Lecture 12: Args & User Args in Command Line

[Xiao-Xin Li](#)

Zhejiang University of Technology

Revision: *2023/05/31*

---

1. `**args`, `**kwargs`

```python
class A:
    def __call__(self, *args, **kwargs):
        """
        值得注意的是，虽然 *args 和 **kwargs 都是“形参”，但它们的名字都是 args
        """
        pass


def test_args_kwargs():
    A()(2, 3, 4, dict(x=2, y=3, z=4))
    A()(2, 3, 4, **dict(x=2, y=3, z=4))
    A()(*[2, 3, 4], **dict(x=2, y=3, z=4))
    A()(*(2, 3, 4), **dict(x=2, y=3, z=4))
    return
```

```python
class B:
    def __call__(self, a, b, c, *args, **kwargs):
        pass


def test_params_args_kwargs():
    # B()(-2, -3, -4, *(2, 3, 4), **dict(x=2, y=3, z=4))
    # 可以越过 *args，直接将 dict 赋值给 **kwargs
    B()(*(2, 3, 4), **dict(x=2, y=3, z=4))
```

```
class C:
    def __call__(self, a, b, c, *args):
        pass


def test_params_args():
    C()(2, 3, 4, *(5, 6))
    C()(*(2, 3, 4), *(5, 6))
```

```
class D:
    def __call__(self, a, b, c, **kwargs):
        pass


def test_params_kwargs():
    D()(*(2, 3, 4), **dict(x=2, y=3, z=4))
```

```
class E:
    def __call__(self, a, b, c, *args, d, e):
        pass


def test_params_args_params():
    # E()(2, 3, 4, *(5, 6), 7, 8)
    E()(2, 3, 4, *(5, 6), d=7, e=8)
```

```
class F:
    def __call__(self, a, b, c, **kwargs, d, e):
        pass
```

2. 命令行参数解析

2.1 直接使用 `ArgumentParser`

```
import sys
from argparse import Namespace, ArgumentParser

```

```
arg_parser = ArgumentParser(description='Training MRI Reconstruction Network.')

# run-mode arguments
arg_parser.add_argument('--train', action='store_true', help='Set system in trai
arg_parser.add_argument('--test', action='store_true', help='Set system in test
arg_parser.add_argument('--debug', action='store_true', help='Set system in debu

# train arguments
arg_parser.add_argument('--batch-size', '-bs', default=5, type=int, help='traini

arg_parser.add_argument('--loss',
                        default='L1Loss',
                        type=str, choices=['SSIMLoss', 'MSELoss', 'L1Loss'],
                        help='Loss function.')  # action=LossAction,

arg_parser.add_argument('--optimizer',
                        default='Adam//lr(1e-4)//StepLR(20,.1)//betas(.8,.99)//w
                        type=str,
                        help='Training optimizer.')

arg_parser.add_argument('--max-epoch', default=50, type=int, help='training epoc

# program start
if __name__ == '__main__':
    args = arg_parser.parse_args(sys.argv[1:])

    train = args.train
    test = args.test

    batch_size = args.batch_size
    loss = args.loss
    max_epoch = args.max_epoch

    pass
```

2.2 对 `ArgumentParser` 进行封装

❓ 如何进行封装?

2.3 使用封装后得到的 `UserArgs` 进行命令行参数解析

```
import sys

```

```python
from args_actions import OptimizerAction
from user_args import UserArgs  # 注意，这里的 UserArgs 对 ArgumentParser 进行了封装

# 需要特别注意的是，`action`所指向的类能够被执行的前提是用户在命令行使用了该参数
args_specs = {
    'run-modes': {
        '--train': dict(action='store_true', help='Set system in training mode.'
        '--test': dict(action='store_true', help='Set system in test mode.'),
        '--debug': dict(action='store_true', help='Set system in debug mode.'),
    },

    'train-args': {  # 'train-args' 表示分组参数的名字，不是命令行参数的名字，用户可以
        '--loss': dict(default='L1Loss', type=str, choices=['SSIMLoss', 'MSELoss
                       help='Loss function.'),  # action=LossAction,
        '--optimizer': dict(default='Adam//lr(1e-4)//StepLR(20,.1)//betas(.8,.99
                            type=str,
                            help='Training optimizer.'),  # 'SGD' action=Optimiz
        '--batch-size/-bs': dict(default=5, type=int, help='training batch size'
        '--max-epoch': dict(default=50, type=int, help='training epochs'),
    },
}


class MyUserArgs(UserArgs):
    def __init__(self, user_args=None):
        UserArgs.__init__(
            self, args_specs, user_args,
            description='Training MRI Reconstruction Network.')


if __name__ == '__main__':
    """ 强制用户必须手动设置实验所需的参数 """
    if len(sys.argv) == 1:
        raise Exception('User does not give any command line argument!')

    # 这里为什么不直接使用 UserArgs，而是要使用 MyUserArgs 对 UserArgs 再次封装?
    args = MyUserArgs(sys.argv[1:])

    train = args.train
    test = args.test

    batch_size = args.batch_size
    loss = args.loss
    max_epoch = args.max_epoch

    optim = args.optim
    lr = args.lr
```

```
    StepLR = args.StepLR
    betas = args.betas
    wd = args.wd

    print()
```

2.4 使用 Action 进一步解析命令行参数中的参数

```
import sys

from args_actions import OptimizerAction
from user_args import UserArgs

# 需要特别注意的是，`action`所指向的类能够被执行的前提是用户在命令行使用了该参数
args_specs = {
    'train-args': {
        '--optimizer':
            dict(default='Adam//lr(1e-4)//StepLR(20,.1)//betas(.8,.99)//wd(5e-4)
                 type=str, action=OptimizerAction,
                 help='Training optimizer.'),
    },
}


class MyUserArgs(UserArgs):
    def __init__(self, user_args=None):
        UserArgs.__init__(
            self, args_specs, user_args,
            description='Training MRI Reconstruction Network.')

```

```
from argparse import Action
from MRIRecon.favourlab.utils.parsers.parser import parse_unit


class OptimizerAction(Action):
    def __call__(self, parser, namespace, values, option_string=None):
        self.members = vars(self).keys()
        params = values.split('//')
        optim, params = params[0], params[1:]
        setattr(namespace, 'optim', optim)
        for unit in params:
            n, v = parse_unit(unit)
```

```
            setattr(namespace, n, v)
```

```
if __name__ == '__main__':
    args = MyUserArgs(sys.argv[1:])

    # 这时，我们会发现，args 中多了几个事先在 args_specs 并未定义的参数
    optim = args.optim
    lr = args.lr
    StepLR = args.StepLR
    betas = args.betas
    wd = args.wd

    print()
```

3. 练习与思考

1. 试对 `ArgumentParser` 进行封装，构建你自己的 `UserArgs` 类，使得**2.3节**和**2.4节**的代码能够正常解析用户命令行参数。
2. `UserArgs` 对 `ArgumentParser` 进行了封装，大大简化了对用户命令行参数的定义。`UserArgs` 看起来是个很有用的工具类，如何在新的项目下对其进行复用呢?