基础概念理解

对于路径来说：Str 和Byte 都是可以接受的,此外或者是Pathlike的接口对象.

首先我们先对Str，Pathlike，Pathlib进行根本上的区分

首先 Str 也就是字符串，是个基本类不多做解释，

Pathlike是一种不可以直接实例化的抽象基类，也就是说只有继承并且重写抽象方法的子类才能进行实例化.（源码再第四页）常见的是pathlib

一个Pathlike子类实例化的例子：

from pathlib import Path  
  
pathlike\_path ='D:/Pathlike\_test'  
str\_path = 'D:/Pathlike\_test'  
p=Path(pathlike\_path)  
t = str\_path  
p=p / 'Images'  
t=t / 'Images'#报错  
print(p)# WindowsPath(‘D:\Pathlike\_test\Images‘)  
print(str(p))# D:\Pathlike\_test\Images  
print(t)#报错

也就是说如果说我们定义了一个Pathlike的子类的实例对象 pathlike\_path

这个对象将会有一个属性Path 如果直接Print（re）则输出的是该对象的地址

总的来说只要有 def \_\_fspath\_\_(path)

都可以成为pathlike的子类 也就是说pathlike类

然后该类有个属性.path来记录地址

所以实例化以后p 就有了如下方法：

p.cwd() # 获取当前路径  
  
p.stat() # 获取当前文件的信息  
  
p.exists() # 判断当前路径是否是文件或者文件夹  
  
p.glob(filename) # 获取路径下的所有符合filename的文件，返回一个generator  
  
p.rglob(filename) # 与上面类似，只不过是返回路径中所有子文件夹的符合filename的文件  
  
p.is\_dir() # 判断该路径是否是文件夹  
  
p.is\_file() # 判断该路径是否是文件  
  
p.iterdir() # 当path为文件夹时，通过yield产生path文件夹下的所有文件、文件夹路径的迭代器  
  
P.mkdir(parents=Fasle) # 根据路径创建文件夹,parents=True时，会依次创建路径中间缺少的文件夹  
p\_news = p / 'new\_dirs/new\_dir'  
p\_news.mkdir(parents=True)  
  
P.open(mode=’r’, buffering = -1, encoding = None, errors = None, newline = None) # 类似于open()函数  
  
p.rename(target) # 当target是string时，重命名文件或文件夹;当target是Path时，重命名并移动文件或文件夹  
  
p.replace(target) # 重命名当前文件或文件夹，如果target所指示的文件或文件夹已存在，则覆盖原文件  
  
p.parent(), p.parents() # parent获取path的上级路径，parents获取path的所有上级路径  
  
p.is\_absolute() # 判断path是否是绝对路径  
  
p.match(pattern) # 判断path是否满足pattern  
  
p.rmdir() # 当path为空文件夹的时候，删除该文件夹  
  
p.name # 获取path文件名  
  
p.suffix # 获取path文件后缀

Pathlike的历史

在 Python 3.4 之前和路径相关操作函数都放在 os 模块里面，尤其是os.path这个子模块，可以说os.path模块非常常用。

而在 Python 3.4，标准库添加了新的模块 - pathlib，它使用面向对象的编程方式来表示文件系统路径。Pathlib就是通过Pathlike来实现的.

所以现在因为Pathlib的优越性，通常对路径的操作都是通过Pathlib来进行

在过去，文件的路径是纯字符串，现在它会是一个pathlib.Path对象class 为Purepath

Purepath的元素接受字符串，Pathlike接口的对象.

在没有pathlike(py3.6)之前Path 对象是不能作为 os 模块下的参数的，需要手动转化成字符串,才能继续作为os下的入参。就是说要额外执行str（Path的对象）

从 Python 3.6 开始，这些接受路径作为参数的函数内部会先通过os.fspath调用 Path 对象的\_\_fspath\_\_方法（也就是Pathlike类）获得字符串类型的路径再去执行逻辑。

根据上面例子其实就可以很好理解

Pathlike类和str的区别

就是 Pathlike 类是专门针对Path处理的类

他有许多处理Path的方法，这个是str 无法具备的

而且它支持 p=p /‘xxx’之类的直接操作

是str 不支持的

以pathlib为例子：

config\_dict = [{"type": type\_,  
 "start": self.time\_base\_line[zip\_pkg]['start\_time'],  
 "end":self.time\_base\_line[zip\_pkg]['end\_time'],  
 "output\_path":'aaaa',  
 "port":channel,  
 "split\_size":100000000}]

output path: C:\Users\wei.chen\Desktop\Wrapper\venv\aaaa\DATALOGGER2F\_20211022\_060000\_20211022\_100000\20211022\_060000\_20211022\_100000\_ETH-LOG5.bin

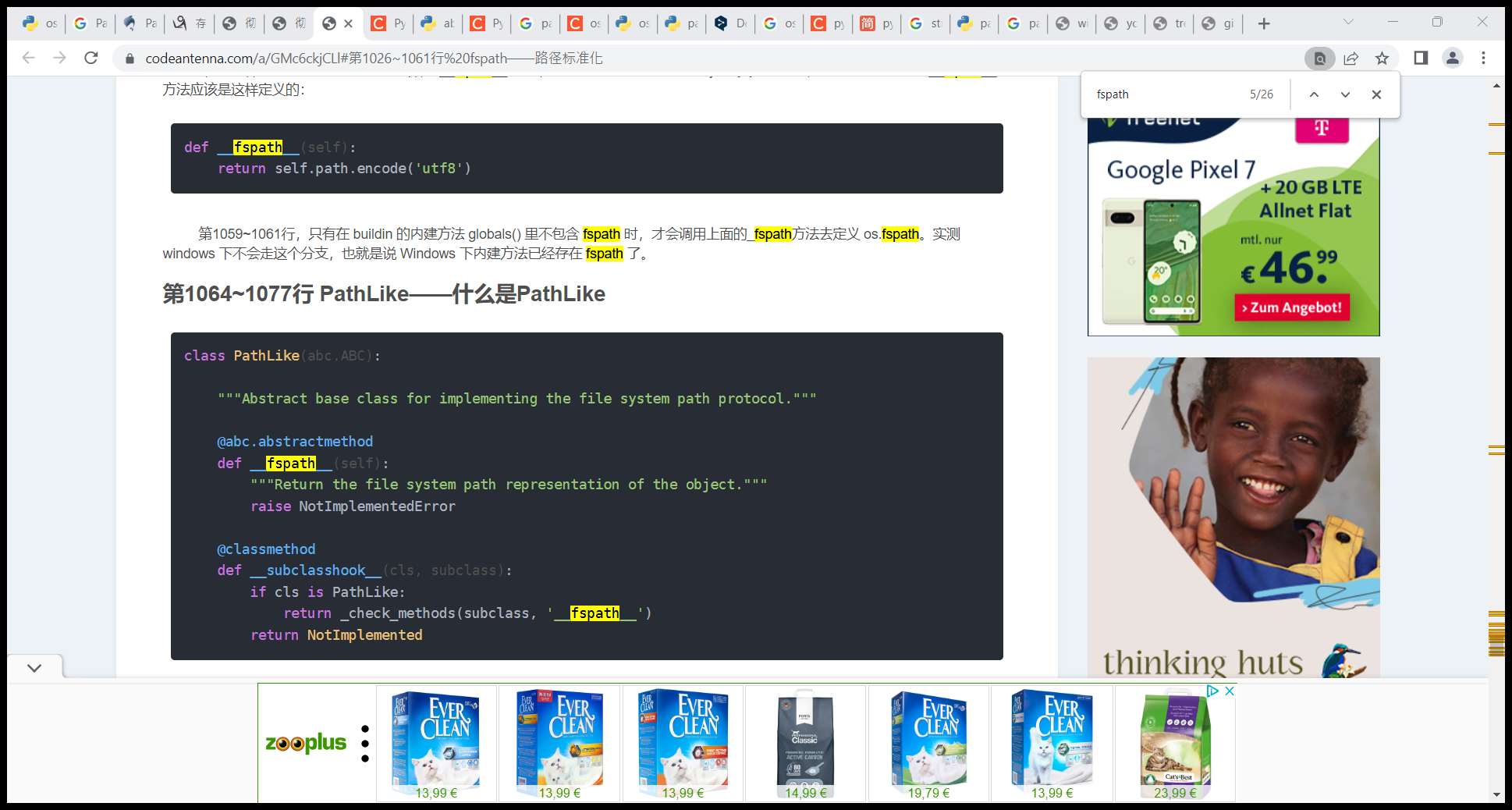
Path =‘aaaa’

是str 无法理解的

Pathlib.path 时

他会理解为工作目录下\aaaa 如果不存在则会自动创建

Os.path和pathlib库里的类都继承了Pathlike.



前面提到好多次，路径要是 PathLike 的，这个抽象类就是为了解释到底什么样的类才是 “ PathLike的”。里面用到了@abc.abstractmethod 装饰器，说明这个类是抽象类不能被实例化。但是可以被继承， \_\_subclasshook\_\_ 检查子类是否实现了\_\_fspath\_\_ 方法。也就是说一个类如果是 “PathLike 的”，首先它必须要实现 \_\_fspath\_\_ 方法。