老男孩教育
oldboyedu.com

内置方法，异常机制

讲师：尚玉杰

老男孩教育
oldboyedu.com

本章目录

- 内置方法
- 异常机制

魔法方法

- Python 的对象天生拥有一些神奇的方法，它们总被双下划线所包围

- 如果你的对象重写了这些方法中的某一个，那么这个方法就会在特殊的情况下被 Python 所调用，你可以定义自己想要的行为，而这一切都是自动发生的

内置方法

- __new__(cls[, ...])
  - __new__ 是在一个对象实例化的时候所调用的第一个方法
- __init__(self[, ...])
  - 构造器，当一个实例被创建的时候调用的初始化方法
- __del__(self)
  - 析构器，当一个实例被销毁的时候调用的方法（建议不要重写）

内置方法

- __len__
  - 通过len(obj)调用，返回值必须设置为int类型
- __hash__ 通过hash(obj)调用（一种消息摘要算法）
  - hash() 用于获取取一个对象（非可变数据类型）的哈希值
- __str__
  - 打印对象时调用

内置方法

- __eq__
  - 比较两个对象

```
def __eq__(self,obj):
    if  self.a == obj.a and self.b == obj.b:
        return True
```

异常处理

- 异常（Exception）是一个事件，该事件可能会在程序执行过程中发生，影响了程序的正常执行
  - raise语句显式的抛出异常
  - Python解释器自己检测到异常并引发它
- 一般情况下，在Python无法正常处理程序时就会发生异常
- 异常是Python对象，表示一个错误
- 当Python程序发生异常时我们需要捕获处理它，否则程序会终止执行

异常处理

- 常见异常：
  - SyntaxError语法错误
  - IndentationError多打一个空格
  - NameError 使用一个还未被赋予对象的变量
  - TypeError 传入对象类型与要求的不符合 （int + str）
  - IOError 输入/输出异常；基本上是无法打开文件
  - ImportError 无法引入模块或包；基本上是路径问题或名称错误
  - IndexError 下标索引超出序列边界
  - KeyError 试图访问字典里不存在的键

异常处理

- 处理异常
  - 程序员编写特定的代码，专门用来捕捉这个异常（这段代码与程序逻辑无关，与异常处理有关）如果捕捉成功则进入另外一个处理分支，执行你为其定制的逻辑，使程序不会崩溃

  - 异常处理的方式1

```
num1=input('>>: ') #输入一个字符串试试
if num1.isdigit():
    int(num1) #我们的正统程序放到了这里,其余的都属于异常处理范畴
elif num1.isspace():
    print('输入的是空格,就执行我这里的逻辑')
elif len(num1) == 0:
    print('输入的是空,就执行我这里的逻辑')
else:
    print('其他情情况,执行我这里的逻辑')
```

异常处理

- 上页的处理方式存在问题
  - 使用if的方式我们只为第一段代码加上了异常处理，但这些if，跟你的代码逻辑并无关系，这样你的代码会因为可读性差而不容易被看懂
  - 这只是我们代码中的一个小逻辑，如果类似的逻辑多，那么每一次都需要判断这些内容，就会让我们的代码特别冗长
  - if判断式的异常处理只能针对某一段代码，对于不同的代码段的相同类型的错误你需要写重复的if来进行处理

异常处理

```
try:
    被检测的代码块
except 异常类型:
    try中一旦检测到异常，就执行这个位置的逻辑
```

异常处理

- 如果你不想在异常发生时结束你的程序，只需在try里捕获它
- 首先，执行 try 子句 （在 try 和 except 关键字之间的部分）
- 如果没有异常发生，except 子句在 try 语句执行完毕后就被忽略了
- 如果在 try 子句执行过程中发生了异常，那么该子句其余的部分就会被忽略
- 如果异常匹配于 except 关键字后面指定的异常类型，就执行对应的 except 子句。然后继续执行 try 语句之后的代码

异常处理

```
try:
    print(1/0)
except ZeroDivisionError:
    print("0不能作为除数")

print("后续语句")
#注意:如果写明异常类型,异常类型要与上面发生的异常匹配,否则捕捉不到
```

异常处理

- 多种捕获（多分支）
  - except可以指定捕获的类型,捕获多种异常
  - 多个except即可,但是之后最多匹配一个异常
  - 如果没有任何一个except语句捕获这异常,则该异常向外抛出

```
try:
    print(1/0)
    raise FileNotFoundError
except ZeroDivisionError:
    print("0不能作为除数")
except FileNotFoundError:
    print("异常 2")

print("后续语句")
```

异常处理

- except ... as ...
  - 使用exception as e:查看异常，以及是否按要求捕获到

```
try:
    print(1/0)
    raise FileNotFoundError
except Exception as e:    #万能异常
    print(e)
```

异常处理

- 如果想要的效果是，无论出现什么异常，我们统一丢弃，或者使用同一段代码逻辑去处理他们，只有一个Exception就足够了

- 如果想要的效果是，对于不同的异常定制不同的处理逻辑，需要用到多分支

异常处理

- 多分支+万能异常
  - 发生的异常中，有一些异常是需要不同的逻辑处理的，剩下的异常统一处理掉即可

```
dic = {1: login,2: register,3: dariy,4: article,5: comment,}
print('''欢迎访问博客园系统：
    1，登录
    2，注册
    3，访问日记页面
    4，访问文章页面
    5，访问评论页面''')
try:
    choice = int(input('请输入：'))
    dic[choice]()
except ValueError:
    print('请输入数字....')
except KeyError:
    print('您输入的选项超出范围...')
except Exception as e:
    print(e)
```

异常处理

- try...except...else组合
  - 与循环中的else类似，try代码中，只要出现了异常，则不会执行else语句，如果不出现异常，则执行else语句
  - 伪代码：

```
try:
    print('扣第一个人钱')
    print('给第二个人加钱')
except ValueError:
    print('必须输入数字。。。')
else:
    print('转账成功')
```

异常处理

- try...excet...finally组合:
  - finally:最后一定要执行的语句块

```
try:
    dic = {'name': 'shang'}
    print(dic[1])
except KeyError:
    print('出现了keyError错误....')
finally:
    print('正常执行')
```

异常处理

- 如果出现异常但是没有成功捕获，finally会在异常发生之前执行

```
try:
    dic = {'name': 'shang'}
    print(dic[1])
except NameError:
    print('出现了NameError错误....')
finally:
    print('执行结束')
```

异常处理

- finally应用场景
- 1. 关闭文件的链接链接，数据等链接时，需要用到finally

```
f = open('file',encoding='utf-8')
try:
    '''各种操作'''
    print(f.read())
    '''但是发生错误了，此时没关闭文件句柄，所以'''
finally:
    f.close()
```

异常处理

- 函数中，finally也会在return之前先执行

```
def func():
    try:
        return 1
    finally:
        print('finally')
func()
```

异常处理

- 循环中

```
while 1:
    try:
        break
    finally:
        print('finally')
```

  - 总结：finally一般是做收尾工作
    - 在一些重要环节出错之前必须一定要做的比如关闭链接的问题时，最好是用上finally作为最后一道防线，收尾。

异常处理

- 主动发出异常
  - 在类的约束中，我们已经用过此方法
  - raise TypeError('类型错误')

- 断言
  - 表示一种强硬的态度，只要assert后面的代码不成立，直接报错，下面的代码就不让你执行
  - assert 条件
  - 代码
  - 代码
  - ..... （类似if）

异常处理

- 自定义异常：
  - python中提供的错误类型可能并不能解决所有错误
  - 如果以后你在工作中，出现了某种异常无法用已知的错误类型捕获（万能异常只能捕获python中存在的异常），那么你就可以尝试自定义异常，只要继承BaseException类即可

异常处理

- 自定义异常：

```
class HelloError(Exception):
    def __init__(self,n):
        self.n=n
try:
    n=input("请输入数字：")
    if not n.isdigit():
        raise HelloError(n)
except HelloError as hi:
    print("HelloError：请输入字符。\n您输入的是：",hi.n)
else:
    print("未发生异常")
```