Python程序设计

异常

刘安
苏州大学，计算机科学与技术学院

http://web.suda.edu.cn/anliu/

运行时错误

- FileNotFoundError & IndexError

```
>>> f = open('xyz/abc.txt')
Traceback (most recent call last):
  File "<pyshell#642>", line 1, in <module>
    f = open('xyz/abc.txt')
FileNotFoundError: [Errno 2] No such file or
directory: 'xyz/abc.txt'
>>>
>>> L = [1, 2, 3]
>>> L[3]
Traceback (most recent call last):
  File "<pyshell#645>", line 1, in <module>
    L[3]
IndexError: list index out of range
```

运行时错误

- TypeError & ZeroDivisionError

```
>>> '1' + 2
Traceback (most recent call last):
  File "<pyshell#647>", line 1, in <module>
    '1' + 2
TypeError: can only concatenate str (not "int
") to str
>>>
>>> 1 / 0
Traceback (most recent call last):
  File "<pyshell#649>", line 1, in <module>
    1 / 0
ZeroDivisionError: division by zero
```

运行时错误

- ValueError & KeyError

```
>>> import math
>>> math.sqrt(-1)
Traceback (most recent call last):
  File "<pyshell#652>", line 1, in <module>
    math.sqrt(-1)
ValueError: math domain error
>>>
>>> D = {'a' : 1, 'b' : 2}
>>> D['c']
Traceback (most recent call last):
  File "<pyshell#655>", line 1, in <module>
    D['c']
KeyError: 'c'
```

运行时错误

- NameError & AttributeError

```
>>> lst.append(3)
Traceback (most recent call last):
  File "<pyshell#669>", line 1, in <module>
    lst.append(3)
NameError: name 'lst' is not defined
>>>
>>> L = []
>>> L.appent(3)
Traceback (most recent call last):
  File "<pyshell#672>", line 1, in <module>
    L.appent(3)
AttributeError: 'list' object has no attribute
'appent'
```

异常

- 异常可以看成是运行时发生的错误
- 异常机制是用来处理运行时错误的方法
- 异常可以根据错误自动的被触发，也可以由代码主动触发
- 异常主要用于错误处理
  - Python默认的异常处理行为：停止程序，显示出错信息
  - 可以使用相应的语句来捕获异常，处理后继续执行程序

默认的异常处理

```
>>> def get(obj, index):
        return obj[index]

>>> x = 'python'
>>> get(x, 6)
Traceback (most recent call last):
  File "<pyshell#424>", line 1, in <module>
    get(x, 6)
  File "<pyshell#422>", line 2, in get
    return obj[index]
IndexError: string index out of range
```

捕获异常

```
>>> def get(obj, index):
        return obj[index]

>>> x = 'python'
>>> get(x, 6)
Traceback (most recent call last):
  File "<pyshell#424>", line 1, in <module>
    get(x, 6)
  File "<pyshell#422>", line 2, in get
    return obj[index]
IndexError: string index out of range
>>>
>>> def test_get():
        try: #把可能发生错误的语句放在try块中
            get(x, 6)
        except IndexError: #如果发生IndexError
            print('索引越界啦') #显示提示信息


>>> test_get()
索引越界啦
```

捕获异常后继续执行

- 当try分句执行时发生错误，自动跳转至相应的except分句
- except分句执行完毕后，继续执行try语句之后的语句

```
>>> def catcher():
        try:
            get(x, 6)
        except IndexError:
            print('索引越界啦')
        print('继续执行啦')


>>> catcher()
索引越界啦
继续执行啦
```

通过raise语句触发异常

```
>>> def my_sqrt(x):
        if not isinstance(x, (int, float)):
            raise TypeError('x必须是数值型')
        elif x < 0:
            raise ValueError('x必须大于等于0')
        else:
            print('x符合条件，可以计算啦')
```

```
>>> my_sqrt('3')
Traceback (most recent call last):
  File "<pyshell#693>", line 1, in <module>
    my_sqrt('3')
  File "<pyshell#692>", line 3, in my_sqrt
    raise TypeError('x必须是数值型')
TypeError: x必须是数值型
```

通过raise语句触发异常

```
>>> def my_sqrt(x):
        if not isinstance(x, (int, float)):
            raise TypeError('x必须是数值型')
        elif x < 0:
            raise ValueError('x必须大于等于0')
        else:
            print('x符合条件，可以计算啦')
```

```
>>> my_sqrt(-1)
Traceback (most recent call last):
  File "<pyshell#700>", line 1, in <module>
    my_sqrt(-1)
  File "<pyshell#699>", line 5, in my_sqrt
    raise ValueError('x必须大于等于0')
ValueError: x必须大于等于0
```

终止行为

- 无论try分句中是否发生异常，finally分句都会被执行

```
>>> x = 'python'
>>>
>>> def test_get_finally():
        try:
            print(get(x, 3))
        finally:
            print('我一定会被执行的')


>>> test_get_finally()
h
我一定会被执行的
```

终止行为

```
>>> def test_get_finally(i):
        try:
            print(get(x, i))
        finally:
            print('执行finally子句')
        print('执行try语句后面的语句')
```

```
>>> test_get_finally(3)
h
执行finally子句
执行try语句后面的语句
```

```
>>> def test_get_finally(i):
        try:
            print(get(x, i))
        finally:
            print('执行finally子句')
        print('执行try语句后面的语句')


>>> test_get_finally(6)
执行finally子句
Traceback (most recent call last):
  File "<pyshell#730>", line 1, in <module>
    test_get_finally(6)
  File "<pyshell#729>", line 3, in test_get_finally
    print(get(x, i))
  File "<pyshell#712>", line 2, in get
    return obj[index]
IndexError: string index out of range
```

try/except/else/finally语句

- try分句 → except分句 → else分句 → finally分句
- 这些关键字具有相同的缩进
- 首先执行try分句
  - 如果触发异常，寻找匹配的except分句并执行
  - 如果没有触发异常，执行else分句
- 无论如何，finally语句块最后会被执行

```
>>> def func(lst):
        m = max(lst)
        avg = len(lst) / sum(lst)
        return lst[3]

>>> def catcher(lst):
        try:
            func(lst)
        except IndexError:
            print('发生IndexError')
        except ZeroDivisionError:
            print('发生ZeroDivisionError')
        except:
            print('发生了IE和ZDE之外的错误')
        else:
            print('没有发生错误，执行else子句')
        finally:
            print('执行finally子句')
```

```
def func(lst):
    m = max(lst)
    avg = len(lst) / sum(lst)
    return lst[3]

def catcher(lst):
    try:
        func(lst)
    except IndexError:
        print('发生IndexError')
    except ZeroDivisionError:
        print('发生ZeroDivisior
    except:
        print('发生了IE和ZDE之外
    else:
        print('没有发生错误，执行else子句')
    finally:
        print('执行finally子句')
```

```
>>> catcher([1, 2, 3])
发生IndexError
执行finally子句
>>>
>>> catcher([1, 2, 3, -6])
发生ZeroDivisionError
执行finally子句
>>>
>>> catcher([1, 2, 3, 4])
没有发生错误，执行else子句
执行finally子句
>>>
>>> catcher([])
发生了IE和ZDE之外的错误
执行finally子句
```

try/except/else/finally语句

- 顺序：try分句 → except分句 → else分句 → finally分句
- try必须有一个except或者finally
- 如果出现else，则必须有至少一个except

情况1 - 没有发生异常

```
>>> def f1():
        try:
            x = 'python'[1]
        except IndexError:
            print('执行except分句')
        finally:
            print('执行finally分句')
        print('执行try语句之后的语句')
```

```
>>> f1()
执行finally分句
执行try语句之后的语句
```

情况2 - 发生异常并被捕获

```
>>> def f2():
        try:
            x = 'python'[10]
        except IndexError:
            print('执行except分句')
        finally:
            print('执行finally分句')
        print('执行try语句之后的语句')


>>> f2()
执行except分句
执行finally分句
执行try语句之后的语句
```

情况3 - 没有发生异常，有else分句

```
>>> def f3():
        try:
            x = 'python'[1]
        except IndexError:
            print('执行except分句')
        else:
            print('执行else分句')
        finally:
            print('执行finally分句')
        print('执行try语句之后的语句')


>>> f3()
执行else分句
执行finally分句
执行try语句之后的语句
```

情况4 - 发生异常，没有捕获

```
>>> def f4():
        try:
            x = 1 / 0
        except IndexError:
            print('执行except分句')
        finally:
            print('执行finally分句')
        print('执行try语句之后的语句')


>>> f4()
执行finally分句
Traceback (most recent call last):
  File "<pyshell#789>", line 1, in <module>
    f4()
  File "<pyshell#788>", line 3, in f4
```

嵌套异常处理

- 当try/except嵌套且触发异常时，只有相符的第一个except分句会执行

```
>>> def test_nested():
        try:
            try:
                print(1 + '1')
            except TypeError:
                print('执行内部except分句')
        except TypeError:
            print('执行外部except分句')


>>> test_nested()
执行内部except分句
```

嵌套异常处理

- 当try/except嵌套且触发异常时，只有相符的第一个except分句会执行

```
>>> def test_nested():
        try:
            try:
                print(1 + '1')
            except IndexError:
                print('执行内部except分句')
        except TypeError:
            print('执行外部except分句')


>>> test_nested()
执行外部except分句
```

嵌套异常处理

- 当try/finally嵌套且触发异常时，每个finally分句都会执行

```
>>> def test_nested():
>>> def print_file_content():
        try:
            fname = input('Enter file to read: ')
            f = open(fname, 'r')
            print(f.read())
        except FileNotFoundError:
            print('File', fname, 'not found.')


>>> test_nested()
执行内部finally分句
执行外部finally分句
Traceback (most recent call last):
  File "<pyshell#809>", line 1, in <module>
```

处理文件打开异常

```
>>> def print_file_content():
        try:
            fname = input('Enter file to read: ')
            f = open(fname, 'r')
            print(f.read())
        except FileNotFoundError:
            print('File', fname, 'not found.')

>>> print_file_content()
Enter file to read: /Users/ryan/Downloads/prime.txt
2 3 5 7
>>>
>>> print_file_content()
Enter file to read: /Users/ryan/Downloads/prime
File /Users/ryan/Downloads/prime not found.
```

```
>>> def print_file_content():
        while True:
            try:
                fname = input('Enter file name: ')
                if not fname: #输入空字符串则退出循环
                    break
                f = open(fname)
                print(f.read())
                f.close()
                break #显示完文件内容后退出循环
            except FileNotFoundError:
                print('File could not be found. Re-enter.')


>>> print_file_content()
Enter file name: /Users/ryan/Downloads/prime
File could not be found. Re-enter.
Enter file name: /Users/ryan/Downloads/prime.txt
2 3 5 7
```

处理文件打开异常

- 注意try分句和else分句里面的内容

```
>>> def print_file_content():
        while True:
            fname = input('Enter file name: ')
            if not fname:
                break
            try:
                f = open(fname)  #可能发生异常
            except FileNotFoundError:
                print('File could not be found. Re-enter.')
            else: #如果打开文件没有发生异常
                print(f.read())
                f.close()
                break
```

处理文件读写异常

- 如果文件打开正常，但是读写文件时发生异常如何处理？

```
>>> def print_file_content():
        f = open('/Users/ryan/Downloads/prime.txt')
        try:
            for line in f:
                print(line)
    finally: #无论发生什么，都要关闭文件
        f.close()
```

使用with/as语句代替try/finally

```
>>> def print_file_content():
        f = open('/Users/ryan/Downloads/prime.txt')
        try:
            for line in f:
                print(line)
        finally: #无论发生什么，都要关闭文件
            f.close()
```

```
>>> def print_file_content():
        myfile = '/Users/ryan/Downloads/prime.txt'
        with open(myfile) as f: #with语句保证f一定被关闭
            for line in f:
                print(line)
```

__name__

- __name__：每个py文件都具有的一个内置属性
- 如果一个文件作为顶层程序文件运行，该属性值被设置为字符串__main__，否则被设置为该文件的名字

show.py

```
1 def show():
2     print(__name__)
3
4 if __name__ == '__main__':
5     show()
```

test.py

```
1 import show
2
3 show.show()
4
5
```

```
下载 — -bash — 54×5
(base) RyandeiMac-Pro:Downloads ryan$ python show.py
__main__
(base) RyandeiMac-Pro:Downloads ryan$ python test.py
show
(base) RyandeiMac-Pro:Downloads ryan$
```