Argparse 教程

发布 *3.9.0a0*

Guido van Rossum
and the Python development team

十一月 19, 2019

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。

注解： 还有另外两个模块可以完成同样的任务，称为 `getopt` (对应于 C 语言中的 `getopt()` 函数) 和被弃用的 `optparse`。还要注意 `argparse` 是基于 `optparse` 的，因此用法与其非常相似。

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]

optional arguments:
  -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

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

程序运行情况如下：

- 我们增加了 `add_argument()` 方法，该方法用于指定程序能够接受哪些命令行选项。在这个例子中，我将选项命名为 `echo`，与其功能一致。
- 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`.
- The variable is some form of 'magic' that `argparse` performs for free (i.e. no need to specify which variable that value is stored in). You will also notice that its name matches the string argument given to the method, `echo`.

Note however that, although the help display looks nice and all, it currently is not as helpful as it can be. For example we see that we got `echo` as a positional argument, but we don't know what it does, other than by guessing or by reading the source code. So, let's make it a bit more useful:

```
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

positional arguments:
  echo        echo the string you use here

optional arguments:
  -h, --help  show this help message and exit
```

现在，做一些更有用的事情怎么样:

```
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'
```

That didn't go so well. That's because `argparse` treats the options we give it as strings, unless we tell it otherwise. So, let's tell `argparse` to treat that input as an integer:

```
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'
```

That went well. The program now even helpfully quits on bad illegal input before proceeding.

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]

optional arguments:
  -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：

- The program is written so as to display something when `--verbosity` is specified and display nothing when not.
- 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，必须指定一些值（任何值）。

The above example accepts arbitrary integer values for `--verbosity`, but for our simple program, only two values are actually useful, `True` or `False`. Let's modify the code accordingly:

```
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]

optional arguments:
  -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`.

- It complains when you specify a value, in true spirit of what flags actually are.
- 留意不同的帮助文字。

4.1 短选项

If you are familiar with command line usage, you will notice that I haven't yet touched on the topic of short versions of the options. It's quite simple:

```
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")
```

And here goes:

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

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

Note that the new ability is also reflected in the help text.

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("the square of {} equals {}".format(args.square, 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
```

(下页继续)

(续上页)

```
$ python3 prog.py 4 --verbose
the square of 4 equals 16
$ python3 prog.py --verbose 4
the square of 4 equals 16
```

- We've brought back a positional argument, hence the complaint.
- 注意顺序无关紧要。

How about we give this program of ours back the ability to have multiple verbosity values, and actually get to use them:

```
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("the square of {} equals {}".format(args.square, answer))
elif args.verbosity == 1:
    print("{}^2 == {}".format(args.square, 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
```

These all look good except the last one, which exposes a bug in our program. Let's fix it by restricting the values the `--verbosity` option can accept:

```
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("the square of {} equals {}".format(args.square, answer))
elif args.verbosity == 1:
    print("{}^2 == {}".format(args.square, 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

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

Note that the change also reflects both in the error message as well as the help string.

Now, let's use a different approach of playing with verbosity, which is pretty common. It also matches the way the CPython executable handles its own verbosity argument (check the output of `python --help`):

```
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("the square of {} equals {}".format(args.square, answer))
elif args.verbosity == 1:
    print("{}^2 == {}".format(args.square, answer))
else:
    print(answer)
```

We have introduced another action, "count", to count the number of occurrences of a specific optional arguments:

```
$ 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

optional arguments:
  -h, --help       show this help message and exit
  -v, --verbosity  increase output verbosity
```

(下页继续)

(续上页)

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

- Yes, it's now more of a flag (similar to `action="store_true"`) in the previous version of our script. That should explain the complaint.
- 它也表现得与“store_true”的行为相似。
- Now here's a demonstration of what the "count" action gives. You've probably seen this sort of usage before.
- And if you don't specify the `-v` flag, that flag is considered to have `None` value.
- As should be expected, specifying the long form of the flag, we should get the same output.
- Sadly, our help output isn't very informative on the new ability our script has acquired, but that can always be fixed by improving the documentation for our script (e.g. via the `help` keyword argument).
- That last output exposes a bug in our program.

让我们修复一下:

```
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",
                    help="increase output verbosity")
args = parser.parse_args()
answer = args.square**2

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

(下页继续)

(续上页)

```
                    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("the square of {} equals {}".format(args.square, answer))
elif args.verbosity >= 1:
    print("{}^2 == {}".format(args.square, answer))
else:
    print(answer)
```

We've just introduced yet another keyword, `default`. We've set it to `0` in order to make it comparable to the other int values. Remember that by default, if an optional argument isn't specified, it gets the `None` value, and that cannot be compared to an int value (hence the `TypeError` exception).

然后：

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

You can go quite far just with what we've learned so far, and we have only scratched the surface. The `argparse` module is very powerful, and we'll explore a bit more of it before we end this tutorial.

6 进行一些小小的改进

What if we wanted to expand our tiny program to perform other powers, not just squares:

```
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("{} to the power {} equals {}".format(args.x, args.y, answer))
elif args.verbosity >= 1:
    print("{}^{} == {}".format(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
```

(下页继续)

(续上页)

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

Notice that so far we've been using verbosity level to *change* the text that gets displayed. The following example instead uses verbosity level to display *more* text instead:

```
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("Running '{}'".format(__file__))
if args.verbosity >= 1:
    print("{}^{} == ".format(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("{} to the power {} equals {}".format(args.x, args.y, answer))
```

(下页继续)

(续上页)

```
else:
    print("{}^{} == {}".format(args.x, args.y, answer))
```

Our program is now simpler, and we've lost some functionality for the sake of demonstration. Anyways, here's the output:

```
$ 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
```

That should be easy to follow. I've added that last output so you can see the sort of flexibility you get, i.e. mixing long form options with short form ones.

Before we conclude, you probably want to tell your users the main purpose of your program, just in case they don't know:

```
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("{} to the power {} equals {}".format(args.x, args.y, answer))
else:
    print("{}^{} == {}".format(args.x, args.y, answer))
```

Note that slight difference in the usage text. Note the `[-v | -q]`, which tells us that we can either use `-v` or `-q`, but not both at the same time:

```
$ 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

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

7 后记

The `argparse` module offers a lot more than shown here. Its docs are quite detailed and thorough, and full of examples. Having gone through this tutorial, you should easily digest them without feeling overwhelmed.