行为型模式-责任链模式

概述：
责任链模式，用于让多个对象处理一个请求时或者用于预先不知道由哪个对象来处理某种特定请求时的场景；其原则如下：
1)存在一个对象链(链表、树或者其他便捷的数据结构)
2)一开始将请求发送给第一个对象让其处理
3)对象决定是否处理该请求
4)对象将请求转发给下一个对象
5)重复该过程，直到达到链尾

场景：
假设有这么一个请假场景：员工若想要请3天以内(包括3天的假)，只需要组长批准就可以了；如果想请3-7天，不仅需要组长批准，部门经理需要最终批准；如果请假大于7天， 不光要组长和直属经理批准，也需要总监最终批准. 类似的场景相信大家都遇到过，那么该如何实现呢？首先想到的当然是if…else…，但一旦遇到需求变动，其臃肿的代码和复杂的耦合缺点都显现出来. 简单分析下需求："假单"在组长 直属经理 总监之间是单向传递关系，像一条链条一样，因而我们可以用一条"链"把他们进行有序连接.

示例代码：

In [2]:
```
from abc import ABCMeta, abstractmethod

class Manager:
    def successor(self, person):
        self.person = person
    @abstractmethod
    def handler(self, *args, **kwargs):
        pass

class group_manager(Manager):
    def __init__(self, successor):
        self.successor = successor

    def handler(self, request):
        if all([request >=1 , request < 3]):
            print('组长批准了你的假单！ 请假%s天'%request)
        else:
            print("组长虽然同意了 但是还需要经理审批假单")
            self.successor.handler(request)

class team_manager(Manager):
    def __init__(self, successor):
        self.successor = successor

    def handler(self, request):
        if all([request >=3, request < 7]):
            print('团队经理批准了你的假单！ 请假%s天'%request)
        else:
            print('经理虽然同意了，但是还需VP审批')
            self.successor.handler(request)

class company_vp(Manager):
    def handler(self, request):
        print("VP同意了你的请假申请!")

if __name__ == '__main__':
    lilei = company_vp()
    lucy = team_manager(lilei)
    hanmei = group_manager(lucy)
    hanmei.handler(10)
    print('#' * 10)
    hanmei.handler(1)
    print('#' * 10)
    hanmei.handler(4)
```

```
组长虽然同意了 但是还需要经理审批假单
经理虽然同意了，但是还需VP审批
VP同意了你的请假申请!
##########
组长批准了你的假单！ 请假1天
##########
组长虽然同意了 但是还需要经理审批假单
团队经理批准了你的假单！ 请假4天
```

责任链模式的优缺点：
优点：
1)将请求者与处理者分离，请求者并不知道请求是被哪个处理者所处理，易于扩展。
缺点：
1)如果责任链比较长，会有比较大的性能问题；
2)如果责任链比较长，若业务出现问题，比较难定位是哪个处理者的问题。