Argparse 教程

发布 *3.11.4*

Guido van Rossum and the Python development team

七月 19, 2023

Python Software Foundation
Email: docs@python.org

Contents

作者 Tshepang Lekhonkhobe

这篇教程旨在作为 `argparse` 的入门介绍，此模块是 Python 标准库中推荐的命令行解析模块。

备注： There are two other modules that fulfill the same task, namely `getopt` (an equivalent for `getopt()` from the C language) and the deprecated `optparse`. Note also that `argparse` is based on `optparse`, and therefore very similar in terms of usage.

1 概念

让我们利用 **ls** 命令来展示我们将要在这篇入门教程中探索的功能：

```
$ ls
cpython  devguide  prog.py  pypy  rm-unused-function.patch
$ ls pypy
ctypes_configure  demo  dotviewer  include  lib_pypy  lib-python ...
$ ls -l
total 20
drwxr-xr-x 19 wena wena 4096 Feb 18 18:51 cpython
drwxr-xr-x  4 wena wena 4096 Feb  8 12:04 devguide
-rwxr-xr-x  1 wena wena  535 Feb 19 00:05 prog.py
drwxr-xr-x 14 wena wena 4096 Feb  7 00:59 pypy
-rw-r--r--  1 wena wena  741 Feb 18 01:01 rm-unused-function.patch
$ ls --help
Usage: ls [OPTION]... [FILE]...
List information about the FILEs (the current directory by default).
Sort entries alphabetically if none of -cftuvSUX nor --sort is specified.
...
```

我们可以从这四个命令中学到几个概念：

- **ls** 是一个即使在运行的时候没有提供任何选项，也非常有用的命令。在默认情况下他会输出当前文件夹包含的文件和文件夹。
- 如果我们想要使用比它默认提供的更多功能，我们需要告诉该命令更多信息。在这个例子里，我们想要查看一个不同的目录，`pypy`。我们所做的是指定所谓的位置参数。之所以这样命名，是因为程序应该如何处理该参数值，完全取决于它在命令行出现的位置。更能体现这个概念的命令如 **cp**，它最基本的用法是 `cp SRC DEST`。第一个位置参数指的是 * 你想要复制的 *，第二个位置参数指的是 * 你想要复制到的位置 *。
- 现在假设我们想要改变这个程序的行为。在我们的例子中，我们不仅仅只是输出每个文件的文件名，还输出了更多信息。在这个例子中，`-l` 被称为可选参数。
- 这是一段帮助文档的文字。它是非常有用的，因为当你遇到一个你从未使用过的程序时，你可以通过阅读它的帮助文档来弄清楚它是如何运行的。

2 基础

让我们从一个简单到（几乎）什么也做不了的例子开始：

```
import argparse
parser = argparse.ArgumentParser()
parser.parse_args()
```

以下是该代码的运行结果：

```
$ python3 prog.py
$ python3 prog.py --help
usage: prog.py [-h]

options:
  -h, --help  show this help message and exit
$ python3 prog.py --verbose
usage: prog.py [-h]
prog.py: error: unrecognized arguments: --verbose
$ python3 prog.py foo
usage: prog.py [-h]
prog.py: error: unrecognized arguments: foo
```

程序运行情况如下：

- 在没有任何选项的情况下运行脚本不会在标准输出显示任何内容。这没有什么用处。
- 第二行代码开始展现出 argparse 模块的作用。我们几乎什么也没有做，但已经得到一条很好的帮助信息。
- --help 选项，也可缩写为 -h，是唯一一个可以直接使用的选项（即不需要指定该选项的内容）。指定任何内容都会导致错误。即便如此，我们也能直接得到一条有用的用法信息。

3 位置参数介绍

举个例子：

```
import argparse
parser = argparse.ArgumentParser()
parser.add_argument("echo")
args = parser.parse_args()
print(args.echo)
```

运行此程序：

```
$ python3 prog.py
usage: prog.py [-h] echo
prog.py: error: the following arguments are required: echo
$ python3 prog.py --help
usage: prog.py [-h] echo

positional arguments:
  echo

options:
  -h, --help  show this help message and exit
$ python3 prog.py foo
foo
```

程序运行情况如下：

- We've added the add_argument() method, which is what we use to specify which command-line options the program is willing to accept. In this case, I've named it echo so that it's in line with its function.
- 现在调用我们的程序必须要指定一个选项。
- The parse_args() method actually returns some data from the options specified, in this case, echo.
- 这一变量是 argparse 免费施放的某种“魔法”（即是说，不需要指定哪个变量是存储哪个值的）。你也可以注意到，这一名称与传递给方法的字符串参数一致，都是 echo。

然而请注意，尽管显示的帮助看起来清楚完整，但它可以比现在更有帮助。比如我们可以知道 echo 是一个位置参数，但我们除了靠猜或者看源代码，没法知道它是用来干什么的。所以，我们可以把它改造得更有用：

```
import argparse
parser = argparse.ArgumentParser()
parser.add_argument("echo", help="echo the string you use here")
args = parser.parse_args()
print(args.echo)
```

然后我们得到：

```
$ python3 prog.py -h
usage: prog.py [-h] echo
```

(下页继续)

(续上页)

```
positional arguments:
  echo        echo the string you use here

options:
  -h, --help  show this help message and exit
```

现在，来做一些更有用的事情：

```
import argparse
parser = argparse.ArgumentParser()
parser.add_argument("square", help="display a square of a given number")
args = parser.parse_args()
print(args.square**2)
```

以下是该代码的运行结果：

```
$ python3 prog.py 4
Traceback (most recent call last):
  File "prog.py", line 5, in <module>
    print(args.square**2)
TypeError: unsupported operand type(s) for ** or pow(): 'str' and 'int'
```

进展不太顺利。那是因为 argparse 会把我们传递给它的选项视作为字符串，除非我们告诉它别这样。所以，让我们来告诉 argparse 来把这一输入视为整数：

```
import argparse
parser = argparse.ArgumentParser()
parser.add_argument("square", help="display a square of a given number",
                    type=int)
args = parser.parse_args()
print(args.square**2)
```

以下是该代码的运行结果：

```
$ python3 prog.py 4
16
$ python3 prog.py four
usage: prog.py [-h] square
prog.py: error: argument square: invalid int value: 'four'
```

做得不错。当这个程序在收到错误的无效的输入时，它甚至能在执行计算之前先退出，还能显示很有帮助的错误信息。

4 可选参数介绍

到目前为止，我们一直在研究位置参数。让我们看看如何添加可选的：

```
import argparse
parser = argparse.ArgumentParser()
parser.add_argument("--verbosity", help="increase output verbosity")
args = parser.parse_args()
if args.verbosity:
    print("verbosity turned on")
```

和输出：

```
$ python3 prog.py --verbosity 1
verbosity turned on
$ python3 prog.py
$ python3 prog.py --help
usage: prog.py [-h] [--verbosity VERBOSITY]

options:
  -h, --help            show this help message and exit
  --verbosity VERBOSITY
                        increase output verbosity
$ python3 prog.py --verbosity
usage: prog.py [-h] [--verbosity VERBOSITY]
prog.py: error: argument --verbosity: expected one argument
```

程序运行情况如下：

- 这一程序被设计为当指定 `--verbosity` 选项时显示某些东西，否则不显示。
- To show that the option is actually optional, there is no error when running the program without it. Note that by default, if an optional argument isn't used, the relevant variable, in this case `args.verbosity`, is given `None` as a value, which is the reason it fails the truth test of the `if` statement.
- 帮助信息有点不同。
- 使用 `--verbosity` 选项时，必须指定一个值，但可以是任何值。

上述例子接受任何整数值作为 `--verbosity` 的参数，但对于我们的简单程序而言，只有两个值有实际意义：`True` 或者 `False`。让我们据此修改代码：

```
import argparse
parser = argparse.ArgumentParser()
parser.add_argument("--verbose", help="increase output verbosity",
                    action="store_true")
args = parser.parse_args()
if args.verbose:
    print("verbosity turned on")
```

和输出：

```
$ python3 prog.py --verbose
verbosity turned on
$ python3 prog.py --verbose 1
usage: prog.py [-h] [--verbose]
prog.py: error: unrecognized arguments: 1
$ python3 prog.py --help
usage: prog.py [-h] [--verbose]

options:
  -h, --help  show this help message and exit
  --verbose   increase output verbosity
```

程序运行情况如下：

- The option is now more of a flag than something that requires a value. We even changed the name of the option to match that idea. Note that we now specify a new keyword, `action`, and give it the value `"store_true"`. This means that, if the option is specified, assign the value `True` to `args.verbose`. Not specifying it implies `False`.
- 当你为其指定一个值时，它会报错，符合作为标志的真正的精神。
- 留意不同的帮助文字。

4.1 短选项

如果你熟悉命令行的用法，你会发现我还没讲到这一选项的短版本。这也很简单：

```
import argparse
parser = argparse.ArgumentParser()
parser.add_argument("-v", "--verbose", help="increase output verbosity",
                    action="store_true")
args = parser.parse_args()
if args.verbose:
    print("verbosity turned on")
```

效果就像这样：

```
$ python3 prog.py -v
verbosity turned on
$ python3 prog.py --help
usage: prog.py [-h] [-v]

options:
  -h, --help     show this help message and exit
  -v, --verbose  increase output verbosity
```

可以注意到，这一新的能力也反映在帮助文本里。

5 结合位置参数和可选参数

我们的程序变得越来越复杂了：

```
import argparse
parser = argparse.ArgumentParser()
parser.add_argument("square", type=int,
                    help="display a square of a given number")
parser.add_argument("-v", "--verbose", action="store_true",
                    help="increase output verbosity")
args = parser.parse_args()
answer = args.square**2
if args.verbose:
    print(f"the square of {args.square} equals {answer}")
else:
    print(answer)
```

接着是输出：

```
$ python3 prog.py
usage: prog.py [-h] [-v] square
prog.py: error: the following arguments are required: square
$ python3 prog.py 4
16
$ python3 prog.py 4 --verbose
the square of 4 equals 16
$ python3 prog.py --verbose 4
the square of 4 equals 16
```

- 我们带回了一个位置参数，结果发生了报错。
- 注意顺序无关紧要。

给我们的程序加上接受多个冗长度的值，然后实际来用用：

```
import argparse
parser = argparse.ArgumentParser()
parser.add_argument("square", type=int,
                    help="display a square of a given number")
parser.add_argument("-v", "--verbosity", type=int,
                    help="increase output verbosity")
args = parser.parse_args()
answer = args.square**2
if args.verbosity == 2:
    print(f"the square of {args.square} equals {answer}")
elif args.verbosity == 1:
    print(f"{args.square}^2 == {answer}")
else:
    print(answer)
```

和输出：

```
$ python3 prog.py 4
16
$ python3 prog.py 4 -v
usage: prog.py [-h] [-v VERBOSITY] square
prog.py: error: argument -v/--verbosity: expected one argument
$ python3 prog.py 4 -v 1
4^2 == 16
$ python3 prog.py 4 -v 2
the square of 4 equals 16
$ python3 prog.py 4 -v 3
16
```

除了最后一个，看上去都不错。最后一个暴露了我们的程序中有一个 bug。我们可以通过限制 `--verbosity` 选项可以接受的值来修复它：

```
import argparse
parser = argparse.ArgumentParser()
parser.add_argument("square", type=int,
                    help="display a square of a given number")
parser.add_argument("-v", "--verbosity", type=int, choices=[0, 1, 2],
                    help="increase output verbosity")
args = parser.parse_args()
answer = args.square**2
if args.verbosity == 2:
    print(f"the square of {args.square} equals {answer}")
elif args.verbosity == 1:
    print(f"{args.square}^2 == {answer}")
else:
    print(answer)
```

和输出：

```
$ python3 prog.py 4 -v 3
usage: prog.py [-h] [-v {0,1,2}] square
prog.py: error: argument -v/--verbosity: invalid choice: 3 (choose from 0, 1, 2)
$ python3 prog.py 4 -h
usage: prog.py [-h] [-v {0,1,2}] square

positional arguments:
  square                display a square of a given number

options:
  -h, --help            show this help message and exit
  -v {0,1,2}, --verbosity {0,1,2}
                        increase output verbosity
```

注意这一改变同时反应在错误信息和帮助信息里。

现在，让我们使用另一种的方式来改变冗长度。这种方式更常见，也和 CPython 的可执行文件处理它自己的冗长度参数的方式一致（参考 `python --help` 的输出）：

```
import argparse
parser = argparse.ArgumentParser()
parser.add_argument("square", type=int,
                    help="display the square of a given number")
parser.add_argument("-v", "--verbosity", action="count",
                    help="increase output verbosity")
args = parser.parse_args()
answer = args.square**2
if args.verbosity == 2:
    print(f"the square of {args.square} equals {answer}")
elif args.verbosity == 1:
    print(f"{args.square}^2 == {answer}")
else:
    print(answer)
```

我们引入了另一种动作"count"，来统计特定选项出现的次数。

```
$ python3 prog.py 4
16
$ python3 prog.py 4 -v
4^2 == 16
$ python3 prog.py 4 -vv
the square of 4 equals 16
$ python3 prog.py 4 --verbosity --verbosity
the square of 4 equals 16
$ python3 prog.py 4 -v 1
usage: prog.py [-h] [-v] square
prog.py: error: unrecognized arguments: 1
$ python3 prog.py 4 -h
usage: prog.py [-h] [-v] square

positional arguments:
  square           display a square of a given number

options:
  -h, --help       show this help message and exit
  -v, --verbosity  increase output verbosity
$ python3 prog.py 4 -vvv
16
```

- 是的，它现在比前一版本更像是一个标志（和 `action="store_true"` 相似）。这能解释它为什么报错。
- 它也表现得与“store_true”的行为相似。
- 这给出了一个关于 `count` 动作的效果的演示。你之前很可能应该已经看过这种用法。
- 如果你不添加 `-v` 标志，这一标志的值会是 `None`。
- 如期望的那样，添加该标志的长形态能够获得相同的输出。
- 可惜的是，对于我们的脚本获得的新能力，我们的帮助输出并没有提供很多信息，但我们总是可以通过改善文档来修复这一问题（比如通过 `help` 关键字参数）。
- 最后一个输出暴露了我们程序中的一个 bug。

让我们修复一下：

```
import argparse
parser = argparse.ArgumentParser()
```

（下页继续）

(续上页)

```
parser.add_argument("square", type=int,
                    help="display a square of a given number")
parser.add_argument("-v", "--verbosity", action="count",
                    help="increase output verbosity")
args = parser.parse_args()
answer = args.square**2

# bugfix: replace == with >=
if args.verbosity >= 2:
    print(f"the square of {args.square} equals {answer}")
elif args.verbosity >= 1:
    print(f"{args.square}^2 == {answer}")
else:
    print(answer)
```

这是它给我们的输出：

```
$ python3 prog.py 4 -vvv
the square of 4 equals 16
$ python3 prog.py 4 -vvvv
the square of 4 equals 16
$ python3 prog.py 4
Traceback (most recent call last):
  File "prog.py", line 11, in <module>
    if args.verbosity >= 2:
TypeError: '>=' not supported between instances of 'NoneType' and 'int'
```

- 第一组输出很好，修复了之前的 bug。也就是说，我们希望任何 >= 2 的值尽可能详尽。
- 第三组输出并不理想。

让我们修复那个 bug：

```
import argparse
parser = argparse.ArgumentParser()
parser.add_argument("square", type=int,
                    help="display a square of a given number")
parser.add_argument("-v", "--verbosity", action="count", default=0,
                    help="increase output verbosity")
args = parser.parse_args()
answer = args.square**2
if args.verbosity >= 2:
    print(f"the square of {args.square} equals {answer}")
elif args.verbosity >= 1:
    print(f"{args.square}^2 == {answer}")
else:
    print(answer)
```

我们刚刚引入了又一个新的关键字 `default`。我们把它设置为 `0` 来让它可以与其他整数值相互比较。记住，默认情况下如果一个可选参数没有被指定，它的值会是 `None`，并且它不能和整数值相比较（所以产生了 `TypeError` 异常）。

然后：

```
$ python3 prog.py 4
16
```

凭借我们目前已学的东西你就可以做到许多事情，而我们还仅仅学了一些皮毛而已。`argparse` 模块是非常强大的，在结束篇教程之前我们将再探索更多一些内容。

6 进行一些小小的改进

如果我们想扩展我们的简短程序来执行其他幂次的运算，而不仅是乘方:

```
import argparse
parser = argparse.ArgumentParser()
parser.add_argument("x", type=int, help="the base")
parser.add_argument("y", type=int, help="the exponent")
parser.add_argument("-v", "--verbosity", action="count", default=0)
args = parser.parse_args()
answer = args.x**args.y
if args.verbosity >= 2:
    print(f"{args.x} to the power {args.y} equals {answer}")
elif args.verbosity >= 1:
    print(f"{args.x}^{args.y} == {answer}")
else:
    print(answer)
```

输出:

```
$ python3 prog.py
usage: prog.py [-h] [-v] x y
prog.py: error: the following arguments are required: x, y
$ python3 prog.py -h
usage: prog.py [-h] [-v] x y

positional arguments:
  x                the base
  y                the exponent

options:
  -h, --help       show this help message and exit
  -v, --verbosity
$ python3 prog.py 4 2 -v
4^2 == 16
```

请注意到目前为止我们一直在使用详细级别来 *更改*所显示的文本。以下示例则使用详细级别来显示 *更多的*文本:

```
import argparse
parser = argparse.ArgumentParser()
parser.add_argument("x", type=int, help="the base")
parser.add_argument("y", type=int, help="the exponent")
parser.add_argument("-v", "--verbosity", action="count", default=0)
args = parser.parse_args()
answer = args.x**args.y
if args.verbosity >= 2:
    print(f"Running '{__file__}'")
if args.verbosity >= 1:
    print(f"{args.x}^{args.y} == ", end="")
print(answer)
```

输出:

```
$ python3 prog.py 4 2
16
$ python3 prog.py 4 2 -v
4^2 == 16
$ python3 prog.py 4 2 -vv
Running 'prog.py'
4^2 == 16
```

6.1 矛盾的选项

So far, we have been working with two methods of an `argparse.ArgumentParser` instance. Let's introduce a third one, `add_mutually_exclusive_group()`. It allows for us to specify options that conflict with each other. Let's also change the rest of the program so that the new functionality makes more sense: we'll introduce the `--quiet` option, which will be the opposite of the `--verbose` one:

```
import argparse

parser = argparse.ArgumentParser()
group = parser.add_mutually_exclusive_group()
group.add_argument("-v", "--verbose", action="store_true")
group.add_argument("-q", "--quiet", action="store_true")
parser.add_argument("x", type=int, help="the base")
parser.add_argument("y", type=int, help="the exponent")
args = parser.parse_args()
answer = args.x**args.y

if args.quiet:
    print(answer)
elif args.verbose:
    print(f"{args.x} to the power {args.y} equals {answer}")
else:
    print(f"{args.x}^{args.y} == {answer}")
```

我们的程序现在变得更简洁了，我们出于演示需要略去了一些功能。无论如何，输出是这样的:

```
$ python3 prog.py 4 2
4^2 == 16
$ python3 prog.py 4 2 -q
16
$ python3 prog.py 4 2 -v
4 to the power 2 equals 16
$ python3 prog.py 4 2 -vq
usage: prog.py [-h] [-v | -q] x y
prog.py: error: argument -q/--quiet: not allowed with argument -v/--verbose
$ python3 prog.py 4 2 -v --quiet
usage: prog.py [-h] [-v | -q] x y
prog.py: error: argument -q/--quiet: not allowed with argument -v/--verbose
```

这应该很容易理解。我添加了末尾的输出这样你就可以看到其所达到的灵活性，即混合使用长和短两种形式的选项。

在我们收尾之前，你也许希望告诉你的用户这个程序的主要目标，以免他们还不清楚:

```
import argparse

parser = argparse.ArgumentParser(description="calculate X to the power of Y")
group = parser.add_mutually_exclusive_group()
group.add_argument("-v", "--verbose", action="store_true")
group.add_argument("-q", "--quiet", action="store_true")
parser.add_argument("x", type=int, help="the base")
parser.add_argument("y", type=int, help="the exponent")
args = parser.parse_args()
answer = args.x**args.y

if args.quiet:
    print(answer)
elif args.verbose:
    print(f"{args.x} to the power {args.y} equals {answer}")
else:
    print(f"{args.x}^{args.y} == {answer}")
```

请注意用法文本中有细微的差异。注意 `[-v | -q]`，它的意思是说我们可以使用 `-v` 或 `-q`，但不能同时使用两者：

```
$ python3 prog.py --help
usage: prog.py [-h] [-v | -q] x y

calculate X to the power of Y

positional arguments:
  x              the base
  y              the exponent

options:
  -h, --help     show this help message and exit
  -v, --verbose
  -q, --quiet
```

7 后记

除了这里显示的内容，`argparse` 模块还提供了更多功能。它的文档相当详细和完整，包含大量示例。完成这个教程之后，你应该能毫不困难地阅读该文档。