Day 3

Mittwoch, 9. September 2020 13:42

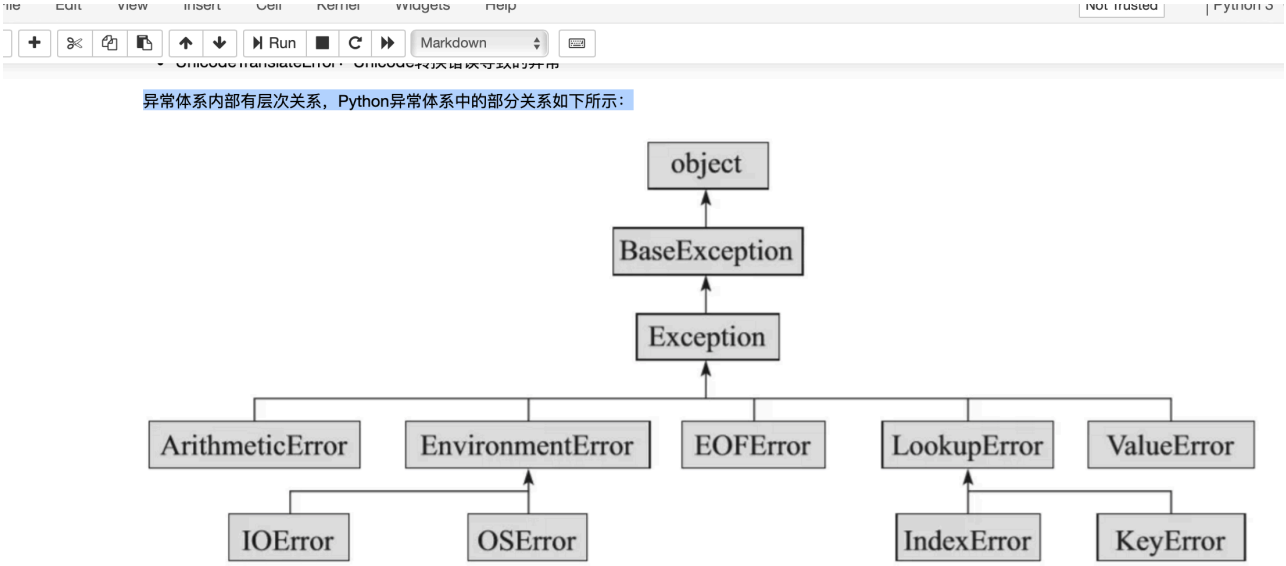
异常处理

1. Python 标准异常总结

- BaseException：所有异常的 **基类**
- Exception：常规异常的 **基类**
- StandardError：所有的内建标准异常的基类
- ArithmeticError：所有数值计算异常的基类
- FloatingPointError：浮点计算异常
- OverflowError：数值运算超出最大限制
- ZeroDivisionError：除数为零
- AssertionError：断言语句（assert）失败
- AttributeError：尝试访问未知的对象属性
- EOFError：没有内建输入，到达EOF标记
- EnvironmentError：操作系统异常的基类
- IOError：输入/输出操作失败
- OSError：操作系统产生的异常（例如打开一个不存在的文件）
- WindowsError：系统调用失败
- ImportError：导入模块失败的时候
- KeyboardInterrupt：用户中断执行
- LookupError：无效数据查询的基类
- IndexError：索引超出序列的范围
- KeyError：字典中查找一个不存在的关键字
- MemoryError：内存溢出（可通过删除对象释放内存）
- NameError：尝试访问一个不存在的变量
- UnboundLocalError：访问未初始化的本地变量
- ReferenceError：弱引用试图访问已经垃圾回收了的对象
- RuntimeError：一般的运行时异常
- NotImplementedError：尚未实现的方法
- SyntaxError：语法错误导致的异常
- IndentationError：缩进错误导致的异常
- TabError：Tab和空格混用
- SystemError：一般的解释器系统异常
- TypeError：不同类型间的无效操作
- ValueError：传入无效的参数
- UnicodeError：Unicode相关的异常

- UnicodeDecodeError：Unicode解码时的异常
- UnicodeEncodeError：Unicode编码错误导致的异常
- UnicodeTranslateError：Unicode转换错误导致的异常

异常体系内部有层次关系，Python异常体系中的部分关系如下所示：



2. Python标准警告总结

- Warning：警告的基类
- DeprecationWarning：关于被弃用的特征的警告
- FutureWarning：关于构造将来语义会有改变的警告
- UserWarning：用户代码生成的警告
- PendingDeprecationWarning：关于特性将会被废弃的警告
- RuntimeWarning：可疑的运行时行为(runtime behavior)的警告
- SyntaxWarning：可疑语法的警告
- ImportWarning：用于在导入模块过程中触发的警告
- UnicodeWarning：与Unicode相关的警告
- BytesWarning：与字节或字节码相关的警告
- ResourceWarning：与资源使用相关的警告

3. try - except 语句

try:
    检测范围
except Exception[as reason]:
    出现异常后的处理代码

try 语句按照如下方式工作：

- 首先，执行try子句（在关键字try和关键字except之间的语句）
- 如果没有异常发生，忽略except子句，try子句执行后结束。
- 如果在执行try子句的过程中发生了异常，那么try子句余下的部分将被忽略。如果异常的类型和except之后的名称相符，那么对应的except子句将被执行。最后执行try - except语句之后的代码。
- 如果一个异常没有与任何的except匹配，那么这个异常将会传递给上层的try中。

【例子】

In [2]:
```
try:
    f = open('test.txt')
    print(f.read())
    f.close()
except OSError:
    print('打开文件出错')

# 打开文件出错
```

```
打开文件出错
```

【例子】

In [3]:
```
try:
    f = open('test.txt')
    print(f.read())
    f.close()
except OSError as error:
    print('打开文件出错\n原因是: ' + str(error))

# 打开文件出错
# 原因是: [Errno 2] No such file or directory: 'test.txt'
```

```
打开文件出错
原因是: [Errno 2] No such file or directory: 'test.txt'
```

一个try语句可能包含多个except子句，分别来处理不同的特定的异常。最多只有一个分支会被执行。

In [4]:
```
try:
    int("abc")
    s = 1 + '1'
    f = open('test.txt')
    print(f.read())
    f.close()
except OSError as error:
    print('打开文件出错\n原因是: ' + str(error))
except TypeError as error:
    print('类型出错\n原因是: ' + str(error))
except ValueError as error:
    print('数值出错\n原因是: ' + str(error))

# 数值出错
# 原因是: invalid literal for int() with base 10: 'abc'
```

```
数值出错
原因是: invalid literal for int() with base 10: 'abc'
```

【例子】

```
In [5]: dict1 = {'a': 1, 'b': 2, 'v': 22}
try:
    x = dict1['y']
except LookupError:
    print('查询错误')
except KeyError:
    print('键错误')
else:
    print(x)

# 查询错误
```

查询错误

try-except-else语句尝试查询不在dict中的键值对，从而引发了异常。这一异常准确地说应属于KeyError，但由于KeyError是LookupError的子类，且将LookupError置于KeyError之前，因此程序优先执行该except代码块。所以，使用多个except代码块时，必须坚持对其规范排序，要从最具针对性的异常到最通用的异常。

```
In [6]: dict1 = {'a': 1, 'b': 2, 'v': 22}
try:
    x = dict1['y']
except KeyError:
    print('键错误')
except LookupError:
    print('查询错误')
else:
    print(x)

# 键错误
```

键错误

【例子】一个 except 子句可以同时处理多个异常，这些异常将被放在一个括号里成为一个元组。

```
In [7]: try:
    s = 1 + '1'
    int("abc")
    f = open('test.txt')
    print(f.read())
    f.close()
except (OSError, TypeError, ValueError) as error:
    print('出错了! \n原因是: ' + str(error))

# 出错了!
# 原因是: unsupported operand type(s) for +: 'int' and 'str'
```

```
出错了!
原因是: unsupported operand type(s) for +: 'int' and 'str'
```

4. try - except - finally 语句

try: 检测范围 except Exception[as reason]: 出现异常后的处理代码 finally: 无论如何都会被执行的代码

不管try子句里面有没有发生异常，finally子句都会执行。

【例子】如果一个异常在try子句里被抛出，而又没有任何的except把它截住，那么这个异常会在finally子句执行后被抛出。

```
In [10]: def divide(x, y):
```

```
    try:
        result = x / y
        print("result is", result)
    except ZeroDivisionError:
        print("division by zero!")
    finally:
        print("executing finally clause")


divide(2, 1)
# result is 2.0
# executing finally clause
divide(2, 0)
# division by zero!
# executing finally clause
divide("2", "1")
# executing finally clause
# TypeError: unsupported operand type(s) for /: 'str' and 'str'
```

```
result is 2.0
executing finally clause
division by zero!
executing finally clause
```

5. try - except - else 语句

如果在try子句执行时没有发生异常，Python将执行else语句后的语句。

try:

检测范围

except:

出现异常后的处理代码

else:

如果没有异常执行这块代码

使用except而不带任何异常类型，这不是一个很好的方式，我们不能通过该程序识别出具体的异常信息，因为它捕获所有的异常。
try: 检测范围 except(Exception1[, Exception2[,...ExceptionN]]]): 发生以上多个异常中的一个，执行这块代码 else: 如果没有异常执行这块代码
【例子】

In [12]:
```
try:
    fh = open("testfile.txt", "w")
    fh.write("这是一个测试文件，用于测试异常!!")
except IOError:
    print("Error: 没有找到文件或读取文件失败")
else:
    print("内容写入文件成功")
    fh.close()

# 内容写入文件成功
```

```
内容写入文件成功
```

注意： `else` 语句的存在必须以 `except` 语句的存在为前提，在没有 `except` 语句的 `try` 语句中使用 `else` 语句，会引发语法错误。

6. raise语句

Python 使用raise语句抛出一个指定的异常。

In [13]:

```
try:
    raise NameError('HiThere')
except NameError:
    print('An exception flew by!')

# An exception flew by!
```

```
An exception flew by!
```