密码字典生成器编写

思路：

1. 收集相关信息
2. 汉字转拼音
3. 不同的信息抽取方式
4. 关键信息组合

```
#!/usr/bin/env python
# coding:utf-8

import time
from pypinyin import lazy_pinyin


class Person:
    NAME = "马永亮"
    PHONE = ["13512345678", ]
    CARD = "220281198309243953"
    BIRTHDAY = ("1983", "09", "24")
    HOMETOWN = ("河南", "郑州", "高新区")
    PLACE = [("北京", "昌平", "清河"), ]
    QQ = ["18746370", ]
    COMPANY = [("极客时间", "geektime"), ]
    SCHOOL = [("清华大学", "清华", "tsinghua")]
    ACCOUNT = ["mage", ]
    PASSWORD = ["old_password", ]


Delimiters = ["", "-", ".", "|", "_", "+", "#", "@"]
Prefix = ["", ]
Suffix = ["", "123", "@", "abc", ".", "123.", "!!!", ]


# 获取拼音
def get_pinyin(word):
    pinyin = ""
    for i in lazy_pinyin(word):
        pinyin = pinyin + ''.join(i)
    return pinyin


# 获取缩写
def get_abbreviation(word):
    result = ""
    for i in word:
        result += get_pinyin(i)[0]
```

```
    return result


# 获取全拼
def get_full_pinyin(word):
    return get_pinyin(word)


# 首字母大写
def get_title(word):
    return word.title()


def get_name_component(person):
    result = []
    # 获取姓名全拼
    result.append(get_pinyin(person.NAME))
    # 获取 姓 全拼
    result.append(get_pinyin(person.NAME[0]))
    # 获取 名 全拼
    result.append(get_pinyin(person.NAME[1:]))
    # 获取首字母大写姓名全拼
    result.append(get_title(get_pinyin(person.NAME)))
    # 获取首字母大写 姓 全拼
    result.append(get_title(get_pinyin(person.NAME[0])))
    # 获取首字母大写 名 全拼
    result.append(get_title(get_pinyin(person.NAME[1:])))
    # 获取缩写姓名拼音(只有首字母)
    result.append(get_abbreviation(person.NAME))
    # 获取缩写 姓 拼音
    result.append(get_abbreviation(person.NAME[0]))
    # 获取缩写 名 拼音
    result.append(get_abbreviation(person.NAME[1:]))
    return result


def get_phone_component(person):
    result = []
    for phone in person.PHONE:
        # 获取手机号
        result.append(phone)
        # 获取手机号后四位
        result.append(phone[-4:])
    return result


def get_card_component(person):
    result = []
    # 获取银行卡号
```

```
    result.append(person.CARD)
    # 获取银行卡号后六位
    result.append(person.CARD[-6:])
    # 获取银行卡号前六位
    result.append(person.CARD[0:6])
    return result


def get_birthday_component(person):
    result = []
    year = person.BIRTHDAY[0]
    month = person.BIRTHDAY[1]
    day = person.BIRTHDAY[2]
    # 获取年/月/日的各种组合
    result.append(year)
    result.append(year[2:])
    result.append(month + day)
    result.append(year + month + day)
    return result


def get_hometown_component(person):
    result = []
    # 获取地址全拼
    result.append(get_pinyin(person.HOMETOWN[0]))
    result.append(get_pinyin(person.HOMETOWN[1]))
    result.append(get_pinyin(person.HOMETOWN[2]))
    # 获取首字母大写的地址全拼
    result.append(get_title(get_pinyin(person.HOMETOWN[0])))
    result.append(get_title(get_pinyin(person.HOMETOWN[1])))
    result.append(get_title(get_pinyin(person.HOMETOWN[2])))
    # 获取缩写的地址拼音
    result.append(get_abbreviation(person.HOMETOWN[0]))
    result.append(get_abbreviation(person.HOMETOWN[1]))
    result.append(get_abbreviation(person.HOMETOWN[2]))
    return result


def get_place_component(person):
    result = []
    for place in person.PLACE:
        result.append(get_pinyin(place[0]))
        result.append(get_pinyin(place[1]))
        result.append(get_pinyin(place[2]))
        result.append(get_title(get_pinyin(place[0])))
        result.append(get_title(get_pinyin(place[1])))
        result.append(get_title(get_pinyin(place[2])))
        result.append(get_abbreviation(place[0]))
        result.append(get_abbreviation(place[1]))
```

```
            result.append(get_abbreviation(place[2]))
    return result


def get_qq_component(person):
    result = []
    for qq in person.QQ:
        result.append(qq)
    return result


# 获取公司信息
def get_company_component(person):
    result = []
    for company in person.COMPANY:
        for name in company:
            result.append(get_pinyin(name))
            result.append(get_title(get_pinyin(name)))
            result.append(get_abbreviation(name))
    return result


# 获取学校信息
def get_school_component(person):
    result = []
    for school in person.SCHOOL:
        for name in school:
            result.append(get_pinyin(name))
            result.append(get_title(get_pinyin(name)))
            result.append(get_abbreviation(name))
    return result


# 获取账号信息
def get_account_component(person):
    result = []
    for account in person.ACCOUNT:
        result.append(get_pinyin(account))
        result.append(get_title(get_pinyin(account)))
        result.append(get_abbreviation(account))
    return result


# 通过不同方式获取各组件信息
def get_all_component(person):
    result = []
    result.append(get_name_component(person))
    result.append(get_phone_component(person))
    result.append(get_card_component(person))
```

```
    result.append(get_birthday_component(person))
    result.append(get_hometown_component(person))
    result.append(get_place_component(person))
    result.append(get_qq_component(person))
    result.append(get_company_component(person))
    result.append(get_school_component(person))
    result.append(get_account_component(person))
    return result


def write_password(password, filename):
    print("[+] %s" % password)
    with open(filename, "a+") as f:
        f.write("%s\n" % password)


def combined_character(compents, Delimiter, prefix, suffix, filename):
    for compent in compents:
        for i in compent:
            if Delimiter == "":
                password = prefix + i + Delimiter + suffix
                write_password(password, filename)
                continue
            password = prefix + i + Delimiter + suffix
            write_password(password, filename)
            password = prefix + Delimiter + i + suffix
            write_password(password, filename)


def combined_character_tow(compents, Delimiter, prefix, suffix, filename):
    for compent_a in compents:
        for compent_b in compents:
            for i in compent_a:
                for j in compent_b:
                    password = prefix + i + Delimiter + j + suffix
                    write_password(password, filename)


def gen_pass():
    compents = get_all_component(Person)  # 获取成分信息
    filename = "password.list"
    # 单组件密码
    for Delimiter in Delimiters:
        for prefix in Prefix:
            for suffix in Suffix:
                combined_character(compents, Delimiter, prefix, suffix, filename)
    # 两组件密码
    for Delimiter in Delimiters:
        for prefix in Prefix:
```

```
            for suffix in Suffix:
                combined_character_tow(compents, Delimiter, prefix, suffix, filename)


if __name__ == "__main__":
    start_time = time.time()
    gen_pass()
    end_time = time.time()
    print("[*] 扫描结束，共计扫描时间：%.2f s" % (end_time - start_time))
```

思考：

如何提高生成效率?

WEB目录扫描程序

使用 logging 模块记录日志

```
# util.py

import logging
from logging import handlers


class Logger(object):
    level_relations = {
        'debug': logging.DEBUG,
        'info': logging.INFO,
        'warning': logging.WARNING,
        'error': logging.ERROR,
        'crit': logging.CRITICAL
    }  # 日志级别关系映射

    def __init__(self, filename, level='info', when='D', backcount=3,
                 fmt='%(asctime)s - %(filename)s[line:%(lineno)d] - %(levelname)s: %
(message)s'):
        self.logger = logging.getLogger(filename)
        format_str = logging.Formatter(fmt)  # 设置日志格式
        self.logger.setLevel(self.level_relations.get(level))  # 设置日志级别
        sh = logging.StreamHandler()  # 往屏幕上输出
        sh.setFormatter(format_str)  # 设置屏幕上显示的格式
        th = handlers.TimedRotatingFileHandler(filename=filename, when=when,
backupCount=backcount,
                                               encoding='utf-8')  # 往文件里写入 指定间隔时
间自动生成文件的处理器
```

```
        # 实例化TimedRotatingFileHandler
        # interval是时间间隔，backupCount是备份文件的个数，如果超过这个个数，就会自动删除，when是间隔的时间单位，单位有以下几种：
        # S 秒
        # M 分
        # H 小时、
        # D 天、
        # W 每星期（interval==0时代表星期一）
        # midnight 每天凌晨
        th.setFormatter(format_str)  # 设置文件里写入的格式
        self.logger.addHandler(sh)  # 把对象加到logger里
        self.logger.addHandler(th)
```

通过配置文件来配置扫描内容

```
{
    // web域名配置文件
    "websites" : "websites.txt",
    // web目录文件
    "dic_folder" : "dir_folder",
    "request_method" : "GET",
    // 指定扫描线程数
    "thread_num" : 16
}
```

管理模块用来读取配置文件，启动扫描任务

```
# mian.py

import json
from concurrent.futures import ThreadPoolExecutor
from webDirScan import *


class Manager(object):
    cfg = None

    def __init__(self):
        try:
            conf = json.load(open('config.json'))
            self.cfg = {'websites': self.read_websites(conf['websites']),
                        'dics': self.read_web_path(conf['dic_folder']),
                        'thread_num': conf['thread_num']}
            self.pool = ThreadPoolExecutor(max_workers=self.cfg['thread_num'])

        except Exception as e:
            logging.error('Manager __init__ error: %s' % e)
```

```
    def start(self):
        try:
            for site in self.cfg['websites']:
                # 每读取一个域名就启动一个线程进行扫描
                w = Worker(site, self.cfg, self.pool)
                w.start()

        except Exception as e:
            logging.error('Manager Start error: %s' % e)

    def read_websites(self, webcfg):
        try:
            websites = []
            with open(webcfg, 'r') as webs:
                for web in webs.readlines():
                    websites.append(web.strip())
            return websites
        except Exception as e:
            logging.error('ReadWebsites error : %s' % e)

    def read_web_path(self, diccfg):
        try:
            dics = []
            doclist = os.listdir(diccfg)
            doclist.sort()
            for filename in doclist:
                filename = diccfg+'/'+filename
                with open(filename, 'r') as f:
                    for i in f.readlines():
                        dics.append(i.strip())
            return dics
        except Exception as e:
            logging.error('dic error : %s , dicname:%s' % (e, filename))


def main():
    w = Manager()
    w.start()


if __name__ == '__main__':
    main()
```

扫描目录负责目录扫描和信息记录

```python
# webDirScan.py

import requests
import threading
import os
import time
from util import Logger
from concurrent.futures import wait
from requests.packages import urllib3

# 忽略HTTPS告警
urllib3.disable_warnings()
g_mutex = threading.Lock()
# 日志文件目录为当前目录所在文件下的log目录
log_dir = os.path.dirname(os.path.abspath(__file__))+'/log/'
# 目录不存在则创建
if not os.path.exists(log_dir):
    os.mkdir(log_dir)
cur_time = time.strftime('%Y-%m-%d', time.localtime(time.time())).replace(':',
'').replace(' ', '_')
LogFile = log_dir + cur_time + '.log'
logging = Logger(LogFile, level='debug').logger


class Worker(object):
    site = None
    cfg = None

    def __init__(self, site, cfg, pool):
        self.site = site
        self.cfg = cfg
        self.pool = pool

    def start(self):
        try:
            scan_list = []
            for web_dir in self.cfg['dics']:
                t = Scanner(self.site, web_dir)
                future = self.pool.submit(t.run())
                scan_list.append(future)
            wait(scan_list)
        except Exception as e:
            logging.error('Start error: %s' % e)


class Scanner():
    site = None
    dics = None
    request_method = None
```

```
    headers = {
        'user-agent': 'Mozilla/5.0 (Windows NT 6.1; WOW64) '
                      'AppleWebKit/537.36 (KHTML, like Gecko) Chrome/43.0.2357.132
Safari/537.36'}

    def __init__(self, site, dics):
        try:
            self.site = site
            self.dics = dics

            if self.site.endswith('/'):
                self.site = self.site[:-1]
            if '://' not in self.site:
                self.site = 'http://' + self.site

        except Exception as e:
            logging.error('Scanner_init error: %s' % e)

    def run(self):
        self.ScanOne(self.site, self.dics)

    def ScanOne(self, site, dic):
        try:
            if not dic.startswith('/'):
                dic = '/' + dic
            url = site + dic

            # 不允许重定向，防止重定向误识别为存在的页面
            res = requests.get(url, verify=False, allow_redirects=False,
headers=self.headers, timeout=8)
            if 200 == res.status_code:
                print('\033[93m[+]\033[96m%d\033[0m    %s' % (res.status_code, url))
                self.WriteFile('./out.txt', url)
            else:
                print('\033[95m[-]\033[31m%d\033[0m    %s' % (res.status_code, url))
        except Exception as e:
            if 'Max retries exceeded with url' in str(e):
                pass
            else:
                logging.error('Start error: %s  ,  url:%s' % (e, url))

    def WriteFile(self, file, msg):
        try:
            # 写文件时加锁
            g_mutex.acquire()
            with open(file, 'a+') as f:
                f.write('%s\r\n' % (str(msg)))
        except Exception as e:
            logging.error('WriteFile error : %s' % e)
```

```
        finally:
            g_mutex.release()
```

指纹识别工具开发

使用python-nmap 模块进行指纹识别并输出

`pip install python-nmap`

```
import optparse  # Import the module

import nmap  # Import the module

pr_blue = '\033[94m'
pr_red = '\033[31m'
pr_yellow = '\033[93m'
pr_green = '\033[96m'


def nmapScan(tgtHost, tgtPort):  # Create the function, this fucntion does the scanning
    nmScan = nmap.PortScanner()
    nmScan.scan(tgtHost, tgtPort)
    state = nmScan[tgtHost]["tcp"][int(tgtPort)]["state"]
    protocol = nmScan[tgtHost]["tcp"][int(tgtPort)]["name"]
    product = nmScan[tgtHost]["tcp"][int(tgtPort)]["product"]
    version = nmScan[tgtHost]["tcp"][int(tgtPort)]["version"]
    extrainfo = nmScan[tgtHost]["tcp"][int(tgtPort)]["extrainfo"]
    result = "%s[*] %s%s tcp/%s %s[%s]\n%s%s %s" % (
        pr_red, pr_blue, tgtHost, tgtPort, pr_green, state, pr_yellow, product,
version)
    if extrainfo:
        result += "os: %s" % extrainfo
    print(result)


def main():  # Main Program
    parser = optparse.OptionParser(
        "usage%prog " + "-H <host> -p <port>"
    )  # Display options/help if required
    parser.add_option("-H", dest="tgtHost", type="string", help="specify host")
    parser.add_option("-p", dest="tgtPort", type="string", help="port")
    (options, args) = parser.parse_args()
    tgtHost = options.tgtHost
    tgtPorts = str(options.tgtPort).split(",")

    if (tgtHost == None) | (tgtPorts[0] == None):
```

```
        print(parser.usage)
        exit(0)

    for tgtPort in tgtPorts:  # Scan the hosts with the ports etc
        nmapScan(tgtHost, tgtPort)


if __name__ == "__main__":
    main()
```

C段WEB服务扫描

Argparse

`argparse` 模块可以让人轻松编写用户友好的命令行接口。程序定义它需要的参数，然后 `argparse` 将弄清如何从 `sys.argv` 解析出那些参数。 `argparse` 模块还会自动生成帮助和使用手册，并在用户给程序传入无效参数时报出错误信息。

```
# 载入模块
import argparse
# 初始化
parser = argparse.ArgumentParser()
# 设置命令行参数
parser.add_argument("echo", action="store_true", help="echo the string you use here")
# 参数解析
args = parser.parse_args()
# 获取参数
print(args.echo)
```

执行结果

```
> python argparse_test.py -h
usage: test.py [-h]

positional arguments:
  echo        echo the string you use here

optional arguments:
  -h, --help  show this help message and exit
```

可以看到自带`-h`参数和 useage 信息

BeautifulSoup

Beautiful Soup提供一些简单的、python式的函数用来处理导航、搜索、修改分析树等功能。它是一个工具箱，通过解析文档为用户提供需要抓取的数据，因为简单，所以不需要多少代码就可以写出一个完整的应用程序。

Beautiful Soup自动将输入文档转换为Unicode编码，输出文档转换为utf-8编码。你不需要考虑编码方式，除非文档没有指定一个编码方式，这时，Beautiful Soup就不能自动识别编码方式了。然后，你仅仅需要说明一下原始编码方式就可以了。

Beautiful Soup将复杂HTML文档转换成一个复杂的树形结构,每个节点都是Python对象,所有对象可以归纳为4种:

- Tag - HTML 中的一个个标签
- NavigableString - 字符串类型
- BeautifulSoup - 表示的是一个文档的全部内容
- Comment - 注释的类型

```
import lxml
from bs4 import BeautifulSoup

html = """
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">

  <head>

  </head>

<div class="body_padded">
  <h1>Vulnerability: Brute Force</h1>

  <div class="vulnerable_code_area">
    <h2>Login</h2>
    <form action="#" method="GET">
      Username:<br />
      <input type="text" name="username"><br />
      Password:<br />
      <input type="password" AUTOCOMPLETE="off" name="password"><br />
      <br />
      <input type="submit" value="Login" name="Login">

    </form>
    <p>Welcome to the password protected area gordonb</p><img
src="http://127.0.0.1/hackable/users/gordonb.jpg" />
  </div>
  <h2>More Information</h2>
  <ul>
```

```
    <li><a href="https://www.owasp.org/index.php/Testing_for_Brute_Force_(OWASP-AT-004)" target="_blank">https://www.owasp.org/index.php/Testing_for_Brute_Force_(OWASP-AT-004)</a></li>
    <li><a href="http://www.symantec.com/connect/articles/password-crackers-ensuring-security-your-password" target="_blank">http://www.symantec.com/connect/articles/password-crackers-ensuring-security-your-password</a></li>
    <li><a href="http://www.sillychicken.co.nz/Security/how-to-brute-force-http-forms-in-windows.html" target="_blank">http://www.sillychicken.co.nz/Security/how-to-brute-force-http-forms-in-windows.html</a></li>
  </ul>
  </body>

</html>
"""
soup = BeautifulSoup(html, 'lxml')  # 文档对象

# 查找a标签,只会查找出一个a标签
# print(soup.a)#<a class="sister" href="http://example.com/elsie" id="xiaodeng"><!--Elsie --></a>

for k in soup.find_all('a'):
    print(k['href'])  # 查a标签的href值
    print(k.string)  # 查a标签的string
```

- 获取tag

```
print(soup.title)
print(soup.head)
print(soup.a)
print(soup.p)
print(type(soup.a))
```

asyncio --- 异步 I/O

asyncio 是用来编写 **并发** 代码的库，使用 **async/await** 语法。

asyncio 被用作多个提供高性能 Python 异步框架的基础，包括网络和网站服务，数据库连接库，分布式任务队列等等。

示例一

```
import asyncio
import time
```

```
async def say_after(delay, what):
    await asyncio.sleep(delay)
    print(what)


async def main():
    print(f"started at {time.strftime('%X')}")

    await say_after(1, 'hello')
    await say_after(2, 'world')

    print(f"finished at {time.strftime('%X')}")

asyncio.run(main())
```

示例二

```
import asyncio
import time


async def say_after(delay, what):
    await asyncio.sleep(delay)
    print(what)


async def main():
    task1 = asyncio.create_task(
        say_after(1, 'hello'))

    task2 = asyncio.create_task(
        say_after(2, 'world'))

    print(f"started at {time.strftime('%X')}")

    # Wait until both tasks are completed (should take
    # around 2 seconds.)
    await task1
    await task2

    print(f"finished at {time.strftime('%X')}")

asyncio.run(main())
```

什么是协程?

协程在等待IO的过程中重复利用线程，也就是说协程本质是通过多路复用来完成的。

协程能保留上一次调用时的状态（即所有局部状态的一个特定组合），每次过程重入时，就相当于进入上一次调用的状态，换种说法：进入上一次离开时所处逻辑流的位置。

C段扫描程序

```
import asyncio
import sys
import requests
from netaddr import IPNetwork
from bs4 import BeautifulSoup


Ports_web = [80, 88, 443, 7001, 8000, 8008, 8888, 8080, 8088, 8089, 8161, 9090]
Ports_other = [21, 22, 445, 1100, 1433, 1434, 1521, 3306, 3389, 6379, 8009, 9200,
11211, 27017, 50070]

COUNT = 0
TIMEOUT_HTTP = 5
TIMEOUT_SOCK = 0.9
PATH = ''


user_agent = "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36
(KHTML, like Gecko)" \
             " Chrome/101.0.4951.41 Safari/537.36"
# 输出字符时的各种颜色前缀
pr_purp = '\033[95m'
pr_blue = '\033[94m'
pr_red = '\033[31m'
pr_yellow = '\033[93m'
pr_green = '\033[96m'
# 结束颜色
pr_end = '\033[0m'


def tag(info):
    return "[" + info + "]"


def get_info(url, keyword):
    try:
        r = requests.get(url, headers={'UserAgent': user_agent}, timeout=TIMEOUT_HTTP,
verify=False,
```

```
                            allow_redirects=True)
            # 解析html
            soup = BeautifulSoup(r.content, "lxml")

            # HTTP头信息分析
            info_code = tag(pr_red + str(r.status_code) + pr_end)
            info_title = tag(pr_blue + soup.title.string.replace('\n', '').replace('\r',
'').replace(
                '\t', '') + pr_end) if soup.title else tag("")

            info_len = tag(pr_purp + str(len(r.content)) + pr_end)
            # 获取请求头中的服务器信息
            if 'Server' in r.headers:
                info_server = " [" + pr_yellow + r.headers['Server']
                info_server += " " + r.headers['X-Powered-By'] + pr_end + "]" if 'X-
Powered-By' in r.headers else "]"
            else:
                info_server = tag("")

            result = info_code + info_title + info_server + info_len

            # HTTP内容，关键字匹配
            key = tag(pr_red + "Keyword!!!" + pr_end) if keyword and keyword in r.text else
""

            return result + key

        except Exception as e:
            # print(e)
            return tag(pr_green + "open" + pr_end)


async def connet(host, sem, keyword):
    """
    先用异步网络请求判断端口是否存在，如果存在，再对web端口进行信息输出
    :param host thread keyword ip
    :param sem: 设置进程
    :param keyword: 对web内容关键字匹配
    :param ip: 获取目标host的IP
    :return info
    """
    global COUNT
    async with sem:
        for port in Ports_web:
            fut = asyncio.open_connection(host=host, port=port)
            try:
                reader, writer = await asyncio.wait_for(fut, timeout=TIMEOUT_SOCK)
                if writer:
                    # 如果没有 443/8443 端口，则使用http协议，否则使用https协议
```

```
                    protocol = "http" if int(port) not in [443, 8443] else "https"
                    url = "{0}://{1}:{2}{3}".format(protocol, host, port, PATH)
                    info = get_info(url, keyword)

                    sys.stdout.write("%s %s\n" % (url, info))
                    COUNT += 1

            except Exception as e:
                # print(e)
                pass


async def scan():
    # 用于并发控制的信号量
    sem = asyncio.Semaphore(60)
    keyword = ""
    c_ip = "192.168.0.1/24"
    # C段IP转换为单个IP的list
    ips = [str(ip) for ip in IPNetwork(c_ip)]
    tasks = []
    for host in ips:
        tasks.append(asyncio.create_task(connet(host, sem, keyword)))
    await asyncio.wait(tasks)


if __name__ == '__main__':
    asyncio.run(scan())
```