第9章 文件操作与异常

AI程序设计课程组

目录页

01 文件操作
02 异常与异常类

文件操作

文件的打开

使用Word编写一份简历的流程：

- 打开新建一个Word文件
- 写入个人简历信息
- 保存文件
- 关闭Word软件

文件的打开

操作文件的整体过程和word写简历的过程类似:

- 打开或新建立一个文件
- 读/写数据
- 关闭文件

文件的打开

在python中，使用open方法打开文件：

```
open(文件名，访问模式)
```

- “文件名”必须要填写
- “访问模式”是可选的

文件的打开

注意

如果使用open函数打开文件时，如果没有注明访问模式，则必须保证文件是存在的，否则会报异常。

```
Traceback (most recent call last):
  File "<input>", line 1, in <module>
FileNotFoundError: [Errno 2] No such file or directory: 'test.txt'
```

文件打开的模式

模式	r	r+	w	w+	a	a+
读	+	+		+		+
写		+	+	+	+	+
创建			+	+	+	+
覆盖			+	+		
指针在开始	+	+	+	+		
指针在结尾					+	+

文件打开的模式

访问模式	说明
r	默认模式。以只读方式打开文件，
w	打开一个文件只用于写入。
a	打开一个文件用于追加。
rb	以二进制格式打开一个文件用于只读。
wb	以二进制格式打开一个文件只用于写入。
ab	以二进制格式打开一个文件用于追加。

文件打开的模式

访问模式	说明
r +	打开一个文件用于读写，文件指针会放在文件开头
w+	打开一个文件用于读写，如已存在则覆盖
a+	打开一个文件用于读写，如已存在，则放末尾
rb+	以二进制格式打开一个文件用于读写
wb+	以二进制格式打开一个文件用于读写。如存在则覆盖
ab+	以二进制格式打开一个文件用于追加

文件的关闭

凡是打开的文件，切记要使用close方法关闭文件。

```
# 新建一个文件，文件名为:test.txt
f = open(itheima.txt', 'w')
# 关闭这个文件
f.close()
```

文件读写

写文件

向文件写数据，需要使用write方法来完成，在操作某个文件时，每调用一次write方法，写入的数据就会追加到文件末尾。

```
f = open('itheima.txt', 'w')
f.write('hello itheima, i am here!')
f.close()
```

方式1：使用read方法读取文件

```
f = open('itheima.txt', 'r')
content = f.read(12)
print(content)
print("-"*30)
content = f.read()
print(content)
f.close()
```

运行结果

```
hello itheim
------------------------------
a, i am here!
hello itheima, i am here!
hello itheima, i am here!
```

方式2：使用readlines方法读取文件

```
f= open('itheima.txt', 'r')
content = f.readlines()
i = 1
for temp in content:
    print("%d:%s" % (i, temp))
    i += 1
f.close()
```

运行结果

```
1:hello itheima, i am here!

2:hello itheima, i am here!

3:hello itheima, i am here!
```

方式3：使用readline方法一行一行读数据

```
f = open('itheima.txt', 'r')
content = f.readline()
print("1:%s"%content)
content = f.readline()
print("2:%s"%content)
f.close()
```

运行结果

```
1:hello itheima, i am here!

2:hello itheima, i am here!
```

方式1：使用tell方法来获取文件当前的读写位置

```
f = open("itheima.txt", "r")
str = f.read(4)
print(“读取的数据是 : ”, str)
position = f.tell()
print("当前文件位置 : ", position)
```

tell()方法返回文件的当前位置，即文件指针当前位置。

方式2：使用seek方法来移动文件读取指针到指定位置

seek(offset, from)方法包含两个参数：

–offset:表示偏移量，也就是代表需要移动偏移的字节数

–from:表示方向，可以指定从哪个位置开始偏移

•0:表示文件开头（默认值）

•1:表示当前位置（只能以rb，wb，ab模式打开文件）

•2:表示文件末尾（只能以rb，wb，ab模式打开文件）

文件重命名和删除

文件的重命名

os模块中的rename()方法可以完成文件的重命名。

格式如下：

```
os.rename(需要修改的文件名, 新的文件名)
```

文件的删除

os模块中的remove()方法可以完成文件的删除操作。

格式如下：

```
os.remove(待删除的文件名)
```

文件的其他操作

文件的相关操作

1. 创建文件夹

os模块的mkdir方法用来创建文件夹，示例如下：

```
import os
os.mkdir("张三")
```

2. 获取当前目录

os模块的getcwd方法用来获取当前的目录，示例如下：

```
import os
os.getcwd()
```

3. 改变默认目录

os模块的chdir方法用来改变默认目录，示例如下：

```
import os
os.chdir("../”)
```

4. 获取目录列表

os模块的listdir方法用于获取目录列表，示例如下：

```
import os
os.listdir (”./”)
```

5. 删除文件夹

os模块的rmdir方法用于删除文件夹，示例如下：

```
import os
os.rmdir (”张三”)
```

异常

异常介绍

在Python中，程序在执行的过程中产生的错误称为异常，比如列表索引越界、打开不存在的文件等。

```
print(a)
open("123.txt","r")
```

这两行代码会报错吗？

异常介绍

报错信息:

NameError: name 'a' is not defined

FileNotFoundError: [Errno 2] No such file or directory: '123.txt'

- 第1个异常的类型为NameError（名称），描述信息为a没有定义;
- 第2个异常为FileNotFoundError，描述信息为没有找到123.txt文件

异常类

异常类

- 所有异常都是基类Exception的成员，它们都定义在exceptions模块中。
- 如果这个异常对象没有进行处理和捕捉，程序就会用所谓的回溯（traceback，一种错误信息）终止执行，这些信息包括错误的名称（例如NameError）、原因和错误发生的行号。

1. NameError

尝试访问一个未声明的变量，会引发NameError。

```
Traceback (most recent call last):
  File "D:/PythonCode/Chapter09/异常.py", line 1, in <module>
    print(foo)
NameError: name 'foo' is not defined
```

2. ZeroDivisionError

当除数为零的时候，会引发ZeroDivisionError异常。

```
Traceback (most recent call last):
  File "D:/PythonCode/Chapter09/异常.py", line 1, in <module>
    1/0
ZeroDivisionError: division by zero
```

3. SyntaxError

当解释器发现语法错误时，会引发SyntaxError异常

```
File "D:/PythonCode/Chapter09/异常.py", line 2
    for i in list
              ^
SyntaxError: invalid syntax
```

4. IndexError

当使用序列中不存在的索引时，会引发IndexError异常

```
Traceback (most recent call last):
  File "D:/PythonCode/Chapter09/异常.py", line 2, in <module>
    list[0]
IndexError: list index out of range
```

5. KeyError

当使用映射中不存在的键时，会引发KeyError异常。

```
Traceback (most recent call last):
  File "D:/PythonCode/Chapter09/异常.py", line 2, in <module>
    myDict['server']
KeyError: 'server'
```

6. FileNotFoundError

试图打开不存在的文件时，会引发FileNotFoundError

```
Traceback (most recent call last):
  File "D:/PythonCode/Chapter09/异常.py", line 1, in <module>
    f = open("test")
FileNotFoundError: [Errno 2] No such file or directory: 'test'
```

7. AttributeError

当尝试访问未知对象属性时，会引发AttributeError异常

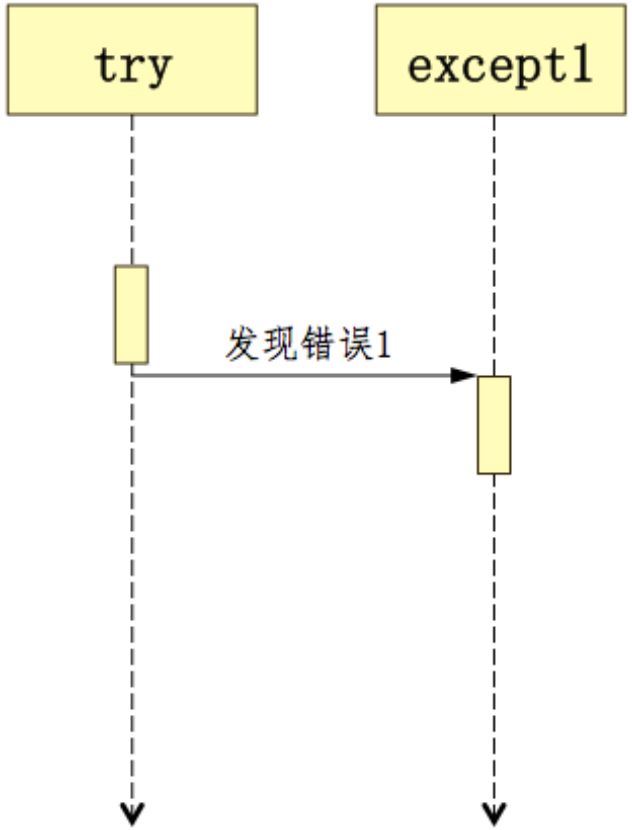
```
Traceback (most recent call last):
  File "D:/PythonCode/Chapter09/异常.py", line 6, in <module>
    print(car.name)
AttributeError: 'Car' object has no attribute 'name'
```

异常处理

捕获简单异常

try-except语句定义了监控异常的一段代码，并提供了处理异常的机制。

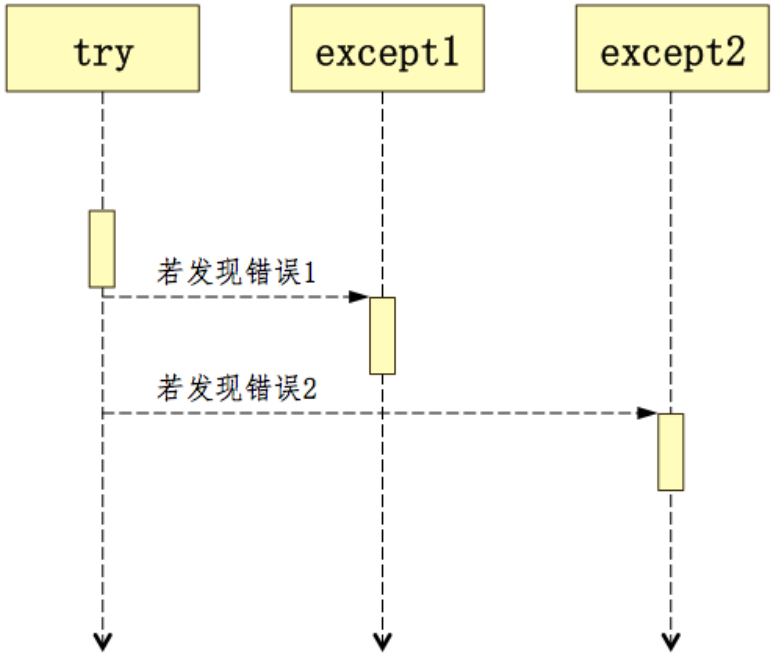
```
try：
        # 语句块
except：
        # 异常处理代码
```



捕获多个异常

处理多个异常的try-except语句格式如下：

```
try：
   # 语句块
except 异常名称1：
    # 异常处理代码1
except异常名称2：
    # 异常处理代码
   …
```



捕获异常的描述信息

当出现多种异常时，为了区分不同的错误信息，可以使用as获取系统反馈的信息。

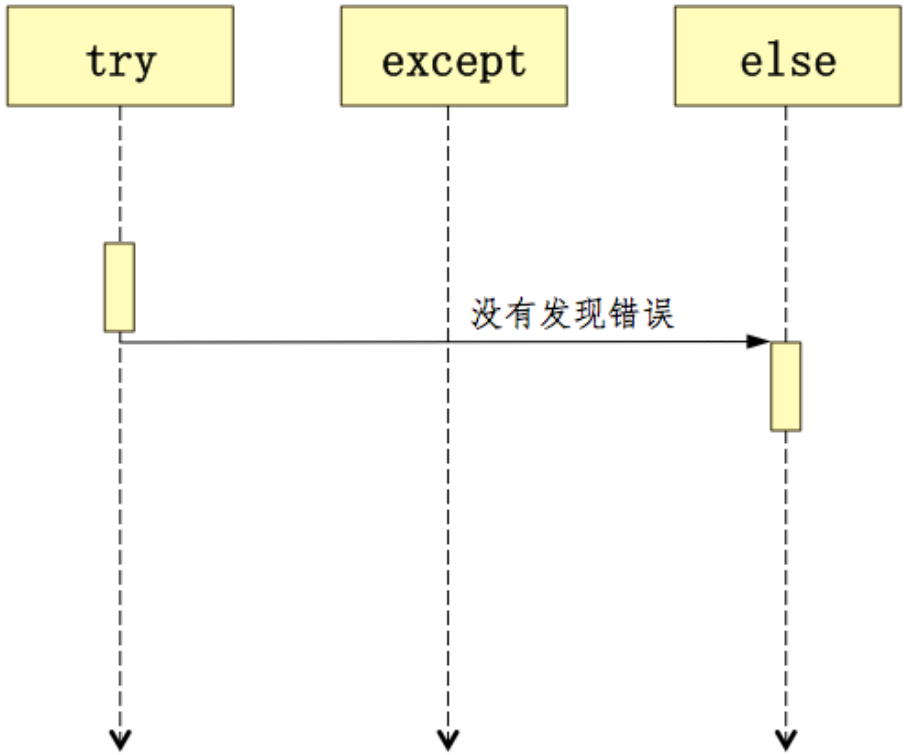
```
# 获取描述信息
except (ZeroDivisionError, ValueError) as result:
    print("捕捉到异常:%s"%result)
```

捕获所有的异常

当程序中出现大量异常时，捕获这些异常是非常麻烦的。这时，我们可以在except子句中不指明异常的类型，这样，不管发生何种类型的异常，都会执行except里面的处理代码。

没有捕获到异常（else）

如果try语句没有捕获到任何的错误信息，就不再执行任何except语句，而是会执行else语句。



终止行为（finally）

在程序中，无论是否捕捉到异常，都必须要执行某件事情，例如关闭文件、释放锁等，这时可以提供finally语句处理。通常情况下，finally用于释放资源。

抛出异常

raise语句

使用raise语句能显示地触发异常，格式如下：

1.raise 异常类名

2.raise 异常类对象

} 引发指定异常类的实例

3.raise → 重新引发刚刚发生的异常

1. 使用类名引发异常

当raise语句指定异常的类名时，会创建该类的实例对象，然后引发异常。

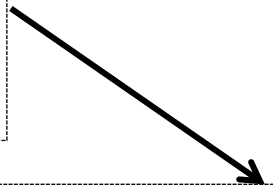
```
raise IndexError
```

→

```
Traceback (most recent call last):
  File "D:/异常.py", line 1, in <module>
    raise IndexError
IndexError
```

2. 使用异常类的实例引发异常

```
index = IndexError()
raise index
```



```
Traceback (most recent call last):
  File "D:/异常.py", line 2, in <module>
    raise index
IndexError
```

3. 传递异常

不带任何参数的raise语句，可以再次引发刚刚发生过的异常，作用就是向外传递异常。

```
try:
    raise IndexError
except:
    print("出错了")
    raise
```

```
出错了
  File "D:/异常.py", line 2, in <module>
    raise IndexError
IndexError
```

4. 指定异常的描述信息

```
raise IndexError("索引下标超出范围")
```

```
Traceback (most recent call last):
  File "D:/异常.py", line 1, in <module>
    raise IndexError("索引下标超出范围")
IndexError: 索引下标超出范围
```

5. 异常引发异常

使用raise...from…可以在异常中抛出另外的异常。

```
try:
    num
except Exception as exception:
    raise IndexError("下标超出范围") from exception
```

try里面只定义了变量num，会引发NameError异常。except子句使用raise...from… 抛出NameError异常后再抛出“下标越界”的异常。

assert语句

- assert语句又称作断言，指的是期望用户满足指定的条件。
- 当用户定义的约束条件不满足的时候，它会触发AssertionError异常，所以assert语句可以当做条件式的raise语句。

assert语句

assert语句格式如下：

assert 逻辑表达式，data

```
if not 逻辑表达式:
    raise AssertionError(data)
```

assert后面紧跟一个逻辑表达式，相当于条件。Data通常是一个字符串，当条件为false时作为异常的描述信息。

assert语句

断言的示例如下：

```
a = 0
assert a!=0,"a的值不能为0"
```

```
Traceback (most recent call last):
  File "D:/异常.py", line 2, in <module>
    assert a!=0,"a的值不能为0"
AssertionError: a的值不能为0
```

自定义异常

自定义异常

- 创建一个继承Exception类的子类，就是自定义异常类。
- 当遇到自己设定的错误时，使用raise语句抛出自定义的异常。

with语句

现在有一个需求，打开foo.txt文本文件，读取完所有的数据以后关闭文件。 示例代码如下：

```
file = open(“/tmp/foo.txt”)   # 打开文件
data = file.read()                  # 读取数据
file.close()                          # 关闭文件
```

文件读取出现问题怎么办？

忘记关闭文件出现问题怎么办？

with语句

为了避免在文件读取的过程中产生这些问题，可以在上述示例中增加处理异常的语句， 加强版本如下：

```
file = open("/tmp/foo.txt")
try:
    data = file.read()
finally:
    file.close()
```

该代码虽然解决了产生异常的可能，但是这段代码过于冗长。此时，在示例中使用with语句处理上下文环境产生的异常，具体如下。

```
with open("/tmp/foo.txt") as file:
    data = file.read()
```

with语句

Python 2.5开始，引入了with语句，with语句适用于对资源进行访问的场合，确保不管使用过程中是否发生异常都会执行必要的“清理”操作，释放资源。

with语句

with语句格式

```
with context_expr [as var]:
    with_body
```

- context_expr：需要返回一个上下文管理器对象，该对象并不赋值给as子句中的 var。
- var：可以是变量或者元组。
- with_body：with语句包裹的代码块。

with语句执行过程：

（1）执行context_expr，生成上下文管理context_manager；

（2）调用上下文管理器的__enter__()方法，如果使用了as子句，就把__enter__() 方法的返回值赋值给 as 子句中的var；

（3）执行语句体with_body。

（4）无论在执行的过程中是否发生异常，都会执行上下文管理器的__exit__() 方法。该方法负责执行程序的“清理”工作，如释放资源等。

with语句执行过程：

（5）如果执行过程中没有出现异常，或者语句体中执行了break、continue或者return语句，则以None 作为参数调用__exit__()方法；如果执行过程中出现异常，则会使用sys.exc_info 得到的异常信息为参数调用__exit__()方法。

（6）出现异常时，如果__exit__()方法返回的结果为False，则会重新抛出异常，让with 之外的语句逻辑来处理异常，这是通用做法；如果返回True，则忽略异常，不再对异常进行处理。

with语句

- 要想使用with语句进行工作，前提是要有上下文管理器。
- 上下文管理器是Python 2.5开始支持的一种语法，用于规定某个对象的使用范围，一旦进入或者离开使用范围，会有特殊的操作被调用。

1. 上下文管理协议

- __enter__(self)：进入上下文管理器时调用此方法，其返回值被放入with-as语句中as说明符指定的变量中。
- _exit__(self, type, value, tb)：离开上下文管理器调用此方法。如果有异常出现，type、value、tb分别为异常的类型、值和追踪信息；如果没有异常，3个参数均设为None。此方法返回值为True或者False，分别指示被引发的异常得到了还是没有得到处理，如果返回False，引发的异常会被传递出上下文。

2. 上下文管理器

- 支持上下文管理协议的对象，用于实现__enter__()和__exit__()方法。上下文管理器定义执行with语句时要建立的运行时上下文，负责执行with语句块上下文中的进入与退出操作。通常情况下，使用with语句调用上下文管理器，也可以通过直接调用其方法来使用。

3. 运行时上下文

- 由上下文管理器创建，通过上下文管理器的__enter__()和__exit__()方法实现。其中，__enter__() 方法在语句体执行之前进入运行时上下文，__exit__() 在语句体执行完后从运行时上下文退出。

本章小结

- 文件操作
- 异常与异常类
- 系统异常及自定义异常
- with和as安装环境