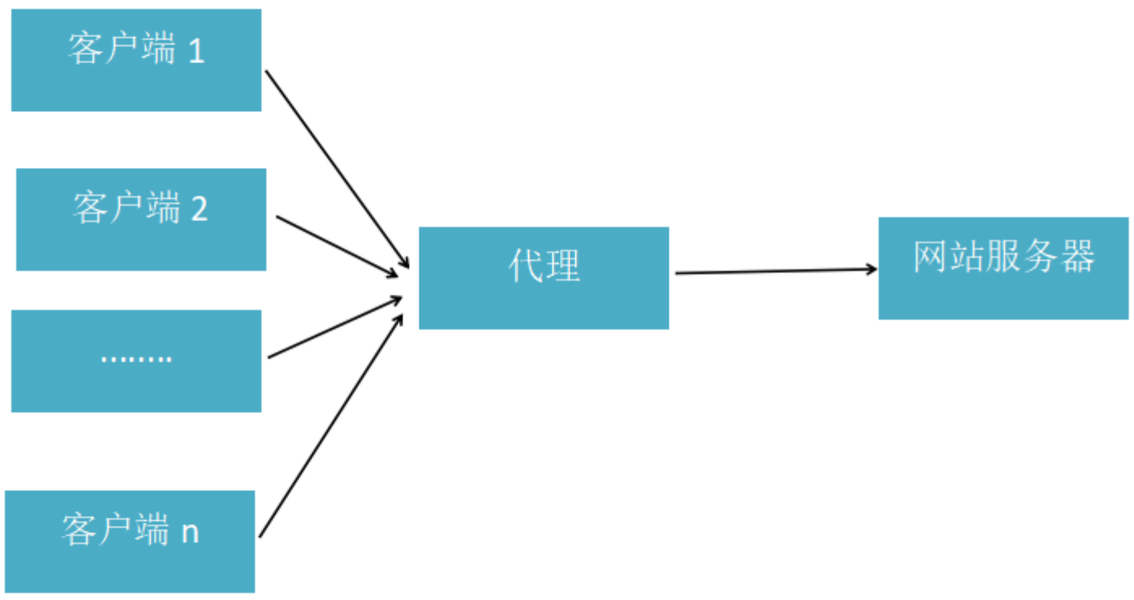
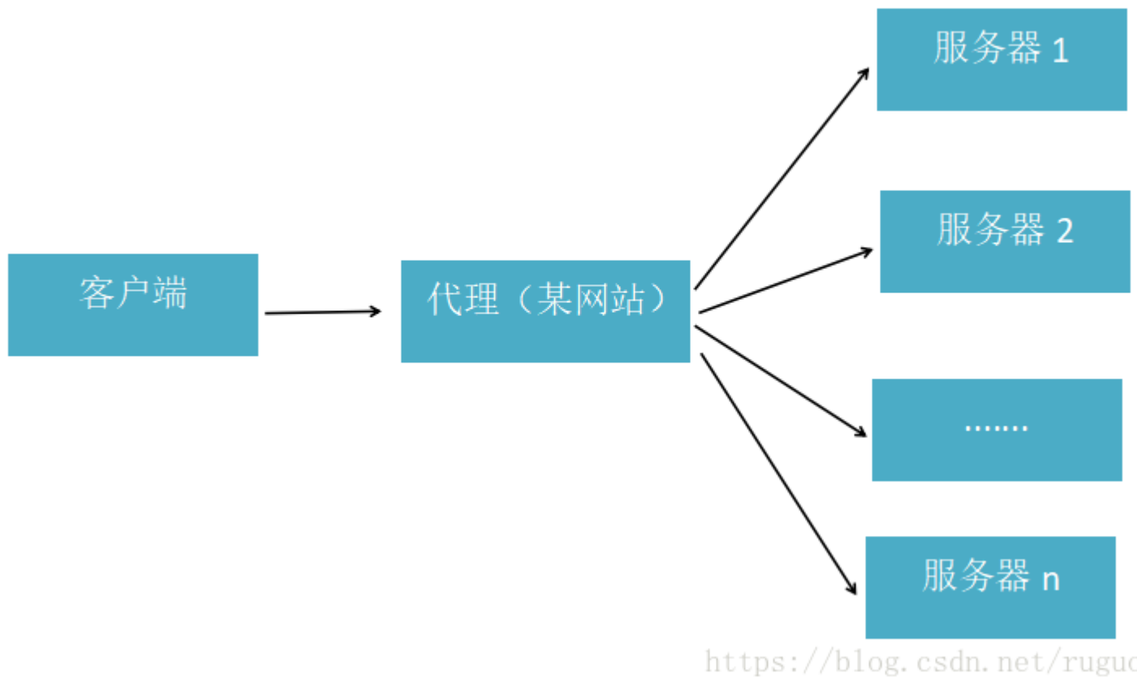
9.结构型模式-代理模式

概述：
　　代理模式在日常生活中很常见，比如你去杂货店买一个插座，而不是去生产插座的工厂去买．比如你去访问某个网站，你并没有访问权限，但你可以通过代理去访问这个网站，然后代理再把内容传给你．讲代理模式之前，先讲下正向代理和反向代理的区别：

正向代理：
　　上面访问网站的例子就是正向代理，可以用下面的流程图展示这一机制．



正向代理：
　　客户端访问某网站，先访问代理，代理去访问某网站，然后把内容返回给客户端，这就是正向代理．正向代理就是我们现实中常用的代理模式，杂货店代理工厂的产品，火车票代售点代理火车票业务等等．



反向代理：
　　客户端去访问某个网站的某个资源，但这个网站没有这个资源，同时这个网站被设置为反向代理，那么这个网站就会从其他服务器上获取这个资源返回给客户端．因此客户端并不知道这个资源是谁提供的，它只要知道代理网站的网站即可．比如我们访问百度，背后有成千上万的反向代理服务器在为我们服务，但我们只要知道baidu.com这个网站就行了，不必知道访问的资料具体来自哪里．nginx是高流量web服务器流行选择，它支持反向代理和负载均衡．

代理模式有3个必要的元素：
　　1)真实的对象（执行业务逻辑，被代理的对象）
　　2)代理类（用户请求的一个接口，对真实目标的保护）
　　3)用户（获取任务的用户请求）

代理模式存在在以下的情形中：
　　1)为真实目标类创建一个对象的代价是昂贵的，一个简单对象被代理类创建是便宜的方法．
　　2)对象必须防止被用户直接使用．
　　3)当实际请求的时候，为真实目标类创建一个对象会有延迟．

一个生活中的例子：
　　让我们想想一个正规办公的场景，为了向一个公司的销售主管谈话，用户首先会向销售主管办公室的接待员打个电话，随后接待员转接电话．在这个例子中，销售主管会是用户希望交谈的目标，接待员就是一个代理，保护主体不受用户直接要求谈话中苦恼．

示例代码：

In [2]:
```
class Manager:
    def work(self):
        print('Manager is working')

    def talk(self):
        print('manager is talking')

class Proxy:
    def __init__(self):
        self.busy = True

    def call(self):
        if  self.busy:
            print('Manager is very busy')
        else:
            self.manager = Manager()
            self.manager.work()
            self.manager.talk()

if __name__ == '__main__':
    p = Proxy()
    p.call()
    print('#' * 10)
    p.busy=False
    p.call()
```

```
Manager is very busy
##########
Manager is working
manager is talking
```

In [ ]: