异常处理

内容

- 异常和异常处理语句
- 抛出异常的方法
- 自定义异常

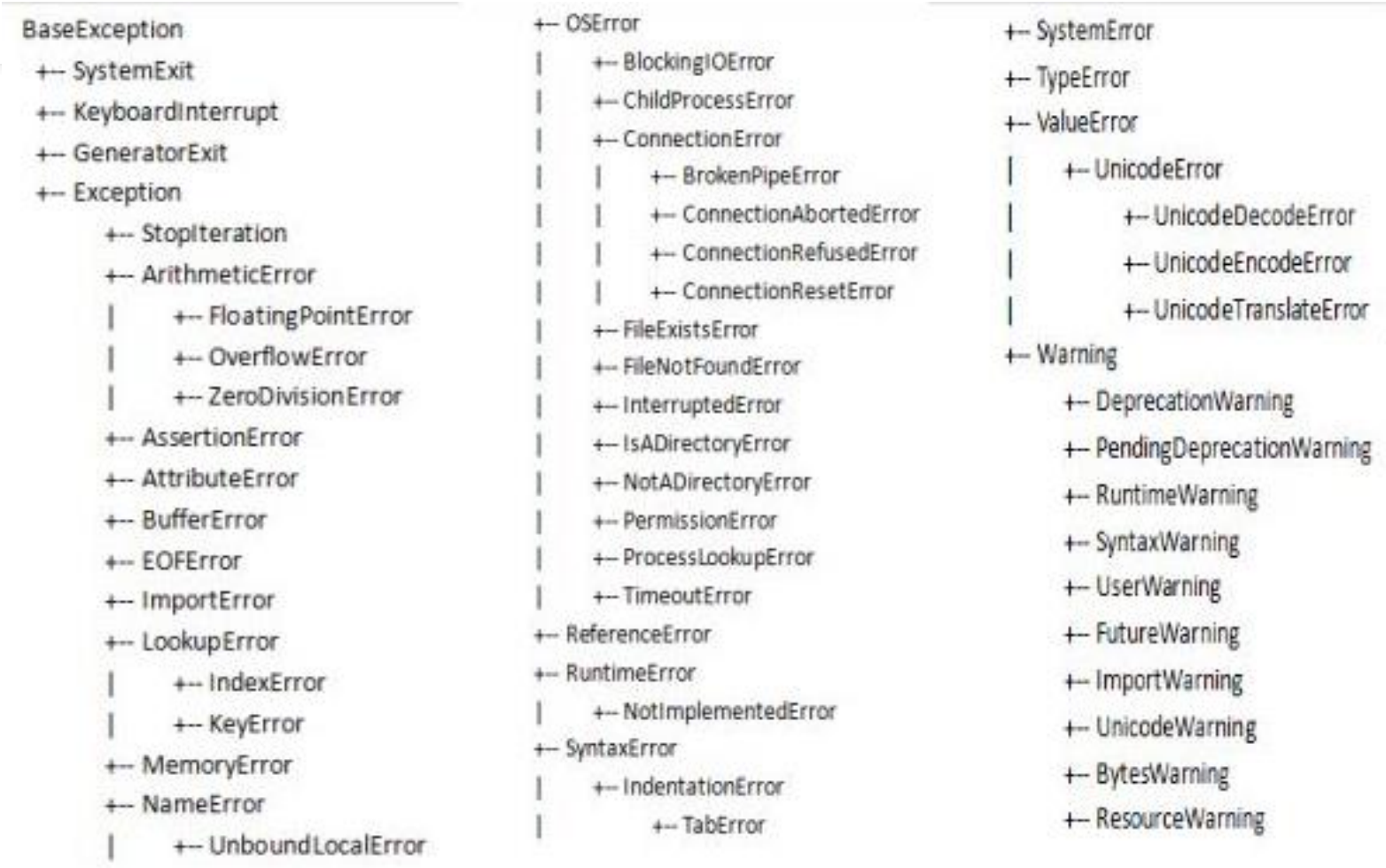
异常（Exception）

- 异常指程序执行期间发生的错误。即：程序运行发生了预计结果之外的情况，从而导致程序无法正常运行
- 异常可分为两种情况：程序遇到逻辑或算法错误；运行过程中的计算机错误

```
>>> 2 / 0
Traceback (most recent call last):
  File "<pyshell#1>", line 1, in <module>
    2 / 0
ZeroDivisionError: division by zero
```

Python中的异常类

在Python中，异常也是一种对象类型。Python中定义了大量的异常类管理异常。每当发生让出现无法继续执行的错误时，都会创建一个相应的异常对象



https://docs.python.org/3/library/exceptions.html#exception-hierarchy

常见异常类型

名称	描述
AssertionError	当assert断言条件为假的时候抛出的异常，一般用于自定义异常处理
AttributeError	当试图访问的对象属性不存在的时候抛出的异常
FileNotFoundError	未找到指定文件
IndexError	索引超出序列范围引发的异常
ImportError	无法引入模块或包，一般是路径问题
IOError	输入输出异常，基本上是无法打开文件
KeyError	字典中查到一个不存在的关键字时引发的异常
KeyboardInterrupt	Crtl-C被按下
MemoryError	当操作耗尽内存时引发的异常
NameError	尝试访问一个未生命的变量时抛出的异常
RuntimeError	运行时错误，一般是在检测到不属于任何其它类别的错误时触发
SyntaxError	解析器遇到语法错误时引发
TypeError	类型错误，通常是不同类型之间的操作会出现此异常
ValueError	参数的类型正确但是值不在指定范围之内

异常处理

- 异常处理（某些时候也称为错误处理）指因为程序执行过程中出现异常而在正常控制流之外采取的行动，提供了处理程序运行时出现的意外或异常情况的方法

- 异常处理使用 try、 except和 finally 关键字来尝试处理可能未成功的操作，并在事后清理资源

异常处理的作用

- 异常处理分离了接收和处理错误的代码
- 异常处理通常是为防止未知错误产生所采取的处理措施
- 合理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
- 异常处理结构可以为用户提供清晰的错误信息，有助于快速修复问题

尽管异常处理机制非常重要也非常有效，但不建议代替程序中常规的检查。在编程时应避免过多依赖异常处理机制提高程序的健壮性

异常处理结构：try...except...

try…except…是最常见、最基本的异常处理结构

```
try:
    try块                                  #被监控的语句，可能会引发异常
except Exception[ as reason]:              #捕获的异常类型
    except块                               #处理异常的代码
```

示例：程序无异常处理时出现异常

```
age = int(input("Please enter your age: "))
print(f"Your age is {age: d}.")
```

```
Please enter your age: abc
Traceback (most recent call last):
  File "C:/Users/DW/AppData/Local/Programs/Python/Python310/exp.ch8_1.py", line 1, in <module>
    age = int(input("Please enter your age: "))
ValueError: invalid literal for int() with base 10: 'abc'
```

异常类型

示例：针对具体异常类型的异常处理

```
try:
    age = int(input("Please enter your age: "))
    print("Your age is %d."%age)
except ValueError:
    print("It's not a number. Try again!")
```

```
Please enter your age: abc
It's not a number. Try again!
```

捕获所有类型的异常：使用BaseException

BaseException是所有Python异常类的基类

```
try:
    try块
except BaseException:                    #捕获所有类型的异常
    except块
```

捕获所有类型的异常：使用Exception

```
try:
    try块
except Exception:                    #捕获非系统类型的异常
    except块
```

捕获所有类型的异常：不使用任何Exception

```
try:
    try块
except:                                    #捕获所有类型的异常
    except块
```

不建议直接捕获所有类型的异常

- 应尽量精准捕捉可能会出现的异常类型，并且有针对性地编写代码进行处理
- 为了避免遗漏导致没有得到处理的异常干扰程序的正常运行，在捕捉了所有可能想到的异常之后，也可以使用异常处理结构的最后一个except捕捉所有类型的异常

异常处理结构：处理多个异常类型

```
try:
    try块                          #被监控的语句，可能会引发异常
except Exception1:                 #捕获异常类型1
    except块1                      #处理异常类型1的代码
except Exception2:                 #捕获异常类型2
    except块2                      #处理异常类型2的代码
……
```

处理多个异常类型示例

```
try:
    x = input("请输入被除数： ")
    y = input("请输入除数： ")
    z = float(x) / float(y)
    print(x,'÷',y,'=',round(z, 2))
except ZeroDivisionError:
    print("除数不能为0")
except ValueError:
    print("被除数和除数应该为数值类型")
```

```
请输入被除数： 13.5
请输入除数： 2.6
13.5 ÷ 2.6 = 5.19
```

```
请输入被除数： x
请输入除数： 3
被除数和除数应该为数值类型
```

```
请输入被除数： 12
请输入除数： 0
除数不能为0
```

处理多个异常类型示例

```
try:
    x = input("请输入被除数：")
    y = input("请输入除数：")
    z = float(x) / float(y)
    print(x,'÷',y,'=',round(z, 2))
except ZeroDivisionError:
    print("除数不能为0")
except ValueError:
    print("被除数和除数应该为数值类型")
except Exception as err: # as err是可选功能，err为错误原因
    print("其他类型错误：")
    print(err)
```

捕获“未知”异常类型，也可使用BaseException

处理多个异常类型的另外一种写法

```
try:
    x = input("请输入被除数：")
    y = input("请输入除数：")
    z = float(x)/float(y)
    print(x,'÷',y,'=',round(z,2))
except (ZeroDivisionError, ValueError) as err:
    print("出错啦：")
    print(err)
```

```
请输入被除数：13.5
请输入除数：a
出错啦：
could not convert string to float: 'a'
```

```
请输入被除数：12
请输入除数：0
出错啦：
float division by zero
```

异常处理结构：try...except...else...

```
try:
    try块                                    #被监控的语句，可能会引发异常
except Exception[ as reason]:    #可以有多个except语句
    except块
else:                                        #若未抛出异常则执行
    else块
```

带有else的异常处理结构

```
try:
    x = input("请输入被除数： ")
    y = input("请输入除数： ")
    z = float(x) / float(y)
except ZeroDivisionError:
    print("除数不能为0")
except ValueError:
    print("被除数和除数应该为数值类型")
else:
    print(x,'÷',y,'=',round(z, 2))
```

```
请输入被除数： 15
请输入除数： 4
15 ÷ 4 = 3.75
```

```
请输入被除数： 4.5
请输入除数： 0
除数不能为0
```

```
请输入被除数： pi
请输入除数： 2
被除数和除数应该为数值类型
```

异常处理结构：try...except...finally...

```
try:
    try块                                    #被监控的语句，可能会引发异常
except Exception[ as reason]:    #可以有多个except语句
    except块
[else:…]                                    #可以有else语句
finally:                                    #无论是否出现异常都会执行
    finally块
```

带有finally的异常处理结构

```
try:
    x = input("请输入被除数：")
    y = input("请输入除数：")
    z = float(x) / float(y)
except ZeroDivisionError:
    print("除数不能为0")
except ValueError:
    print("被除数和除数应该为数值类型")
else:
    print(x,'÷',y,'=',round(z, 2))
finally:
    print("执行finally操作")
```

```
请输入被除数：22
请输入除数：6.8
22 ÷ 6.8 = 3.24
执行finally操作
```

```
请输入被除数：4.3
请输入除数：0
除数不能为0
执行finally操作
```

带有finally的异常处理结构：出现未捕获异常

```
#example907.py
try:
    x = input("请输入被除数：")
    y = input("请输入除数：")
    z = float(x)/float(y)
except ZeroDivisionError:
    print("除数不能为0")
else:
    print(x,'÷',y,'=',round(z,2))
finally:
    print("执行finally操作")
```

```
请输入被除数：pi
请输入除数：3
执行finally操作
Traceback (most recent call last):
  File "C:/Users/DW/AppData/Local/Programs/Python/Python310/exp.ch8_7.py",
line 4, in <module>
    z = float(x) / float(y)
ValueError: could not convert string to float: 'pi'
```

异常处理结构：finally块中尽量不要使用return

```
def demo_div(x, y):
    try:
        return x / y
    except:
        pass
    finally:
        return -1
```

```
>>> demo_div(2, 5)
-1
>>> demo_div(2, 0)
-1
```

特殊的异常处理结构：静默失败

```
try:
    try块
except Exception1:
    except块1
……
except Exception:
    pass
……
```

出现指定异常类型时，什么都不会发生，不会出现异常退出traceback，也没有任何输出

什么都不做

内容

- 异常和异常处理语句
- 抛出异常的方法
- 自定义异常

主动抛出异常

- 系统默认的异常类型总是有限的，如果当程序运行时，产生的特殊数据并不在默认的异常类型之内时，就可以选择主动抛出异常，以便程序进行后续的异常捕捉处理

```
raise [Exception]

#Exception应为已定义的异常类型，如不写则抛出RunTimeError
```

主动抛出异常

例如当输入年龄时，输入的数据应该在0至120岁之间。如果输入是数字但不在此范围内，系统自带的异常类型是无法处理的，因为只要是数字都会认为是正确的输入。如果你的程序需要处理这种异常的输入，就可以主动抛出异常

```
#example0809.py
age = int(input("Please enter your age: "))
if 0 <= age <= 120:
    print(f"Your age is: {age: d}.")
else:
    raise ValueError
```

```
Please enter your age: 23
Your age is:  23.
```

```
Please enter your age: 130
Traceback (most recent call last):
  File "C:/Users/DW/AppData/Local/Programs/Python/Python310
/example0809.py", line 7, in <module>
    raise ValueError
ValueError
```

主动抛出异常并进行异常处理

```
#example0810.py
try:
    age = int(input("Please enter your age: "))
    if 0 <= age <= 120:
        print(f"Your age is: {age: d}.")
    else:
        raise ValueError(f"{age: d} is not a valid age.")
except ValueError as err:
    print(f"You entered incorrect age. {err}")
```

```
Please enter your age: 130
You entered incorrect age.  130 is not a valid age.
```

异常提示

单独的raise：重新抛出异常

```
#example0811.py
try:
    1/0
except:
    print("错误")
```

```
错误
```

```
#example0812.py
try:
    1/0
except:
    print("错误")
    raise
```

```
错误
Traceback (most recent call last):
  File "C:/Users/DW/AppData/Local/Programs/Python/Python310/example0812.py",
line 3, in <module>
    1/0
ZeroDivisionError: division by zero
```

内容

- 异常和异常处理语句
- 抛出异常的方法
- 自定义异常

自定义异常类型

自定义异常类型的父类是Exception

```
class MyOwnException(Exception):
    pass
```

对于自定义异常类型，在进行异常处理时需要通过raise抛出异常

```
try:
    ……
    raise(MyOwnException)
exception MyOwnException:
    ……
```

```
#example0813.py
class MyOwnException(Exception):
    def __init__(self, length, minlen=6):
        Exception.__init__(self)
        self.length = length
        self.minlen = minlen
while(True):
    try:
        str = input("Please enter a string: ")
        if len(str)<6:
            raise MyOwnException(len(str))
    except MyOwnException as err:
        print(f"MyOwnException: Your string length is {err.length: d}.")
        print(f"Please input at least {err.minlen: d}.")
    else:
        print("No exception!")
        break
```

```
Please enter a string: ab
MyOwnException: Your string length is  2.
Please input at least  6.
Please enter a string: hello world
No exception!
```

异常处理：总结

- 异常处理结构可以提高程序的容错性和健壮性，但不建议过多依赖异常处理结构
- 异常处理结构中主要的关键字包括try、except、else和finally
- 可以使用BaseException或Exception捕获所有异常
- 可以继承Python内建的异常类实现自定义的异常类