1. 聚焦爬虫的一般步骤?

答: 1> 确定目标： 确定目标数据所在的URL，确定目标数据字段。

2> 爬: 模拟浏览器发起请求，获取服务器响应数据。

3> 取: 从响应数据中解析目标数据。

4> 存：数据持久化，将解析到的结构化数据，保存到数据库。

2. 爬取下来的数据如何去重，说一下具体的算法依据?

答: 1> 通过MD5生成电子指纹来判断页面是否改变。

2> nutch去重。nutch中digest是对采集的每一个网页内容的32位哈希值，如果两个网页内容完全一样，他们的digest值肯定会一样。

3. 写爬虫是用多进程好? 还是多线程好? 为什么?

答: IO密集型代码(文件处理、网络爬虫等)，多线程能够有效提升效率(单线程下有IO操作会进行IO等待，造成不必要的时间浪费，而开启多线程能在线程A等待时，自动切换到线程B，可以不浪费CPU资源，从而能提升程序执行效率)。在实际的数据采集过程中，既考虑网速和响应的问题，也需要考虑自身机器的邮件情况，来设置多进程或多线程。

4. Post和Get的区别?

答: 1> GET请求：请求的数据会附在URL之后，以`?`分割URL和传输数据，多个参数用`&`连接。URL的编码格式采用的是ASCII编码，而不是unicode，即是说所有的非ASCII字符都要编码之后再传输。

POST请求: POST请求会把请求的数据放置在HTTP请求的请求体中。

2> 传输数据的大小

在HTTP规范中，没有对URL的长度和传输的数据大小进行限制。但是在实际开发过程中，对于GET，特定的浏览器和服务器对URL的长度有限制。因此，在使用GET请求时，传输数据会受到URL长度的限制。

对于POST，由于不是URL传值，理论上是不会受限制的，但是实际上各个服务器会规定对POST提交数据大小进行限制。

3> 安全性

POST的安全性比GET的安全性高。比如：通过GET提交数据，用户名和密码将明文出现在URL上，因为(1)登录页面有可能被浏览器缓存，(2)其他人查看浏览器的历史纪录，那么别人就可以拿到你的账号和密码了，除此之外，使用GET提交数据还可能会造成Cross-site request forgery攻击。

5. 常用的应对反爬虫方法?

答: 1> 添加代理；2> 减低访问频率；3> 添加User-Agent；

4> 动态HTML数据加载；5> 验证码处理；6> Cookie。

6. 什么是URL?

答: Uniform Resource Locator，缩写：URL，即统一资源定位符，也就是我们说的网址，统一资源定位符是对可以从互联网上得到的资源的位置和访问方法的一种简洁的表示，是互联网上标准资源的地址。互联网上的每个文件都有一个唯一的URL，它包含的信息指出文件的位置以及浏览器应该怎么处理它。

7. 简单说一下你对Scrapy的了解?

答: scrapy是一套用Python编写的异步爬虫框架，基于Twisted实现，运行于Linux/Windows/MacOS等多种环境，具有速度快、扩展性强、使用简便等特点。即便是新手也能迅速掌握并编写出所需要的爬虫程序。scrapy可以在本地运行，也能部署到云端（scrapyd）实现真正的生产级数据采集系统。

8. Scrapy的优缺点?

答: 优点: scrapy是异步的；采取可读性更强的xpath代替正则；强大的统计和log 系统；同时在不同的url上爬行；支持shell方式，方便独立调试；写middleware，方便写一些统一的过滤器；通过管道的方式存入数据库。

缺点：基于Python的爬虫框架，扩展性比较差；基于twisted框架，运行中的exception是不会干掉reactor，并且异步框架出错后不会停掉其他任务，数据出错后难以察觉。

9. Scrapy框架和requests模块对比?

答: Scrapy是封装起来的框架，包含了下载器、解析器，日志及异常处理，基于多线程，twisted的方式处理，对于固定单个网站的爬取开发，有优势，但是对于多网站爬取，并发及分布式处理方面，不够灵活，不便调整与拓展。

requests是一个HTTP库，只是用来进行请求，对于HTTP请求，它是一个强大的库，下载、解析全部自己处理，灵活性更高，高并发与分布式部署也非常灵活，对于功能可以更好实现。

10. 描述下Scrapy框架运行的机制?

答: 从stats_url里获取第一批url并发送请求，请求由引擎交给调度器入请求队列，获取完毕后，调度器将请求队列里的请求交给下载器去获取请求对应的响应资源，并将响应交给自己编写的解析方法做提取处理：a) 若提取出需要的数据，则交给管道文件处理；b) 若提取出url，则继续执行之前的步骤(发送url请求，并由引擎将请求交给调度器入队列)，直到请求队列里

没有请求，程序结束。

11. 写一个Python中的单例模式

```python
class Singleton(object):
  __instance = None

  def __new__(cls, *args, **kwargs):
    if not cls._instance:
      cls.__instance = super().__new__(cls, *args, **kwargs)

    return cls.__instance
```

12. 对`if __name__ == '__main__':`的理解

`__name__`是当前模块名，当模块被直接运行时模块名为`__main__`，也就是当前的模块，当模块被导入时，模块名就不是`__main__`，即代码将不会执行。

13. 实现模拟登录的方式有哪些?

1> 使用一个具有登录状态的cookie，结合请求报头一起发送，可以直接发送get请求，访问登录后才能访问的页面。

2> 先发送登录界面的get请求，在登录页面的HTML里获取登录需要的数据(如果需要的话)，然后结合账户密码，再发送post请求，即可登录成功。然后根据获取的cookie信息，继续访问之后的页面。

14. 简单介绍下Scrapy的异步处理

scrapy框架的异步机制是基于twisted异步网络框架处理的，在settings.py文件里可以设置具体的并发量数值(默认是并发量16)。

15. 描述用浏览器访问www.baidu.com的过程。

1、 浏览器获取输入的域名www.baidu.com

2、 浏览器向DNS请求解析www.baidu.com的IP地址

3、域名系统DNS解析出百度服务器的IP地址

4、 浏览器发出HTTP请求，请求百度首页

5、浏览器与该服务器建立TCP连接(默认端口号80)

6、 IP的重要功能是寻址和路由

7、服务器通过HTTP响应把首页文件发送给浏览器

8、TCP连接释放

9、 浏览器将首页文件进行解析，并将Web页显示给用户。

16. cookie和session的区别?

1、cookie数据存放在浏览器端，session数据存放在服务器上。

2、cookie不是很安全，别人可以分析存在本地的cookie并进行cookie欺骗，考虑到安全应当使用session。

3、session会在一定时间内保存在服务器上。当访问增多，会比较占用服务器的性能，考虑到减轻服务器性能方面，应当使用cookie。

4、单个cookie保存的数据不能超过4k，很多浏览器都限制一个站点最多保存20个cookie。

5、建议：将登录信息等重要信息存放为session，其他信息如果需要保留，可以放在cookie中。

17. 列出HTTP协议的常用状态码。

- 100~199：表示服务器成功接收部分请求，要求客户端继续提交其余请求才能完成整个处理过程。
- 200~299：表示服务器成功接收请求并已完成整个处理过程。常用200（OK 请求成功）。
- 300~399：为完成请求，客户需进一步细化请求。例如：请求的资源已经移动一个新地址、常用302（所请求的页面已经临时转移至新的url）、307和304（使用缓存资源）。
- 400~499：客户端的请求有错误，常用404（服务器无法找到被请求的页面）、403（服务器拒绝访问，权限不够）。
- 500~599：服务器端出现错误，常用500（请求未完成。服务器遇到不可预知的情况）。

18. 简述python标准库json的使用。

1、json.loads： 把Json格式字符串解码转换成Python对象

2、json.dumps: 把一个Python对象编码转换成Json字符串

3、json.load: 读取文件中json形式的字符串元素 转化成python类型

4、json.dump: 将Python内置类型序列化为json对象后写入文件

19.怎样让scrapy框架发送一个post请求?

使用FormRequest:

```
class MySpider(scrapy.Spider):

  def start_request(self):
    url = "https://www.taobao.com/login"

    yield scrapy.FormRequest(
      url=url,
      formdata={'email': 'xxxx', 'password': 'yyyy'}
      callback=self.parse_page
    )

  def parse_page(self, response):
    # do something
```

20. re模块中match和search的区别

match: 只从字符串的开始与正则表达式匹配，匹配成功返回match object，否则返回none；

search: 将字符串的所有字串尝试与正则表达式匹配，如果所有的字串都没有匹配成功，返回none，否则返回match object；

21. Python如何给程序传参?

sys模块中的argv是一个列表型变量，存放着启动python程序的命令行参数。

```
$ cat test.py
import sys

print(sys.argv)
                                    $ python test.py 1 2 3 4 5
['test.py', '1', '2', '3', '4', '5']
```

22. Python中如何实现类似三元表达式功能?

在python中的格式为

```
为真时的结果 if 判定条件 else 为假时的结果
```

[例] 如果5大于3， 输出1，否则输出0

```
1 if 5>3 else 0
```

23. 什么是迭代器，实现偶数迭代器?

Python 支持在容器中进行迭代的概念。迭代器对象自身需要支持以下两个方法，它们共同组成了 迭代器协议:

`iterator.__iter__()` 返回迭代器对象本身。

`iterator.__next__()` 从容器中返回下一项。 如果已经没有项可返回，则会引发 StopIteration 异常。

[例] 偶数迭代器

```
class EvenIterator(object):
    """偶数迭代器"""

    def __init__(self, n):
        """
        :param n: int, 指明生成数列的前n个数
        """
        # cur为计数器，表示计算第cur个偶数
        self.cur = 0
        self.n = n
        # 保存每次生成的偶数
        self.res = 0

```

```
    def __next__(self):
        """计算下一个数"""
        if self.cur < self.n:
            self.res += 2
            self.cur += 1
            return self.res
        else:
            raise StopIteration

    def __iter__(self):
        return self


if __name__ == '__main__':
    ei = EvenIterator(5)
    for num in ei:
        print(num, end=" ")
```

24. 什么是生成器，实现斐波拉契数列生成器?

在定义函数时使用yield关键字，即为生成器

[例] 斐波拉契数列(0, 1, 1, 2, 3, 5, 8, 13, 21, 34.....)生成器

```python
def fib(n):
  cur = 0
  a, b = 0, 1
  res = 0
  while cur < n:
    yield a
    a, b = b, a + b
    cur += 1

if __name__ == '__main__':
  for x in fib(10):
    print(x, end='\t')
```

25. 使用列表推导式取出列表中值为偶数，且下标为偶数的元素。

```
In [3]: nums = [1, 3, 4, 5, 6]

In [4]: [nums[i] for i in range(0, len(nums), 2) if nums[i] % 2 == 0]
Out[4]: [4, 6]
```

26. 对深拷贝、浅拷贝的理解

如果希望赋值时可变对象不进行引用，而是重新分配地址空间并将数据复制，可以利用标准库copy模块。其中主要的函数有copy.copy和copy.deepcopy。

1> copy.copy仅仅复制父对象，不会复制父对象内部的子对象。

2> copy.deepcopy复制父对象和子对象

```python
import copy

list1 = [1, 2, ['a', 'b']]
```

```
list2 = list1
list3 = copy.copy(list1)
list4 = copy.deepcopy(list1)
list1.append(3)
list1[2].append('c')

print('list1 = ', list1)
print('list2 = ', list2)
print('list3 = ', list3)
print('list4 = ', list4)

# 结果:
# list1 =  [1, 2, ['a', 'b', 'c'], 3]
# list2 =  [1, 2, ['a', 'b', 'c'], 3]
# list3 =  [1, 2, ['a', 'b', 'c']]
# list4 =  [1, 2, ['a', 'b']]
```

27. python标准库os常用方法。

os.path.abspath(path): 获取给定path的绝对路径。

os.path.exists(path): 判断给定路径(文件或目录)是否存在。

os.rename(src, dst): 将文件scr的文件名修改为dst。

os.remove(path): 删除一个文件。

os.mkdir(path): 创建一个文件夹。

os.listdir(path): 以列表形式给出指定目录下的所有文件和文件夹。

28. Python中函数的5种参数类型

位置参数、默认参数、不定长参数、关键字参数、命名关键字参数。

29. 给定两个list A, B，请找出A，B中相同元素和不同元素

使用集合

```
In [1]: list_a = [1, 3, 4, 5, 6]

In [2]: list_b = [2, 4, 6, 7, 8]

In [3]: same = list(set(list_a) & set(list_b))

In [4]: same
Out[4]: [4, 6]

In [5]: diff = list(set(list_a) ^ set(list_b))

In [6]: diff
Out[6]: [1, 2, 3, 5, 7, 8]
```

30. 上下文管理器

Python 的 with 语句支持通过上下文管理器所定义的运行时上下文这一概念。 此对象的实现使用了一对专门方法，允许用户自定义类来定义运行时上下文，在语句体被执行前进入该上下文，并在语句执行完毕时退出该上下文：

文件备份，with版:

```
with open("test.txt") as f_read, open("test[copy].txt", "w")
as f_write:
    f_write.write(f_read.read())
```

31. 怎么判断网站是否更新?

使用MD5数字签名:

每次下载网页是，把无服务器返回的数据流ResponseStream先放在内存缓冲区，然后对ResponseStream生成MD5数字签名S1，下次下载同样生成签名S2，比较S2和S1，如果相同，则页面没有更新，否则页面就已经更新了。

32. 内置函数map的语法及示例

1> 语法

```
map(function, iterable, ...)
```

2> 功能

map()会根据提供的函数对指定序列做映射。

第一个参数function以参数序列中的每一个元素调用function函数，返回包括每次function函数返回值的新列表。

3> 示例

```
# 计算列表各元素的平方
In [1]: map(lambda x:x**2,[1,2,3,4])
Out[1]: [1, 4, 9, 16]

# 传入两个列表，对应元素求和
In [2]: map(lambda x,y:x+y,[1,3,5,7,9],[2,4,6,8,10])
Out[2]: [3, 7, 11, 15, 19]
```

33. 内置函数zip的语法及示例

1> 语法

```
zip([iterable, ...])
```

2> 描述

zip()函数用于将可迭代对象作为参数，将对象中对应的元素打包成一个元组，然后返回由这些元组组成的列表。如果各个迭代器的元素个数不一致，则返回列表长度与最短的对象相同，利用*号操作，可以将元组解压为列表。

3> 示例:

```
In [4]: a = [1,2,3]

In [5]: b = [4,5,6]

In [6]: c = [4,5,6,7,8]

In [7]: zipped = zip(a,b);zipped
Out[7]: [(1, 4), (2, 5), (3, 6)]

In [8]: zip(a,c)
Out[8]: [(1, 4), (2, 5), (3, 6)]

In [10]: zip(*zipped)  # zip逆操作
Out[10]: [(1, 2, 3), (4, 5, 6)]
```

34. 内置函数filter的语法及示例

1> 语法

```
filter(function, iterable)
```

2> 描述

filter()函数用于过滤序列。

3> 示例

```
# 过滤出列表中所有奇数
In [14]: def is_odd(n):
    ...:     return n % 2 == 1
    ...:

In [15]: filter(is_odd, [1,2,3,4,5,6,7,8,9,10])
Out[15]: [1, 3, 5, 7, 9]
```

35. 列表嵌套字典结构按照字典的键或值排序

给定学生信息如下，请按照age进行升序排列

```
In [7]: students = [
    ...:     {"name": "Bob", "age":19},
    ...:     {"name": "Mike", "age":18},
    ...:     {"name": "Andy", "age":20}
    ...: ]

In [9]: students.sort(key=lambda x: x["age"])

In [10]: students
Out[10]:
[{'name': 'Mike', 'age': 18},
 {'name': 'Bob', 'age': 19},
 {'name': 'Andy', 'age': 20}]
```

36. 函数reduce的语法及示例

1> 语法

```
reduce(function, iterable[, initializer])
```

2> 描述

reduce()函数会对参数序列中元素进行累积。

3> 示例

```
In [12]: reduce(lambda x,y:x+y,range(10))
Out[12]: 45
```

4> 注意

python2中为内置函数，python3中需`from functools import reduce`

37. 字符串反转

方法一: 使用切片

```
In [20]: s = ['h', 'e', 'l', 'l', 'o']

In [21]: s[::-1]
Out[21]: ['o', 'l', 'l', 'e', 'h']
```

方法二: 递归

```
In [22]: def reverse_str(s: list, lo: int, hi: int):
    ...:     if lo < hi:
    ...:         s[lo], s[hi] = s[hi], s[lo]
    ...:         reverse_str(s, lo + 1, hi - 1)

In [25]: reverse_str(s, 0, len(s) - 1)

In [26]: s
Out[26]: ['o', 'l', 'l', 'e', 'h']
```

38. is和==的区别

is比较的是两个对象的id值是否相等，也就是比较两个对象是否为同一个实例对象，是否指向同一个内存地址。

==比较的是两个对象的内容是否相等，默认会调用对象的`__eq__()`方法。

39. json格式文件转换成csv文件

json格式文件，一个字典一行，每行最后一个逗号，之后另起一行写下一个数据项

格式如下:

```
{'name': 'Mike', 'age': 18},
{'name': 'Bob', 'age': 19},
{'name': 'Andy', 'age': 20}
```

参考代码

```
import pandas as pd

with open("test.json") as f:
  datas = [line.replace(",\n", "") for line in f]

pd.DataFrame(datas).to_csv("test.csv")
```

40. 对于for-else结构的理解

Python提供了一种很多编程语言都不支持的功能，那就是可以在循环内部的语句块后面直接编写else块。注意，只有当循环没有被中断(即没有遇到break)，才会遇到else。

[例] 输出2~100所有的素数(因数只有1和本身)

```
for x in range(2, 101):
  for i in range(2, x):
    if x % i == 0:
      break
  else:
    print("%d 是一个素数." % x)
```

41. 统计给定字符串中，不同字符出现次数

```
In [1]: from collections import Counter

In [2]: s = "hello world"

In [3]: dict(Counter(s))
Out[3]: {'h': 1, 'e': 1, 'l': 3, 'o': 2, ' ': 1, 'w': 1, 'r': 1, 'd': 1}
```

42. python字符串常用API

函数名	功能	用例
replace	替换字符串，如果替换后的字符是空字符。(删除)	s = "hello world" s.replace("world", "python")
strip	两端去空白，能去除"\n", "\t", " "(空格)	s = "\n\t zixingAI \t\t\n " s.strip()
split	以指定的字符(参数)切割字符串	s = "www.zixinga.com" s.split(".")
join	以指定的字符拼接字符串	li = ['www', 'zixinga', 'com'] ".".join(li)

43. 常用正则规则

模式	描述
.	通配符，匹配任一字符(1. 任何字符都可以匹配; 2. 只匹配一个)
\d	匹配一个数字(digit)
\D	匹配一个非数字
\w	匹配一个单词(word) 外延: 数字 or 字母 or 下划线
\W	非\w
\s	匹配一个空白(space)字符。外延: \n, \t, ' '
\S	非\s
*	量词，0或者无穷多个 字符个数>= 0
+	量词，1或者无穷多个 字符个数>=1
?	非贪婪

44. 常用xpath语法

表达式	说明	示例
/	从根节点开始选取	/html/body/div[3]/div/div[1]/div[2]/div[1]/div/h2
//	选择所有指定的结点	//div
.	当前结点	./div
..	父节点	../div
@	选取属性	//a/@href
text()	获取标签内的字符	//li/text()
node/elem[i]	选取node结点下的第i个elem元素 注意：下标从1开始。	ul/li[2]
node/elem[last()]	选取node结点下的最后一个elem元素	ul/li[last()]
//elem[@attr='xxx']	选取所有属性attr为xxx的elem元素	//div[@class='content']

45. 常用bs4语法

css选择器

写 CSS 时，标签名不加任何修饰，类名前加`.`，id名前加`#`，用到的方法是`soup.select()`，返回类型是`list`。

1> 通过标签名查找

```
soup.select('a')
```

2> 通过类名查找

```
soup.select('.sister')
```

3> 通过id名查找

```
soup.select('#link1')
```

4> 组合查找: 组合查找即和写 class 文件时，标签名与类名、id名进行的组合原理是一样的，例如查找 p 标签中，id 等于 link1的内容，二者需要用空格分开

```
soup.select('p #link1')
```

直接子标签查找，则使用`>`分隔

```
soup.select("head > title")
```

5> 属性查找: 查找时还可以加入属性元素，属性需要用中括号括起来，注意属性和标签属于同一节点，所以中间不能加空格，否则会无法匹配到。

```
soup.select('a[class="sister"]')
```

6> 获取内容

以上的 select 方法返回的结果都是列表形式，可以遍历形式输出，然后用 get_text() 方法来获取它的内容。

46. Python的标识符命名规则与命名规范

命名规则: 只能有数字、字母、下划线组成，且数字不能开头，区分大小写。

命名规范: 小驼峰、大驼峰、下划线连接。(其中，类名使用大驼峰，变量名和函数名使用下划线连接)

47. 搜索插入位置

给定一个排序数组和一个目标值，在数组中找到目标值，并返回其索引。如果目标值不存在于数组中，返回它将会被按顺序插入的位置。

你可以假设数组中无重复元素。

示例 1:

```
输入: [1,3,5,6], 5
输出: 2
```

思路: 使用二分查找

```
class Solution:
    def searchInsert(self, nums: List[int], target: int) ->
int:
        if not nums:
            return 0
        low = 0
        high = len(nums) - 1
        while low <= high:
            mid = (low + high) // 2
            if nums[mid] == target:
                return mid
            elif nums[mid] < target:
```

```
                low = mid + 1
            else:
                high = mid - 1
        return low
```

48. 有效的括号

给定一个只包括 '('，')'，'{'，'}'，'['，']' 的字符串，判断字符串是否有效。

有效字符串需满足：

左括号必须用相同类型的右括号闭合。 左括号必须以正确的顺序闭合。 注意空字符串可被认为是有效字符串。

示例 1:

```
输入: "()"
输出: true
```

示例 4:

```
输入: "([)]"
输出: false
```

参考代码

```python
class Solution:
    def isValid(self, s: str) -> bool:
        # 字符串长度为奇数，说明不是有效的括号
        if len(s) % 2 != 0:
            return False
        # 建立左右括号的之间的映射关系
        brackets = {")": "(", "]": "[", "}": "{"}
        # 建立栈
        st = []
        for ch in s:
            # 若匹配到左括号
```

```
            if ch in ["(", "[", "{"]:
                st.append(ch)  # 入栈
            elif ch in [")", "]", "}"]:  # 若匹配到右括号
                if not st:
                    return False
                # 与栈顶元素进行匹配
                if brackets[ch] == st[-1]:
                    st.pop()  # 匹配成功删除栈顶元素
                    continue
                else:
                    return False
        # 有效括号到最后检测到结束，栈中应没有元素
        return not st
```

49. 买卖股票的最佳时机

给定一个数组，它的第 i 个元素是一支给定股票第 i 天的价格。

如果你最多只允许完成一笔交易（即买入和卖出一支股票），设计一个算法来计算你所能获取的最大利润。

注意你不能在买入股票前卖出股票。

示例 1:

```
输入: [7,1,5,3,6,4]
输出: 5
解释: 在第 2 天（股票价格 = 1）的时候买入，在第 5 天（股票价格 = 6）的时候卖出，最大利润 = 6-1 = 5 。
     注意利润不能是 7-1 = 6，因为卖出价格需要大于买入价格。
```

示例 2:

```
输入: [7,6,4,3,1]
输出: 0
解释: 在这种情况下，没有交易完成，所以最大利润为 0。
```

参考代码

```python
def max_profit_dynamic(prices):
    if not prices:
        return 0
    min_buy = prices[0]
    max_profit = 0
    for x in prices[1:]:
        min_buy = min(min_buy, x)
        max_profit = max(max_profit, x - min_buy)
    return max_profit
```

50. 最大数

给定一组非负整数，重新排列它们的顺序使之组成一个最大的整数。

示例 1:

```
输入: [10,2]
输出: 210
```

示例 2:

```
输入: [3,30,34,5,9]
输出: 9534330
```

说明: 输出结果可能非常大，所以你需要返回一个字符串而不是整数。

参考代码

```python
class Solution:
    def largestNumber(self, nums):
        """
        :type nums: List[int]
        :rtype: str
        """
        from functools import cmp_to_key
        num = sorted(map(str, nums),key=cmp_to_key(lambda x,y:int(y + x) - int(x + y)))
        ans = "".join(num).lstrip("0")
        return ans or "0"
```