马哥教育

主讲：马永亮(马哥)
QQ:113228115
客服QQ：2813150558，1661815153
http://www.magedu.com
http://mageedu.blog.51cto.com

马哥教育

Python异常

主讲：马永亮(马哥)
QQ:113228115
客服QQ：2813150558，1661815153
http://www.magedu.com
http://mageedu.blog.51cto.com

Python异常

- Python的运行时错误称作异常
  - 语法错误：软件的结构上有错误而导致不能被解释器解释或不能被编译器编译
  - 逻辑错误：由于不完整或不合法的输入所致，也可能是逻辑无法生成、计算或者输出结果需要的过程无法执行等

```
>>> f1 = open('/tmp/a.txt')

Traceback (most recent call last):
  File "<pyshell#1>", line 1, in <module>
    f1 = open('/tmp/a.txt')
IOError: [Errno 2] No such file or directory: '/tmp/a.txt'
```


Python异常

- Python异常是一个对象，表示错误或意外情况
- 在Python检测到一个错误时，将触发一个异常
  - Python可以通过异常传导机制传递一个异常对象，发出一个异常情况出现的信号
  - 程序员也可以在代码中手动触发异常

- Python异常也可以理解为：程序出现了错误而在正常控制流以外采取的行为
  - 第一阶段：解释器触发异常，此时当前程序流将被打断
  - 第二阶段：异常处理，如忽略非致命性错误、减轻错误带来的影响等

异常的功用

- 错误处理
  - Python的默认处理：停止程序，打印错误消息
  - 使用try语句处理异常并从异常中恢复
- 事件通知
  - 用于发出有效状态信号
- 特殊情况处理
  - 无法调整代码去处理的场景
- 终止行为
  - try/finally语句可确保执行必需的结束处理机制
- 非常规控制流程
  - 异常是一种高级跳转(goto)机制

检测和处理异常

- 异常通过try语句来检测
  - 任何在try语句块里的代码都会被监测，以检查有无异常发生
- try语句主要有两种形式
  - try-except：检测和处理异常
    - 可以有多个except
    - 支持使用else子句处理没有探测异常的执行的代码
  - try-finally：仅检查异常并做一些必要的清理工作
    - 仅能有一个finally
- try语句的复合形式
  - try-except-finally

try-except语句

- 定义了进行异常监控的一段代码，并且提供了处理异常的机制
- 语法
  - try:
        try_suite
    except Exception[, reason]:
        except_suite
  - 例子

```
>>> try:
...     f1 = open('/tmp/a.txt','r')
... except IOError, e:
...     print 'Could not open file:', e
...
Could not open file: [Errno 2] No such file or directory: '/tmp/a.txt'
```

try-except-else语句

- try语句可以带多个except子句，还可以有一个可选的else子句，语法格式如下
  - try:
    try_suite
    except Exception1[, reason]:
    suite_exception1
    except (Exception2, Exception3, ...)[,reason]:
    suite_
    ...
    except:
    suite_
    else:
    else_suite

一次捕获多个异常时要定义为元组

空except语句用于捕获一切异常

  - except分句个数没有限制，但else只能有一个
  - 没有异常发生时，else分句才会执行
  - 没有符合的except分句时，异常会向上传递到程序中的之前进入的try中或者到进程的顶层

try-finally语句

- 无论异常是否发生，finally子句都会执行
  - 常用于定义必需进行的清理动作，如关闭文件或断开服务器连接等
- finally中的所有代码执行完毕后会继续向上一层引发异常
- 语法
  - try:
    try_suite
    finally:
    finally_suite

例子：

```
>>> try:
...     f1 = open('/tmp/a.txt','r')
...     f1.write('hello world')
... finally:
...     f1.close()
...
Traceback (most recent call last):
  File "<stdin>", line 3, in <module>
IOError: File not open for writing
```

关闭文件后继续向上触发异常

总结：try语句的分句形式

分句形式	说明
except:	捕捉所有（其他）异常类型
except *name*:	只捕捉特定的异常
except *name*, *value*:	捕捉所列的异常和其额外的数据（或实例）
except (*name1*, *name2*):	捕捉任何列出的异常
except (*name1*, *name2*), *value*:	捕捉任何列出的异常，并取得其额外数据
else:	如果没有引发异常，就运行
finally:	总是会运行此代码块

Reference: 《Python学习手册（第4版）》

try-except-else-finally语句

- 语法：
  - ```
    try:
        try_suite
    except Exception1:
        suite1_exception1
    except (Exception2, Exception3):
        suite23_exception23
    …
    else:
        else_suite
    finally:
        finally_suite
    ```
  - 可以替换为在try-finally语句中嵌套try-except语句的形式


    ```

自定义异常

- raise语句可显式触发异常
  - raise [SomeException [, args [, traceback]]]
    - SomeException：可选，异常的名字，仅能使用字符串、类或实例
    - args：可选，以元组的形式传递给异常的参数
    - traceback：可选，异常触发时新生成的一个用于异常-正常化的跟踪记录，多用于重新引发异常时

```
>>> def CrossProduct(seq1, seq2):
        if not seq1 or not seq2:
                raise ValueError, "Sequence argements must be non-empty"
        return [(x1, x2) for x1 in seq1 for x2 in seq2]

>>> seq1=[]
>>> seq2=[]
>>> CrossProduct(seq1, seq2)

Traceback (most recent call last):
  File "<pyshell#25>", line 1, in <module>
    CrossProduct(seq1, seq2)
  File "<pyshell#22>", line 3, in CrossProduct
    raise ValueError, "Sequence argements must be non-empty"
ValueError: Sequence argements must be non-empty
```

raise语句的用法大全

rasie 语法	描　　述
raise *exclass*	触发一个异常，从 exclass 生成一个实例（不含任何异常参数）
raise *exclass*()	同上，但现在不是类；通过函数调用操作符（function calloperator：“()”）作用于类名生成一个新的 exclass 实例，同样也没有异常参数
raise *exclass*，*args*	同上，但同时提供的异常参数 args，可以是一个参数也可以是元组
raise *exclass*(*args*)	同上
raise *exclass*，*args*，*tb*	同上，但提供一个跟踪记录（traceback）对象 tb 供使用
raise *exclass*，*instance*	通过实例触发异常（通常是 exclass 的实例）；如果实例是 exclass 的子类实例，那么这个新异常的类型会是子类的类型（而不是 exclass）；如果实例既不是 exclass 的实例也不是 exclass 子类的实例，那么会复制此实例为异常参数去生成一个新的 exclass 实例
raise *instance*	通过实例触发异常：异常类型是实例的类型；等价于 raise instance.__class__，instance（同上）
raise *string*	（过时的）触发字符串异常
raise *string*，*args*	同上，但触发伴随着 args
raise *string*，*args*，*tb*	同上，但提供了一个跟踪记录（traceback）对象 tb 供使用
raise	（1.5 新增）重新触发前一个异常，如果之前没有异常，触发 TypeError

异常对象

- Python异常是内置的经典类Exception的子类的实例
  - 为了向后兼容，Python还允许使用字符串或任何经典类实例
  - Python2.5之后，Exception是从BaseException继承的新式类
- Python自身引发的所有异常都是Exception的子类的实例
- 大多的标准异常都是由StandardError派生的，其有3个抽象的子类
  - ArithmeticError
    - 由于算术错误而引发的异常基类
    - OverflowError，ZeroDivisionError，FloatingPointError
  - LookupError
    - 容器在接收到一个无效键或索引时引发的异常的基类
    - IndexError，KeyError
  - EnvironmentError
    - 由于外部原因而导致的异常的基类
    - IOError，OSError，WindowsError

标准异常类

- AssertionError
  - 断言语句失败
- AttributeError
  - 属性引用或赋值失效
- FloatingPointError
  - 浮点型运算失败
- IOError
  - I/O操作失败
- ImportError
  - import语句不能找到要导入的模块，或者不能找到该模块特别请求的名称
- IndentationError
  - 解析器遇到了一个由于错误的缩进而引发的语法错误
- IndexError
  - 用来索引序列的整数超出了范围

标准异常类

- KeyError
  - 用来索引映射的键不在映射中
- KeyboardInterrupt
  - 用户按了中断键(Ctrl+c，Ctrl+Break或Delete键)
- MemoryError
  - 运算耗尽内存
- NameError
  - 引用了一个不存在的变量名
- NotImplementedError
  - 由抽象基类引发的异常，用于指示一个具体的子类必须覆盖一个方法
- OSError
  - 由模块os中的函数引发的异常，用来指示平台相关的错误
- OverflowError
  - 整数运算的结果太大导致溢出

标准异常类

- SyntaxError
  - 语法错误
- SystemError
  - Python本身或某些扩展模块中的内部错误
- TypeError
  - 对某对象执行了不支持的操作
- UnboundLocalError
  - 引用未绑定值的本地变量
- UnicodeError
  - 在Unicode的字符串之间进行转换时发生的错误
- ValueError
  - 应用于某个对象的操作或函数，这个对象具有正确的类型，但确有不适当的值
- WindowsError
  - 模块os中的函数引发的异常，用来指示与Windows相关的错误
- ZeroDivisionError
  - 除数为0

自定义异常类

- 自定义异常和多重继承
  - 较有效的方法是从自定义异常类和标准异常类进行多重继承，例如
    - class CustomAttributeError(CustomException, AttributeError):
      pass
- 标准库中使用的其它异常
  - Python标准库中的许多模块都定义了自己的异常类，如socket中的socket.error
    - 等同于自定义的异常类

assert语句

- assert语句用于在程序中引入调试代码
  - assert *condition*[, *expression*]
    - 如果condition条件满足，则assert不做任何操作
    - 如果condition条件不满足，则assert使用expression作为参数实例化AssertionError并引发结果实例
- 注意：如果运行Python时使用了-O优化选项，则assert将是一个空操作：编译器不为assert语句生成代码
  - 运行Python时不使用-O选项，则__debug__内置变量为True，否则其值为False
- assert语句相当于下面的代码
  - if __debug__:

```
if not condition:
    raise AssertionError, <expression>
```


with-as语句

关于马哥教育

- 博客：http://mageedu.blog.51cto.com
- 主页：http://www.magedu.com
- QQ：2813150558，1661815153，113228115
- QQ群：203585050，279599283

马哥教育

Thank You!