

公告

wiki和教程：www.pythonav.com

免费教学视频：[B站](#)：凸头统治地球

高级专题教程：[网易云课堂](#)：武沛齐

聊技术，加武Sir微信



武沛齐

扫一扫上面的二维码图案，加我微信



Python技术交流群：737658057

软件测试开发交流群：721023555

昵称：武沛齐
园龄：8年
粉丝：9944
关注：44
[+加关注](#)

我的标签

- Python(17)
- ASP.NET MVC(15)
- python之路(7)
- Tornado源码分析(5)
- 每天一道Python面试题(5)
- crm项目(4)
- 面试都在问什么？(2)
- Python开源组件 - Tyrion(1)
- Python面试315题(1)
- Python企业面试题讲解(1)

积分与排名

积分 - 425245
排名 - 779

随笔分类

- JavaScript(1)
- MVC(15)
- Python(17)
- 面试都在问什么系列？【图】(2)
- 其他(37)

随笔 - 140 文章 - 164 评论 - 887

Python之路【第五篇】：面向对象及相关

面向对象基础

基础内容介绍详见一下两篇博文：

- [面向对象初级篇](#)
- [面向对象进阶篇](#)

其他相关

一、 isinstance(obj, cls)

检查是否obj是否是类 cls 的对象

```
1 class Foo(object):
2     pass
3
4 obj = Foo()
5
6 isinstance(obj, Foo)
```

二、 issubclass(sub, super)

检查sub类是否是 super 类的派生类

```
1 class Foo(object):
2     pass
3
4 class Bar(Foo):
5     pass
6
7 issubclass(Bar, Foo)
```



三、 异常处理

1、 异常基础

在编程过程中为了增加友好性，在程序出现bug时一般不会将错误信息显示给用户，而是现实一个提示的页面，通俗来说就是不让用户看见大黄页！！

```
1 try:
2     pass
3 except Exception,ex:
4     pass
```

需求：将用户输入的两个数字相加



```
while True:
    num1 = raw_input('num1:')
    num2 = raw_input('num2:')
    try:
        num1 = int(num1)
        num2 = int(num2)
        result = num1 + num2
    except Exception, e:
        print '出现异常，信息如下：'
        print e
```

企业面试题及答案(1)
请求响应(6)
设计模式(9)
微软C#(34)

随笔档案

- 2020年6月(1)
- 2020年5月(1)
- 2019年11月(1)
- 2019年10月(1)
- 2019年9月(4)
- 2018年12月(1)
- 2018年8月(1)
- 2018年5月(2)
- 2018年4月(1)
- 2017年8月(1)
- 2017年5月(1)
- 2017年3月(1)
- 2016年10月(1)
- 2016年7月(1)
- 2015年10月(1)
- 2015年8月(1)
- 2015年7月(1)
- 2015年6月(2)
- 2015年4月(2)
- 2014年3月(3)
- 2014年1月(3)
- 2013年12月(2)
- 2013年11月(2)
- 2013年10月(7)
- 2013年8月(17)
- 2013年7月(1)
- 2013年6月(14)
- 2013年5月(23)
- 2013年4月(3)
- 2013年3月(13)
- 2013年2月(1)
- 2012年11月(26)

相册

git(14)

最新评论

1. Re:1. 路过面了个试就拿到2个offer。是运气吗?
听了大王的讲课，觉得大王真是厉害，年轻有为!

--Xiyue666

2. Re:Celery
s3.py中 使用async为变量是关键字，会报错把?

--killer-147

3. Re:不吹不擂，你想要的Python面试都在这里了【315+道题】



2、异常种类

python中的异常种类非常多，每个异常专门用于处理某一项异常!!!



AttributeError 试图访问一个对象没有的树形，比如foo.x，但是foo没有属性x
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
IndentationError 语法错误（的子类）；代码没有正确对齐
IndexError 下标索引超出序列边界，比如当x只有三个元素，却试图访问x[5]
KeyError 试图访问字典里不存在的键
KeyboardInterrupt Ctrl+C被按下
NameError 使用一个还未被赋予对象的变量
SyntaxError Python代码非法，代码不能编译（个人认为这是语法错误，写错了）
TypeError 传入对象类型与要求的不符合
UnboundLocalError 试图访问一个还未被设置的局部变量，基本上是由于另有一个同名的全局变量，导致你以为正在访问它
ValueError 传入一个调用者不期望的值，即使值的类型是正确的



ArithmeticError
AssertionError
AttributeError
BaseException
BufferError
BytesWarning
DeprecationWarning
EnvironmentError
EOFError
Exception
FloatingPointError
FutureWarning
GeneratorExit
ImportError
ImportWarning
IndentationError
IndexError
IOError
KeyboardInterrupt
KeyError
LookupError
MemoryError
NameError
NotImplementedError
OSError
OverflowError
PendingDeprecationWarning
ReferenceError
RuntimeError
RuntimeWarning
StandardError
StopIteration
SyntaxError
SyntaxWarning
SystemError
SystemExit
TabError
TypeError
UnboundLocalError

先赞了再说 日后再说 哈哈

--Xiyue666

4. Re:Python开发【第十九篇】：Python操作MySQL

武大大 我学到这里想放弃了 怎么办？学不进去了！

--Xiyue666

5. Re:为什么很多IT公司不喜欢进过培训机构的人呢？

@toEverybody 您搁这混淆概念呢？？明明是在讲两个国家间以前的技术人才的区别，这咋能让培训班的背锅？那科班的那些孩子呢？那些计算机专业的研究生们呢？思想深度不是培训不培训就能划分清楚的。...

--温故而新

```
UnicodeDecodeError
UnicodeEncodeError
UnicodeError
UnicodeTranslateError
UnicodeWarning
UserWarning
ValueError
Warning
ZeroDivisionError
```



```
dic = ["wupeiqi", 'alex']
try:
    dic[10]
except IndexError, e:
    print e
```

```
dic = {'k1':'v1'}
try:
    dic['k20']
except KeyError, e:
    print e
```

```
s1 = 'hello'
try:
    int(s1)
except ValueError, e:
    print e
```

对于上述实例，异常类只能用来处理指定的异常情况，如果非指定异常则无法处理。

```
1 # 未捕获到异常，程序直接报错
2
3 s1 = 'hello'
4 try:
5     int(s1)
6 except IndexError,e:
7     print e
```

所以，写程序时需要考虑try代码块中可能出现的任意异常，可以这样写：

```
1 s1 = 'hello'
2 try:
3     int(s1)
4 except IndexError,e:
5     print e
6 except KeyError,e:
7     print e
8 except ValueError,e:
9     print e
```

万能异常 在python的异常中，有一个万能异常：Exception，他可以捕获任意异常，即：

```
1 s1 = 'hello'
2 try:
3     int(s1)
4 except Exception,e:
5     print e
```

接下来你可能要问了，既然有这个万能异常，其他异常是不是就可以忽略了！

答：当然不是，对于特殊处理或提醒的异常需要先定义，最后定义Exception来确保程序正常运行。

```
1 s1 = 'hello'
2 try:
3     int(s1)
4 except KeyError,e:
5     print '键错误'
6 except IndexError,e:
7     print '索引错误'
8 except Exception, e:
9     print '错误'
```

3、异常其他结构

```
1 try:
2     # 主代码块
3     pass
4 except KeyError,e:
5     # 异常时, 执行该块
6     pass
7 else:
8     # 主代码块执行完, 执行该块
9     pass
10 finally:
11     # 无论异常与否, 最终执行该块
12     pass
```

4、主动触发异常

```
1 try:
2     raise Exception('错误了。。。')
3 except Exception,e:
4     print e
```

5、自定义异常

```
1 class WupeiqiException(Exception):
2
3     def __init__(self, msg):
4         self.message = msg
5
6     def __str__(self):
7         return self.message
8
9 try:
10     raise WupeiqiException('我的异常')
11 except WupeiqiException,e:
12     print e
```

6、断言

```
1 # assert 条件
2
3 assert 1 == 1
4
5 assert 1 == 2
```

四、反射

python中的反射功能是由以下四个内置函数提供：hasattr、getattr、setattr、delattr，改四个函数分别用于对对象内部执行：检查是否含有某成员、获取成员、设置成员、删除成员。

```
1 class Foo(object):
2
3     def __init__(self):
4         self.name = 'wupeiqi'
5
6     def func(self):
```

```

7         return 'func'
8
9     obj = Foo()
10
11     ##### 检查是否含有成员 #####
12     hasattr(obj, 'name')
13     hasattr(obj, 'func')
14
15     ##### 获取成员 #####
16     getattr(obj, 'name')
17     getattr(obj, 'func')
18
19     ##### 设置成员 #####
20     setattr(obj, 'age', 18)
21     setattr(obj, 'show', lambda num: num + 1)
22
23     ##### 删除成员 #####
24     delattr(obj, 'name')
25     delattr(obj, 'func')

```

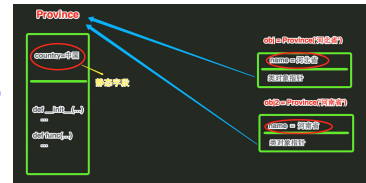
详细解析：

当我们要访问一个对象的成员时，应该是这样操作：

```

1     class Foo(object):
2
3         def __init__(self):
4             self.name = 'alex'
5
6         def func(self):
7             return 'func'
8
9     obj = Foo()
10
11     # 访问字段
12     obj.name
13     # 执行方法
14     obj.func()

```



那么问题来了？

a、上述访问对象成员的 name 和 func 是什么？

答：是变量名

b、obj.xxx 是什么意思？

答：obj.xxx 表示去obj中或类中寻找变量名 xxx，并获取对应内存地址中的内容。

c、需求：请使用其他方式获取obj对象中的name变量指向内存中的值“alex”

```

class Foo(object):

    def __init__(self):
        self.name = 'alex'

# 不允许使用 obj.name
obj = Foo()

```

答：有两种方式，如下：

```

class Foo(object):

```

```

def __init__(self):
    self.name = 'alex'

def func(self):
    return 'func'

# 不允许使用 obj.name
obj = Foo()

print obj.__dict__['name']

```

```

class Foo(object):

    def __init__(self):
        self.name = 'alex'

    def func(self):
        return 'func'

# 不允许使用 obj.name
obj = Foo()

print getattr(obj, 'name')

```

d、比较三种访问方式

- obj.name
- obj.__dict__['name']
- getattr(obj, 'name')

答: 第一种和其他种比, ...
 第二种和第三种比, ...

```

#!/usr/bin/env python
#coding:utf-8
from wsgiref.simple_server import make_server

class Handler(object):

    def index(self):
        return 'index'

    def news(self):
        return 'news'

def RunServer(environ, start_response):
    start_response('200 OK', [('Content-Type', 'text/html')])
    url = environ['PATH_INFO']
    temp = url.split('/')[1]
    obj = Handler()
    is_exist = hasattr(obj, temp)
    if is_exist:
        func = getattr(obj, temp)
        ret = func()

        return ret
    else:
        return '404 not found'

```

```
if __name__ == '__main__':  
    httpd = make_server('', 8001, RunServer)  
    print "Serving HTTP on port 8000..."  
    httpd.serve_forever()
```

结论：反射是通过字符串的形式操作对象相关的成员。**一切事物都是对象！！**

```
#!/usr/bin/env python  
# -*- coding:utf-8 -*-  
  
import sys  
  
def s1():  
    print 's1'  
  
def s2():  
    print 's2'  
  
this_module = sys.modules[__name__]  
  
hasattr(this_module, 's1')  
getattr(this_module, 's2')
```

类也是对象

```
1 class Foo(object):  
2  
3     staticField = "old boy"  
4  
5     def __init__(self):  
6         self.name = 'wupeiqi'  
7  
8     def func(self):  
9         return 'func'  
10  
11     @staticmethod  
12     def bar():  
13         return 'bar'  
14  
15 print getattr(Foo, 'staticField')  
16 print getattr(Foo, 'func')  
17 print getattr(Foo, 'bar')
```

模块也是对象

```
#!/usr/bin/env python  
# -*- coding:utf-8 -*-  
  
def dev():  
    return 'dev'
```

```

1  #!/usr/bin/env python
2  # -*- coding:utf-8 -*-
3
4  """
5  程序目录:
6      home.py
7      index.py
8
9  当前文件:
10     index.py
11 """
12
13
14 import home as obj
15
16 #obj.dev()
17
18 func = getattr(obj, 'dev')
19 func()

```

设计模式

一、单例模式

单例，顾名思义单个实例。

学习单例之前，首先来回顾下面面向对象的内容：

python的面向对象由两个非常重要的两个“东西”组成：类、实例

面向对象场景一：

如：创建三个游戏人物，分别是：

- 苍井井，女，18，初始战斗力1000
- 东尼木木，男，20，初始战斗力1800
- 波多多，女，19，初始战斗力2500

```

# ##### 定义类 #####
class Person:

    def __init__(self, na, gen, age, fig):
        self.name = na
        self.gender = gen
        self.age = age
        self.fight = fig

    def grassland(self):
        """注释：草丛战斗，消耗200战斗力"""

        self.fight = self.fight - 200

# ##### 创建实例 #####

cang = Person('苍井井', '女', 18, 1000) # 创建苍井井角色
dong = Person('东尼木木', '男', 20, 1800) # 创建东尼木木角色
bo = Person('波多多', '女', 19, 2500) # 创建波多多角色

```

面向对象场景二：

如：创建对数据库操作的公共类

- 增

- 删
- 改
- 查

```

# ##### 定义类 #####

class DbHelper(object):

    def __init__(self):
        self.hostname = '1.1.1.1'
        self.port = 3306
        self.password = 'pwd'
        self.username = 'root'

    def fetch(self):
        # 连接数据库
        # 拼接sql语句
        # 操作
        pass

    def create(self):
        # 连接数据库
        # 拼接sql语句
        # 操作
        pass

    def remove(self):
        # 连接数据库
        # 拼接sql语句
        # 操作
        pass

    def modify(self):
        # 连接数据库
        # 拼接sql语句
        # 操作
        pass

# ##### 操作类 #####

db = DbHelper()
db.create()
```

实例：结合场景二实现Web应用程序

```

#!/usr/bin/env python
#coding:utf-8
from wsgiref.simple_server import make_server

class DbHelper(object):

    def __init__(self):
        self.hostname = '1.1.1.1'
        self.port = 3306
        self.password = 'pwd'
        self.username = 'root'

    def fetch(self):
```

```

        # 连接数据库
        # 拼接sql语句
        # 操作
        return 'fetch'

    def create(self):
        # 连接数据库
        # 拼接sql语句
        # 操作
        return 'create'

    def remove(self):
        # 连接数据库
        # 拼接sql语句
        # 操作
        return 'remove'

    def modify(self):
        # 连接数据库
        # 拼接sql语句
        # 操作
        return 'modify'

class Handler(object):

    def index(self):
        # 创建对象
        db = DbHelper()
        db.fetch()
        return 'index'

    def news(self):
        return 'news'

def RunServer(envIRON, start_response):
    start_response('200 OK', [('Content-Type', 'text/html')])
    url = envIRON['PATH_INFO']
    temp = url.split('/')[1]
    obj = Handler()
    is_exist = hasattr(obj, temp)
    if is_exist:
        func = getattr(obj, temp)
        ret = func()
        return ret
    else:
        return '404 not found'

if __name__ == '__main__':
    httpd = make_server('', 8001, RunServer)
    print "Serving HTTP on port 8001..."
    httpd.serve_forever()

```



对于上述实例，每个请求到来，都需要在内存里创建一个实例，再通过该实例执行指定的方法。

那么问题来了...如果并发量大的话，内存里就会存在非常多**功能上一模一样的对象**。存在这些对象肯定会消耗内存，对于这些功能相同的对象可以在内存中仅创建一个，需要时都去调用，也是极好的!!!

铛铛 铛铛 铛铛铛铛铛，单例模式出马，单例模式用来保证内存中**仅存在一个实例**!!!

通过面向对象的特性，构造出单例模式：

```

1  # ##### 单例类定义 #####
2  class Foo(object):
3
4      __instance = None
5
6      @staticmethod
7      def singleton():
8          if Foo.__instance:
9              return Foo.__instance
10         else:
11             Foo.__instance = Foo()
12             return Foo.__instance
13
14 # ##### 获取实例 #####
15 obj = Foo.singleton()

```

对于Python单例模式，创建对象时不能再直接使用：obj = Foo()，而应该调用特殊的方法：obj = Foo.singleton()。

```

#!/usr/bin/env python
#coding:utf-8
from wsgiref.simple_server import make_server

# ##### 单例类定义 #####
class DbHelper(object):

    __instance = None

    def __init__(self):
        self.hostname = '1.1.1.1'
        self.port = 3306
        self.password = 'pwd'
        self.username = 'root'

    @staticmethod
    def singleton():
        if DbHelper.__instance:
            return DbHelper.__instance
        else:
            DbHelper.__instance = DbHelper()
            return DbHelper.__instance

    def fetch(self):
        # 连接数据库
        # 拼接sql语句
        # 操作
        pass

    def create(self):
        # 连接数据库
        # 拼接sql语句
        # 操作
        pass

    def remove(self):
        # 连接数据库
        # 拼接sql语句
        # 操作
        pass

    def modify(self):
        # 连接数据库

```

```

# 拼接sql语句
# 操作
pass

class Handler(object):

    def index(self):
        obj = DbHelper.singleton()
        print id(single)
        obj.create()
        return 'index'

    def news(self):
        return 'news'

def RunServer(envron, start_response):
    start_response('200 OK', [('Content-Type', 'text/html')])
    url = envron['PATH_INFO']
    temp = url.split('/')[1]
    obj = Handler()
    is_exist = hasattr(obj, temp)
    if is_exist:
        func = getattr(obj, temp)
        ret = func()
        return ret
    else:
        return '404 not found'

if __name__ == '__main__':
    httpd = make_server('', 8001, RunServer)
    print "Serving HTTP on port 8001..."
    httpd.serve_forever()

```



总结：单利模式存在的目的是保证当前内存中仅存在单个实例，避免内存浪费！！



作者：武沛齐

出处：<http://www.cnblogs.com/wupeiqi/>

本文版权归作者和博客园共有，欢迎转载，但未经作者同意必须保留此段声明，且在文章页面明显位置给出原文连接。

好文要顶

关注我

收藏该文



武沛齐

关注 - 44

粉丝 - 9944

+加关注

18

0

posted @ 2015-12-03 21:47 武沛齐 阅读(31930) 评论(5) 编辑 收藏

评论列表

#1楼 2017-01-04 11:40 樊宇豪

回复 引用

拍砖，为什么描述符不讲？感觉很经典。

支持(0) 反对(0)

#2楼 2018-04-15 08:48 scw89757+

回复 引用

非常好的文章 感谢分享

支持(0) 反对(0)

#3楼 2018-08-09 20:24 野生大魔王

@ scw89757+
非常好

回复 引用

支持(0) 反对(0)

#4楼 2018-08-09 20:25 野生大魔王

学习

回复 引用

支持(0) 反对(0)

#5楼 2019-07-29 14:07 opss

反射 中delattr (obj, 'func') 不能执行， 方法是类命名空间吧
delattr(obj, 'name')
delattr(obj, 'func') # 异常

回复 引用

支持(0) 反对(0)

最新评论 刷新页面 返回顶部

发表评论

编辑 预览

B 🔗 <> “ ☒

支持 Markdown

提交评论 退出 订阅评论

[Ctrl+Enter快捷键提交]

- 【推荐】超50万行VC++源码：大型组态工控、电力仿真CAD与GIS源码库
- 【推荐】了不起的开发者，挡不住的华为，园子里的品牌专区
- 【推荐】有道智云周年庆，API服务大放送，注册即送100元体验金！
- 【推荐】首次公开！三代技术人深度对话，《云上朗读者》开放下载



相关博文：

- Python之路【第五篇】：面向对象及相关
- Python之路【第五篇】：面向对象及相关面向对象基础
- Python之路【第五篇】：面向对象及相关
- Python之路【第五篇】：面向对象及相关
- Python之路【第五篇】：面向对象及相关
- » 更多推荐...

最新 IT 新闻：

- 腾讯黑鲨3S评测：升级120Hz刷新率 提升整机游戏体验
- 理想成功IPO，我们和它的投资人聊了聊李想与理想
- 1000万份微信支付“摇免单”：每单最高200元

- 市值跌去99%、冯鑫被捕一年：暴风集团濒临“暴风退”
 - 董明珠称空调一晚一度电都是骗人的：企业不敢再打这样的广告
- » 更多新闻...