@classmethod 多态总结

[toc]

Effective Python 24 条. 以 MapReduce 流程为例.

常规方法构建 MapReduce

```python
import os
from threading import Thread


# InputData 基类
class InputData(object):
    def read(self):
        raise NotImplementedError


# InputData 具体子类

class PathInputData(InputData):
    def __init__(self, path):
        super().__init__()
        self.path = path

    def read(self):
        return open(self.path).read()


# MapReduce 工作线程，基类
class Worker(object):
    def __init__(self, input_data):
        # input_date 为 PathInputData的实例
        self.input_data = input_data
        self.result = None

    def map(self):
        raise NotImplementedError

    def reduce(self):
        raise NotImplementedError


# 子类，换行符计数器
class LineCountWorker(Worker):
    def map(self):
        data = self.input_data.read()
        self.result = data.count('\n')

    def reduce(self, other):
        self.result += other.result


"""
需要手工写流程协调上面定义的各个组件
并实现 MapReduce
"""


def generate_inputs(data_dir):
    for name in os.listdir(data_dir):
        yield PathInputData(os.path.join(data_dir, name))


def create_workers(input_list):
    workers = []
    for input_data in input_list:
        workers.append(LineCountWorker(input_data))
    return workers


def execute(workers):
    threads = [Thread(target=w.map) for w in workers]
    for thread in threads: thread.start()
    for thread in threads: thread.join()

    first, rest = workers[0], workers[1:]
    for worker in rest:
        first.reduce(worker)
    return first.result


# 组合
def mapreduce(data_dir):
    inputs = generate_inputs(data_dir)
    workers = create_workers(inputs)
    return execute(workers)


from tempfile import TemporaryDirectory, NamedTemporaryFile
import uuid


def write_test_file(tmpdir):
    for i in range(100):
        with open(os.path.join(tmpdir, "_{}.txt".format(i)), 'w') as f:
            f.write(str(uuid.uuid4()) + '\n')


def test():
    with TemporaryDirectory() as tmpdir:
        write_test_file(tmpdir)
        result = mapreduce(tmpdir)
    print('There are {} lines'.format(result))

if __name__ =='__main__':
    test()
```

代码问题在于 MapReduce 不够通用, 每次都要手工写流程组合各个类模块. 因此引出 `@classmethod` 形式的多态.

@classmethod 形式的多态

> classmethod 修饰符对应的函数不需要实例化，不需要 self 参数，但第一个参数需要是表示自身类的 cls 参数，可以来调用类的属性，类的方法，实例化对象等。

```python
import os
from threading import Thread


class GenericInputData(object):
    def read(self):
        raise NotImplementedError

    # 第一个参数为 cls
    @classmethod
    def generate_inputs(cls, config):
        raise NotImplementedError


class PathInputData(GenericInputData):
    def __init__(self, path):
        super().__init__()
        self.path = path

    def read(self):
        return open(self.path).read()

    @classmethod
    def generate_inputs(cls, config):
        data_dir = config['data_dir']
        for name in os.listdir(data_dir):
            # 类的多态，以 cls 形式构造 PathInputData 对象
            yield cls(os.path.join(data_dir, name))


# MapReduce 工作线程，基类
class GenericWorker(object):
    def __init__(self, input_data):
        # input_date 为 PathInputData的实例
        self.input_data = input_data
        self.result = None

    def map(self):
        raise NotImplementedError

    def reduce(self):
        raise NotImplementedError

    @classmethod
    def create_workers(cls, input_class, config):
        workers = []
        for input_data in input_class.generate_inputs(config):
            workers.append(cls(input_data))
        return workers


# 子类，换行符计数器
class LineCountWorker(GenericWorker):
    def map(self):
        data = self.input_data.read()
        self.result = data.count('\n')

    def reduce(self, other):
        self.result += other.result

# 组合
def execute(workers):
    threads = [Thread(target=w.map) for w in workers]
    for thread in threads: thread.start()
    for thread in threads: thread.join()

    first, rest = workers[0], workers[1:]
    for worker in rest:
        first.reduce(worker)
    return first.result

def mapreduce(worker_class, input_class, config):
    workers = worker_class.create_workers(input_class, config)
    return execute(workers)

```

```python

from tempfile import TemporaryDirectory, NamedTemporaryFile
import uuid


def write_test_file(tmpdir):
    for i in range(100):
        with open(os.path.join(tmpdir, "_{}.txt".format(i)), 'w') as f:
            f.write(str(uuid.uuid4()) + '\n')


def test():
    with TemporaryDirectory() as tmpdir:
        write_test_file(tmpdir)
        config = {'data_dir': tmpdir}
        result = mapreduce(LineCountWorker, PathInputData, config)
    print('There are {} lines'.format(result))


if __name__ == '__main__':
    test()
```

此时, 需要给`mapreduce`函数传入更多的参数, 但不需要重新写辅助流程来组个各个模块了.