构建Python包的五个简单准则简介

<http://www.jb51.net/article/67825.htm>

创建一个软件包（package）似乎已经足够简单了，也就是在文件目录下搜集一些模块，再加上一个\_\_init\_\_.py文件，对吧？我们很容易看出来，随着时间的推移，通过对软件包的越来越多的修改，一个设计很差的软件包可能会出现循环依赖问题，或是可能变得不可移植和不可靠。

**1. \_\_init\_\_.py 仅为导入服务**

对于一个简单的软件包，你可能会忍不住把工具方法，工厂方法和异常处理都丢进\_\_init\_\_.py，千万别这样！

一个结构良好的\_\_init\_\_.py文件，仅为一个非常重要的目的来服务：从子模块导入。你的\_\_init\_\_.py应该看起来像这个样子：



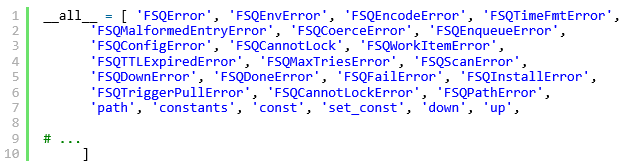
**2.使用\_\_init\_\_.py来限制导入顺序**

把方法和类置于软件包的作用域中，这样用户就不需要深入软件包的内部结构，使你的软包变得易用。

作为调和导入顺序的唯一地方。

使用得当的话，\_\_init\_\_.py 可以为你提供重新组织内部软件包结构的灵活性，而不需要担心由内部导入子模块或是每个模块导入顺序所带来的副作用。因为你是以一个特定的顺序导入子模块，你的\_\_init\_\_.py 对于他程序员来讲应该简单易懂，并且能够明显的表示该软件包所能提供的全部功能。

文档字符串，以及在软件包层面对\_\_all\_\_属性的赋值应当是\_\_init\_\_.py中唯一与导入无关的代码：



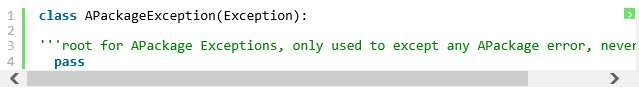
**3.使用一个模块来定义所有的异常**

你也许已经注意到了，\_\_init\_\_.py中的第一个导入语句从exceptions.py子模块中导入了全部的异常。从这里出发，你将看到，在大多数的软件包中，异常被定义在引起它们的代码附近。尽管这样可以为一个模块提供高度的完整性，一个足够复杂的软件包会通过如下两种方式，使得这一模式出现问题。

通常一个模块/程序需要从一个子模块导入一个函数, 利用它导入代码并抛出异常。为了捕获异常并保持一定的粒度，你需要导入你需要的模块，以及定义了异常的模块（或者更糟，你要导入一系列的异常）。这一系列衍生出来的导入需求，是在你的软件包中编织一张错综复杂的导入之网的始作俑者。你使用这种方式的次数越多，你的软件包内部就变的越相互依赖，也更加容易出错。

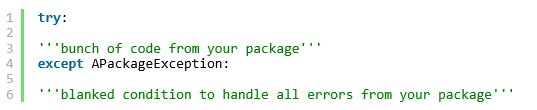
随着异常数量的不断增长，找到一个软件包可能引发的全部异常变的越来越难。把所有的异常定义在一个单独的模块中，提供了一个方便的地方，在这里，程序员可以审查并确定你的软件包所能引发全部潜在错误状态。

你应该为你的软件包的异常定义一个基类:



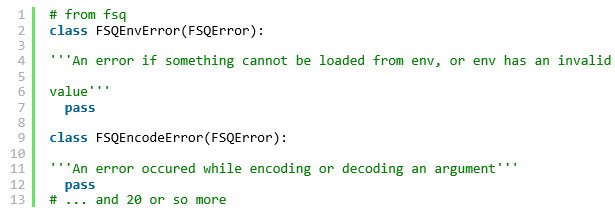
**root for APackage Exceptions, only used to except any APackage error, never raised**

然后确保你的软件包在任何错误状态下，只会引发这个基类异常的子类异常，这样如果你需要的话，你就可以阻止全部的异常：

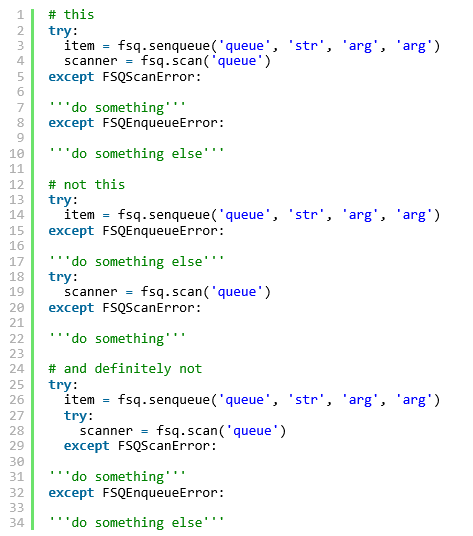


对于一般的错误状态，这里有一些重要的异常处理已经被包括在标准库中了（例如，TypeError, ValueError等）

灵活地定义异常处理并保持足够的粒度：



在你的异常处理中保持更大的[粒度](https://github.com/axialmarket/fsq/blob/master/fsq/exceptions.py)，有利于让程序员们在一个try/except中包含越来越大的，互相不干涉的代码段。



在异常定义时保持高度的粒度，会减少错综复杂的错误处理，并且允许你把正常执行指令和错误处理指令分别开来，使你的代码更加易懂和更易维护。

**4. 在软件包内部只进行相对导入**

在子模块中你时常见到的一个简单错误，就是使用软件包的名字来导入软件包。

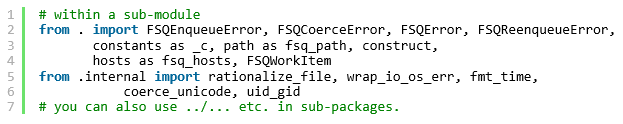
clipboard.png

这样做会导致两个不好的结果：

子模块只有当软件包被安装在 PYTHONPATH 内才能正确运行。

子模块只有当这个软件包的名字是 a\_package 时才能正确运行。

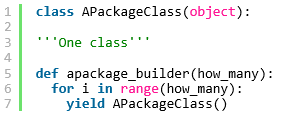
尽管第一条看上去并不是什么大问题，但是考虑一下，如果你在 PYTHONPATH 下的两个目录中，有两个同名的软件包。你的子模块可能最终导入了另一个软件包，你将无意间使得某个或某些对此毫无戒备的程序员（或是你自己）debug 到深夜。



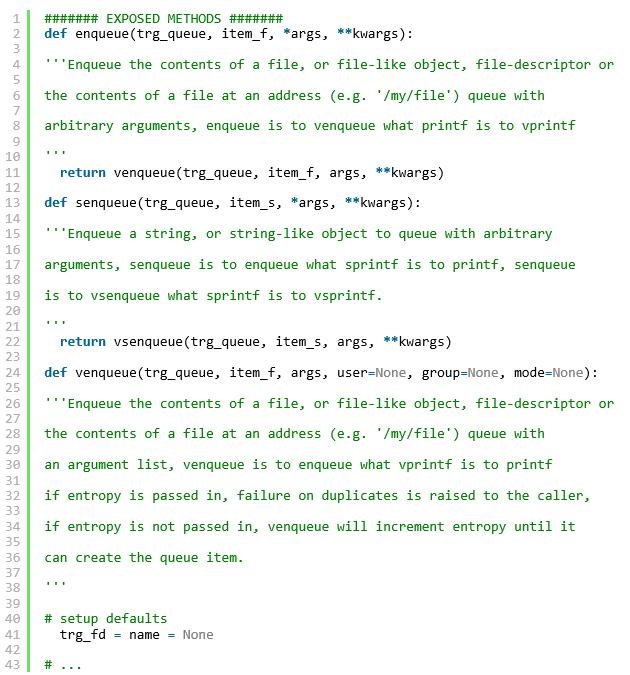
**5. 让模块保持较小的规模**

你的模块应当比较小。记住，那个使用你软件包的程序员会在软件包作用域进行导入，同时你会使用你的 \_\_init\_\_.py 文件来作为一个组织工具，来暴露一个完整的接口。

好的做法是一个模块只定义一个类，伴随一些帮助方法和工厂方法来协助建立这个模块。



如果你的模块暴露了一些方法，把一些相互依赖的方法分为一组放进一个模块，并且把不相互依赖的方法移动到单独的模块中：



上面的例子是 [fsq/enqueue.py](https://github.com/axialmarket/fsq/blob/master/fsq/enqueue.py)，它暴露了一系列的方法来为同一个功能提供不同的接口（就像 simplejson 中的l oad/loads）。尽管这个例子足够直观，让你的模块保持较小规模需要一些判断，但是一个好的原则是：

当你有疑问的时候，就去创建一个新的子模块吧。