13.3 解析命令行选项¶

问题¶

你的程序如何能够解析命令行选项（位于sys.argv中）

解决方案¶

`argparse` 模块可被用来解析命令行选项。下面一个简单例子演示了最基本的用法：

```
# search.py
'''
Hypothetical command-line tool for searching a collection of
files for one or more text patterns.
'''
import argparse
parser = argparse.ArgumentParser(description='Search some files')

parser.add_argument(dest='filenames',metavar='filename', nargs='*')

parser.add_argument('-p', '--pat',metavar='pattern', required=True,
                    dest='patterns', action='append',
                    help='text pattern to search for')

parser.add_argument('-v', dest='verbose', action='store_true',
                    help='verbose mode')

parser.add_argument('-o', dest='outfile', action='store',
                    help='output file')

parser.add_argument('--speed', dest='speed', action='store',
                    choices={'slow','fast'}, default='slow',
                    help='search speed')

args = parser.parse_args()

# Output the collected arguments
print(args.filenames)
print(args.patterns)
print(args.verbose)
print(args.outfile)
print(args.speed)
```

该程序定义了一个如下使用的命令行解析器：

```
bash % python3 search.py -h
usage: search.py [-h] [-p pattern] [-v] [-o OUTFILE] [--speed {slow,fast}]
                 [filename [filename ...]]

Search some files

positional arguments:
  filename

optional arguments:
  -h, --help            show this help message and exit
  -p pattern, --pat pattern
                        text pattern to search for
  -v                    verbose mode
  -o OUTFILE            output file
  --speed {slow,fast}   search speed
```

下面的部分演示了程序中的数据部分。仔细观察print()语句的打印输出。

```
bash % python3 search.py foo.txt bar.txt
usage: search.py [-h] -p pattern [-v] [-o OUTFILE] [--speed {fast,slow}]
```

```
                [filename [filename ...]]
search.py: error: the following arguments are required: -p/--pat

bash % python3 search.py -v -p spam --pat=eggs foo.txt bar.txt
filenames = ['foo.txt', 'bar.txt']
patterns  = ['spam', 'eggs']
verbose   = True
outfile   = None
speed     = slow

bash % python3 search.py -v -p spam --pat=eggs foo.txt bar.txt -o results
filenames = ['foo.txt', 'bar.txt']
patterns  = ['spam', 'eggs']
verbose   = True
outfile   = results
speed     = slow

bash % python3 search.py -v -p spam --pat=eggs foo.txt bar.txt -o results \
           --speed=fast
filenames = ['foo.txt', 'bar.txt']
patterns  = ['spam', 'eggs']
verbose   = True
outfile   = results
speed     = fast
```

对于选项值的进一步处理由程序来决定，用你自己的逻辑来替代 `print()` 函数。

讨论¶

`argparse` 模块是标准库中最大的模块之一，拥有大量的配置选项。 本节只是演示了其中最基础的一些特性，帮助你入门。

为了解析命令行选项，你首先要创建一个 `ArgumentParser` 实例， 并使用 `add_argument()` 方法声明你想要支持的选项。 在每个 `add_argument()` 调用中，`dest` 参数指定解析结果被指派给属性的名字。 `metavar` 参数被用来生成帮助信息。`action` 参数指定跟属性对应的处理逻辑， 通常的值为 `store` ,被用来存储某个值或将多个参数值收集到一个列表中。 下面的参数收集所有剩余的命令行参数到一个列表中。在本例中它被用来构造一个文件名列表：

```
parser.add_argument(dest='filenames',metavar='filename', nargs='*')
```

下面的参数根据参数是否存在来设置一个 `Boolean` 标志：

```
parser.add_argument('-v', dest='verbose', action='store_true',
                    help='verbose mode')
```

下面的参数接受一个单独值并将其存储为一个字符串：

```
parser.add_argument('-o', dest='outfile', action='store',
                    help='output file')
```

下面的参数说明允许某个参数重复出现多次，并将它们追加到一个列表中去。 `required` 标志表示该参数至少要有一个。`-p` 和 `--pat` 表示两个参数名形式都可使用。

```
parser.add_argument('-p', '--pat',metavar='pattern', required=True,
                    dest='patterns', action='append',
                    help='text pattern to search for')
```

最后，下面的参数说明接受一个值，但是会将其和可能的选择值做比较，以检测其合法性：

```
parser.add_argument('--speed', dest='speed', action='store',
                    choices={'slow','fast'}, default='slow',
                    help='search speed')
```

一旦参数选项被指定，你就可以执行 `parser.parse()` 方法了。 它会处理 `sys.argv` 的值并返回一个结果实例。 每个参数值会被设置成该实例中 `add_argument()` 方法的 `dest` 参数指定的属性值。

还很多种其他方法解析命令行选项。 例如，你可能会手动的处理 `sys.argv` 或者使用 `getopt` 模块。 但是，如果你采用

本节的方式，将会减少很多冗余代码，底层细节 `argparse` 模块已经帮你处理了。 你可能还会碰到使用 `optparse` 库解析选项的代码。 尽管 `optparse` 和 `argparse` 很像，但是后者更先进，因此在新的程序中你应该使用它。