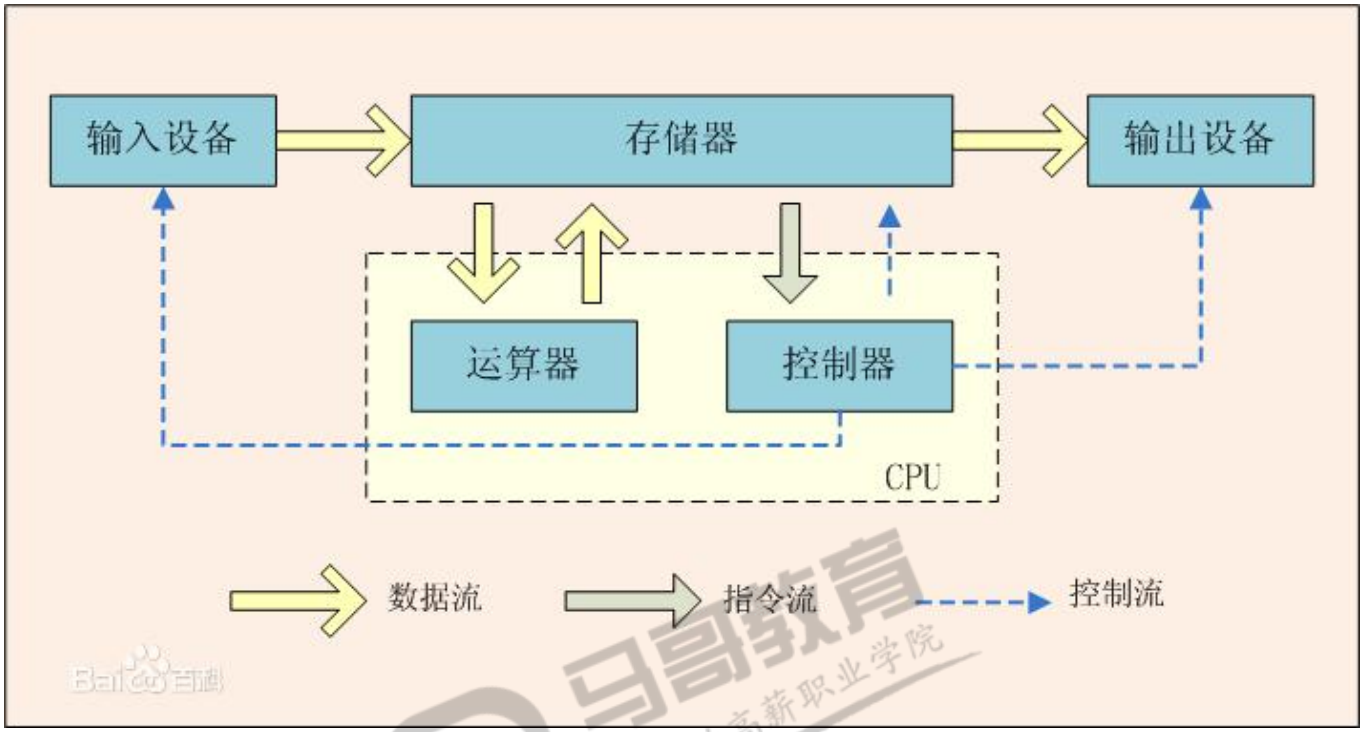
文件操作

冯诺依曼体系架构



- CPU由运算器和控制器组成
  - 运算器，完成各种算数运算、逻辑运算、数据传输等数据加工处理
  - 控制器，控制程序的执行
  - 存储器，用于记忆程序和数据，例如内存
  - 输入设备，将数据或者程序输入到计算机中，例如键盘、鼠标
  - 输出设备，将数据或程序的处理结果展示给用户，例如显示器、打印机等

一般说IO操作，指的是文件IO，如果指的是网络IO，都会直接说网络IO

文件IO常用操作

column	column
open	打开
read	读取
write	写入
close	关闭

readline	行读取
readlines	多行读取
seek	文件指针操作
tell	指针位置

打开操作

open(file, mode='r', buffering=-1, encoding=None, errors=None, newline=None, closefd=True, opener=None)

打开一个文件，返回一个文件对象(流对象)和文件描述符。打开文件失败，则返回异常

基本使用：
创建一个文件test，然后打开它，用完**关闭**

```
f = open("test") # file对象
# windows <_io.TextIOWrapper name='test' mode='r' encoding='cp936'>
# linux <_io.TextIOWrapper name='test' mode='r' encoding='UTF-8'>
print(f.read()) # 读取文件
f.close() # 关闭文件
```

文件操作中，最常用的操作就是读和写。
文件访问的模式有两种：文本模式和二进制模式。不同模式下，操作函数不尽相同，表现的结果也不一样。

open的参数

file

打开或者要创建的文件名。如果不指定路径，默认是当前路径

mode模式

描述字符	意义
r	缺省的，表示只读打开
w	只写打开
x	创建并写入一个新文件
a	写入打开，如果文件存在，则追加
b	二进制模式
t	缺省的，文本模式
+	读写打开一个文件。给原来只读、只写方式打开提供缺失的读或者写能力

在上面的例子中，可以看到默认是文本打开模式，且是只读的。

```
# r模式
f = open('test') # 只读还是只写?
f.read()
f.write('abc')
f.close()

f = open('test', 'r') # 只读
f.write('abc')
f.close()

f = open('test1', 'r') # 只读，文件不存在

# w模式
f = open('test','w') # 只写打开
f.write('abc')
f.close()

>>> cat test # 看看内容

f = open('test', mode='w')
f.close()

>>> cat test # 看看内容

f = open('test1', mode='w')
f.write('123')
f.close()

>>> cat test1 # 看看内容
```

open默认是只读模式r打开已经存在的文件。

r
只读打开文件，如果使用write方法，会抛异常。
如果文件不存在，抛出FileNotFoundError异常

w
表示只写方式打开，如果读取则抛出异常
如果文件不存在，则直接创建文件
如果文件存在，则清空文件内容

```
f = open('test2','x')
f.read() #
f.write('abcd')
f.close()

f = open('test2','x') #
```

x
文件不存在，创建文件，并只写方式打开
文件存在，抛出FileExistsError异常

```
f = open('test2','a')
f.read() #

f.write('abcde')
f.close()

>>> cat test2

f = open('test2','a')
f.write('\n hello')
f.close()

>>> cat test2

f = open('test3','a')
f.write('test3')
f.close()
```

a

文件存在，只写打开，追加内容
文件不存在，则创建后，只写打开，追加内容

r是只读，wxa都是只写。
wxa都可以产生新文件，w不管文件存在与否，都会生成全新内容的文件；a不管文件是否存在，都能在打开的文件尾部追加；x必须要求文件事先不存在，自己造一个新文件

文本模式t
字符流，将文件的字节按照某种字符编码理解，按照字符操作。open的默认mode就是rt。

二进制模式b
字节流，将文件就按照字节理解，与字符编码无关。二进制模式操作时，字节操作使用bytes类型

```
f = open("test3",'rb') # 二进制只读
s = f.read()
print(type(s)) # bytes
print(s)
f.close() # 关闭文件

f = open("test3",'wb') # IO对象
s = f.write("马哥教育".encode())
print(s) # 是什么
f.close()
```

思考：windows下，执行下面的代码

```
f = open("test3",'rw') #

f = open("test3",'r+')
s = f.read()
f.write("马哥教育")
print(f.read()) # 没有显示，为什么
f.close()

f = open("test3",'r+')
s = f.write("magedu") #
print(f.read())
f.close()

>>> cat test3

f = write('test3', 'w+')
```

```
r.read() #
f.close()

>>> cat test3


f = open('test3', 'a+')
f.write('mag')
f.read()
f.close()

>>> cat test3

f = open('test3', 'a+')
f.write('edu')
f.close()

>>> cat test3

f = open('test3', 'x+') #

f = open('test4', 'x+') #
f.write('python')
f.read()
f.close()
>>> cat test4
```


+

为r、w、a、x提供缺失的读写功能，但是，获取文件对象依旧按照r、w、a、x自己的特征。

+不能单独使用，可以认为它是为前面的模式字符做增强功能的。

文件指针

上面的例子中，已经说明了有一个指针。

文件指针，指向当前字节位置

mode=r，指针起始在0

mode=a，指针起始在EOF

tell() 显示指针当前位置

seek(offset[, whence])
移动文件指针位置。offest偏移多少字节，whence从哪里开始。

文本模式下
whence 0 缺省值，表示从头开始，offest只能正整数
whence 1 表示从当前位置，offest只接受0
whence 2 表示从EOF开始，offest只接受0

```
# 文本模式
f = open('test4','r+')
f.tell() # 起始
f.read()
f.tell() # EOF
f.seek(0) # 起始
f.read()
f.seek(2,0)
f.read()
f.seek(2,0)
f.seek(2,1) # offset必须为0
f.seek(2,2) # offset必须为0
f.close()

# 中文
f = open('test4','w+')
f.write('马哥教育')
f.tell()
f.close()
f = open('test4','r+')
f.read(3)
f.seek(1)
f.tell()
f.read() #
f.seek(2) # f.seek(3)
f.close()
```


文本模式支持从开头向后偏移的方式。
whence为1表示从当前位置开始偏移，但是只支持偏移0，相当于原地不动，所以没什么用。
whence为2表示从EOF开始，只支持偏移0，相当于移动文件指针到EOF。
seek是按照字节偏移的。

二进制模式下

whence 0 缺省值，表示从头开始，offest只能正整数
whence 1 表示从当前位置，offest可正可负
whence 2 表示从EOF开始，offest可正可负

```
# 二进制模式
f = open('test4','rb+')
f.tell() # 起始
f.read()
f.tell() # EOF
f.write(b'abc')
f.seek(0) # 起始
f.seek(2,1) # 从当前指针开始，向后2
f.read()
f.seek(-2,1) # 从当前指针开始，向前2

f.seek(2,2) # 从EOF开始，向后2
f.seek(0)
f.seek(-2,2) # 从EOF开始，向前2
f.read()

f.seek(-20,2) # OSError
f.close()
```

二进制模式支持任意起点的偏移，从头、从尾、从中间位置开始。
向后seek可以超界，但是向前seek的时候，不能超界，否则抛异常。

buffering：缓冲区

-1 表示使用缺省大小的buffer。如果是二进制模式，使用io.DEFAULT_BUFFER_SIZE值，默认是4096或者8192。如果是文本模式，如果是终端设备，是行缓存方式，如果不是，则使用二进制模式的策略。

- 0 只在二进制模式使用，表示关buffer
- 1 只在文本模式使用，表示使用行缓冲。意思就是见到换行符就flush
- 大于1 用于指定buffer的大小

buffer 缓冲区
缓冲区一个内存空间，一般来说是一个FIFO队列，到缓冲区满了或者达到阈值，数据才会flush到磁盘。

flush() 将缓冲区数据写入磁盘
close() 关闭前会调用flush()

io.DEFAULT_BUFFER_SIZE 缺省缓冲区大小，字节

先看二进制模式

```
import io

f = open('test4','w+b')
print(io.DEFAULT_BUFFER_SIZE)
f.write("magedu.com".encode())
# cat test4
f.seek(0)
# cat test4
f.write("www.magedu.com".encode())
f.flush()
f.close()

f = open('test4','w+b', 4)
f.write(b"mag")
# cat test4
f.write(b'edu')
# cat test4
f.close()
```

文本模式

```
# buffering=1，使用行缓冲
f = open('test4','w+', 1)
f.write("mag") # cat test4
f.write("magedu"*4) # cat test4
f.write('\n') # cat test4
f.write('Hello\nPython') # cat test4
f.close()

# buffering>1，使用指定大小的缓冲区
f = open('test4','w+', 15)
f.write("mag") # cat test4
f.write('edu') # cat test4
f.write('Hello\n') # cat test4
f.write('\nPython') # cat test4
f.write('a' * (io.DEFAULT_BUFFER_SIZE - 20)) # 设置为大于1没什么用
f.write('\nwww.magedu.com/python')
```

```
f.close()
```

buffering=0
这是一种特殊的二进制模式，不需要内存的buffer，可以看做是一个FIFO的文件。

```
f = open('test4','wb+', 0)
f.write(b"m") # cat test4
f.write(b"a") # cat test4
f.write(b"g") # cat test4
f.write(b"magedu"*4) # cat test4
f.write(b'\n') # cat test4
f.write(b'Hello\nPython')
f.close()
```

buffering	说明
buffering=-1	t和b，都是io.DEFAULT_BUFFER_SIZE
buffering=0	b 关闭缓冲区 t 不支持
buffering=1	b 就1个字节 t 行缓冲，遇到换行符才flush
buffering>1	b模式表示行缓冲大小。缓冲区的值可以超过io.DEFAULT_BUFFER_SIZE，直到设定的值超出后才把缓冲区flush t模式，是io.DEFAULT_BUFFER_SIZE，flush完后把当前字符串也写入磁盘

似乎看起来很麻烦，一般来说，只需要记得：

1. 文本模式，一般都用默认缓冲区大小
2. 二进制模式，是一个个字节的操作，可以指定buffer的大小
3. 一般来说，默认缓冲区大小是个比较好的选择，除非明确知道，否则不调整它
4. 一般编程中，明确知道需要写磁盘了，都会手动调用一次flush，而不是等到自动flush或者close的时候

encoding：编码，仅文本模式使用

None 表示使用缺省编码，依赖操作系统。windows、 linux下测试如下代码

```
f = open('test1','w')
f.write('啊')
f.close()
```

windows下缺省GBK（0xB0A1），Linux下缺省UTF-8（0xE5 95 8A）

其它参数

errors
什么样的编码错误将被捕获
None和strict表示有编码错误将抛出ValueError异常；ignore表示忽略

newline
文本模式中，换行的转换。可以为None、'' 空串、'\r'、'\n'、'\r\n'
读时，None表示'\r'、'\n'、'\r\n'都被转换为'\n'；''表示不会自动转换通用换行符；其它合法字符表示换行符就是指定字符，就会按照指定字符分行
写时，None表示'\n'都会被替换为系统缺省行分隔符os.linesep；'\n'或''表示'\n'不替换；其它合法字符表示'\n'会被替换为指定的字符

```
f = open('o:/test', 'w')
f.write('python\rwww.python.org\nwww.magedu.com\r\npython3')
f.close()

newlines = [None, '', '\n', '\r\n']
for nl in newlines:
    f = open('o:/test', 'r+' , newline=nl) # 缺省替换所有换行符
    print(f.readlines())
    f.close()
```

closefd
关闭文件描述符，True表示关闭它。False会在文件关闭后保持这个描述符。fileobj.fileno()查看

read

read(size=-1)
size表示读取的多少个字符或字节；负数或者None表示读取到EOF

```
f = open('o:/test4','r+', 0)
f.write("magedu")
f.write('\n')
f.write('马哥教育')
f.seek(0)
f.read(7)
f.close()
```

```
# 二进制
f = open('test4','rb+')
f.read(7)
f.read(1)
f.close()
```

行读取

readline(size=-1)
一行行读取文件内容。size设置一次能读取行内几个字符或字节。

readlines(hint=-1)
读取所有行的列表。指定hint则返回指定的行数。

```
# 按行迭代
f = open('test') # 返回可迭代对象

for line in f:
    print(line)

f.close()
```


write

write(s)，把字符串s写入到文件中并返回字符的个数
writelines(lines)，将字符串列表写入文件。

```
f = open('test', 'w+')

lines = ['abc', '123\n', 'magedu'] # 提供换行符
f.writelines(lines)

f.seek(0)
print(f.read())
f.close()
```

close

flush并关闭文件对象。
文件已经关闭，再次关闭没有任何效果。

其他

seekable() 是否可seek
readable() 是否可读
writable() 是否可写
closed 是否已经关闭

上下文管理

问题的引出

在Linux中，执行

```
# 下面必须这么写
lst = []
for _ in range(2000):
    lst.append(open('test'))
# OSError: [Errno 24] Too many open files: 'test'

print(len(lst))
```

lsof 列出打开的文件。没有就# yum install lsof
$ lsof -p 1427 | grep test | wc -l
lsof -p 进程号
ulimit -a 查看所有限制。其中open files就是打开文件数的限制，默认1024

```
for x in lst:
    x.close()
```

将文件一次关闭，然后就可以继续打开了。再看一次lsof。

如何解决？

1、异常处理
当出现异常的时候，拦截异常。但是，因为很多代码都可能出现OSError异常，还不好判断异常就是应为资源限制产生的。

```
f = open('test')
try:
    f.write("abc") # 文件只读，写入失败
finally:
    f.close() # 这样才行
```

使用finally可以保证打开的文件可以被关闭。

2、上下文管理

一种特殊的语法，交给解释器去释放文件对象

上下文管理

```
del f
with open('test') as f:
    f.write("abc") # 文件只读，写入失败

# 测试f是否关闭
f.closed # f的作用域
```

上下文管理

1. 使用with … as 关键字
2. 上下文管理的语句块并不会开启新的作用域
3. with语句块执行完的时候，会自动关闭文件对象

另一种写法

```
f1 = open('test')
with f1:
    f1.write("abc") # 文件只读，写入失败

# 测试f是否关闭
f1.closed # f1的作用域
```

对于类似于文件对象的IO对象，一般来说都需要在不使用的时候关闭、注销，以释放资源。
IO被打开的时候，会获得一个文件描述符。计算机资源是有限的，所以操作系统都会做限制。就

是为了保护计算机的资源不要被完全耗尽，计算资源是共享的，不是独占的。
一般情况下，除非特别明确的知道资源情况，否则不要提高资源的限制值来解决问题。

练习

指定一个源文件，实现copy到目标目录。

例如把/tmp/test.txt 拷贝到 /tmp/test1.txt

有一个文件，对其进行单词统计，不区分大小写，并显示单词重复最多的10个单词。

简单处理后，大概的得数如下：

```
the, 136
is, 60
a, 54
path, 52
if, 42
and, 39
to, 34
of, 33
on, 32
return, 30
```


实际上有效的**path**很多，作为合法的单词**path**统计应该有**100**多个。
对单词做进一步处理后，统计如下：

```
path, 137
the, 136
is, 60
a, 59
os, 50
if, 43
and, 40
to, 34
of, 33
on, 33
```

马哥教育
IT人的高薪职业学院