第7部分 异常处理结构

异常处理结构与单元测试

- 异常是指程序运行时引发的错误，引发错误的原因有很多，例如除零、下标越界、文件不存在、网络异常、类型错误、名字错误、字典键错误、磁盘空间不足，等等。

- 如果这些错误得不到正确的处理将会导致程序终止运行，而合理地使用异常处理结构可以使得程序更加健壮，具有更强的容错性，不会因为用户不小心的错误输入或其他运行时原因而造成程序终止。也可以使用异常处理结构为用户提供更加友好的提示。

- 程序出现异常或错误之后是否能够调试程序并快速定位和解决存在的问题也是程序员综合水平和能力的重要体现方式之一。

1.1 异常的概念与表现形式

- 当程序执行过程中出现错误时Python会自动引发异常。
- 异常处理是因为程序执行过程中由于输入不合法导致程序出错而在正常控制流之外采取的行为。
- 严格来说，语法错误和逻辑错误不属于异常，但有些语法错误往往会导致异常，例如由于大小写拼写错误而试图访问不存在的对象，或者试图访问不存在的文件，等等。当Python检测到一个错误时，解释器就会指出当前程序流已经无法再继续执行下去，这时候就出现了异常。

1.1 异常的概念与表现形式

- 异常表现形式：

```
>>> 2 / 0                                  #除0错误
Traceback (most recent call last):
  File "<pyshell#9>", line 1, in <module>
    2 / 0
ZeroDivisionError: division by zero
>>> 'a' + 2                                #操作数类型不支持，略去异常的详细信息
TypeError: Can't convert 'int' object to str implicitly
>>> {3, 4, 5} * 3                          #操作数类型不支持
TypeError: unsupported operand type(s) for *: 'set' and 'int'
>>> print(testStr)                         #变量名不存在
NameError: name 'testStr' is not defined
>>> fp = open(r'D:\test.data', 'rb')   #文件不存在
FileNotFoundError: [Errno 2] No such file or directory: 'D:\\test.data'
>>> len(3)                                 #参数类型不匹配
TypeError: object of type 'int' has no len()
>>> list(3)                                #参数类型不匹配
TypeError: 'int' object is not iterable
```

1.2 Python内置异常类层次结构

```
BaseException
 +-- SystemExit
 +-- KeyboardInterrupt
 +-- GeneratorExit
 +-- Exception
      +-- StopIteration
      +-- ArithmeticError
      |    +-- FloatingPointError
      |    +-- OverflowError
      |    +-- ZeroDivisionError
      +-- AssertionError
      +-- AttributeError
      +-- BufferError
      +-- EOFError
      +-- ImportError
      +-- LookupError
      |    +-- IndexError
      |    +-- KeyError
      +-- MemoryError
      +-- NameError
      |    +-- UnboundLocalError
```

```
      +-- OSError
      |    +-- BlockingIOError
      |    +-- ChildProcessError
      |    +-- ConnectionError
      |    |    +-- BrokenPipeError
      |    |    +-- ConnectionAbortedError
      |    |    +-- ConnectionRefusedError
      |    |    +-- ConnectionResetError
      |    +-- FileExistsError
      |    +-- FileNotFoundError
      |    +-- InterruptedError
      |    +-- IsADirectoryError
      |    +-- NotADirectoryError
      |    +-- PermissionError
      |    +-- ProcessLookupError
      |    +-- TimeoutError
      +-- ReferenceError
      +-- RuntimeError
      |    +-- NotImplementedError
      +-- SyntaxError
      |    +-- IndentationError
      |         +-- TabError
```

```
      +-- SystemError
      +-- TypeError
      +-- ValueError
      |    +-- UnicodeError
      |         +-- UnicodeDecodeError
      |         +-- UnicodeEncodeError
      |         +-- UnicodeTranslateError
      +-- Warning
           +-- DeprecationWarning
           +-- PendingDeprecationWarning
           +-- RuntimeWarning
           +-- SyntaxWarning
           +-- UserWarning
           +-- FutureWarning
           +-- ImportWarning
           +-- UnicodeWarning
           +-- BytesWarning
           +-- ResourceWarning
```

1.3 异常处理结构

（1）try...except...

- 其中try子句中的代码块包含可能会引发异常的语句，而except子句则用来捕捉相应的异常。
- 如果try子句中的代码引发异常并被except子句捕捉，就执行except子句的代码块；
- 如果try中的代码块没有出现异常就继续往下执行异常处理结构后面的代码；
- 如果出现异常但没有被except捕获，继续往外层抛出，如果所有层都没有捕获并处理该异常，程序崩溃并将该异常呈现给最终用户。
- 该结构语法如下：

```
try:
    #可能会引发异常的代码，先执行一下试试
except Exception[ as reason]:
    #如果try中的代码抛出异常并被except捕捉，就执行这里的代码
```

1.3 异常处理结构

（2）try...except...else...

- 如果try中的代码抛出了异常并且被except语句捕捉则执行相应的异常处理代码，这种情况下就不会执行else中的代码；
- 如果try中的代码没有引发异常，则执行else块的代码。
- 该结构的语法如下：

```
try:
    #可能会引发异常的代码
except Exception [ as reason]:
    #用来处理异常的代码
else:
    #如果try子句中的代码没有引发异常，就继续执行这里的代码
```

1.3 异常处理结构

（3）try...except...finally...

- 在这种结构中，无论try中的代码是否发生异常，也不管抛出的异常有没有被except语句捕获，finally子句中的代码总是会得到执行。该结构语法为：

```
try:
    #可能会引发异常的代码
except Exception [ as reason]:
    #处理异常的代码
finally:
    #无论try子句中的代码是否引发异常，都会执行这里的代码
```

1.3 异常处理结构

（4）可以捕捉多种异常的异常处理结构

- 一旦try子句中的代码抛出了异常，就按顺序依次检查与哪一个except子句匹配，如果某个except捕捉到了异常，其他的except子句将不会再尝试捕捉异常。该结构类似于多分支选择结构，语法格式为：

```
try:
    #可能会引发异常的代码
except Exception1:
    #处理异常类型1的代码
except Exception2:
    #处理异常类型2的代码
except Exception3:
    #处理异常类型3的代码
...
```

1.3 异常处理结构

（5）同时包含else子句、finally子句和多个except子句的异常处理结构

```
>>> def div(x, y):
    try:
        print(x / y)
    except ZeroDivisionError:
        print('ZeroDivisionError')
    except TypeError:
        print('TypeError')
    else:
        print('No Error')
    finally:
        print("executing finally clause")
```

1.3 异常处理结构

```
>>> div(3,5)
0.6
No Error
executing finally clause
>>> div('3',5)
TypeError
executing finally clause
>>> div(3,0)
ZeroDivisionError
executing finally clause
```

```
17  local _gcd
18  _gcd = function (a, b)
19      if b == 0 then
20          return a
21      end
22
23      return _gcd(b, a % b)
24  end
25
```

1.4 断言

❖断言语句的语法是：

```
assert expression[, reason]
```

✓ 当判断表达式expression为真时，什么都不做；如果表达式为假，则抛出异常。

✓assert语句一般用于开发程序时对特定必须满足的条件进行验证，仅当__debug__为True时有效。当Python脚本以-O选项编译为字节码文件时，assert语句将被移除以提高运行速度。

1.4 断言

```
>>> a = 3
>>> b = 5
>>> assert a==b, 'a must be equal to b'
Traceback (most recent call last):
  File "<pyshell#17>", line 1, in <module>
    assert a==b, 'a must be equal to b'
AssertionError: a must be equal to b

>>> try:
    assert a==b, 'a must be equal to b'
except AssertionError as reason:
    print('%s:%s'%(reason.__class__.__name__, reason))

AssertionError:a must be equal to b
```