# 一、代码

## 1. 模拟浏览器的代理

user\_agent\_forHeader.py

（1）使用说明：

from user\_agent\_forHeader import user\_agent

headers = {"User-agent": user\_agent[0]}

或者，

session.headers.update({"User-agent": user\_agent[0]})

（2）特别注意：

有时可能会因为会话仍在保持连接，而因为在多个代码模块建立多个会话请求，而导致错误。

## 2. Json文件的保存和加载

|  |  |  |
| --- | --- | --- |
| 1 | saveJson.py | 使用说明：  from saveJson import save\_json  save\_json(“文件路径”, list或dict) |
| 2 | loadJson.py | 使用说明：  from loadJson import load\_json  list\_name = load\_json('文件路径') |

## 1. 爬虫代码（智能合约爬取）

|  |  |  |
| --- | --- | --- |
| 1 | crawl\_contracts\_update.py | 实时更新24h内的爬虫内容。 |
| 2 | crawl\_contracts\_yesterday.py | （1）获取最近24h内的爬虫内容；  （2）获取昨天一天内的爬虫内容。 |
| 3 | crawl\_contracts.py | 当前时刻下获取最新的500条智能合约。 |

## 2. 以太币单位转换

ether\_value\_convert.py

（1）介绍：

etherscan上展示的资金流动，其所使用的以太币的单位不太统一，为了在后续更方便的进行计算、比较等操作，统一单位是必要的。

（2）函数说明

在我们的实现中，主要的函数为：

value\_convert2ether(string\_num\_with\_currencyUnit)。

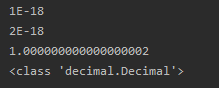
|  |  |  |
| --- | --- | --- |
| 函数名称 | value\_convert2ether() | （示例） |
| 输入 | 待转化的、携带单位的以太币表示。 | "2 Wei" |
| 返回值 | 不携带单位的以太币表示  （默认统一的单位是“ether”） | 2E-18 |

（3）使用：

from ether\_value\_convert import value\_convert2ether

string\_num\_with\_currencyUnit = "1 Wei"  
finished\_converted\_ether\_value = value\_convert2ether(string\_num\_with\_currencyUnit)  
print(finished\_converted\_ether\_value)  
print(finished\_converted\_ether\_value+finished\_converted\_ether\_value)  
print(finished\_converted\_ether\_value+finished\_converted\_ether\_value+Decimal('1'))  
print(type(finished\_converted\_ether\_value))

输出为：



## 3. etherscan.py

（1）介绍：

利用etherscan API实现的简单客户端。该客户端可以通过etherscan提供的API接口实现对一些可用信息的获取。

（2）函数说明：

当前的实现中，所有的函数基本实现在Client类中。

* get\_contract\_byte\_code\_by\_address(address, tag=BYTE\_CODE\_TAG\_LATEST)

|  |  |
| --- | --- |
| 函数名称 | get\_contract\_byte\_code\_by\_address() |
| 输入 | 智能合约的地址 |
| 返回值 | 智能合约的字节码 |

* get\_contract\_source\_code\_by\_address(address)

|  |  |
| --- | --- |
| 函数名称 | get\_contract\_source\_code\_by\_address |
| 输入 | 智能合约的地址 |
| 返回值 | 智能合约的源码 |

* list\_transactions\_by\_address(address,

transaction\_type=TRANSACTION\_TYPE\_NORMAL,

start\_block=0,

end\_block=99999999,

page=1,

offset=REQUEST\_LIMIT,

sort="asc")

|  |  |
| --- | --- |
| 函数名称 | list\_transactions\_by\_address() |
| 输入 | （1）address：可以是智能合约账户的地址，也可以是EOA地址； （2）transaction\_type：  TRANSACTION\_TYPE\_NORMAL或者  TRANSACTION\_TYPE\_INTERNAL  （3）sort："asc" 或者 "desc" |
| 返回值 | 对应账户地址的内部或外部交易列表，该列表是正序或逆序排列的。 |

（3）使用：

from etherscan import Client

Client.函数名称

## 4. 专用于创建请求会话

create\_session.py

（1）介绍：

因为在代码实现过程中，总需要与etherscan进行交互，因此，创建请求会话会在各种代码文件被用到，作为一段共享代码，将其单独生成为一个代码文件。

（2）使用：

from create\_session import get\_session

session = get\_session()

## 5. （一组）tx详情页的调用链

try\_get\_eao2targetAddr\_ifhas\_inCalls\_from\_txHash.py

共有4个函数组成：

* get\_single\_txSoup(txHash,session)

作用: 请求并解析tx详情页。注: etherscanCN上没有advanced模式

|  |  |
| --- | --- |
| 函数名称 | get\_single\_txSoup() |
| 参数 | 参数1: txHash  参数2: session |
| 返回值 | tx详情页的soup。 |

* get\_single\_tx\_callDict(soup)

作用: 获取单个tx详情页下的调用链信息, 获取到的调用链以字典的形式返回。

|  |  |
| --- | --- |
| 函数名称 | get\_single\_tx\_callDict() |
| 参数 | soup |
| 返回值 | 单个tx详情页的调用链。 |

返回的字典形式为:  
{  
 'tx\_origin':   
 { 'From': '0xe0b4789d1cb42f0ee149bfbc12f016dbd55db33b',   
 'To': '0x6f49baef279f8c40d7479f5be6a9bbf5fc607af0',   
 'Transfer': '1 Ether'  
 },   
 'call\_0':   
 {  
 'From': '0x6f49baef279f8c40d7479f5be6a9bbf5fc607af0',   
 'To': '0x95d34980095380851902ccd9a1fb4c813c2cb639',   
 'Transfer': '1 Ether'},   
 'call\_1\_0':  
 {  
 'From': '0x95d34980095380851902ccd9a1fb4c813c2cb639',   
 'To': '0x6f49baef279f8c40d7479f5be6a9bbf5fc607af0',   
 'Transfer': '1 Ether'  
 },   
 'call\_1\_0\_0\_0':   
 {  
 'From': '0x95d34980095380851902ccd9a1fb4c813c2cb639',   
 'To': '0x6f49baef279f8c40d7479f5be6a9bbf5fc607af0',   
 'Transfer': '1 Ether'  
 }  
}

* get\_txhash2calls\_dict(unique\_TxHash\_list)

作用: 对一组txs, 用这些tx的hash作为key, 调用链字典作为value, 组合生成一组tx的调用链集合, 并以字典的形式返回。

|  |  |
| --- | --- |
| 函数名称 | get\_txhash2calls\_dict() |
| 参数 | unique\_TxHash\_list。一组txs组成的列表。 |
| 返回值 | 一组tx的调用链集合构成的字典。 |

返回的字典的形式为:  
{  
 tx\_hash: 该单个tx的call\_dict  
}

* get\_isToAddrAContract(soup)

作用: 判断tx详情页中的To字段中的地址是否为智能合约的地址。如果是, 返回True; 如果不是, 返回False.

|  |  |
| --- | --- |
| 函数名称 | get\_isToAddrAContract() |
| 参数 | soup |
| 返回值 | True或False。 |

## 6. 关注的外部交易hash列表（单个地址上）

try\_get\_txs\_from\_singleAddr.py

共有两个函数组成：

* getAll\_txs\_list(address\_single,session)

作用: 利用etherscan提供的API接口获取当前(EOA或SC的)地址的外部交易(详情页)信息的列表。

|  |  |
| --- | --- |
| 函数名称 | getAll\_txs\_list() |
| 参数 | 参数1: address\_single。可以是EOA地址, 也可以是SC的地址。  参数2: session。 |
| 返回值 | 一组**外部**交易(详情页)信息的列表。 |

* get\_concerned\_txs\_list(all\_txs\_list,session)

作用: 本函数的作用只是助力获取有效的、最值得关注的调用链。

排除了两种不需要关注的情况:

(1) to字段的内容如果是""时,最常见的情况就是智能合约的创建。在获取调用链方面, 智能合约创建的相关交易确实不是值得我们关注的(注: 对于其他情况, 则需要另外讨论)。

(2) 若当前tx详情页的To字段判断为不是智能合约,则该tx不可能包含内部交易,因此也不太值得关注。

|  |  |
| --- | --- |
| 函数名称 | get\_concerned\_txs\_list() |
| 参数 | 参数1: all\_txs\_list。一组交易(详情页)信息的列表。  参数2: session |
| 返回值 | concerned\_txHashs\_list。一组值得关注的tx的hash组成的列表。 |

注：参数1 (all\_txs\_list)对应的交易一般应为外部交易，因为内部交易的父tx一定包含内部调用链，所以内部交易的父hash都是值得关注的。

## 7. 获取内部（父）交易hash列表（单个）

try\_get\_internalTxs\_from\_singleAddr.py

主要由2个函数组成。

* getAll\_internal\_txs\_list(address\_single, session)

（1）作用介绍：

获取对应单个地址（EOA或SC账户）的所有内部交易(父tx)信息列表。

（PS: 获取的信息中，有效信息较少）

（2）函数说明：

|  |  |
| --- | --- |
| 函数名称 | getAll\_internal\_txs\_list() |
| 参数 | 参数1: address\_single, 智能合约地址或EOA账户地址  参数2: session, 已经建立的会话。 |
| 返回值 | internal\_txs\_list. 与某个账户地址关联的所有内部交易信息列表。 |

（3）返回值的形式如下：

[

{'blockNumber': '5079425',

'timeStamp': '1518475652',

'hash': '0xed5eaf959224dff10fb705e16af81017374f3f3856ada926571aa9f8ef8e0b63',

'from': '0x6f49baef279f8c40d7479f5be6a9bbf5fc607af0',

'to': '0x95d34980095380851902ccd9a1fb4c813c2cb639',

'value': '1000000000000000000',

'contractAddress': '',

'input': '',

'type': 'call',

'gas': '559457',

'gasUsed': '0',

'traceId': '0',

'isError': '0',

'errCode': ''

},

......

]

* get\_internal\_txs\_uniquePTH\_list(all\_internal\_list)

（1）作用介绍：

接收一个交易信息列表，从中提取有效信息，获取其对应交易的hash地址，最终返回记录tx hash的列表。

（2）函数说明：

|  |  |
| --- | --- |
| 函数名称 | get\_internal\_txs\_uniquePTH\_list() |
| 参数 | all\_internal\_list.  某个地址相关的所有内部(父)交易列表（非advanced模式下）. |
| 返回值 | parent\_txHash\_list.  记录tx hash的列表。 |

（3）生成的tx hash的列表，主要用于助力get\_txhash2calls\_dict()函数的执行，获取（一组）tx详情页的调用链。

## 8. 创建溯源：创建者EOA创建的合约

try\_get\_allCreatedSCs\_byhpCreator.py

由3个函数组成。

* get\_contract\_creator(all\_txs\_list, address\_single,session)

作用：获取相关（创建者）EOA创建的所有合约。

1. 当address\_single为智能合约，all\_txs\_list为该智能合约的结果时: 该函数会首先定位合约的创建者, 获取创建者的（外部）交易列表; 然后获取该创建者创建的所有合约（地址）。

2. 当address\_single为EOA，all\_txs\_list为该EOA的结果时: 该函数会获取该EOA账户创建的所有合约（地址）。

|  |  |
| --- | --- |
| 函数名称 | get\_contract\_creator() |
| 参数 | 参数1: all\_txs\_list, 智能合约或者EOA账户的外部交易列表。  参数2: address\_single, 智能合约或EOA账户的地址  参数3: 已建立的session链接。 |
| 返回值 | contract\_creation\_eoa2sc\_dict。  key：EOA地址；  value：相关（创建者）EOA创建的所有合约构成的列表。 |

* get\_contract\_creator\_withByteCode(all\_txs\_list, address\_single,session)

作用:

1. 当address\_single为智能合约，all\_txs\_list为该智能合约的结果时: 该函数会首先定位合约的创建者, 获取创建者的交易列表; 然后获取该创建者创建的所有合约（地址+字节码）。

2. 当address\_single为eoa，all\_txs\_list为该eoa的结果时: 该函数会获取该eoa账户创建的所有合约（地址+字节码）。

|  |  |
| --- | --- |
| 函数名称 | get\_contract\_creator\_withByteCode() |
| 参数 | 参数1: all\_txs\_list, 智能合约或者eoa账户的外部交易列表。  参数2: address\_single, 智能合约或eoa账户的地址  参数3: 已建立的session链接。 |
| 返回值 | contract\_creation\_eoa2sc\_dict。  key：EOA地址；  value：{一组EOA创建的SC: 对应字节码}构成的字典 |

* get\_uniqueByteCode\_withRepeatTimes(Eoa\_sc\_bytes\_dict)

作用:

对于给定的合约字节码, 给出两个列表, 分别是m\_same\_sc和createdby\_n\_eoa。

m\_same\_sc记录了具有相同字节码的合约地址;

createdby\_n\_eoa记录了创建具有相同字节码合约的账户地址。

|  |  |
| --- | --- |
| 函数名称 | get\_uniqueByteCode\_withRepeatTimes() |
| 参数 | Eoa\_sc\_bytes\_dict。  EOA账户所创建的所有智能合约(地址+字节码)的字典. |
| 返回值 | unique\_contract\_dict。  unique\_contract = {bytes: {m\_same\_sc:[same\_numbers, scAddrx], createdby\_n\_eoa:[num, eoax]} |

注：对于hp，当越多个合约地址处都拥有相同的字节码，则可疑度越高，对应的EOA更需要被关注；

当EOA创建相同hp字节码的次数越多，该EOA越可疑，需要被高度关注。

## 9. 调用链溯源：蜜罐相关tx+tx的EOA调用

try\_get\_all\_counts\_withUseHPtimes.py

在该文件中包含两个函数：

* get\_all\_relatedTxCalls\_dict(hp\_addr,session)

（1）作用介绍：

在使用到hp的所有相关交易（外部tx+内部的父tx）中, 对应每个交易, 获取它们的调用链。

（2）函数说明：

|  |  |
| --- | --- |
| 函数名称 | get\_all\_relatedTxCalls\_dict() |
| 参数 | 参数1: hp\_addr, 蜜罐地址;  参数2: session, 已经建立连接的session会话。 |
| 返回值 | pth2calls\_dict：与hp相关的所有tx的调用链集合。 |

（3）关联函数（调用到的函数）：

getAll\_txs\_list()：获取hp\_addr的外部交易列表.

get\_concerned\_txs\_list()：在hp\_addr的外部tx列表中，筛选值得关注的交易列表.

get\_txhash2calls\_dict()：对关注的tx列表中的每个交易, 获取其对应的调用链.

getAll\_internal\_txs\_list()：获取hp\_addr的所有内部交易列表.

get\_internal\_txs\_uniquePTH\_list：对每个内部交易, 获取其对应的父交易的hash地址，最终返回父hash的交易列表.

* get\_all\_mayUseHP\_addr2times\_dict(hp\_relatedTxCalls\_dict)

（1）作用介绍：

在使用到hp的所有相关交易中, 定位这些交易的创建者eoa, 以及他们发起过几次调用hp的交易。

（2）函数说明：

|  |  |
| --- | --- |
| 函数名称 | get\_all\_mayUseHP\_addr2times\_dict() |
| 参数 | hp\_relatedTxCalls\_dict：与hp相关的所有tx的调用链集合。 |
| 返回值 | all\_mayUseHP\_addr2times\_dict：  key是与hp相关的tx的创建者EOA,  value是发起相关tx的次数。 |

注：使用hp\_relatedTxCalls\_dict做参数的原因是：相关的信息可以直接从中获取，而不需要额外的网页请求和内容解析。

（3）返回的字典形式如下：

{

'0xe0b4789d1cb42f0ee149bfbc12f016dbd55db33b': 2,

'0x6b35c9f9670c0641e3d5e80642992223d1ffec7b': 1,

'0x3476e2628006a7f74285c44b501437b415b08299': 1

}

## 10. 继调用链溯源：可疑EOA创建的合约

（可能在某些情况下，可疑EOA并不包括hp的创建者EOA）

try\_get\_allCreatedSc\_byUseHpEOAs.py

主要包含1个函数：

* get\_all\_CreatedSC\_byUsedHpEOA\_dict(all\_mayUseHP\_EOA2times\_dict, session)

（1）作用: 其实就是为每个地址做了一个循环。用于批量获取一堆账户地址所创建的智能合约（地址+字节码）。

（2）函数说明：

|  |  |
| --- | --- |
| 函数名称 | get\_all\_CreatedSC\_byUsedHpEOA\_dict() |
| 参数 | 参数1: all\_mayUseHP\_EOA2times\_dict, 记录了所有调用蜜罐的账户地址，及它们调用hp的交易次数。  参数2: 已建立的session连接。 |
| 返回值 | all\_CreatedSC\_byUsedHpEOA\_dict。  key: 可疑EOA账户  value: 可疑EOA账户创建的智能合约（{地址: 字节码}） |

（3）关联函数（调用到的函数）：

getAll\_txs\_list(eoaAddr, session)：用于获取对应EOA账户的外部交易列表.

get\_contract\_creator\_withByteCode(all\_txs\_list, eoaAddr, session)：对应每个EOA账户创建过哪些智能合约, 获取它们的地址及字节码.

save\_json ('all\_CreatedSC\_byUsedHpEOA.json', all\_CreatedSC\_byUsedHpEOA\_dict)

：将返回值存储为all\_CreatedSC\_byUsedHpEOA.json文件.

（4）关联文件：

all\_CreatedSC\_byUsedHpEOA.json文件：记录从调用链上溯源的可疑EOA账户（发起了调用hp的tx）创建的智能合约（的地址+字节码）。

## 11. 二层向前溯源：hp调用链上SC的EOA创建的SCs

try\_get\_hpChain\_related\_scAddrs\_Bytes.py

主要由3个函数组成：

* get\_related\_scAddrs\_hpcalls(hp\_relatedTxCalls\_dict, hp\_addr)

（1）作用介绍：关注hp相关调用链上的SC.

收集hp所有相关交易的所有调用链中的所有智能合约(hp除外)的地址。

（2）函数说明：

|  |  |
| --- | --- |
| 函数名称 | get\_all\_CreatedSC\_byUsedHpEOA\_dict() |
| 参数 | 参数1: hp\_relatedTxCalls\_dict, 记录了hp所有相关交易(外部+内部)的调用链；  参数2: hp\_addr, 蜜罐合约的地址 |
| 返回值 | all\_related\_sc\_addrs\_set。  记录了hp相关调用链中的所有智能合约的地址（不包含hp的地址）。 |

注：hp\_relatedTxCalls\_dict字典中记录的调用链中，在他们的“tx\_origin”关联的“To”字段中的地址一定是智能合约的地址，因为在外部交易筛选的时候，确定了这种情况。

* get\_hpChain\_related\_createdEOA\_secondRelatedSCs(all\_related\_sc\_addrs\_set, session)

（1）作用介绍：

对hp所有相关交易的所有调用链中的所有智能合约(hp除外)，

它们的创建者都创建过哪些智能合约, 获取所有被这些创建者创建的智能合约(地址+字节码)。

（2）函数说明：

|  |  |
| --- | --- |
| 函数名称 | get\_hpChain\_related\_createdEOA\_secondRelatedSCs() |
| 参数 | 参数1: all\_related\_sc\_addrs\_set, 记录了hp所有相关交易的所有调用链中的所有智能合约的地址(hp除外)。  参数2: 已建立的session连接。 |
| 返回值 | firstAndSecondLayer\_createdSCs\_byRelatedEOA\_dict.  记录hp相关调用链上的SC的创建者还创建过的智能合约（地址+字节码）。 |

（3）关联函数（调用的函数）：

getAll\_txs\_list(related\_scAddr, session)：这里是获取SC相关的外部交易列表.d

get\_contract\_creator\_withByteCode (all\_txs\_list, related\_scAddr,session)：获取合约创建者EOA创建的所有合约（{地址：字节码}）

save\_json('firstAndSecondLayer\_createdSCs\_byRelatedEOA.json',

firstAndSecondLayer\_createdSCs\_byRelatedEOA\_dict)

（PS: 被关注SC的创建者也是需要被关注的对象。）

（4）关联文件：

firstAndSecondLayer\_createdSCs\_byRelatedEOA.json：记录hp相关调用链上的SC的创建者还创建过的智能合约（地址+字节码）。

* get\_hpChain\_otherSCs\_bytes(hpchain\_related\_sc\_addrs\_set, secondLayer\_createdSCs\_dict)

（1）作用介绍：

这个函数好像没什么用, 我就是从链上的相关合约往前溯源的, 现在通过这个函数又从往前溯源的结果中, **向后定位**了属于hp相关调用链上的合约, 并将它们记录、保存。

（2）函数说明：

|  |  |
| --- | --- |
| 函数名称 | get\_hpChain\_otherSCs\_bytes() |
| 参数 | 参数1: hpchain\_related\_sc\_addrs\_set.  记录了hp所有相关交易的所有调用链中的所有智能合约的地址。  参数2: secondLayer\_createdSCs\_dict.  记录hp相关调用链上的SC的创建者还创建过的智能合约（地址+字节码）。 |
| 返回值 | hpchain\_related\_sc\_addr2bytes\_dict。  记录了参数1和参数2的交集地址，即再次记录了链上的智能合约（{地址:字节码}）。 |

（3）关联函数：

save\_json('hpchain\_related\_sc\_addr2bytes.json', hpchain\_related\_sc\_addr2bytes\_dict)。

（4）关联文件：

hpchain\_related\_sc\_addr2bytes.json：从往前溯源的结果中, **向后定位**了属于hp相关调用链上的合约, 并将它们记录、保存。

## 12. 最长公共子序列（不需要在原序列中占用连续的位置）

try\_get\_Longest\_Common\_Subsequence.py

包含两种求解公共子序列的方法.

* 方法一：longestCommonSubsequence(text1, text2)
* 方法二：space\_efficient\_lcs(str\_a, str\_b)

函数说明：

|  |  |
| --- | --- |
| 函数名称 | longestCommonSubsequence() 或 space\_efficient\_lcs() |
| 输入 | 参数1：第一个字符串；  参数2：第二个字符串。 |
| 返回值 | 公共子序列（或其**长度**） |

## 13. 最长公共子串（连续）

try\_get\_Longest\_Common\_Substring.py

主要由1种方法构成。

* space\_efficient\_lcs(str\_a, str\_b)

（1）作用介绍：

求解两个字符串的公共子串。

（2）函数说明

|  |  |
| --- | --- |
| 函数名称 | space\_efficient\_lcs() |
| 输入 | 参数1：第一个字符串；  参数2：第二个字符串。 |
| 返回值 | 公共子串的长度。 |

（3）使用示例：

from try\_get\_Longest\_Common\_Substring import space\_efficient\_lcs

str1 = “xxx”

str2 = “xxxxx”

len\_lcstring = space\_efficient\_lcs(str1, str2)

print(len\_lcstring/min(len(str1),len(str2))) #求比例

## 14. 字节匹配/同源性分析

find\_sc\_similarWithHp.py

主要由1个函数组成。

* get\_suspicious\_scAddr\_list(all\_CreatedSC\_byUsedHpEOA\_dict,

thred\_line = 0.8)

（1）作用介绍：

对所有可疑账户创建的智能合约, 基于最长公共子串与hp合约字节码做同源性分析。将**相似性大于阈值的智能合约地址**记录在列表中。

（2）函数说明：

|  |  |
| --- | --- |
| 函数名称 | get\_suspicious\_scAddr\_list() |
| 输入 | all\_CreatedSC\_byUsedHpEOA\_dict.  记录所有可疑账户创建的所有智能合约（地址+字节码）的字典。 |
| 返回值 | suspicious\_scAddr\_list。  记录可疑的、可能是蜜罐的智能合约地址。 |

## 15. 获取蜜罐列表的创建时间：

get\_CreateTime\_of\_hp.py

获取到的文件是dict\_hp\_createTime.json

# 二、json文件

## 1. all\_CreatedSC\_byUsedHpEOA.json

{  
 EOA账户: {  
 创建的智能合约-1: 字节码,  
 创建的智能合约-2: 字节码,  
 创建的智能合约-x: 字节码  
 },  
 "0x6b35c9f9670c0641e3d5e80642992223d1ffec7b": {  
 "0x76ddf37e9418d5251e7c254a6dabb062027328ba": 字节码,  
 "0xabb4b9b48ceaefc290464025bdd4ef324f8b23c4": 字节码  
 },  
 "0x3476e2628006a7f74285c44b501437b415b08299": {  
 "0x65e5909d665cbda128de96aa9eb0160729eac1b0": 字节码,  
 "0x18a473e3a50f84b5d4f51af688670d23383993f5": 字节码,  
 "0x95d34980095380851902ccd9a1fb4c813c2cb639": 字节码  
 }  
}

## 2. contract\_creation\_eoa2sc.json

contract\_creation\_eoa2sc.json

{

EOA账户: {  
 创建的智能合约-1: 字节码,  
 创建的智能合约-2: 字节码,  
 创建的智能合约-x: 字节码  
 },  
 "0x3476e2628006a7f74285c44b501437b415b08299": {  
 "0x65e5909d665cbda128de96aa9eb0160729eac1b0": 字节码,  
 "0x18a473e3a50f84b5d4f51af688670d23383993f5": 字节码,  
 "0x95d34980095380851902ccd9a1fb4c813c2cb639": 字节码  
 }  
}

## 3. external\_txhash2calls.json

external\_txhash2calls.json

（1）介绍：

尝试记录由一个外部交易发起的调用链。

最简单的调用链是只有“tx\_origin”一个记录。

{

外部交易的hash地址：{

"tx\_origin": {

"From": 一定是一个EOA账户地址,

"To": 可以是EOA地址，也可以是SC地址,

"Transfer": （To为SC） & （Transfer为0）,是创建合约。

},  
 "0xcee0c2333438be10f1de753bb9222ebe3619fafbafa66fb011ee8277735fa1c2": {  
 "tx\_origin": {  
 "From": "0x3476e2628006a7f74285c44b501437b415b08299",  
 "To": "0x95d34980095380851902ccd9a1fb4c813c2cb639",  
 "Transfer": "1.0000003467625344 Ether"  
 }  
 }  
}

## 4. firstAndSecondLayer\_createdSCs\_byRelatedEOA.json

## 5. internal-txs\_from\_singleAddr.json

## 6. internal\_UniquePTH2calls.json

## 7. latest\_hrs\_sc\_urls.json

已转移到61.7服务器上了。

## 8. sc\_abiCode.json

## 9. sc\_byteCode.json

## 10. sc\_sourceCode.json

## 11. txs\_from\_singleAddr.json

## 12. unique\_contract.json

## 13. unique\_contract\_fromUsedHpEOAs.json