

28 | 如何合理利用assert？

2019-07-12 景霄

Python核心技术与实战

[进入课程 >](#)



讲述：冯永吉

时长 09:09 大小 8.39M



你好，我是景霄。

相信你平时在写代码时，肯定或多或少看到过 `assert` 的存在。我也曾在日常的 `code review` 中，被一些同事要求增加 `assert` 语句，让代码更加健壮。


不过，尽管如此，我发现在很多情况下，`assert` 还是很容易被忽略，人们似乎对这么一个“不起眼”的东西并不关心。但事实上，这个看似“不起眼”的东西，如果能用好，对我们的程序大有裨益。

说了这么多，那么究竟什么是 `assert`，我们又该如何合理地使用 `assert` 呢？今天这节课，我就带你一起来学习它的用法。

什么是 `assert` ?

Python 的 `assert` 语句，可以说是一个 `debug` 的好工具，主要用于测试一个条件是否满足。如果测试的条件满足，则什么也不做，相当于执行了 `pass` 语句；如果测试条件不满足，便会抛出异常 `AssertionError`，并返回具体的错误信息（optional）。

它的具体语法是下面这样的：

 复制代码


```
1 assert_stmt ::= "assert" expression ["," expression]
```

我们先来看一个简单形式的 `assert expression`，比如下面这个例子：

 复制代码

```
1 assert 1 == 2
```

它就相当于下面这两行代码：

 复制代码

```
1 if __debug__:
2     if not expression: raise AssertionError
```

再来看 `assert expression1, expression2` 的形式，比如下面这个例子：

 复制代码

```
1 assert 1 == 2, 'assertion is wrong'
```

它就相当于下面这两行代码：

 复制代码

```
1 if __debug__:
2     if not expression1: raise AssertionError(expression2)
```

这里的 `__debug__` 是一个常数。如果 Python 程序执行时附带了 `-O` 这个选项，比如 `Python test.py -O`，那么程序中所有的 `assert` 语句都会失效，常数 `__debug__` 便为 `False`；反之 `__debug__` 则为 `True`。

不过，需要注意的是，直接对常数 `__debug__` 赋值是非法的，因为它的值在解释器开始运行时就已经决定了，中途无法改变。


此外，一定记住，不要在使用 `assert` 时加入括号，比如下面这个例子：

 复制代码

```
1 assert(1 == 2, 'This should fail')
2 # 输出
3 <ipython-input-8-2c057bd7fe24>:1: SyntaxWarning: assertion is always true, perhaps remove parentheses
4     assert(1 == 2, 'This should fail')
```

如果你按照这样来写，无论表达式对与错（比如这里的 `1 == 2` 显然是错误的），`assert` 检查永远不会 fail，程序只会给你 `SyntaxWarning`。

正确的写法，应该是下面这种不带括号的写法：

 复制代码

```
1 assert 1 == 2, 'This should fail'
2 # 输出
3 AssertionError: This should fail
```

总的来说，`assert` 在程序中的作用，是对代码做一些 internal 的 self-check。使用 `assert`，就表示你很确定。这个条件一定会发生或者一定不会发生。


举个例子，比如你有一个函数，其中一个参数是人的性别，因为性别只有男女之分（这里只指生理性别），你便可以使用 `assert`，以防止程序的非法输入。如果你的程序没有 bug，

那么 `assert` 永远不会抛出异常；而它一旦抛出了异常，你就知道程序存在问题了，并且可以根据错误信息，很容易定位出错误的源头。

assert 的用法

讲完了 `assert` 的基本语法与概念，我们接下来通过一些实际应用的例子，来看看 `assert` 在 Python 中的用法，并弄清楚 `assert` 的使用场景。


第一个例子，假设你现在使用的极客时间正在做专栏促销活动，准备对一些专栏进行打折，所以后台需要写一个 `apply_discount()` 函数，要求输入为原来的价格和折扣，输出是折后的价格。那么，我们可以大致写成下面这样：

 复制代码

```
1 def apply_discount(price, discount):
2     updated_price = price * (1 - discount)
3     assert 0 <= updated_price <= price, 'price should be greater or equal to 0 and less
4     return updated_price
```


可以看到，在计算新价格的后面，我们还写了一个 `assert` 语句，用来检查折后价格，这个值必须大于等于 0、小于等于原来的价格，否则就抛出异常。

我们可以试着输入几组数，来验证一下这个功能：

 复制代码

```
1 apply_discount(100, 0.2)
2 80.0
3
4 apply_discount(100, 2)
5 AssertionError: price should be greater or equal to 0 and less or equal to original price
```


显然，当 `discount` 是 0.2 时，输出 80，没有问题。但是当 `discount` 为 2 时，程序便抛出下面这个异常：

 复制代码

```
1 AssertionError: price should be greater or equal to 0 and less or equal to original price
```

这样一来，如果开发人员修改相关的代码，或者是加入新的功能，导致 discount 数值的异常时，我们运行测试时就可以很容易发现问题。正如我开头所说，assert 的加入，可以有效预防 bug 的发生，提高程序的健壮性。


再来看一个例子，最常见的除法操作，这在任何领域的计算中都经常会遇到。同样还是以极客时间为例，假如极客时间后台想知道每个专栏的平均销售价格，那么就需要给定销售总额和销售数目，这样平均销售价格便很容易计算出来：

 复制代码

```
1 def calculate_average_price(total_sales, num_sales):
2     assert num_sales > 0, 'number of sales should be greater than 0'
3     return total_sales / num_sales
```

同样的，我们也加入了 assert 语句，规定销售数目必须大于 0，这样就可以防止后台计算那些还未开卖的专栏的价格。

除了这两个例子，在实际工作中，assert 还有一些很常见的用法，比如下面的场景：


 复制代码

```
1 def func(input):
2     assert isinstance(input, list), 'input must be type of list'
3     # 下面的操作都是基于前提：input 必须是 list
4     if len(input) == 1:
5         ...
6     elif len(input) == 2:
7         ...
8     else:
9         ...
```

这里函数 func() 里的所有操作，都是基于输入必须是 list 这个前提。是不是很熟悉的需求呢？那我们就很有必要在开头加一句 assert 的检查，防止程序出错。

当然，我们也要根据具体情况具体分析。比如上面这个例子，之所以能加 `assert`，是因为我们很确定输入必须是 `list`，不能是其他数据类型。

如果你的程序中，允许 `input` 是其他数据类型，并且对不同的数据类型都有不同的处理方式，那你就应该写成 `if else` 的条件语句了：


 复制代码

```
1 def func(input):
2     if isinstance(input, list):
3         ...
4     else:
5         ...
```

assert 错误示例

前面我们讲了这么多 `assert` 的使用场景，可能给你一种错觉，也可能会让你有些迷茫：很多地方都可以使用 `assert`，那么，很多 `if` 条件语句是不是都可以换成 `assert` 呢？这么想可就不准确了，接下来，我们就一起来看几个典型的错误用法，避免一些想当然的用法。

还是以极客时间为例，我们假设下面这样的场景：后台有时候需要删除一些上线时间较长的专栏，于是，相关的开发人员便设计出了下面这个专栏删除函数。

 复制代码

```
1 def delete_course(user, course_id):
2     assert user_is_admin(user), 'user must be admin'
3     assert course_exist(course_id), 'course id must exist'
4     delete(course_id)
```

极客时间规定，必须是 `admin` 才能删除专栏，并且这个专栏课程必须存在。有的同学一看，很熟悉的需求啊，所以在前面加了相应的 `assert` 检查。那么我想让你思考一下，这样写到底对不对呢？

答案显然是否定的。你可能觉得，从代码功能角度来说，这没错啊。但是在实际工程中，基本上没人会这么写。为什么呢？

要注意，前面我说过，`assert` 的检查是可以被关闭的，比如在运行 Python 程序时，加入 `-O` 这个选项就会让 `assert` 失效。因此，一旦 `assert` 的检查被关闭，`user_is_admin()` 和 `course_exist()` 这两个函数便不会被执行。这就会导致：

任何用户都有权限删除专栏课程；

并且，不管这个课程是否存在，他们都可以强行执行删除操作。

这显然会给程序带来巨大的安全漏洞。所以，正确的做法，是使用条件语句进行相应的检查，并合理抛出异常：

 复制代码


```
1 def delete_course(user, course_id):
2     if not user_is_admin(user):
3         raise Exception('user must be admin')
4     if not course_exist(course_id):
5         raise Exception('course id must exist')
6     delete(course_id)
```

再来看一个例子，如果你想打开一个文件，进行数据读取、处理等一系列操作，那么下面这样的写法，显然也是不正确的：

 复制代码

```
1 def read_and_process(path):
2     assert file_exist(path), 'file must exist'
3     with open(path) as f:
4         ...
```

因为 `assert` 的使用，表明你强行指定了文件必须存在，但事实上在很多情况下，这个假设并不成立。另外，打开文件操作，也有可能触发其他的异常。所以，正确的做法是进行异常处理，用 `try` 和 `except` 来解决：

 复制代码

```
1 def read_and_process(path):
2     try:
3         with open(path) as f:
```

```
4         ...
5     except Exception as e:
6         ...
```

总的来说，`assert` 并不适用 `run-time error` 的检查。比如你试图打开一个文件，但文件不存在；再或者是你试图从网上下载一个东西，但中途断网了等等，这些情况下，还是应该参照我们前面所讲的[错误与异常](#)的内容，进行正确处理。

总结

今天这节课，我们一起学习了 `assert` 的用法。`assert` 通常用来对代码进行必要的 `self check`，表明你很确定这种情况一定发生，或者一定不会发生。需要注意的是，使用 `assert` 时，一定不要加上括号，否则无论表达式对与错，`assert` 检查永远不会 `fail`。另外，程序中的 `assert` 语句，可以通过 `-O` 等选项被全局 `disable`。

通过这节课的几个使用场景，你能看到，`assert` 的合理使用，可以增加代码的健壮度，同时也方便了程序出错时开发人员的定位排查。

不过，我们也不能滥用 `assert`。很多情况下，程序中出现的不同情况都是意料之中的，需要我们用不同的方案去处理，这时候用条件语句进行判断更为合适。而对于程序中的一些 `run-time error`，请记得使用异常处理。

思考题

最后，给你留一个思考题。在平时的工作学习中，你用过 `assert` 吗？如果用过的话，是在什么情况下使用的？有遇到过什么问题吗？

欢迎在留言区写下你的经历，还有今天学习的心得和疑惑，与我一起分享。也欢迎你把这篇文章分享给你的同事、朋友，我们一起交流，一起进步。

Python 核心技术与实战

系统提升你的 Python 能力

景霄

Facebook 资深工程师



新版升级：点击「👤 请朋友读」，20位好友免费读，邀请订阅更有**现金**奖励。

© 版权归极客邦科技所有，未经许可不得传播售卖。页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 27 | 学会合理分解代码，提高代码可读性

下一篇 29 | 巧用上下文管理器和With语句精简代码

精选留言 (13)

写留言



图·美克尔

2019-07-15

写测试代码时用

展开 ∨



倾

2019-07-15

一般不怎么用，全部使用异常处理的。

展开 ∨





更好的自己

2019-07-12

个人认为assert的使用应该是，有没有assert程序都能够正常运行，但有了assert可以使我们的代码后期维护更加方便

展开 ∨



小侠龙旋风

2019-07-12

印象中好像就这个用法比较常用一点：

```
assert isinstance(input, list), 'input must be type of list'
```

展开 ∨

作者回复: 这个是一个常用的例子，但是文中所讲的其实都挺常用的



阿西吧

2019-07-12

正式上线时也建议用assert??

展开 ∨

作者回复: 线上代码里也有assert的



程序员人生

2019-07-12

老师，没有用过assert唉。看你介绍，好像用来调试程序用



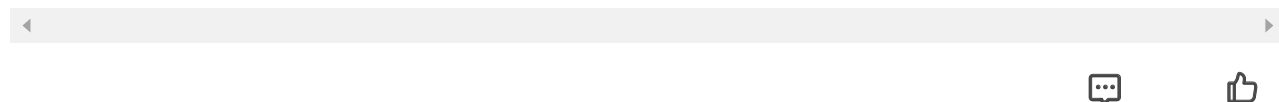
稳

2019-07-12

我记得以前看过assert会严重影响运行性能，所以一直不在代码里用。工作中，主要是单元测试用，想跟老师了解fb的规定

展开 ▾

作者回复: 没有这个说法



carpe_diem

2019-07-12

assert主要用于开发和测试阶段，使用assert时，应该是思考一下，当去掉assert语句之后，代码逻辑是否仍然正确。

展开 ▾



天凉好个秋

2019-07-12

感觉只要可以disable，那还是需要对相同问题加入if判断啊。是否是像其他同学说的自测阶段才能用assert呢？并且对于可能出现的相同问题需要用assert写一遍，再用if写一遍？



enjoylearning

2019-07-12

assert想用但不确信什么情况下用，一直以为python没法做debug判断，原来也有个类似，我们线上经常出现一些诡异的bug，不知道可不可以开启这个

展开 ▾



安排

2019-07-12

自测阶段才能用assert吧？上线长时间运行的程序都会disable掉assert吧？



AllenGFLiu

2019-07-12

以前看其他课程，倒是都没讲过 assert这一点，看起来觉得assert还是大有用武之地的，

一定找机会用一下。



Hurt

2019-07-12

可以这么理解吗 只是用来作为自我检查 而不是用来做你代码的判断条件的

