py2exe使用方法详解

最近有很多朋友对网上的py2exe使用教程看不懂，我在这里发布图文详解。

平台：python2.51

py2exe:0.69 for python2.5

这里以打包e://hello.py为例

第一步：

在e盘新建hello.py：

helo.py代码如下：

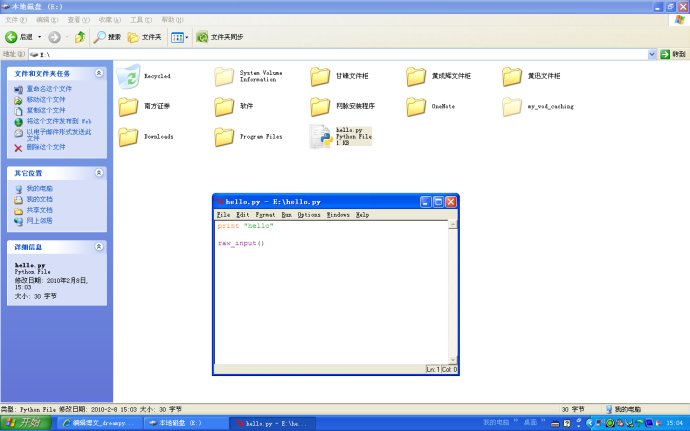
--------------------

print "hello"

raw\_input()

--------------------

图片：

[](http://photo.blog.sina.com.cn/showpic.html#blogid=649ebd460100hpgh&url=http://static16.photo.sina.com.cn/orignal/649ebd46g7f116b0609df)

第二步：可以在任何路径任何位置新建一个名为setup.py的文件，这里把setup.py建在e盘根目录

e:\setup.py:

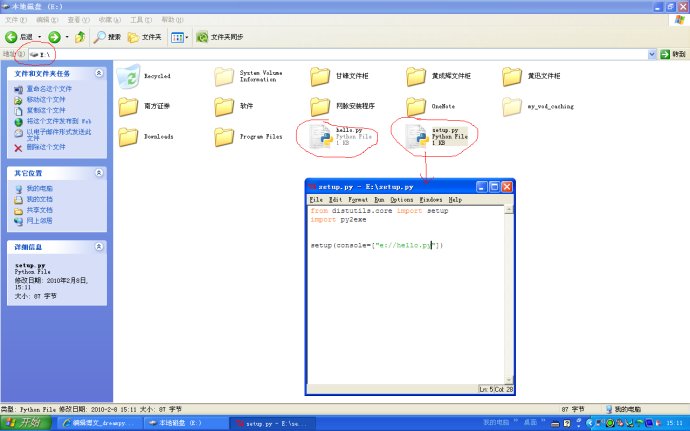
代码如下：

from distutils.core import setup  
import py2exe

setup(console=[r’e:/hello.py’])

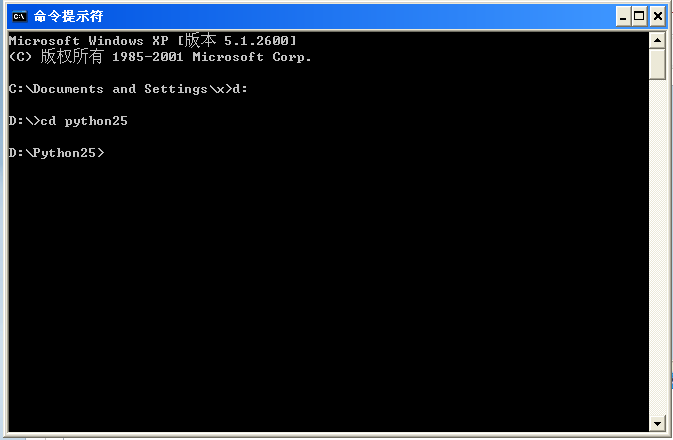
               #这里是要打包

                的py文件的

                路径。  
[](http://photo.blog.sina.com.cn/showpic.html#blogid=649ebd460100hpgh&url=http://static16.photo.sina.com.cn/orignal/649ebd46g7f11868c247f)  
  
第三步：

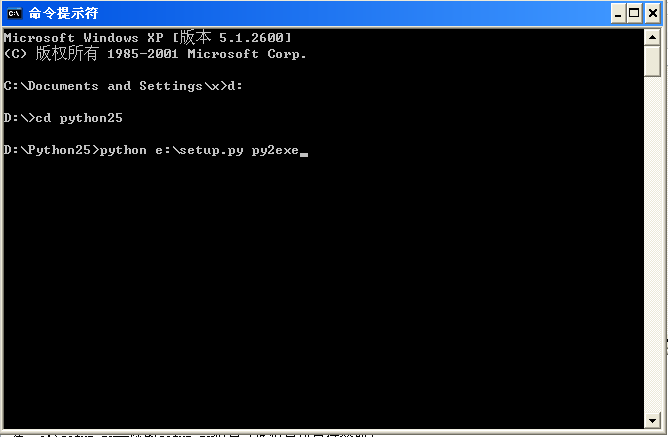
打开命令提示符(cmd)。注意：不是python解析器，看到网上很多人都把其理解为python解析器，注意：python解析器！！！是cmd

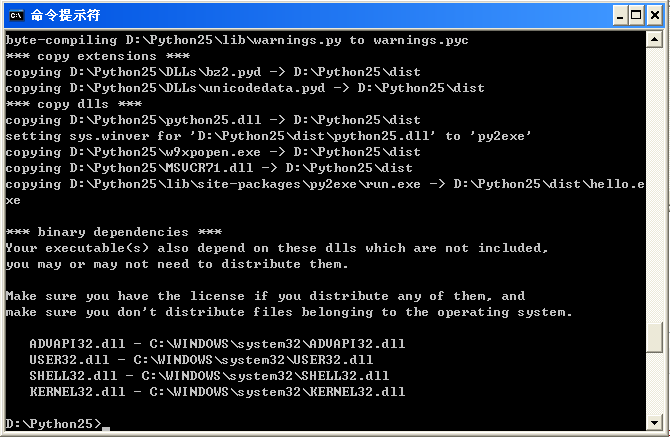
[](http://photo.blog.sina.com.cn/showpic.html#blogid=649ebd460100hpgh&url=http://static15.photo.sina.com.cn/orignal/649ebd46g7f11bb301bee)  
  
然后切换到你的python25目录，这里我的python25目录为d:\python25

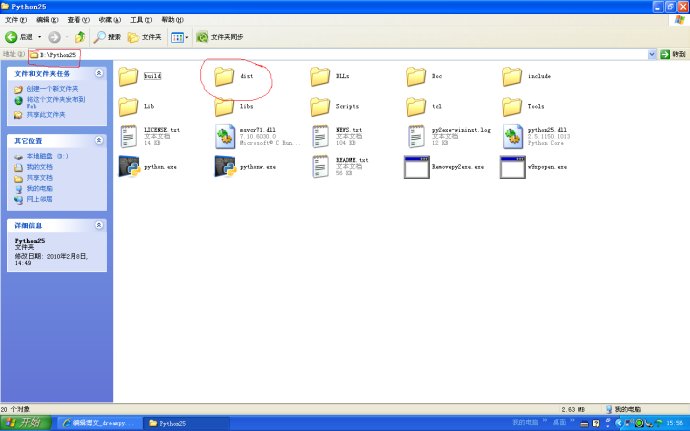
[](http://photo.blog.sina.com.cn/showpic.html#blogid=649ebd460100hpgh&url=http://static15.photo.sina.com.cn/orignal/649ebd46g7f11ea3b802e)  
  
然后在cmd下输入：

python e:\setup.py py2exe

注：e:\setup.py是你的setup.py位置（此位置可自行替换）

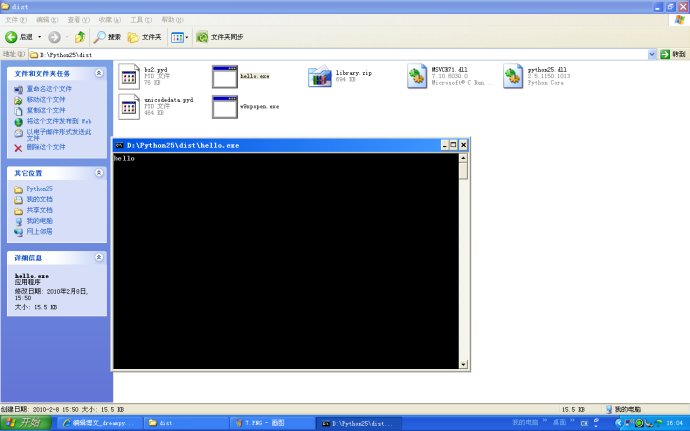
[](http://photo.blog.sina.com.cn/showpic.html#blogid=649ebd460100hpgh&url=http://static7.photo.sina.com.cn/orignal/649ebd46g7f1201e4e536)  
  
然后按Enter继续，看见一大堆输出信息。

[](http://photo.blog.sina.com.cn/showpic.html#blogid=649ebd460100hpgh&url=http://static13.photo.sina.com.cn/orignal/649ebd46g7f121a9c7c9c)  
  
搞定！现在去d:\python25目录看看是否生成了名为 dist 的文件夹

[](http://photo.blog.sina.com.cn/showpic.html#blogid=649ebd460100hpgh&url=http://static16.photo.sina.com.cn/orignal/649ebd46g7f1229e37cef)  
  
进入文件夹dist，点击helo.exe就可以运行你打包的程序了。

而dist文件夹就是你打包好的程序文件夹，记住，发布的时候要把里面的所有文件都一同发布！！！

运行结果：

[](http://photo.blog.sina.com.cn/showpic.html#blogid=649ebd460100hpgh&url=http://static3.photo.sina.com.cn/orignal/649ebd46g7f12461aa402)  
  
如果你要打包\*.pyw文件，只要把setup.py里的console改成windows即可

如果还不懂，可以试试这个图形化打包的软件：[点击这里查看](http://blog.sina.com.cn/s/blog_649ebd460100hoaz.html)。再不懂可以留言问我！

|  |
| --- |
| py2exe是一种python发布工具，可以用来将 python 脚本转换成 windows 下的可执行程序，而且不需要安装 python 环境便可运行。  py2exe 现在可以用来创建使用了 wxPython, Tkinter, Pmw, PyGTK, pygame, win32com client 以及 server 等模块的程序。  要使用py2exe编译工具，首先要安装。python自身不带这个功能。  关于py2exe的基本用法：  1、首先需要我们自己手写一个setup.py程序，内容为：  from distutils.croe import setup ''' distutils 是python 内建的模块。这里从distutils里导入setup方法。''' import py2exe                            ''' 导入py2exe模块'''  setup(console=['myprogram.py'])   '''console后面跟的是一个列表。此处的myprogram.py是你要编译的.py程序 这里我使用了console这个参数，它表示的是程序运行在命令提示符下，也就是DOS窗口里。 或                                                      者说是CMD.exe里。如果你的python里使用了图形组建，即：GUI，那么就要把console改 成                                                      windows。'''  把上面这段代码保存后，我们切换到cmd.exe里，运行下面的命令：  python setup.py py2exe  就会出现编译过程，最后会生成两个文件夹：build 和dist ，其中dist目录里面存放的是你编译好的可执行文件myprogram.exe，此时，你就可以直接运行这个生成的.exe文件了。另一个build目 录是临时文件可以删除。当你在其他没有python环境的系统上运行必要的文件： \*.pyd,  library.zip,  MSVCR71.dll python\*.dll  这几个文件是必须要和生成的.exe 文件放在一起。  以上是最基本的编译方法。  不过在我测试的时候出现了问题。  我用pyqt4写了一个最简单的小程序，功能就是显示一个窗口。文件名称是window.py 代码如下：  #!/usr/bin/python #filename:window.py import sys from PyQt4 import QtCore,QtGui  class Mywindow(QtGui.QMainWindow): def \_\_init\_\_(self): QtGui.QMainWindow.\_\_init\_\_(self) self.setWindowTitle('my first pyqt window') self.resize(300,200)  app=QtGui.QApplication(sys.argv) mywindow=Mywindow() mywindow.show() app.exec\_()  用来编译的setup.py代码如下：  #!/usr/bin/python #filename:setup.py  from distutils.core import setup import py2exe  setup(windows=["window.py"])  整个编译过程都很顺利，可是当我运行编译好的window.exe文件的时候，出现了错误提示：  Traceback (most recent call last): File "window.py", line 5, in <module> File "PyQt4\QtCore.pyc", line 12, in <module> File "PyQt4\QtCore.pyc", line 10, in \_\_load ImportError: No module named sip  这让我很困惑，后来去搜索了一下得到如下解决方法：  #!/usr/bin/python #filename:setup.py  from distutils.core import setup import py2exe  setup(windows=[{"script":"myprogram.py"}], options={"py2exe":{"includes":["sip"]}})  这样再次编译一下，程序可以正常运行，这其中的关键是在与setup这个函数可以带很多参数，我就是因为不了解setup所以出现了比较困惑的问题。  关于更多py2exe的使用方法和更多参数介绍，目前还没找到更详细的介绍，只在<http://li2z.cn/>里面找到了这一段：不过我觉得写算是比较详细的了。其中我做了些小小的改动。在此对作者表示歉意！     1. 打包文件最好要和你的主程序文件放在同一目录下.我曾经试着把打包文件放在上级目录,结果在搜索依赖库的时候遇到了很多麻烦. 2. 打包文件本身其实也是一个python的程序,虽然一般只有一个setup函数.此函数可以有 name,description,version,console,windows,options,data\_files 等参数. 3. 要生成一个或多个命令行的exe,setup需要一个List型的名为console参数,指名源py文件. 4. 要生成一个或多个GUI的exe,setup需要一个List型的名为windows参数,指名源py文件.List的每项都可以有图标等选项, 就是icon\_resources哪里,图标也是个Tuple的List,但是windows一般只会显示出第一个图标,所以一个就够了. 5. 如果你想用png文件直接当图标,那很可能会在生成exe的时候,引起python崩溃,所以还是赶紧转成ico吧 6. 如果你和我一样,生成的GUI的exe文件,在执行的时候,报 ImportError: No module named cairo ,那最好在setup函数里加上options,里面有include cairo的选项. 7. 如果你也有数据文件(比如glade文件,图片文件等),py2exe默认并不会帮你一起打包,你需要在setup函数中加传 data\_files参数.data\_files需要是一个List,每项可以是直接指向数据文件的字符串,也可以是一个Tuple;如果是前者,数据文 件会直接在生成的dist目录下面,如果是后者,Tuple的第一个元素是目标目录,第二个元素是是个源文件名的List,效果是把所有的源文件存放在 dist下的指定的目标目录下.(呃,好长好拗口…) 8. 在py2exe生成的环境中,\_\_file\_\_这个应用将失效,这个问题的解决方法可以参看<http://www.py2exe.org/index.cgi/WhereAmI> 这里.个人觉得Alternate Solution更加靠谱. 9. 好了,现在如果你已经写好了打包文件(win\_packer.py),也对源代码进行了应有的修改,你就可以执行 python win\_packer.py py2exe 来生成exe了,执行此命令会在当前目录下多出一个 dist 目录,你只要把dist目录复制出来,运行里面的exe,就可以了 10. 最后,如果你的GUI程序还是运行不了,你还需要手工将你的GTK+安装安装目录(默认是C:\GTK)下的etc和lib目录复制到dist目录。 |

### py2exe用法

from:http://www.dev.idv.tw/mediawiki/index.php/Py2exe簡介  
Description   
py2exe是一個工具程式，可以將python程式轉換成為單一的執行檔案。在某些情況下使用者也許不想安裝整個python套件或是相關的檔  
案。透過  
py2exe可以將您的工具包裝成單一的執行檔，及數個相關檔案。使用者只要下載這幾個檔案就可以執行該程式了。因為py2exe會自動將該工具的或自動  
搜尋您的python程式中所有用到的模組及檔案加以集中包裝至這數個檔案中。  
關於py2exe的相關資訊可以在下面的網站中找到。  
[url=http://sourceforge.net/projects/py2exe/]http://sourceforge.net/projects/py2exe/[/url]  
且讓我用一個範例來說明。我寫了一個wxPython的程式。大家都知道wxPython就是python及wxWidgets的結合。對  
於使用者來說，  
要同時在他們的電腦中安裝python及wxPython，其實是個蠻沈重的負擔。所以我打算使用py2exe將我寫的工具包裝起來。我的程式總共有兩個  
檔案分別是：HtmlConv.py及MainFrame.py。  
此時，我們需要準備小小的python程式，透過這個程式來呼叫py2exe包裝我們的程式。我們將這個程式取名為setup.py。其內容如下：  
# setup.py  
from distutils.core import setup  
import py2exe  
setup(windows=["HtmlConv.py"])  
讓我們來看一下這個檔案的內容。首先，我們可以看到這個python程式引入了distutils這個模組，這個模組是python的標準模  
組之一。py2exe其實算是他的擴充模組之一，所以需要載入這個模組中的setup部分。然後，再引入py2exe這個模組。  
因為我們的程式很簡單，所以只需要簡單的呼叫setup  
function。並且在在參數中指定您的主要程式的檔名。您或許會懷疑，我們怎麼不用指定MainFrame.py這個檔案呢？因為  
MainFrame.py是被HtmlConv.py這個檔案所import的，py2exe在處理HtmlConv.py時，就會處理到  
MainFrame.py了。 有了這三個檔案後，我們需要透過command的方式呼叫python來執行我們的setup.py程式。指令如下：  
python.exe setup.py py2exe  
您也可以透過下面的命令來瞭解py2exe提供了哪些command line參數：  
python.exe setup.py py2exe --help  
接著，您會看到有些訊息跑出來，如果都沒有錯誤出現，您會看到出現了兩個新的目錄名為build以及dist。其中build是py2exe再製作過程中的一個暫存目錄。您可以不予理會。而dist目錄中就是您要的東西了。我們的範例中dist目錄產生了下面這些檔案：  
HtmlConv.exe  
library.zip  
python23.dll  
w9xpopen.exe  
wxc.pyd  
xmsw24uh.dll  
使用者只需要將這幾個檔案拷貝回去執行HtmlConv.exe就可以執行我們的工具程式。這對於使用者來說，無疑是非常方便的。  
現在讓我們重新看看setup.py這個檔案的內容。因為我的工具程式是視窗介面的，不需要出現console視窗。因此，我透過指定  
windows=["HtmlConv.py"] 的方式來指定主要檔案。如果您的程式是屬於console介面的，您就應該使用  
console=["xxx.py"] 的方式來指定您的程式。此外，py2exe也支援Windows  
Service及COM元件類型的程式。分別使用下面的方式指定之：  
service=["MyService"]  
或  
com\_server=["win32com.server.interp"]  
py2exe其實是透過python的modulefinder來找到所需的module，再將他們包裝在一起。您在dist目錄中所看到的  
library.zip就是這些module的壓縮檔。不過，也正因為如此，如果是一些您程式中所需要的資料檔，如圖片，音效等等。py2exe並沒有辦  
法幫您自動包裝起來。不過，您還是可以透過手動的方式在setup.py中指定setup  
function的參數來告訴py2exe您還有哪些檔案。假設我們原來程式有一個子目錄名為bmp，下面放了一些程式用到的圖形檔。另外，還有一個子目  
錄名為wav，放了一些程式會用到的wav檔案。此時，我們的setup function參數應修改為如下：  
setup(windows=["HtmlConv.py"],  
      data\_files=[("bmp",  
                   ["bmp/logo.bmp", "bmp/title.gif"]),  
                  ("wav",  
                   glob.glob("sounds\*.wav"))])  
data\_files這個參數是一個list，list內的每個元素都是一個pair。pair的第一個元素是您所指定的檔案的目的地。第二個元素  
是一個list，  
列出了那些檔案的來源。且讓我們仔細看看上面的範例。第一組pair，告訴py2exe在dist目錄下面另外建立一個bmp目錄，並且將原來bmp目錄  
中的的logo.bmp及title.gif檔案複製到dist目錄中的bmp子目錄。如果您不希望建立子目錄，可以將目的目錄指定為空白字串。此時，  
py2exe會將您指定的檔案複製到dist目錄下。  
而第二組pair告訴py2exe在dist目錄中建立wav子目錄。然後將sounds目錄中的所有副檔名為wav的檔案拷貝過去。由於我們用了  
glob的弁遄C所以別忘了在setup.py中import glob模組。  
預設的狀況下，製作出來的執行檔所使用的icon是windows下的預設icon。其實，您可以自己將他修改成其他的icon。首先您先要準備好一個  
icon檔案。以上面的例子來說，我們準備一個名為HtmlConv.ico的檔案(檔名未必要與python程式相同)。接著，將上面的範例修改如下：  
setup(windows=["HtmlConv.py", {"script":"HtmlConv.py", "icon\_resources":[(1, "HtmlConv.ico")]}],  
      data\_files=[("bmp",  
                   ["bmp/logo.bmp", "bmp/title.gif"]),  
                  ("wav",  
                   glob.glob("sounds\*.wav"))])  
重新製作執行檔後，該執行檔就會是您所指定的icon囉。py2exe的使用對於會使用python的programmer來說，是非常簡單。而對於使用者而言，不用安裝不必要的東西就可以使用。  
py2exe實在是非常值得推薦的工具。   
另外一个比较好的例子：  
from:http://d887419.spaces.live.com/blog/cns!481a2ec4e1e2501d!682.entry  
from distutils.core import setup  
import py2exe  
setup(  
    name = 'testGlade',  
    description = 'Some Test including Glade, Python and GTK in win32',  
    version = '1.0',  
    windows = [  
                  {  
                      'script': 'xxxxx.py',  
                      'icon\_resources': [(1, "your\_program\_icon\_file.ico")],  
                  }  
              ],  
    options = {  
                  'py2exe': {  
                      'packages':'encodings',  
                      'includes': 'cairo, pango, pangocairo, atk, gobject', # including the dynamic library  
                  }  
              },  
    data\_files=[  
                   'xxxxx.glade',     # xxxxx.glade is the project file made by Glade  
                   'readme.txt'  
               ]  
)  
下面是py2exe的help：  
>>> help(py2exe)  
Help on package py2exe:  
NAME  
    py2exe - builds windows executables from Python scripts  
FILE  
    c:\python25\lib\site-packages\py2exe\\_\_init\_\_.py  
DESCRIPTION  
    New keywords for distutils' setup function specify what to build:  
        console  
            list of scripts to convert into console exes  
        windows  
            list of scripts to convert into gui exes  
        service  
            list of module names containing win32 service classes  
        com\_server  
            list of module names containing com server classes  
        ctypes\_com\_server  
            list of module names containing com server classes  
        zipfile  
            name of shared zipfile to generate, may specify a subdirectory,  
            defaults to 'library.zip'  
    py2exe options, to be specified in the options keyword to the setup function  
:  
        unbuffered - if true, use unbuffered binary stdout and stderr  
        optimize - string or int (0, 1, or 2)  
        includes - list of module names to include  
        packages - list of packages to include with subpackages  
        ignores - list of modules to ignore if they are not found  
        excludes - list of module names to exclude  
        dll\_excludes - list of dlls to exclude  
        dist\_dir - directory where to build the final files  
        typelibs - list of gen\_py generated typelibs to include (XXX more text n  
eeded)  
    Items in the console, windows, service or com\_server list can also be  
    dictionaries to further customize the build process.  The following  
    keys in the dictionary are recognized, most are optional:  
        modules (SERVICE, COM) - list of module names (required)  
        script (EXE) - list of python scripts (required)  
        dest\_base - directory and basename for the executable  
                    if a directory is contained, must be the same for all target  
s  
        create\_exe (COM) - boolean, if false, don't build a server exe  
        create\_dll (COM) - boolean, if false, don't build a server dll  
        bitmap\_resources - list of 2-tuples (id, pathname)  
        icon\_resources - list of 2-tuples (id, pathname)  
        other\_resources - list of 3-tuples (resource\_type, id, datastring)  
PACKAGE CONTENTS  
    boot\_com\_servers  
    boot\_common  
    boot\_ctypes\_com\_server  
    boot\_service  
    build\_exe  
    mf  
    py2exe\_util  
    resources (package)  
CLASSES  
    distutils.dist.Distribution  
        Distribution  
    class Distribution(distutils.dist.Distribution)  
     |  Methods defined here:  
     |  
     |  \_\_init\_\_(self, attrs)  
     |  
     |  ----------------------------------------------------------------------  
     |  Methods inherited from distutils.dist.Distribution:  
     |  
     |  announce(self, msg, level=1)  
     |  
     |  dump\_option\_dicts(self, header=None, commands=None, indent='')  
     |  
     |  finalize\_options(self)  
     |      Set final values for all the options on the Distribution  
     |      instance, analogous to the .finalize\_options() method of Command  
     |      objects.  
     |  
     |  find\_config\_files(self)  
     |      Find as many configuration files as should be processed for this  
     |      platform, and return a list of filenames in the order in which they  
     |      should be parsed.  The filenames returned are guaranteed to exist  
     |      (modulo nasty race conditions).  
     |  
     |      There are three possible config files: distutils.cfg in the  
     |      Distutils installation directory (ie. where the top-level  
     |      Distutils \_\_inst\_\_.py file lives), a file in the user's home  
     |      directory named .pydistutils.cfg on Unix and pydistutils.cfg  
     |      on Windows/Mac, and setup.cfg in the current directory.  
     |  
     |  get\_command\_class(self, command)  
     |      Return the class that implements the Distutils command named by  
     |      'command'.  First we check the 'cmdclass' dictionary; if the  
     |      command is mentioned there, we fetch the class object from the  
     |      dictionary and return it.  Otherwise we load the command module  
     |      ("distutils.command." + command) and fetch the command class from  
     |      the module.  The loaded class is also stored in 'cmdclass'  
     |      to speed future calls to 'get\_command\_class()'.  
     |  
     |      Raises DistutilsModuleError if the expected module could not be  
     |      found, or if that module does not define the expected class.  
     |  
     |  get\_command\_list(self)  
     |      Get a list of (command, description) tuples.  
     |      The list is divided into "standard commands" (listed in  
     |      distutils.command.\_\_all\_\_) and "extra commands" (mentioned in  
     |      self.cmdclass, but not a standard command).  The descriptions come  
     |      from the command class attribute 'description'.  
     |  
     |  get\_command\_obj(self, command, create=1)  
     |      Return the command object for 'command'.  Normally this object  
     |      is cached on a previous call to 'get\_command\_obj()'; if no command  
     |      object for 'command' is in the cache, then we either create and  
     |      return it (if 'create' is true) or return None.  
     |  
     |  get\_command\_packages(self)  
     |      Return a list of packages from which commands are loaded.  
     |  
     |  get\_option\_dict(self, command)  
     |      Get the option dictionary for a given command.  If that  
     |      command's option dictionary hasn't been created yet, then create it  
     |      and return the new dictionary; otherwise, return the existing  
     |      option dictionary.  
     |  
     |  handle\_display\_options(self, option\_order)  
     |      If there were any non-global "display-only" options  
     |      (--help-commands or the metadata display options) on the command  
     |      line, display the requested info and return true; else return  
     |      false.  
     |  
     |  has\_c\_libraries(self)  
     |  
     |  has\_data\_files(self)  
     |  
     |  has\_ext\_modules(self)  
     |  
     |  has\_headers(self)  
     |  
     |  has\_modules(self)  
     |  
     |  has\_pure\_modules(self)  
     |  
     |  has\_scripts(self)  
     |  
     |  is\_pure(self)  
     |  
     |  parse\_command\_line(self)  
     |      Parse the setup script's command line, taken from the  
     |      'script\_args' instance attribute (which defaults to 'sys.argv[1:]'  
     |      -- see 'setup()' in core.py).  This list is first processed for  
     |      "global options" -- options that set attributes of the Distribution  
     |      instance.  Then, it is alternately scanned for Distutils commands  
     |      and options for that command.  Each new command terminates the  
     |      options for the previous command.  The allowed options for a  
     |      command are determined by the 'user\_options' attribute of the  
     |      command class -- thus, we have to be able to load command classes  
     |      in order to parse the command line.  Any error in that 'options'  
     |      attribute raises DistutilsGetoptError; any error on the  
     |      command-line raises DistutilsArgError.  If no Distutils commands  
     |      were found on the command line, raises DistutilsArgError.  Return  
     |      true if command-line was successfully parsed and we should carry  
     |      on with executing commands; false if no errors but we shouldn't  
     |      execute commands (currently, this only happens if user asks for  
     |      help).  
     |  
     |  parse\_config\_files(self, filenames=None)  
     |  
     |  print\_command\_list(self, commands, header, max\_length)  
     |      Print a subset of the list of all commands -- used by  
     |      'print\_commands()'.  
     |  
     |  print\_commands(self)  
     |      Print out a help message listing all available commands with a  
     |      description of each.  The list is divided into "standard commands"  
     |      (listed in distutils.command.\_\_all\_\_) and "extra commands"  
     |      (mentioned in self.cmdclass, but not a standard command).  The  
     |      descriptions come from the command class attribute  
     |      'description'.  
     |  
     |  reinitialize\_command(self, command, reinit\_subcommands=0)  
     |      Reinitializes a command to the state it was in when first  
     |      returned by 'get\_command\_obj()': ie., initialized but not yet  
     |      finalized.  This provides the opportunity to sneak option  
     |      values in programmatically, overriding or supplementing  
     |      user-supplied values from the config files and command line.  
     |      You'll have to re-finalize the command object (by calling  
     |      'finalize\_options()' or 'ensure\_finalized()') before using it for  
     |      real.  
     |  
     |      'command' should be a command name (string) or command object.  If  
     |      'reinit\_subcommands' is true, also reinitializes the command's  
     |      sub-commands, as declared by the 'sub\_commands' class attribute (if  
     |      it has one).  See the "install" command for an example.  Only  
     |      reinitializes the sub-commands that actually matter, ie. those  
     |      whose test predicates return true.  
     |  
     |      Returns the reinitialized command object.  
     |  
     |  run\_command(self, command)  
     |      Do whatever it takes to run a command (including nothing at all,  
     |      if the command has already been run).  Specifically: if we have  
     |      already created and run the command named by 'command', return  
     |      silently without doing anything.  If the command named by 'command'  
     |      doesn't even have a command object yet, create one.  Then invoke  
     |      'run()' on that command object (or an existing one).  
     |  
     |  run\_commands(self)  
     |      Run each command that was seen on the setup script command line.  
     |      Uses the list of commands found and cache of command objects  
     |      created by 'get\_command\_obj()'.  
     |  
     |  ----------------------------------------------------------------------  
     |  Data and other attributes inherited from distutils.dist.Distribution:  
     |  
     |  common\_usage = "Common commands: (see '--help-commands' for more...'\n..  
.  
     |  
     |  display\_option\_names = ['help\_commands', 'name', 'version', 'fullname'..  
.  
     |  
     |  display\_options = [('help-commands', None, 'list all available command..  
.  
     |  
     |  global\_options = [('verbose', 'v', 'run verbosely (default)', 1), ('qu..  
.  
     |  
     |  negative\_opt = {'quiet': 'verbose'}  
DATA  
    \_\_version\_\_ = '0.6.9'  
VERSION  
    0.6.9