OOP七大设计原则-单一职责原则

概述：
    单一职责就是，一个类负责一项职责．就是希望一个类只有一个引起它变化的职责，如果一个类有多个职责，就相当于把多个职责耦合在一起了．而我们进行框架设计的其中一个原则就是希望低耦合，所以任何高耦合的设计都是应该尽力避免的，这样当修改一个功能时，可以显著降低对其其他功能的影响．

适用场景：

    1.一个类负责多项职责的场景

    同一个类C负责两个或多个不同的职责：职责func1...职责funcN．当类C中的某个职责发生改变需要修改类C时，有可能导致该类C原本运行正常的其他职责发生功能故障．此时所以我们需要考虑将类C拆分成两个或多个类，将类C中经常发生变化的单个职责封装成一个类．这样新封装类中的职责发生变化时不会导致类C拆分后生成的其他类发生故障．

    2.一个类负责一项职责时"职责扩散"场景
    在实际项目中经常会出现"职责扩散"的现象，比如类P原本只有单个职责func1，但由于需求变更或其它原因，职责1被细分为职责func11和职责func12导致类P无法正常工作．

示例代码：现有一logging类原本只有单个职责"离线收集日志"

In [2]:
```
class logging(object):
    def __init__(self):
        pass

    def collect(self, log_dir):
        print('starting collect log %s' %log_dir)

class client:
    def __init__(self):
        pass
    def act(self):
        l = logging()
        l.collect('/var/log/messages')
        l.collect('/tmp/logs')

if __name__ == '__main__':
    c = client()
    c.act()
```

```
starting collect log /var/log/messages
starting collect log /tmp/logs
```

现在新增需求希望给日志做实时大数据分析

违反单一职责原则的代码：

In [1]:
```
class logging(object):
    def __init__(self):
        pass

    def collect(self, log_dir):
        print('starting collect log %s' %log_dir)

    def analysis(self, log_dir):
        print("logging analysis %s" %log_dir)

class client:
    def __init__(self):
        pass
    def act(self):
        l = logging()
        l.collect('/var/log/messages')
        l.analysis('/var/log/messages')
        print('#' * 20)
        l.collect('/tmp/logs')
        l.analysis('/tmp/logs')

if __name__ == '__main__':
    c = client()
    c.act()
```

```
starting collect log /var/log/messages
logging analysis /var/log/messages
####################
starting collect log /tmp/logs
logging analysis /tmp/logs
```

遵循了单一职责原则的代码如下：

In [3]:
```

class logging(object):
    def __init__(self):
        pass

    def collect(self, log_dir):
        print('starting collect log %s' %log_dir)

class logAnalysis(object):
    def __init__(self):
        pass

    def analysis(self, log_dir):
        print("logging analysis %s" %log_dir)

class client:
    def __init__(self):
        pass
    def act(self):
        l = logging()
        l.collect('/var/log/messages')
        l.collect('/tmp/logs')
        print("#" * 20)
        w = logAnalysis()
        w.analysis('/var/log/messages')
        w.analysis('/tmp/logs')

if __name__ == '__main__':
    c = client()
    c.act()
```

```
starting collect log /var/log/messages
starting collect log /tmp/logs
####################
logging analysis /var/log/messages
logging analysis /tmp/logs
```

注意事项：
    1.尽量保证定义的类或模块都是单一职责的，尽可能编写短小的类．
    2.单个类的职责越少，则类与类之间的耦合度就越弱，受其他类的约束和牵制就越少，可扩展性就越强．
    3.职责拆分时尽量把变化频繁的部分单独拆分、封装．

```
4.单一职责的要点在于职责的粒度，但是具体需要把职责细化到哪一步，取决于项目需求和业务的复杂度.
5.单一职责不是刻板的教条，程序设计的主旨还是高内聚、低耦合.
6.极端场景下，一味要求所有类都是100%的单一职责可能会带来类数量的爆炸  开发效率及程序性能的降低；需要均衡考虑、灵活运用.
```

In [ ]: