## 第15章 办事不必亲自出面——代理模式

## 15.1办理签证居然这么麻烦

熊小猫：兔小白，现在有时间吗？

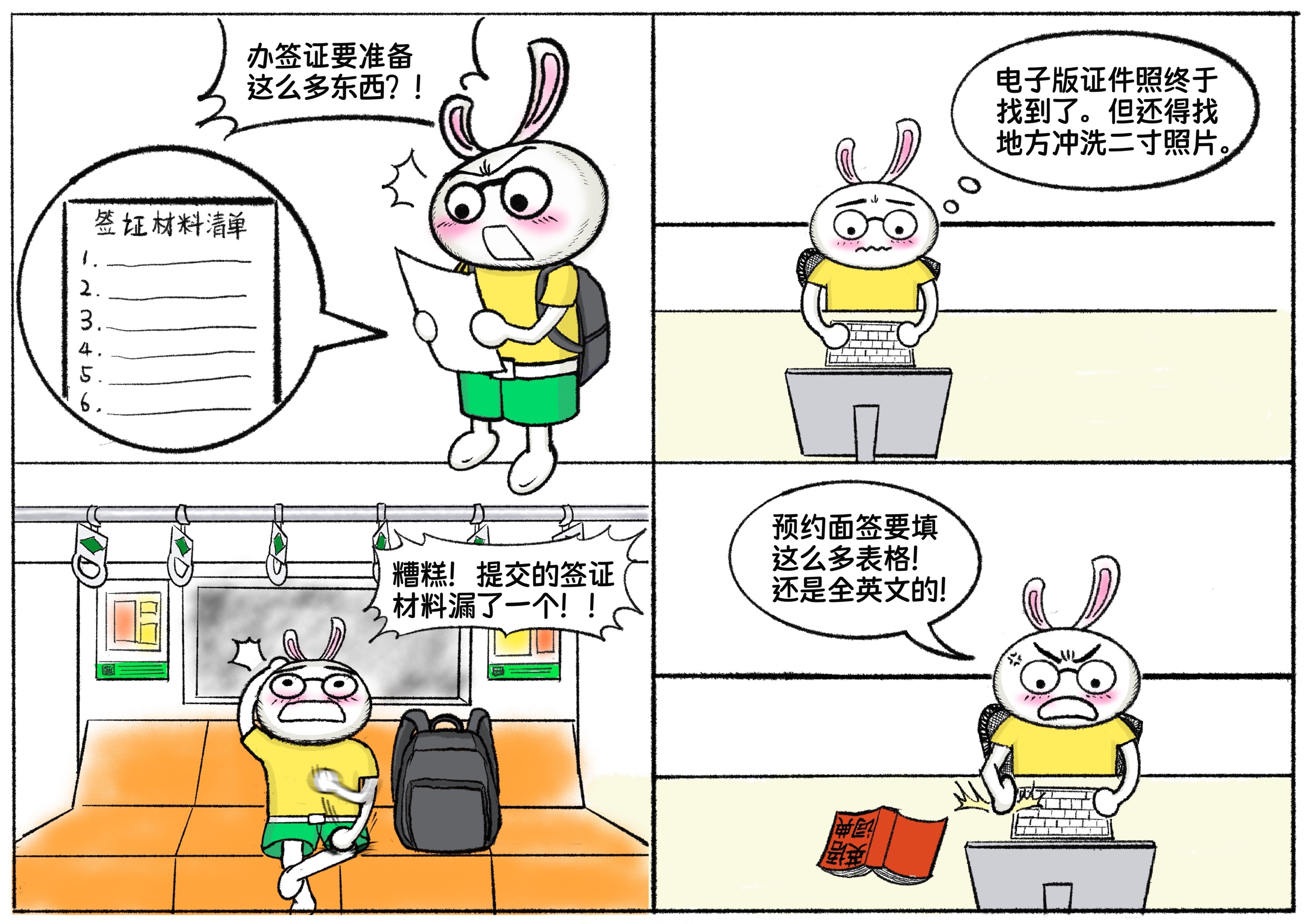
兔小白：有时间，你不是又要给我讲设计模式吧？

熊小猫：哈哈，我有那么好为人师吗？我下个月要休假，去冰岛玩一个星期。我想和你交代一下工作上的事情。咱们抓紧点时间，我下午还得去办理签证。

兔小白：我听你说到签证，就想起了两个月前，我办理签证的痛苦经历。我因为要去欧洲旅游，提前两个月就开始办理签证，办理的过程非常曲折，不堪回首。

熊小猫：哦？我倒想听听你都经历了什么。

兔小白：办理签证需要准备各种资料，准备过程非常麻烦。需要开各种证明、打印各种表单。最郁闷的是我在递交材料的时候漏了一份文件。我发现后赶紧补交了上去，还好没有影响签证申请。我还需要在网站上填写面签预约表，填写的内容非常多。我的英文不太好，磕磕绊绊填了好久才搞定。好在面签还比较顺利。



熊小猫：哈哈，没想到你办理签证这么费劲！我这次找的签证代理机构帮忙办理，非常轻松！办理全程基本不用我动脑子，都是代理在主导、跟进。我更像是代理的助理，他让我干啥，我就干啥。而且代理会帮我检查申请材料，绝对不会出现漏了材料的问题。

兔小白：之前我不知道，下次一定也要找代理。你给我讲讲代理是怎么帮你申请签证的？

熊小猫：签证代理机构的工作方式是这样的......代理？代理也是一种设计模式呀。咱们今天就以办理签证为例子，讲一讲代理模式怎么样？

兔小白：既能学习办理签证的知识，又能学习设计模式，何乐而不为？

熊小猫：那咱们现在就开始！

## 15.2 自己办理签证

熊小猫：咱们做一个在程序中实现签证办理的练习。需求简化一下，假设签证的办理流程只需要三大步骤——准备材料、准备照片和面签。你来根据你自己办理签证的经历完成代码。

兔小白：办理过程我熟悉呀，才刚刚办完。这个程序好写。

十五分钟后，兔小白完成了开发。

兔小白：首先定义申请人接口Applicant，接口中的3个方法对应准备材料、准备照片、面签申请签证。

public interface Applicant {  
    void prepareMaterial();  
    void preparePhoto();  
    void applyVisa();  
}

签证人类VisaApplicant实现Applicant接口。实现逻辑就是我之前办理签证的步骤。

public class VisaApplicant implements Applicant {  
    private String name;  
​  
    public VisaApplicant(String name) {  
        this.name = name;  
   }  
​  
    public void prepareMaterial() {  
        System.out.println("申请人"+name+"-准备签证材料");  
        System.out.println("申请人"+name+"-打印材料");  
   }  
​  
    public void preparePhoto() {  
        System.out.println("申请人"+name+"-准备签证电子照片");  
        System.out.println("申请人"+name+"-打印照片");  
   }  
​  
    public void applyVisa() {  
        System.out.println("申请人"+name+"-预约面签");  
        System.out.println("申请人"+name+"-面签");  
   }  
}

在客户端代码中，Applicant对象依次执行申请签证步骤的3个方法。

Applicant rabbit = new VisaApplicant("兔小白");  
​  
 rabbit.prepareMaterial();  
 rabbit.preparePhoto();  
 rabbit.applyVisa();

执行结果如下，符合预期。

申请人兔小白-准备签证材料  
申请人兔小白-打印材料  
申请人兔小白-准备签证电子照片  
申请人兔小白-打印2寸照片  
申请人兔小白-预约面签  
申请人兔小白-面签

熊小猫：你将每一个申请步骤都做了细化，很不错。现在你想一想，是否每个步骤里面所有的事情都需要你亲自去做呢？

兔小白：其实很多事情不需要我亲自去做，比如打印材料、冲洗照片，谁来做都可以呀！但是准备材料、准备照片文件，特别是面签，必须我亲自去做。

熊小猫：对呀，其实你的精力应该用在必须你去做的那些事情上，其他的小事交给代理办就好了。你看我，找代理办理签证多轻松！下一版程序，咱们将代理人加上。

## 15.3 代理人办理签证

熊小猫：增加代理人后，那些不是必须由你亲自做的小事情，都可以让代理人来做。在这之外，代理人还可以承担一些附加职责，比如帮助签证人审核材料。这次把这个需求也加入进来。

在你动手前，我还得再啰嗦两句。对于客户类来说，代理人和签证人应该是透明的。客户类不会因为要和代理人打交道，而改变自己的业务逻辑。

兔小白：有道理呀！大使馆不会因为是代理人在帮忙办理签证，而改变自己的工作流程。**想实现这一点，应该让代理人和签证人，都实现同一个Applicant接口。**

熊小猫：代理人在办理过程中，执行到只有签证人才能执行的步骤，自然就会通知签证人。比如今天下午我要去面签，就是代理人提前预约好通知我的。

兔小白：明白啦！**签证人要受到代理人的调用！**代码已经在我脑海里浮现出来了。

10分钟后，兔小白在程序中引入了代理人。

兔小白：VisaApplicantProxy是代理人类，它同样实现Applicant接口。对于客户类来说，VisaApplicantProxy和VisaApplicant是透明的，因为二者实现同一个接口。由于代理人在办理签证过程中需要调用签证人完成任务，所以VisaApplicantProxy对象持有VisaApplicant对象的引用。VisaApplicantProxy对象对接口的实现会调用VisaApplicant对象对应的方法。我把不是必须由签证人做的事情从VisaApplicant移到了VisaApplicantProxy中，比如打印材料、打印照片、预约面签。在prepareMaterial方法中，我为代理人增加了检查材料的职责。

public class VisaApplicantProxy implements Applicant {  
​  
    private VisaApplicant visaApplicant;  
​  
    public VisaApplicantProxy(VisaApplicant visaApplicant) {  
        this.visaApplicant = visaApplicant;  
   }  
​  
    public void prepareMaterial() {  
        visaApplicant.prepareMaterial();  
        System.out.println("代理为"+visaApplicant.getName()+"-检查材料");  
        System.out.println("代理为"+visaApplicant.getName()+"-打印材料");  
   }  
​  
    public void preparePhoto() {  
        visaApplicant.preparePhoto();  
        System.out.println("代理为"+visaApplicant.getName()+"-打印2寸照片");  
   }  
​  
    public void applyVisa() {  
        System.out.println("代理为"+visaApplicant.getName()+"-预约面签");  
        visaApplicant.applyVisa();  
   }  
}

VisaApplicant类也做了相应的修改，把不需要本人亲自做的事情移除掉，只保留必须本人亲自办理的职责，比如准备材料、准备照片和面签。

public class VisaApplicant implements Applicant {  
    private String name;  
​  
    public VisaApplicant(String name) {  
        this.name = name;  
   }  
​  
    public String getName() {  
        return name;  
   }  
​  
    public void prepareMaterial() {  
        System.out.println("申请人"+name+"-准备签证材料");  
   }  
​  
    public void preparePhoto() {  
        System.out.println("申请人"+name+"-准备签证电子照片");  
   }  
​  
    public void applyVisa() {  
        System.out.println("申请人"+name+"-面签");  
   }  
}

客户端增加一行代码来创建VisaApplicantProxy代理对象，后面的事情就交给代理对象去做啦！

VisaApplicant rabbit = new VisaApplicant("兔小白");  
 Applicant applicantProxy = new VisaApplicantProxy(rabbit);  
​  
 applicantProxy.prepareMaterial();  
 applicantProxy.preparePhoto();  
 applicantProxy.applyVisa();

执行结果如下，代理人负责完成一部分签证办理工作。

申请人兔小白-准备签证材料  
代理为兔小白-检查材料  
代理为兔小白-打印材料  
申请人兔小白-准备签证电子照片  
代理为兔小白-打印2寸照片  
代理为兔小白-预约面签  
申请人兔小白-面签

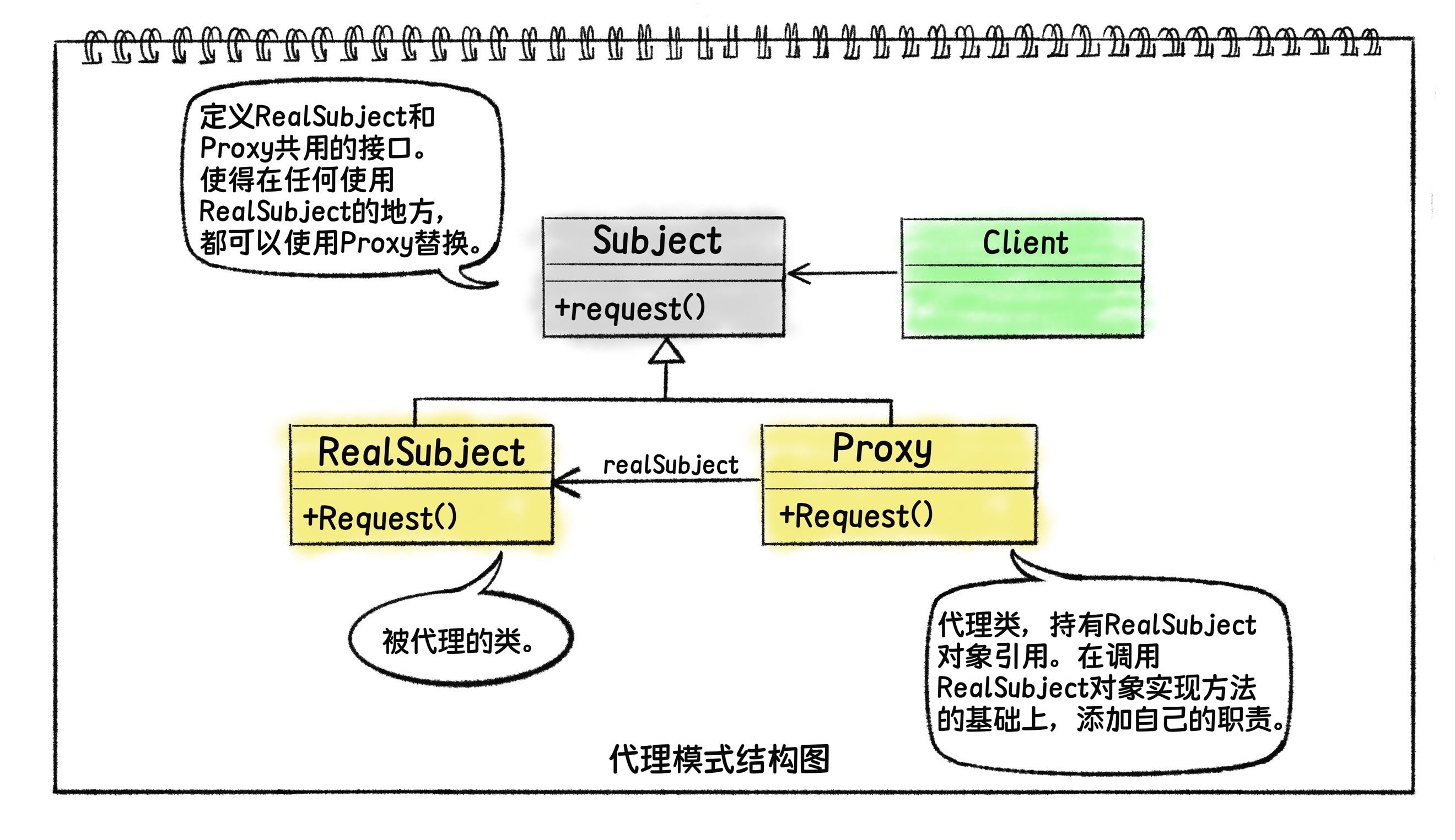
熊小猫：代码写的不错，看来我的提醒你全都听进去了！这版程序其实已经使用了代理模式。VisaApplicant实现办理签证的核心职责。VisaApplicantProxy作为代理类，以VisaApplicant的职责为基础附加职责。

兔小白：代理模式只是将职责进行合理划分吗？很多设计模式都有这个效果呀！我觉得我还没有体会到代理模式的精髓。

熊小猫：代理模式的使用非常广泛，应用场景也非常多，远远不止划分职责这么简单。接下来让我们看看代理模式适用的场景！

## 15.4 代理模式适用的场景

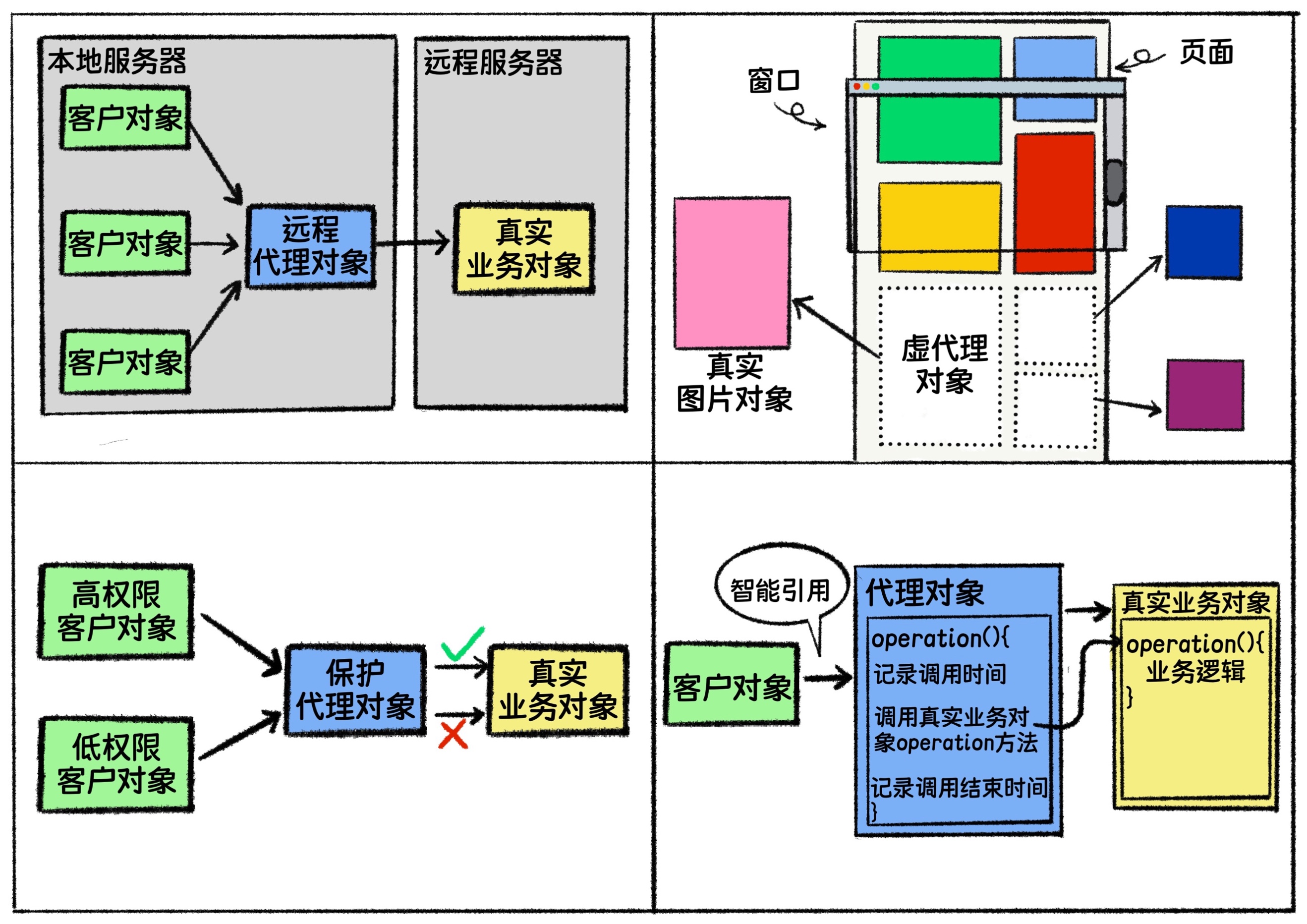
熊小猫：我们先来看看代理模式的结构图



代理模式的结构和办理签证程序的结构基本一致。Subject定义了对象的接口。RealSubject实现该接口。Proxy同样实现Subject接口，这样Proxy对象就可以替代RealSubject对象。Proxy持有RealSubject对象引用，它在调用RealSubject对象实现接口的基础上，添加自己的职责。

代理模式的优点是消除了Client和RealSubject间的直接依赖。Client想要使用RealSubject，需要通过Proxy对象。Proxy对象相当于“中间商”，它在中间可以做很多有附加值的事情。下面是使用代理模式的典型场景。

1. 远程代理。客户端程序需要访问远程服务器的对象时，可以在客户端本地创建代理对象，客户端通过代理对象访问远程服务器上的对象。这样做的好处，一是代理对象可以屏蔽跨网络调用的细节，客户端就像调用本地对象一样。二是不同客户端程序可以复用代理对象的网络访问功能，避免重复开发。
2. 虚代理。虚代理可以用于“占位”。类似于上学时候在图书馆，很多人拿书本占座位。其实占座用的书本就是占座人的虚代理。程序中那些创建开销很大的对象，例如图片列表页中的图片对象，可以先通过虚代理占位。虚代理根据图片尺寸仅画出边框，当页面滚动到图片附近时，虚代理再触发真正图片对象的创建。虚代理可以提升系统性能，并且避免系统资源浪费。
3. 保护代理。如果真实对象有不同访问权限，可以通过保护代理来实现。在保护代理中实现对真实对象的访问控制。
4. 智能引用。所谓的智能，是指代理类可以在真实类的接口实现基础上，附加更多功能。代理对象在调用真实对象的方法前、后，可以加入自己的处理逻辑。例如调用计数、记录调用时间、执行时长等等。



兔小白：我之前把代理模式想简单了，没想到它的用武之地还真不少！

熊小猫：从结构上看，代理模式是一种层次结构，在客户类和真实类之间加入了一层代理类，从而可以在真实类的基础上添加更多职责。上面列出的场景比较具体，其实想象空间还可以更大。

## 15.5 代理模式与装饰模式的比较

兔小白：说到层次结构，我觉得代理模式和装饰模式有相似之处。装饰模式就像俄罗斯套娃，装饰对象一层套一层，同样是层次结构。只不过代理模式套了一层，装饰模式套了多层。

熊小猫：从结构上看，代理模式和装饰模式确实类似，都是一个接口和两个实现类。其实装饰模式也运用的是代理思想。代理模式甚至可以看做是装饰模式退化为仅有一层装饰。二者做的事情都是在某个业务类的基础上添加职责，但是二者的使用场景存在较大区别。

代理模式中的代理类，它所添加的职责和真实类实现的功能关系不大。换句话讲，即使没有代理类，真实类一般也可以满足业务需求，只是缺失了安全控制、日志记录、性能等非功能性需求。

在装饰模式中，业务分散在装饰类和实现类中。虽然装饰类也是基于组件类添加职责，但它添加的是业务职责。例如美颜相机，美颜的功能在装饰类中实现，那么没有装饰类，又何谈美颜相机呢？为了应对灵活多变的业务，装饰模式通过巧妙的设计，使得装饰对象可以层层嵌套，动态扩展原始组件对象的职责。

兔小白：我可以这样理解吗？代理模式主要用于附加业务需求之外的职责，而装饰模式则是为了实现灵活多变的业务需求。

熊小猫：这样理解没有问题。其实从代理模式和装饰模式的名字上，也可以看得出来。既然是代理，那么主要业务肯定不是代理的职责，代理只负责跑腿和提供一些附加价值。装饰则是在主要职责的基础之上，不断叠加职责，从而构成完整的业务功能。

但在实际应用时会更加灵活，没必要拘泥于字面上的限制。假如装饰对象没有叠加使用的需求，也就是说只会存在一层装饰，那么采用代理模式也未尝不可。采用结构更为复杂的装饰模式反而徒增麻烦。

在工作中，我们说到的“代理”不一定指代理模式。例如A类实现某个功能，实现方式主要是通过调用B类来实现。A类也可以称为B类的代理。这种场景非常常见，代理的思想在设计模式中也被广泛运用。