Python 程序设计

Xia Tian
Email: xiat(at)ruc.edu.cn

Renmin University of China

May 25, 2016

CH8 异常、调试与测试

- 异常
  - Python 的异常处理使用方法
  - Python 的异常继承关系
  - 利用 raise 抛出异常
- 调试
  - print 调试法
  - logging 调试法
  - assert
  - pdb
- 单元测试

异常处理

- 程序在编写过程中，有大量情况需要考虑
  - 除数是否为 0
  - 打开文件时，需要判断文件是否存在，有无权限
  - ...
- 异常可以简化这一处理过程
- Python 的错误处理机制
  - try...except...finally...

try

```
1  try:
2      print('try...')
3      r = 10 / 0
4      print('result:', r)
5  except ZeroDivisionError as e:
6      print('except:', e)
7  finally:
8      print('finally...')
9  print('END')
10
```

try...
except: division by zero
finally...
END

try

```
1  try:
2      print('try...')
3      r = 10 / 2
4      print('result:', r)
5  except ZeroDivisionError as e:
6      print('except:', e)
7  finally:
8      print('finally...')
9  print('END')
10
```

try...
result: 5.0
finally...
END

Python 异常处理的规则

- 遇到第一个满足条件的异常，执行该异常下的语句，忽略后续的其它异常
- finally 永远会被执行

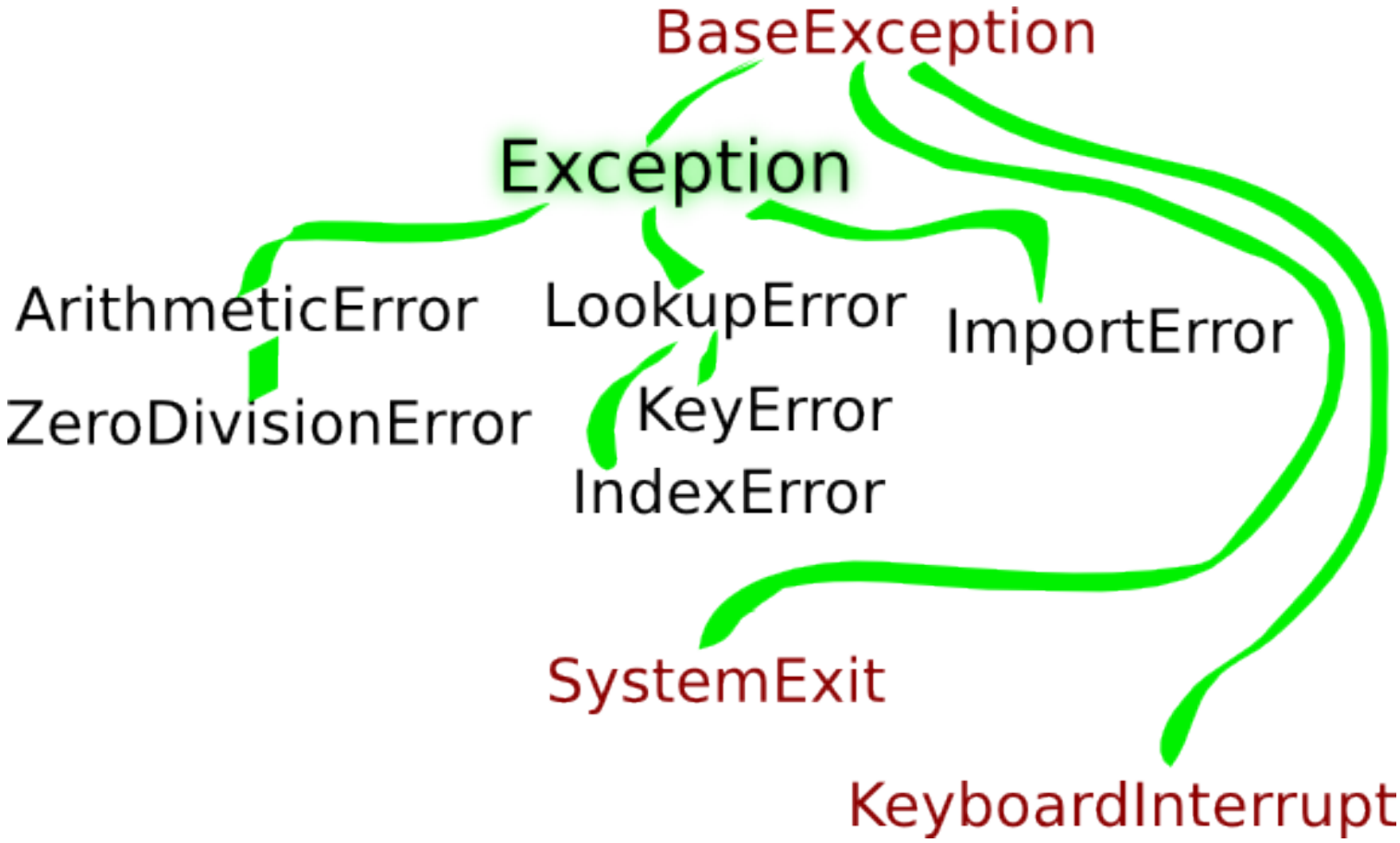
Python 异常的继承关系

- Python 的错误也是 class，都继承自 BaseException
  - 在使用 except 时需要注意的是，它不但捕获该类型的错误，还把其子类也“一网打尽”。

```
lst = [x for x in range(10)]
try:
    n = lst[15]
except LookupError as e:
    print('LookupError ', e)
except IndexError as e:
    print('IndexError ', e)

```

Python 异常继承关系示例



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

抛出异常

- 可以用 raise 语句来引发一个异常。异常/错误对象必须有一个名字，且它们应是 Error 或 Exception 类的子类。

raise 示例 I

```
1  class Student(object):
2      @property
3      def score(self):
4          return self._score
5
6      @score.setter
7      def score(self, value):
8          if not isinstance(value, int):
9              raise ValueError('score必须是整数!')
10         if value < 0 or value > 100:
11             raise ValueException('score必须在0到100之间')
12         self._score = value
```

raise 示例 II

```
1 s = Student()
2 s.score = 60
3 print(s.score)
4 s.score = 999 # Error!
```

异常可以自定义 I

- 例如：把以上的 score 判断条件不满足时，抛出的异常更改为自定义异常

```
1  class ScoreException(Exception):
2      def ___init___(self, msg):
3          self.msg = msg
4
5      def ___str___(self):
6          return 'ScoreException: ' + repr(self.msg)
7
8  class Student(object):
9      @property
10     def score(self):
11         return self.__score
12
```

异常可以自定义 II

```
13      @score.setter
14      def score(self, value):
15          if not isinstance(value, int):
16              raise ScoreException('score必须是整数!')
17          if value < 0 or value > 100:
18              raise ScoreException('score必须在0到100之间')
19          self._score = value
20
21  try:
22      s = Student()
23      s.score = 60
24      print(s.score)
25      s.score = 999
26  except ScoreException as e:
```

异常可以自定义 III

```
27      print(e)
28
```

异常总结

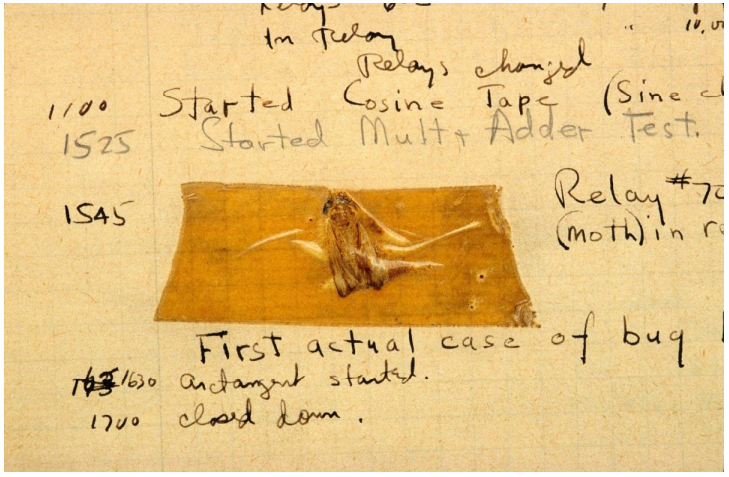
- Python 内置的 try...except...finally 用来处理错误十分方便
  - 出错时，会分析错误信息并定位错误发生的代码位置更为关键
- 程序也可以主动抛出错误，让调用者来处理相应的错误
  - 应在文档中写清楚可能会抛出哪些错误，以及错误产生的原因

调试

Bug and Debug

格蕾丝 · 赫柏 (Grace Murray Hopper)

赫柏是一位为美国海军工作的电脑专家。1945 年的一天，赫柏对 Harvard Mark II 设置好 17000 个继电器进行编程后，技术人员在进行整机运行时，它突然停止了工作。于是他们爬上去找原因，发现这台巨大的计算机内部一组继电器的触点之间有一只飞蛾，这显然是由于飞蛾受光和热的吸引，飞到了触点上，然后被高电压击死。所以在报告中，赫柏用胶条贴上飞蛾，并把“bug”来表示”一个在电脑程序里的错误”。



- 程序一次编写就能成功运行的概率很小，通常会有各种各样的错误需要调试，因此，掌握错误的调试方法非常重要。

常用的调试方法

- 观察出错的提示信息
- 利用 print 函数输出信息，观察输出结果和预期结果是否一致
- 通过日志 logging 替代 print
- 利用 assert 断言
- pdb

利用 print 进行调试

```
1 books = ['Python', 'XML', 'Information Retrieval']
2 for book in books:
3     print(book)
4
5 print('Run here!')
6 if book.price > 50:
7     print('High price.')
8
```

以上代码会抛出异常信息：

```
Traceback (most recent call last):
  File "bug.py", line 5, in <module>
    if book.price > 50:
AttributeError: 'str' object has no attribute 'price'
```

如果我们觉得前三行代码不太可能出问题，而问题很可能在后面，那

利用 logging 进行调试

print() 的结果默认输出到控制台上，不够灵活和方便，可以使用 logging 进行控制

```
1  import logging
2  logging.basicConfig(level=logging.INFO)
3
4  books = ['Python', 'XML', 'Information Retrieval']
5  for book in books:
6      logging.info(book)
7
8  logging.debug('Run here!')
9  if book.price > 50:
10     logging.warn('High price.')
11
```

assert 调试

assert 断言用于判断给定的逻辑表达式是否成立，如果不成立，就会抛出 AssertionError 异常

```
1 books = ['Python', 'XML', 'Information Retrieval']
2 assert len(books) == 3
```

启动 Python 解释器时可以用-O 参数来关闭 assert，此时的 assert 语句可看作是 pass

pdb

Python 的调试器，可以单步运行 Python 脚本，查看当前运行的代码，查看变量值

bug.py:

```
1 books = ['Python', 'XML', 'Information Retrieval']
2 for book in books:
3     print(book)
4
5 if book.price > 50:
6     print('High price.')
7
```

运行：pdb bug.py
或者：python -m pdb bug.py

pdb

- n: 执行下一条语句
- p xxx: 查看变量 xxx 的当前值
- l: 列出

pdb.set_trace()

在可能出错的地方放置 pdb.set_trace()，程序运行到该条语句时，会暂停并进入 pdb 调试环境，此时，可以用 p 指令查看变量，或者用 c 继续运行

bug2.py:

```
1 import pdb
2 books = ['Python', 'XML', 'Information Retrieval']
3 for book in books:
4     print(book)
5
6 pdb.set_trace()
7 if book.price > 50:
8     print('High price.')
9
```

测试

- 代码规范检查
- 单元测试

代码规范性检查工具

- pep8
- pylint
- pyflakes(略)

pep8

- PEP: Python 增强建议 (Python Enhancement Proposals)
  - describe and document the way python language evolves.
  - 完整的 PEP 索引列表：`https://www.python.org/dev/peps/`
- PEP 8: Style Guide for Python Code
  - Pythonic way to write code
  - `https://www.python.org/dev/peps/pep-0008/`

pep8 示例

- 新建 finder.py(源代码见下张幻灯片)
- 打开终端，进入 finder.py 文件所在目录
- 执行命令：

```
pep8  finder.py
```

输出结果

```
finder.py:3:1: E302 expected 2 blank lines, found 1
finder.py:19:1: W293 blank line contains whitespace
finder.py:19:1: W391 blank line at end of file
```

finder.py 源代码

```
1  # coding: utf-8
2
3  def first_index_of(sorted_name_list, name, start=0):
4      '''获取排序列表sorted_name_list中指定名称元素的位置
5
6      如果元素name在sorted_name_list中存在，则返回其下标，
7      下标从0开始计数；如果不存在，则返回-1.
8
9      >>> first_index_of([1,2,3], 1)
10     0
11     >>> first_index_of([1,2,3], 5)
12     -1
13
14     '''
15     try:
16         return sorted_name_list.index(name)
17     except ValueError as e:
18         return -1
19
```

pylint

- `https://www.pylint.org/`
  - star your python code!
  - Coding Standard
    - checking line-code's length,
    - checking if variable names are well-formed according to your coding standard
    - checking if imported modules are used
  - Error detection
    - checking if declared interfaces are truly implemented
    - checking if modules are imported
    - and much more ...

pylint

- 安装
  - sudo pip install pylint
- 使用
  - pylint some_file.py
  - 最终输出综合评分结果，例如：

pylint 输出结果片断

Global evaluation

Your code has been rated at 8.38/10

单元测试

- 单元测试是用来对一个模块、一个函数或者一个类来进行正确性检验的测试工作。
- TDD：Test-Driven Development
  - 敏捷开发中的一项常用技术和设计方法，即通过测试来推动整个开发的进行，在明确需要开发的功能后，首先思考如何对功能进行测试，进而完成测试用例代码的编写，然后实现具体产品功能，满足之前设计的测试用例。
- Python 的单元测试模块
  - unittest：最初由 Steve Purcell 编写，以前叫 PyUnit
  - doctest: 一个依赖于 Python 的 doc 字符串的测试工具
- 第三方提供的单元测试工具 (略)
  - nose
  - py.test

unittest

- 类似于 Java 中的 JUnit
- 提供了一个 TestCase 基类，该基类拥有一个用来验证输入输出的方法集合
- 示例
  - 假设有一个工具方法模块 utils.py, 里面包含一个求平均数的函数 average
  - 现在需要编写对应的单元测试

test_utils.py 代码清单 1

```
1  # coding: utf-8
2
3  import unittest
4  from utils import average
5
6
7  class UtilsTest(unittest.TestCase):
8      '''
9      utils模块的单元测试
10     '''
11     def setUp(self):
12         print('测试准备处理...')
13
14     def tearDown(self):
15         print('测试收尾工作.')
16
17     def test_average(self):
```

test_utils.py 代码清单 II

```
18          self.assertEqual(average(1, 2, 3), 2)
19          self.assertNotEqual(average(2, 2.4), 2)
20
21  if __name__ == '__main__':
22      unittest.main()
```

运行 “python test_utils.py” 进行测试

utils.py 代码清单

```
1 def average(*numbers):
2     return sum(numbers) / len(numbers)
```

doctest

- doctest：在 docstring 中编写测试代码
- 优点：
  - 可以通过示例创建文档和测试
  - 文档示例总是最新的
- 注意：
  - doctest 应该只被用于在文档中提供人类可读的示例
  - 帮助文档有可能因为加入过多的测试代码而破坏了其可阅读性
- 例子：
  - 为上例中的 utils.py 中的 average 函数添加 docstring

utils2.py 代码清单 1

```
# coding: utf-8

def average(*numbers):
    """
    计算输入的若干个数字的平均值.

    >>> average(1, 2, 3)
    2.0

    >>> average(0)
    0.0

    >>> average(2, 5)
    3.5
    """
    return sum(numbers) / len(numbers)

```

utils2.py 代码清单 II

```
18
19 if __name__ == '__main__':
20     import doctest
21     doctest.testmod()
```

第三方测试框架

- unittest 过于僵化，不易扩展
  - 必须继承 TestCase
  - 必须使用 TestCase 本身所提供的断言方法
  - 测试方法名称前需要加 test
  - ...
- 全部使用 doctest 替代单元测试会破坏文档的可阅读性
  - ...
- 提出了一些第三方测试框架
  - nose
  - py.test

—END—

Web 服务器

- 最简单的 Python Web 服务器
- Flask

Simple Http Server

- 最轻便的 Web 服务器[1]：
- 启动方式：
  - python -m http.server

[1]参数 m 的作用参考：`http://www.tuicool.com/articles/jMzqYzF`

Flask

- Flask 是一种简便的基于 Python 语言的 Web 应用程序开发框架
  - Flask is a microframework for Python based on Werkzeug, Jinja 2 and good intentions. And before you ask: It's BSD licensed!
- 相关中文文档可在线参考：
  - `http://dormousehole.readthedocs.io/en/latest/quickstart.html`
- 安装：
  - pip install Flask
- 文档
  - http://flask.pocoo.org/docs/0.10/.latex/Flask.pdf

Flask 简单例子

```
1  from flask import Flask
2
3  app = Flask(__name__, static_folder='.', static_url_path='')
4
5
6  @app.route('/')
7  def home():
8      return app.send_static_file('index.html')
9
10
11 @app.route('/fac/<n>')
12 def factorial(n):
13     total = 1
14     print(type(n))
15     m = int(n)
16     for i in range(m):
17         total = total*(i+1)
18     return total
19
```

Flask 简单例子 I

```
1  from flask import Flask
2  from flask import request
3
4  app = Flask(__name__, static_folder='.', static_url_path='')
5
6
7  @app.route('/')
8  def home():
9      return app.send_static_file('index.html')
10
11
12 @app.route('/echo/<thing>')
13 def echo(thing):
14     return "Say hello to my little friend: %s" % thing
15
16
17 @app.route('/fetch', methods=['GET', 'POST'])
```

Flask 简单例子 II

```
18 def fetch():
19     if request.method == 'POST':
20         url = request.form['url']
21         return url
22     return "Sorry"
23
24 app.run(port=9999, debug=True)
```

练习

实现一个 Web 程序，在网页上输入一个 url 地址，提交后，通过浏览器显示该地址所包含的所有图片。

科研人员在线社交媒体行为分析

- 编写一个 Web 应用程序，记录每个高校在社交媒体上的活动信息
  - 第一步完成基本信息的收集管理
- 基本信息包括
  - 学校名称, 学院, 专业, 姓名, 性别, 出生年, 简介, 微博 UID, 微博注册日期, 最后访问日期
- 基本信息管理功能：
  - 基本信息录入
  - 重复检测（根据学校、学院、姓名判断重复，或者根据微博 UID 判断重复）
  - 检索统计：