallet-init**命令，你完全可以把一个格式正确的 wallet.json 文件放在 **./wallets** 目录下，这样也算完成了初始化。

现在你已经完成了使用Atomicals CLI工具的所有必备条件，可以开始你的Atomicals 之旅了！

下面是一些可选操作。

9. 更改 Atomicals RPC 索引节点（可选）

代码根目录中的 **.env**文件存放着节点的配置信息，更改此文件可以配置节点信息。

```
(base) → atomicals-js git:(master) x ls -al
total 4768
drwxr-xr-x@ 26 williamsandy staff 832 12 5 17:42 .
drwxr-xr-x  4 williamsandy staff 128 12 5 17:11 ..
-rw-r--r--@  1 williamsandy staff 510 12 5 17:08 .env
-rw-r--r--@  1 williamsandy staff 510 12 5 17:08 .env.example
drwxr-xr-x@ 12 williamsandy staff 384 12 5 17:08 .git
drwxr-xr-x@  3 williamsandy staff  96 12 5 17:08 .github
-rwxr-xr-x@  1 williamsandy staff 345 12 5 17:08 .gitignore
-rw-r--r--@  1 williamsandy staff 841 12 5 17:08 Dockerfile
-rw-r--r--@  1 williamsandy staff 1097 12 5 17:08 LICENSE
-rw-r--r--@  1 williamsandy staff 3150 12 5 17:08 README.md
-rw-r--r--@  1 williamsandy staff 9203 12 5 17:08 a.png
-rw-r--r--@  1 williamsandy staff 97732 12 5 17:08 atomicals.jpg
-rw-r--r--@  1 williamsandy staff 439959 12 5 17:08 banner-full.png
-rw-r--r--@  1 williamsandy staff 252393 12 5 17:08 banner.png
-rw-r--r--@  1 williamsandy staff 1834 12 5 17:08 bitcoin.png
drwxr-xr-x@ 13 williamsandy staff 416 12 5 17:32 dist
-rw-r--r--@  1 williamsandy staff 788397 12 5 17:08 donate.png
```

大家可以使用文本编辑器编辑 .env 文件来更改节点：

```
.env+ [?]
1 # Install your own ElectrumX, Proxy and Service to help decentralization
2 # https://github.com/atomicals/electrumx-proxy
3 # https://github.com/atomicals/atomicals-electrumx
4 #
5 # PUBLIC ELECTRUMX PROXIES
6 #
7 # Bitcoin:
8 # ELECTRUMX_PROXY_BASE_URL=https://ep.atomicals.xyz/proxy
9 #
10 # Bitcoin (Testnet):
11 # ELECTRUMX_PROXY_BASE_URL=https://eptestnet.atomicals.xyz/proxy
12
13 ELECTRUMX_PROXY_BASE_URL=https://ep.atomicals.xyz/proxy
14 WALLET_PATH=./wallets
15 WALLET_FILE=wallet.json
16 # testnet or livenet or regtest
17 NETWORK=livenet
18
```

可以看到 .env 文件总共只有4行配置是有效的，以 # 开头的行都是注释！

而第 13 行的 **ELECTRUMX_PROXY_BASE_URL** 就是节点配置信息，比如我想使用 NextDAO 的公共节点可以这么写：

```
ELECTRUMX_PROXY_BASE_URL=https://ep.nextdao.xyz/proxy
```

如果自己配置了节点，也可以改成自己的节点位置，比如我在我局域网的10.110.8.8

的6000端口搭建了 Atomicals ElectrumX Server服务，那么可以写成：
ELECTRUMX_PROXY_BASE_URL=http://10.110.8.8:6000/proxy
或者我在本机的5050端口搭建了节点，那么可以写成：

ELECTRUMX_PROXY_BASE_URL=http://localhost:5050/proxy

可用的公共节点列表请参考：

<https://x.com/wusimpl/status/1729829443097526572?s=20>

搭建个人 atom 节点可参考：[Web5 | BTC 生态](#)

10. 简化 CLI 命令的使用（可选）

安装另一个包管理工具yarn可简化命令的使用。

原来的命令是：

```
node dist/cli.js <command> <option>
```

安装 yarn 之后，命令是：

```
yarn cli <command> <option>
```

也就是 **node dist/cli.js**和 **yarn cli**等效。

四、查看 Container 元数据

可以简单的把 Atomicals 的 Container 理解为以太坊上的 NFT Collection，即一个 NFT 集合。

CLI 工具提供了查询 container 元数据的方法，这对我们想 mint 这个 container 的用户来说特别重要，因为其中包括了稀有度分级、难度、支付给项目方的费用、mint 激活时间等信息。

查询命令：

```
yarn cli get-container-items <container name> 0 0
```

或者

```
yarn cli get-container <container name>
```

比如我想查询 crabada 的信息：

```
yarn cli get-container-items crabada 0 0
```

输出了一个堆东西，是一个 json 文件，由于文件太长，我这里只展示部分内容。

1. 首先是容器的名字：

```
{
  "$request_container_status": {
    "status": "verified",
    "verified_atomical_id": "21191ed476f9c4ac775c924ed084f8e1aed3ad9e5bc3c65ce4447088b4c542dc10",
    "note": "Successfully verified and claimed container for current Atomical"
  },
  "$request_container": "crabada",
  "subtype": "container",
  "$container": "crabada",
  "$sealed": "daef5bf69f1af352c7689da5694bcdf80cc9b5c66dba60fbc02ce9a9f9231d7ei0",
  "$container_dmint_status": {
    "status": "valid",

```

2. 然后是 mint 规则，这个最为重要：

```

subtype : container ;
"$container": "crabada",
"$sealed": "daef5bf69f1af352c7689da5694bcdf80cc9b5c66dba60fbc02ce9a9f9231d7ei0",
"$container_dmint_status": {
  "status": "valid",
  "dmint": {
    "v": "1",
    "rules": [
      {
        "o": {
          "512031994d8ee86db69f327bc54f675d8ff6349fc0e28d1defe52dd17e4fbf4026dd": {
            "v": 88888
          }
        },
        "p": "^[0-7]$",
        "bitworkc": "81981981.9"
      },
      {
        "o": {
          "512031994d8ee86db69f327bc54f675d8ff6349fc0e28d1defe52dd17e4fbf4026dd": {
            "v": 79999
          }
        },
        "p": "^[8-9]$|^1-6[0-9]$|^7[0-1]$",
        "bitworkc": "8198198.1"
      },
      {
        "o": {
          "512031994d8ee86db69f327bc54f675d8ff6349fc0e28d1defe52dd17e4fbf4026dd": {
            "v": 59999
          }
        },
        "p": "^7[2-9]$|^8-9[0-9]$|^1-3[0-9][0-9]$|^4[0-7][0-9]$|^48[0-5]$",
        "bitworkc": "819819.8"
      },
      {
        "o": {
          "512031994d8ee86db69f327bc54f675d8ff6349fc0e28d1defe52dd17e4fbf4026dd": {
            "v": 39999
          }
        }
      }
    ]
  }
}

```

rules 里面是一个数组，里面有若干条规则，每条规则用 { } 包裹。我们来看第一条规则：

```

1  {
2    "o": {
3      "512031994d8ee86db69f327bc54f675d8ff6349fc0e28d1defe52dd17e4fbf4026dd": {
4        "v": 88888
5      }
6    },
7    "p": "^[0-7]$",
8    "bitworkc": "81981981.9"
9  }

```

o字段：第一行表示需要向脚本

512031994d8ee86db69f327bc54f675d8ff6349fc0e28d1defe52dd17e4fbf4026

dd输出 88888 sats

p字段：匹配该规则的 item，用正则表达式规定。例如正则表达式`^[0-7]$`的含义就是匹配以 0~7 开头的并且以7结尾的 item。比如你想打的 NFT 的编号是 0~7，那么你就需要符合这条规则。

bitworkc字段：这表明你要提交该 mint 交易时需要的工作量。例如`81981981.9`表示你的 commit tx id 前8位必须是89981981，第9位必须是 9~f 的数字（16进制）。需要匹配八位数字，这个 POW 的难度可以说是非常难了，一般匹配 7 位数都需要花上个人电脑 CPU 几天的时间来计算。

综上所述，第一条规则的意思就是：mint token id 以0~7开头的 NFT 需要支付 88888 聪的费用，并且需要完成前缀为81981981.9的POW证明。

我们再来看最后一条规则：

```
{
  "o": {
    "512031994d8ee86db69f327bc54f675d8ff6349fc0e28d1defe52dd17e4fbf4026dd": {
      "v": 19999
    }
  },
  "p": ".*",
  "bitworkc": "8198.10"
}
```

有了上面的解释，这条规则的含义就很清晰了，除了前面几条规则匹配到的 Token ID，剩下的所有其他 NFT，需要支付 19999 聪的费用，并完成前缀为 8198.10 的 POW证明（.*的意思就是匹配所有情况）。

3. 激活高度与总量

```
    },  
    "p": ".*",  
    "bitworkc": "8198.10"  
  }  
],  
  "merkle": "b7df859ac9e630771d2ddef1a5",  
  "immutable": true,  
  "mint_height": 819819,  
  "items": 10000  
},
```

在规则后面，就显示了 `mint_height` 和 `items` 信息，他们一个是开始 mint 的区块高度，一个是这个容器所含的 NFT 总量。

4. 其他信息

返回的整个 json 还包含了很多其他可能有用的东西，比如这个容器的唯一编号 `atomical_id`，有兴趣的朋友请自行研究吧。

另外我将 Atomicals 的官方文档爬了下来做了一个 GPTs，开通了 ChatGPT Plus 的朋友

有问题可以问它噢！但不保证答案正确哈：<https://chat.openai.com/g/g-WlsAjMyLg-atomicals-guide>

五、查重

查重是 dmint 你看中的 NFT 之前的必备工作，因为你不知道链上是否有其他人已经提交了该 NFT 的 mint，所以在 mint 之前一定要查重，并且在 POW 计算的过程中，每产生一个新的区块就要查重一次，如果发现有人已经 mint 了你的编号，马上换其他编号。

这里有一个极端情况——你和你的对手（mint 同一个 token id 的人）在同一个区块

提交了 commit 交易，据群友的经验，似乎 gas 高的那一方会成为获胜者，所以真正的查重还需要查询内存池里面的交易，但这已经超出了本文的范畴。

查重的命令很简单：

```
yarn cli get-container-item <container name> <token id>
```

例如我想打capybaras的8601号 NFT，它长这样：



那么命令就是：

```
yarn cli get-container-item capybaras 8601
```

返回的结果也是 json：

```
{
  "success": true,
  "data": {
    "status": "verified",
    "candidate_atomical_id": "0000e32c0baea9c29f61c095f1aa869017b7bfd45921bb842i0",
    "atomical_id": "0000e32c0baea9c29f61c095f1aa869017b7bfd45921bb842i0",
    "candidates": [
      {
        "tx_num": 930581206,
        "atomical_id": "0000e32c0baea9c29f61c095f1aa869017b7bfd45921bb842i0",
        "txid": "0000e32c0baea9c29f61c095f1aa869017b7bfd45921bb842i0",
        "commit_height": 819809,
        "reveal_location_height": 819809
      }
    ],
    "type": "item"
  }
}
```

可以看到 status 为 verified，这说明这个 NFT 已经被 mint，且距离 mint 成功已经过了四个区块，是铁打不动的再也没人能抢走的了。

如果 status 是 pending，就说明 mint 交易已经成功，但是还没有过四个区块，这个时候仍是悬而未定的，但是也和你没关系了，直接换另一个 token id 吧。

如果 status 是 null，那么就可以 mint，但是不能确保内存池中是否已经存在和你 token id 相同的 commit 交易。

```
(base) → atomicals-js git:(master) x yarn cli get-container-item crabada 1111
yarn run v1.22.19
warning ../../../../package.json: No license field
$ node dist/cli.js get-container-item crabada 1111
{
  "success": true,
  "data": {
    "status": null,
    "candidate_atomical_id": null,
    "atomical_id": null,
    "candidates": [],
    "type": "item"
  }
}
✨ Done in 0.52s.
```

我们来看 toothy 6666 号的查重信息（截取了部分）：

```
{
  "success": true,
  "data": {
    "status": "verified",
    "candidate_atomical_id": "161864d654a0ab0c867f9e82d1029c0e9c8b453e9f0ad77ebb8e3c95acd05a69i0",
    "atomical_id": "161864d654a0ab0c867f9e82d1029c0e9c8b453e9f0ad77ebb8e3c95acd05a69i0",
    "candidates": [
      {
        "tx_num": 928338397,
        "atomical_id": "161864d654a0ab0c867f9e82d1029c0e9c8b453e9f0ad77ebb8e3c95acd05a69i0",
        "txid": "161864d654a0ab0c867f9e82d1029c0e9c8b453e9f0ad77ebb8e3c95acd05a69",
        "commit_height": 819181,
        "reveal_location_height": 819181
      },
      {
        "tx_num": 928338569,
        "atomical_id": "1618b7cf064890d2fa9e49b048bd5b8b0247ccf66a5785033a541927994a237ei0",
        "txid": "1618b7cf064890d2fa9e49b048bd5b8b0247ccf66a5785033a541927994a237e",
        "commit_height": 819181,
        "reveal_location_height": 819181
      },
      {
        "tx_num": 928341699,
        "atomical_id": "16188d260d3584c9b83117e9578feb00158096ba71c9b61a25d42bfa6ce106bci0",
        "txid": "16188d260d3584c9b83117e9578feb00158096ba71c9b61a25d42bfa6ce106bc",
        "commit_height": 819182,
        "reveal_location_height": 819182
      }
    ]
  }
}
```

可以看到 candidates 字段有3个元素，很明显第2和第3个打废了，第2个虽然和第1个在同一个区块成交（819181），但是第一个给的 gas 应该高一些，我们来用 txid 在区块链浏览器查一下：

候选人1:

交易		712 confirmations
时间戳	2023-12-01 07:19 (5天之前)	手续费 10,600 聪 US\$4.00
交易特征	SegWit Taproot RBF	费率 68.8 聪/字节
		有效收费率 166 聪/字节 多缴3倍
		CPFP ⓘ

候选人2:

交易		712 confirmations
时间戳	2023-12-01 07:19 (5天之前)	手续费 6,760 聪 US\$2.55
交易特征	SegWit Taproot RBF	费率 43.9 聪/字节
		有效收费率 100 聪/字节 多缴2倍
		CPFP ⓘ

可以看到候选人1给的 gas 是 166，而候选人2只有100。

六、mint

有了上面的铺垫之后，mint 也就是一个命令的事：

```
yarn cli mint-item <container name> <token id> <manifest file>
```

需要着重解释的是 <manifest file> 这个字段，它是你 token id 对应的 json 文件的路径，里面包含 <token id> 号 NFT 的元数据，包括这个 NFT 图片的二进制数据。它通常由项目方提供下载。

如果你需要 mint-item 命令的详细帮助，可以执行：

```
yarn cli mint-item -h
```

```

Mint item non-fungible token (NFT) Atomical from a decentralized container

Arguments:
  containerName      string
  itemName           string
  manifestFile       string

Options:
  --rbf              Whether to enable RBF for transactions.
  --owner <string>   Owner of the parent Atomical. Used for direct subrealm
                     minting.
  --initialowner <string> Initial owner wallet alias to mint the Atomical into
  --satsbyte <number>   Satoshis per byte in fees (default: "15")
  --satsoutput <number> Satoshis to put into the minted atomical (default: "100
0")
  --funding <string>    Use wallet alias WIF key to be used for funding and cha
nge
  --container <string>  Name of the container to request
  --bitworkc <string>   Whether to put any bitwork proof of work into the token
mint. Applies to the commit transaction.
  --bitworkr <string>   Whether to put any bitwork proof of work into the token
mint. Applies to the reveal transaction.
  --disablechalk       Whether to disable the real-time chalked logging of eac
h
                           hash for mining. Improvements mining performance to set
this
                           flag
  -h, --help           display help for command
🌟 Done in 0.93s.

```

这里值得关注的是上面红框中的 5 个选项。

--rbf: 如果开启, 这笔交易就可以加速了, 如果你后面发现 gas 不够, 那么开启这个选项就可以使用 Sparrow Wallet 等支持加速的钱包软件来加速该笔交易。这应该是最近才更新的功能。

--initialowner: mint 到的 NFT 发到哪一个地址, 也就是我们通常说的接收地址 (receive address) 。

--satsbyte: 这个没什么好说的, 就是 gas fee, 听说已经修复了实际 gas 比给的 gas 高很多的 bug, 我还没测试。

--funding: 就是你要用哪个钱包付款, 就是我们通常说的 funding address, 你 mint 的时候跳出来的那个二维码地址, 就是 funding address。

--disablechalk: 关闭挖矿时每个哈希实时记录的功能。设置此项可以提高挖矿性能。

要注意 **initialowner**和 **funding**选项，后面跟的是这个钱包的别名，而不是钱包的地址，例如我的 wallet.json 长这样：

```
{
  "phrase": "merry hungry rich suspect culture sibling woman split regular glimpse dog screen",
  "primary": {
    "address": "bc1p3025cncek2vc9cxgptep55mh6dz6lqrmvz2nekwe3hj00trwdxrs5wq8tl",
    "path": "m/86'/0'/0'/0/0",
    "WIF": "L5FCGqD9GbT9wf8rsJ75TpYvXTdVq6DZ9Sh7gFB5RdYwiefzkauv"
  },
  "funding": {
    "address": "bc1pps7ds367xlnn7a3kptnfhtjq3eqn5g0mv2uvfdy4mwhkqyxck2qkp0usq",
    "path": "m/86'/0'/0'/1/0",
    "WIF": "L22cv3mjUQunZsQivCxQbHiYbgbRk3FGsDXodHf3JUWnpZ3HgQxQ"
  },
  "imported": {
    "aaa": {
      "address": "bc1pqxa0wgvagme3fvqmadg4au35v84xn3jlsfqs7kmrwkl8t876rs6r4pj5",
      "path": "m/44'/0'/0'/0/0",
      "WIF": "L4muX6fUuEenDLKAoyTLW6Ci9txFyfwPx8qBLup9yLyhwHYx5obA"
    },
    "bbb": {
      "address": "bc1pd8nytu9rp3n3d27les6tvc5ytlh36a3k0vrla2a3kwr9l7ta55wq34efye",
      "path": "m/44'/0'/0'/1/0",
      "WIF": "L39Kdt6GjbxFA6nx1ywuQ4bH4bcm9rQsB5sPcogt4rZAnhGPjs1r"
    }
  }
}
```

我导入了两个钱包，一个取名为 AAA，一个 BBB，那么我就可以吧 AAA 或者 BBB 作为 接收钱包或者发送钱包，可能的命令例如：

```
yarn cli mint-item crabada 1111 ./crabada/item-1111.json --
initialowner AAA --funding BBB --satsbyte 40
```

七、购买 NFT

dmint 为项目方设计了高级支付规则，其实就是我们通常意义上的支付代币购买 NFT。上文所述皆为 Free Mint 项目的 mint 操作，只需要给手续费，而增加了高级支付规则的容器，需要用户在 mint 完成后，向项目方支付指定的数额的 btc 才是真正的拥有了这个 NFT。整个 mint 周期的费用组成如下：

总费用 = commit + reveal + satsoutput + advancedPayment

commit 和 reveal 的费用很好理解，这个由矿工收取，satsoutput 是放入 atomicals 中的 sats 的数量，严格意义上来说不算费用，而最后一项

advancedPayment 包括需要支付给项目方的 NFT 购买费用 + 这笔支付交易需要支付的矿工手续费，对于 free mint（容器元数据的 rules 字段中没有"o"字段，只有"p"和"bitworkc"字段）的项目来说，没有这一项，而开通了高级支付规则的容器，例如 scientists 和 crabada 等，需要额外的支付才能真正的拥有 NFT。

!! 重要声明：本节是本文难度最大的一章，如果你看不懂，也无法确定每一步，请不要进行操作，以免造成资金损失。

我们以 mint crabada 第 8 号 NFT 为例讲解整个 mint 流程。

1. mint-item

这是 mint NFT 必备的流程，请参见第六节。

2. 获取项目方的收款地址

通过查询 crabada 的元数据可知（查询容器元数据的方法在第四节），8 号 NFT 命中下面的规则——即需要向

512031994d8ee86db69f327bc54f675d8ff6349fc0e28d1defe52dd17e4fbf4026dd 支付 79999 sats，并且完成前缀规则为 8198198.1 的 POW 计算证明。

```
{
  "o": {
    "512031994d8ee86db69f327bc54f675d8ff6349fc0e28d1defe52dd17e4fbf4026dd": {
      "v": 79999
    }
  },
  "p": "^[8-9]$|^1-6[0-9]$|^7[0-1]$",
  "bitworkc": "8198198.1"
},
```

512031994d8ee86db69f327bc54f675d8ff6349fc0e28d1defe52dd17e4fbf4026dd 是个什么东西呢？我们叫它**锁定脚本**（locking script or scriptPubkey），每一笔交易的输出都有一个锁定脚本，指示这个输出所携带的聪的拥有者，对于一般脚本来说，锁定脚本通常包含公钥或者公钥哈希，持有公钥对应的私钥的人就能够动用这笔资金。

所以锁定脚本里面一般是包含比特币地址（严谨地说是公钥或者公钥哈希）的，注意看，上文的锁定脚本以 5120 开头，这正好是 P2TR 类型地址的锁定脚本，我们可以使用 CLI 自带的 script-address 命令解析这个输出脚本，得到比特币地址：

```
yarn cli script-address <输出脚本>
```

例如：

```
yarn cli script-address
```

```
512031994d8ee86db69f327bc54f675d8ff6349fc0e28d1defe52dd17e4fb  
f4026dd
```

得到结果：

```
(base) → atomicals-js git:(master) ✖ yarn cli script-address 5120fcfc1c83d55034c206f896a8f36  
yarn run v1.22.19  
warning ../../../../package.json: No license field  
$ node dist/cli.js script-address 5120fcfc1c83d55034c206f896a8f36403678de0004b4393d739d089000  
Address: bc1pln7peq742q6vyphcj650xeqrv7x7qqztgwfwwwws3yqqz4xqjp2q7g84cu  
-----  
✨ Done in 1.15s.
```

bc1pln7peq742q6vyphcj650xeqrv7x7qqztgwfwwwws3yqqz4xqjp2q7g84cu 就是项目方的收款地址，记录下来，后面要用。

3. 获取 atomical_id

Atomicals 协议的每个 Atomicals 对象都有一个全局唯一的 id 用于标识，我们需要对这个 Atomicals 对象付款，所以肯定要达到这个对象的 id，不然命令怎么会知道你付款的是哪一个 NFT 呢？

获取 atomical id 的命令也很简单，上文已经介绍过了：

```
yarn cli get-container-item <容器名称> <NFT 编号>
```

例如我们这里的例子，查找 crabada 第 8 号 NFT 的信息：

```
yarn cli get-container-item capybaras 8
```

```
(base) → atomicals-js git:(master) ✖ yarn cli get-container-item capybaras 8  
yarn run v1.22.19  
warning ../../../../package.json: No license field  
$ node dist/cli.js get-container-item capybaras 8  
{  
  "success": true,  
  "data": {  
    "status": null,  
    "candidate_atomical_id": null,  
    "atomical_id": null,  
    "candidates": [],  
    "type": "item"  
  }  
}  
✨ Done in 1.06s.
```


可以看到这个 NFT 目前是没有被打的，为了方便说明，我们用 toothy 6666 号的数据：

```
{
  "success": true,
  "data": {
    "status": "verified",
    "candidate_atomical_id": "161864d654a0ab0c867f9e82d1029c0e9c8b453e9f0ad77ebb8e3c95acd05a69i0",
    "atomical_id": "161864d654a0ab0c867f9e82d1029c0e9c8b453e9f0ad77ebb8e3c95acd05a69i0",
    "candidates": [
      {
        "tx_num": 928338397,
        "atomical_id": "161864d654a0ab0c867f9e82d1029c0e9c8b453e9f0ad77ebb8e3c95acd05a69i0",
        "txid": "161864d654a0ab0c867f9e82d1029c0e9c8b453e9f0ad77ebb8e3c95acd05a69",
        "commit_height": 819181,
        "reveal_location_height": 819181
      },
      {
        "tx_num": 928338569,
        "atomical_id": "1618b7cf064890d2fa9e49b048bd5b8b0247ccf66a5785033a541927994a237ei0",
        "txid": "1618b7cf064890d2fa9e49b048bd5b8b0247ccf66a5785033a541927994a237e",
        "commit_height": 819181,
        "reveal_location_height": 819181
      },
      {
        "tx_num": 928341699,
        "atomical_id": "16188d260d3584c9b83117e9578feb00158096ba71c9b61a25d42bfa6ce106bci0",
        "txid": "16188d260d3584c9b83117e9578feb00158096ba71c9b61a25d42bfa6ce106bc",
        "commit_height": 819182,
        "reveal_location_height": 819182
      }
    ]
  }
}
```

假设我就是第一个候选人，并且 status 已经是 verified 了，说明我可以开始付款了。注意：如果你查询后发现你不是第一个候选人，那这个 NFT 和你基本无缘了，这在上文说得很清楚，并且你需要等到四个区块确认之后，才能开始付款，也就是 status 是 verified 状态，而不是 pending 状态，pending 状态的 NFT 还没有经过四个区块的确认。这里我们的 atomical_id 是 161864d654a0ab0c867f9e82d1029c0e9c8b453e9f0ad77ebb8e3c95acd05a69i0，需要把它记录下来，后面要用，这里我们假设这个 atomical_id 是 crabada 8 号 NFT 的。

3. 使用 transfer-builder 命令进行支付

我们先来整理一下上文有用的数据：

NFT: crabada 第 8 号

atomical_id:

161864d654a0ab0c867f9e82d1029c0e9c8b453e9f0ad77ebb8e3c95a
cd05a69i0

支付地址:

bc1pln7peq742q6vyphcj650xeqrv7x7qqztgwfwwwws3yqqz4xqjp2q7g8
4cu

支付金额: 79999 sats

很好, 下面我们可以进行最后一步了, 使用 CLI 提供的 transfer-builder 命令进行高级支付了:

```
Usage: Atomicals CLI Utility transfer-builder [options]
Transfer plain regular UTXOs to another addresses

Options:
  --owner <string>          Use wallet alias WIF key to move the Atomical
  --funding <string>        Use wallet alias WIF key to be used for funding and change
  --satsbyte <number>       Satoshis per byte in fees (default: "15")
  --nofunding                Do not ask for separate funding, use existing utxo
  --skipvalidation           Do not do FT transfer validation on broadcast (danger)
  --atomicalreceipt <string> Attach an atomical id to a pay receipt
  --atomicalreceipttype <string> Attach receipt type p or d
  -h, --help                display help for command
🌟 Done in 1.10s.
```

transfer-builder 命令提供 7 个选项, 下面一一解释。

--owner: 不懂

--funding: 使用哪个钱包来支付, 如果省略此项, 应该是默认用 funding address

--satsbyte: 手续费价格, 不用解释了吧

--atomicalreceipt: 这里就是填 atomical_id 的地方, 指定需要向哪个 atomical 对象支付。

--atomicalreceipttype: 可以填 d 或者 p, d 代表 dmint 对象, p 代表子领域对象, 如果大家对 atomicals 协议的领域 (realm) 有所了解的话, 相信知道子领域是什么。

我们要 mint 的是 container dmint item, 所以 atomicalreceipttype 的值当然填 d。

例如我们要支付刚刚打的 crabada 第 8 号 NFT，那么完整命令就是：

```
yarn cli transfer-builder --satsbyte 50 --atomicalreceipt  
161864d654a0ab0c867f9e82d1029c0e9c8b453e9f0ad77ebb8e3c95acd05  
a69i0 --atomicalreceipttype d
```

命令会让你选择funding钱包的utxo，接下来会让你输入付款地址，这个付款地址就是步骤2获取到的

bc1pln7peq742q6vyphcj650xeqrv7x7qqztgwfwwwws3yqqz4xqjp2q7g84cu，最后等待提交内存池确认。

注意：高级支付的机制是只要在 15 个区块内，完成了支付（交易确认），这个 NFT 就属于你了，如果超过15 个区块还没有确认，那么你前面的所有费用都打水漂了，其他的人就可以来 mint 这个 NFT 了，如果你还想要这个 NFT，那么你和所有人又站在同一起跑线上了。

由于目前没有合适的容器项目让我测试（crabada 实在太贵），所以最后这一步的支付参考的 chenandzhu 的推文，后面如果有合适的项目，我会补上一些细节和截图。

本节参考了：

@chenandzhu: <https://x.com/chenandzhu/status/1730593973918630314?s=20>

aandds website: <https://aandds.com/blog/bitcoin-taproot.html>

八、结语

感谢能坚持看到此处的朋友！Atomicals Proctocol 很年轻，但这里已经聚集了很多优秀的 builder 和 contributors，还有很多看好 Atomicals 的人，如果你认为 Atomicals 未来也大有可为，欢迎关注我的推特，获得更多 Atomicals 相关的知识、脚本等。

个人推特: [wusimpl \(@wusimpl\) / Twitter](#)