python实验中遇到的问题及解决方法_Python中遇到的小问题及解决方法汇总

原创

啊倩 于 2021-02-10 08:59:43 发布 2382 收藏 1
文章标签： python实验中遇到的问题及解决方法

本文链接：https://blog.csdn.net/weixin_42545163/article/details/113979511

本文会把学习过程中遇到的一些小问题和解决办法放在这里，以便于大家能够更好地学习python。

一、Python的异常处理

因为想到自己不断尝试写小程序的话会用到抛出异常信息来判断哪里出现了问题:

usage: raise [Exception [, args [, trackback]]]

上面是Python的raise的用法,下面是自己用这个方法实现异常的抛出方法:

```
def check_args(args):

if not args.host:

msg = 'Args missing! One of the following args should be specified \n' \

'--host 192.168.1.1 \n' \

'-f TargetFile \n'

raise Exception(msg)

#参考别人的代码模式,我这样写来抛出异常.
```

二、list转str

这个问题是因为自己的无知吧:

命令行传入的host(即ip地址)是list形式,想要通过list转为str的格式之后来进行socket.connect() , 报错:

自己想象的姿势:

```
client.connect((str(args.host), args.p))
```

正确的姿势:

```
client.connect((''.join(args.host), args.p))
```

上图证明自己的愚蠢(可能下次还会犯同样的错):

三、argparse函数

出发的动机是因为自己写的小程序要通过命令行的形式传参并执行命令.

用自己的简单小实例来显示函数的具体用法吧:

def parse_args():

parser = argparse.ArgumentParser(prog = 'hello',

formatter_class = argparse.RawTextHelpFormatter,

description = '* A tiny toy for fun *\n'

'By ST(www.********)',

usage = 'hellPLC.py [options]')

parser.add_argument('-host', metavar = 'HOST [HOST2 HOST3 ...]', type = str,

default = '', nargs = '*',

help = 'Scan the host form command line')

代码很简单,一看就懂大概函数是什么样子,重点是add_argument的参数,当你传入的cmd参数是这种方式的话:

我们可以看到,需要在-p之后跟一个int型的参数才可以,还有另外一种形式.

parser.add_argument('-p', metavar = 'PORT', type = int,

default = '')

这种方式不需要跟参数,只需要类似于"python -h"这种形式就可以执行并得到想要的结果:

这里就需要对这个函数的各个参数的功能有个基本的了解,这样才能用起来舒服.

parser.add_argument('-b', default= False, dest='b', action='store_true',

help = 'Get the base info')

下图就是我们可以跟的参数,自己的问题就是区分清楚action和dest这两个参数,还有default,type等.

```
- option_strings -- A list of command-line option strings which
should be associated with this action.

- dest -- The name of the attribute to hold the created object(s)

- nargs -- The number of command-line arguments that should be
consumed. By default, one argument will be consumed and a single
value will be produced. Other values include:
- N (an integer) consumes N arguments (and produces a list)
- '?' consumes zero or one arguments
- '*' consumes zero or more arguments (and produces a list)
- '+' consumes one or more arguments (and produces a list)
Note that the difference between the default and nargs=1 is that
with the default, a single value will be produced, while with
nargs=1, a list containing a single value will be produced.

- const -- The value to be produced if the option is specified and the
option uses an action that takes no values.

- default -- The value to be produced if the option is not specified.

- type -- A callable that accepts a single string argument, and
returns the converted value. The standard Python types str, int,
float, and complex are useful examples of such callables. If None,
str is used.

- choices -- A container of values that should be allowed. If not None,
after a command-line argument has been converted to the appropriate
type, an exception will be raised if it is not a member of this
collection.

- required -- True if the action must always be specified at the
command line. This is only meaningful for optional command-line
arguments.
                                                    www.jumtu.com
- help -- The help string describing the argument.

- metavar -- The name to be used for the option's argument with the
help string. If None, the 'dest' value will be used as the name.
"""
```

四、正则匹配

这个东西有点高深了，最初的想法是使用正则来转变接收到的数据的格式，问过达哥之后原本recv到的数据在encode之后的类型是"str"，之前自己的想法是把数据转成list格式，之后提取之类的方便，但"str"类型也可以直接利用偏移来进行分析也可以，直接用"str[]"就可以搞定，因为这个过程中不需要可视化的打印出来，需要打印的是分析之后的结果，所以正则先用不到。

不过还是要把觉得不错的链接放一下：

Python入门篇之正则表达式

https://www.jb51.net/article/56436.htm

Python 匹配任意字符(包括换行符)的正则表达式写法

https://www.jb51.net/article/20654.htm

五、格式化字符串

temp = '123456'

print("word:%s" %temp)

output: word:123456

很简单就可以搞定,但是当时想要返回取多个返回值,一下子蒙住不知道怎么搞了,呵呵了~

f.write ("Block Type: %s \n"

"Block count: %s \n"

%(block_type, block_count))

return block_type, block_count

六、文件读写操作

这个问题别人的博客写的很好很详细了

f = open(r'C:\Movie\test2.txt','w')

直接打开一个文件，如果文件不存在则创建文件，只能只用写命令

f = open(r'C:\Movie\test2.txt')

打开一个文件，只能只用读命令

f.read([size])

size为读取的长度，以byte为单位，如果不写则读取全部内容

f.readline([size])

读一行，如果定义了size，有可能返回的只是一行的一部分。每读取一次，文件操作符向下移动一行。

f.readlines([size])

把文件每一行作为一个list的一个成员，并返回这个list。其实它的内部是通过循环调用readline()来实现的。如果提供size参数，size是表示读取内容的总长，也就是说可能只读到文件的一部分。

关于open模式,参数不同达到的效果也是不同的,比如我需要的是在一个日志文件中不断的追加新的东西而不是每次都直接覆盖掉,所以我用到了"a+"

关于open模式的参数：

w 以写方式打开

a 以追加模式打开

r+ 以读写模式打开

w+ 以读写模式打开

a+ 以读写模式打开 (我用到了这个模式,读写模式不断追加新的东西)

rb 以二进制读模式打开

wb 以二进制写模式打开

ab 以二进制追加模式打开

rb+ 以二进制读写模式打开11 wb+ 以二进制读写模式打开12 ab+ 以二进制读写模式打开

还有一些操作的区别,我当时用到的问题是f.readline,每次都是读文件的一行,没有把全部的内容都读出来,所以区别还是很明显,要区分清楚.

写操作的用法和区别:

f.write(“str”)

把str写到文件中，write()方法不会在str后加上一个换行符

f.writelines(seq)

把seq的内容全部写到文件中(多行一次性写入)。这个函数也只是忠实地写入，不会在每行后面加上任何东西。

f.close()

关闭文件。在读命令或者写命令结束时，需要用关闭。如果文件关闭后依然操作，会抛出ValueError: I/O operation on closed file

f.tell()

返回文件操作标记的当前位置，以文件的开头为起点

fp.next()

返回下一行内容，并将文件操作标记位移到下一行。把一个file用于for … in file这样的语句时，就是调用next()函数来实现遍历的。

fp.seek(offset[,whence])

将文件操作标记为移动到offset位置。

七、遇到的报错情况

自己不想要定义太多的函数,个人感觉是想要将不同功能的函数进行分类,因此我尝试用到了类,知道自己这么做可能是想法上就有些不对的,但是想要瞎试试看看效果,所以结果就是报错了.

如图,这是别人遇到的方法,和我的是一样的,我没有定义静态的函数,所以每次使用之前都需要进行实例化才能调用.

```
TypeError: unbound method a() must be called with A instance as first argument (got nothing instead)
```

错误原因：函数a()非静态方法，故需实例化然后才能使用，改正如下：

```
class A:
    def a(self):
        print("I'm a")

obj = A()
obj.a()
```


或者将a()改为静态方法即可，如下：

```
class A:
    @staticmethod
    def a():
        print ("I'm a")

A.a()
```

总结

以上就是这篇文章的全部内容了，希望本文的内容对大家的学习或者工作能带来一定的帮助，如果有疑问大家可以留言交流。