## 第20章 房产中介的存在价值——中介者模式

## 20.1 中介的价值有几何

兔小白：熊小猫，下午我要请假去签租房合同。

熊小猫：没问题。生活不易啊，你又得交出去一大笔钱，除了房租，还有中介费。

兔小白：没办法，不通过中介很难找到房子。你说中介赚钱也太容易了，只是带我看看房，几千块钱中介费就到手了。

熊小猫：中介的工作可不是你说的这么简单。中介公司要开发房源APP，中介需要收集房源，还得带租房人看房。签合的同时候，有中介规范流程，双方也会更安心。中介赚的也是辛苦钱。

兔小白：主要是房源问题，其它对我来说都是小事。

熊小猫：那我问你，为什么房主更喜欢将房源交给中介呢？

兔小白：因为房主图省事嘛！交给中介，房主就不用操心了，否则那么多租房人都联系房主，房主也忙不过来呀！

熊小猫：对呀，这就是中介存在的价值，他使得租房人和房主不用直接接触，消除了双方的多对多依赖。

兔小白：听起来怎么像在分析程序设计......你说的有些道理，但和交的中介费相比，我还是觉得中介提供的价值不够大。

熊小猫：那我推荐你一种新的中介形式，不但不用交中介费，而且中介提供的服务还会更多！

兔小白：还有这等好事？趁着我还没交中介费，赶快给我说说！

熊小猫：没问题！不过咱们先趁此机会学习一种设计模式——中介者模式。还是先从一个练习开始吧。

## 20.2 让中介出局会怎样

熊小猫：咱们先看看没有中介会怎么样。你来写一个没有中介的租房程序。租房的过程很简单，租房人交租金，房主收租金。

兔小白：这太简单了，我这就敲代码。首先定义房主HouseOwner类，它只有一个收房租的方法。

public class HouseOwner {  
    public void acceptRent(){  
        System.out.println("房主：已收到租金");  
   }  
}

接下来是租房人Tenant类，它依赖HouseOwner类。rentHouse方法用来向房主交房租。

public class Tenant {  
    private HouseOwner houseOwner;  
​  
    public Tenant(HouseOwner houseOwner) {  
        this.houseOwner = houseOwner;  
   }  
​  
    public void rentHouse(){  
        System.out.println("租房人：我要交租金");  
        houseOwner.acceptRent();  
   }  
}

客户端类代码如下。

HouseOwner houseOwner = new HouseOwner();  
Tenant tenant = new Tenant(houseOwner);  
​  
tenant.rentHouse();

程序输出如下。

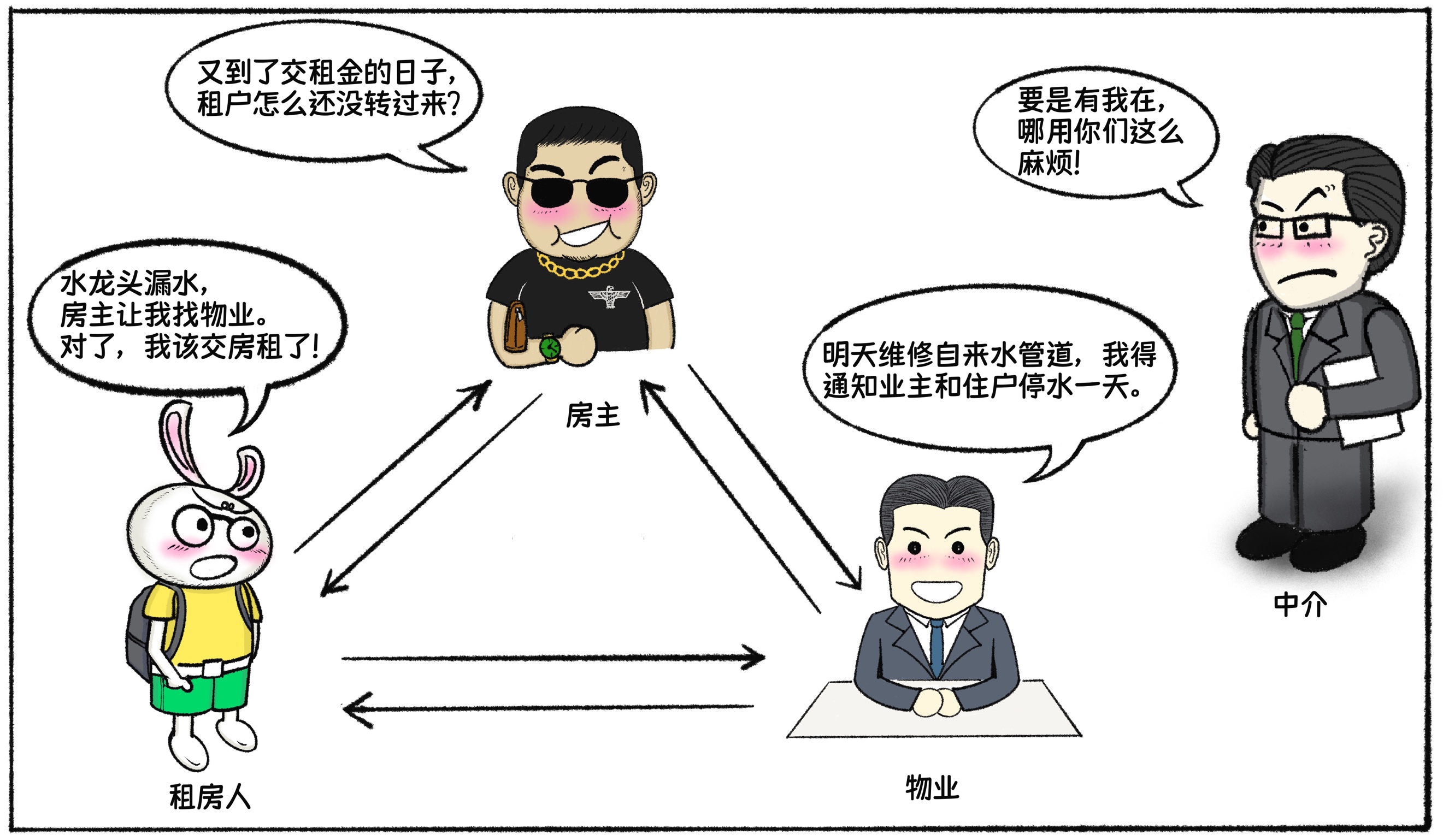
租房人：我要交租金  
房主：已收到租金

熊小猫：程序很简单，租房的过程并不复杂。如果只是参与租房过程，中介的价值确实有限。因此，中介也在不断进化自己的业务模式。现在有一种新的租房模式，中介能够体现更大的价值。

兔小白：我对这种新的租房模式很感兴趣，说来听听？

熊小猫：在整个租房周期中，这套房子涉及到的相关方不止租房人和房主，至少还有物业公司。我先将练习的需求加上物业公司。房主要向物业公司交物业费，物业公司需要向房主和租房人发布通知。租房人还会找物业修理基础设施，例如自来水管道。你看看如何修改程序。

兔小白：这里面的关系有点复杂,我先画图梳理一下。



兔小白：租房人、房主、物业，构成了三角依赖关系。他们的关系真的是说不清，理还乱呀!

熊小猫：如果需求中再加入保洁公司，那么关系会更复杂。房主、租房人、物业都可以找保洁公司打扫卫生。还可以继续加入供暖公司，房主需要交取暖费，供暖公司也会给房主和租房人发通知。

兔小白：再加上保洁公司和供暖公司，我就彻底理不清头绪了。即使我硬着头皮开发完，早晚也得出问题。程序肯定不能这样设计。

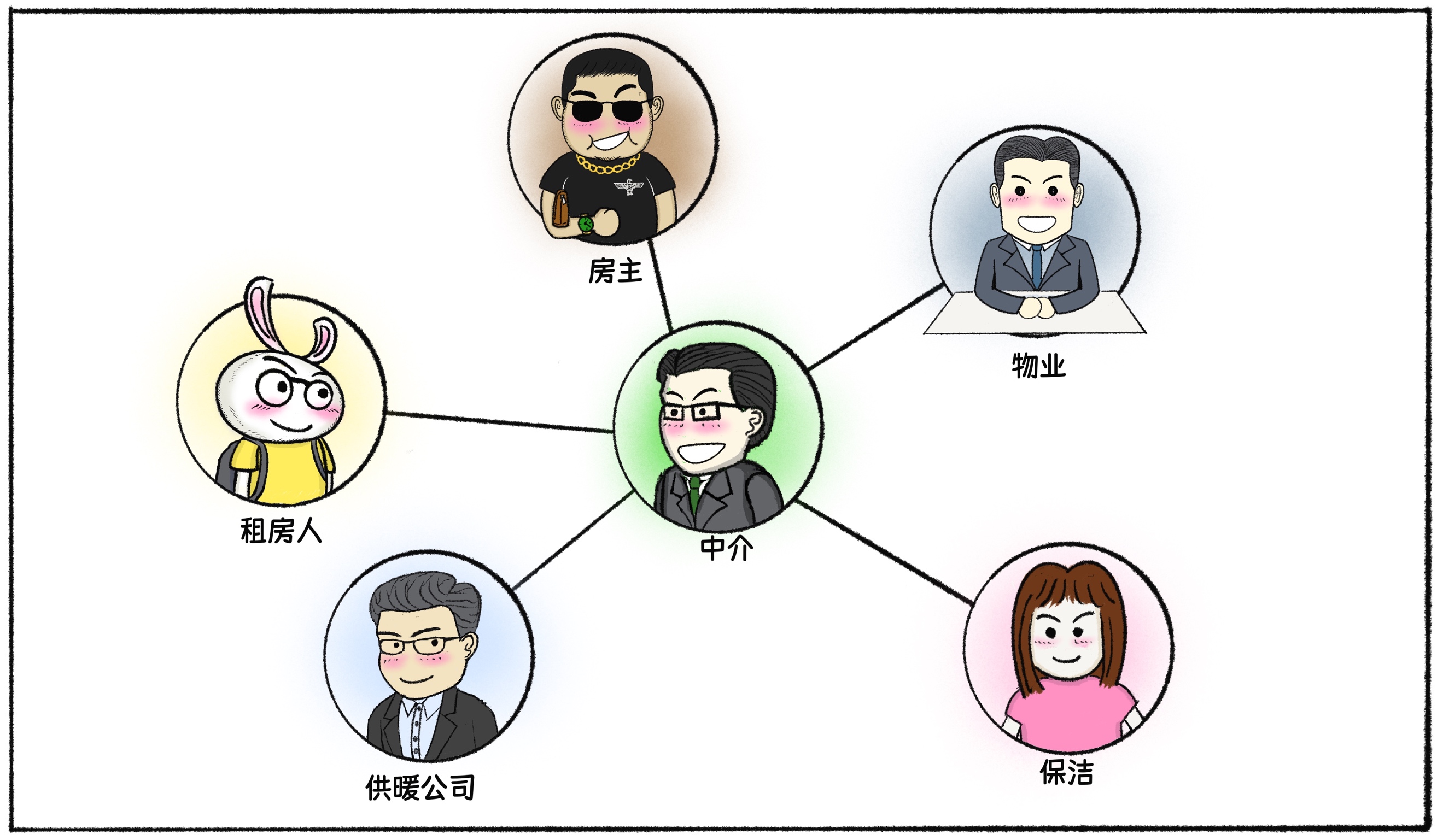
熊小猫：哪里混乱，哪里就有优化的空间，所以中介也是瞅准了这块空间，开展了新的业务。

## 20.3 中介协调，多方受益

兔小白：新的业务就是你说的不用交中介费的业务吧？

熊小猫：没错，我来给你介绍一下。现在房产中介有一种公寓式房源。中介从房主这里收房，你的房租交给中介，中介再交给房主。租房期间，房子的一切事宜都可以找中介解决，比如水管漏水，不用你找物业，直接找中介，中介会联系物业修理；房主要缴物业费，也可以找中介代缴；物业发通知，只需要发给中介，中介转给房主和租房人。

有了中介后，各个相关方都只依赖中介。哪怕加上保洁公司、供暖公司，也不会让相关方的依赖关系变得杂乱无序。相关方以中介为中心，构成“星”状结构。



兔小白：在出租期内，这套房子的所有相关方，有任何事情要办，都只需要找中介，中介负责协调办理。这样中介的价值确实凸显了出来。

熊小猫：房子的相关方越多，中介就越能体现他的价值。你的程序加上中介类，同样也能消除复杂的依赖关系。快去修改吧！

20分钟后兔小白改好了程序。

兔小白：这版程序增加了中介类EstateMediator。它负责接收相关方的请求，并转发给能够处理该请求的相关方。我最后再详细介绍它。

我们先看租房人Tenant类。acceptNotification方法用于接收通知，供中介对象调用。在rentHouse方法中，调用中介的方法交房租和修理自来水管道，不再需要调用房主和物业公司的方法。Tenant类只依赖EstateMediator类。

public class Tenant {  
    private EstateMediator estateMediator;  
​  
    public Tenant(EstateMediator estateMediator) {  
        this.estateMediator = estateMediator;  
   }  
​  
    public void acceptNotification(String notification){  
        System.out.println("租房人：已收到通知\""+notification+"\"");  
   }  
​  
    public void rentHouse(){  
        System.out.println("租房人：我要交租金");  
        estateMediator.handleRentFee();  
​  
        System.out.println("租房人：我要修理自来水管道");  
        estateMediator.fixWaterPipe();  
   }  
}

HouseOwner类有两个方法供中介调用，一个是acceptRent方法接收租金，另一个是acceptNotification方法用于接收通知。它还有一个方法payPropertyFee用于交物业费，通过调用中介的方法来实现，不需要调用物业公司的方法。

public class HouseOwner {  
    private EstateMediator estateMediator;  
​  
    public HouseOwner(EstateMediator estateMediator) {  
        this. estateMediator = estateMediator;  
   }  
​  
    public void  payPropertyFee(){  
        System.out.println("房主：我要交物业费");  
        estateMediator.handlePropertyFee();  
   }  
​  
    public void acceptRent(){  
        System.out.println("房主：已收到租金");  
   }  
​  
    public void acceptNotification(String notification){  
        System.out.println("房主：已收到通知\""+notification+"\"");  
   }  
}

物业公司PropertyCompany类中的publishNotification方法，通过中介发布通知，并不需要直接发给房主或者租房人。handlePropertyFee方法用于收取物业费。fixWaterPipe用于修理自来水管道。这2个方法供中介调用。

public class PropertyCompany {  
    private EstateMediator estateMediator;  
​  
    public PropertyCompany(EstateMediator estateMediator) {  
        this.estateMediator = estateMediator;  
   }  
​  
    public void handlePropertyFee() {  
        System.out.println("物业公司：已收到物业费，我给您开发票");  
   }  
​  
    public void fixWaterPipe(){  
        System.out.println("物业公司：自来水管道已经修好");  
   }  
​  
    public void publishNotification(String notification) {  
        System.out.println("物业公司：我发布了消息，内容是\"" + notification+"\"");  
        estateMediator.acceptNotification(notification);  
   }  
}

下面重点介绍相关方协作的中心——中介类EstateMediator。它封装了以上3个相关方的职责，统一对外提供调用接口。3个相关方能做的事情，EstateMediator都能做，但它其实只是传话筒。它的核心职责是整合资源、组织协作，而不是业务实现。

public class EstateMediator {  
    private HouseOwner houseOwner;  
    private Tenant tenant;  
    private PropertyCompany propertyCompany;  
​  
    public void init(HouseOwner houseOwner, Tenant tenant,  
                     PropertyCompany propertyCompany){  
        this.houseOwner = houseOwner;  
        this.tenant = tenant;  
        this.propertyCompany = propertyCompany;  
   }  
​  
    public void handlePropertyFee(){  
        propertyCompany.handlePropertyFee();  
   }  
​  
    public void fixWaterPipe(){  
        propertyCompany.fixWaterPipe();  
   }  
​  
    public void handleRentFee(){  
        houseOwner.acceptRent();  
   }  
​  
    public void acceptNotification(String notification){  
        tenant.acceptNotification(notification);  
        houseOwner.acceptNotification(notification);  
   }  
}

客户端代码中，先创建3个相关方对象以及中介对象，然后调用3个相关方的方法执行租房、交物业费、发布通知操作。

EstateMediator estateMediator = new EstateMediator();  
​  
HouseOwner houseOwner = new HouseOwner(estateMediator);  
PropertyCompany propertyCompany = new PropertyCompany(estateMediator);  
Tenant tenant = new Tenant(estateMediator);  
​  
estateMediator.init(houseOwner,tenant,propertyCompany);  
​  
tenant.rentHouse();  
houseOwner.payPropertyFee();  
propertyCompany.publishNotification("11月5日修理自来水管道，8：00-9：00停水");

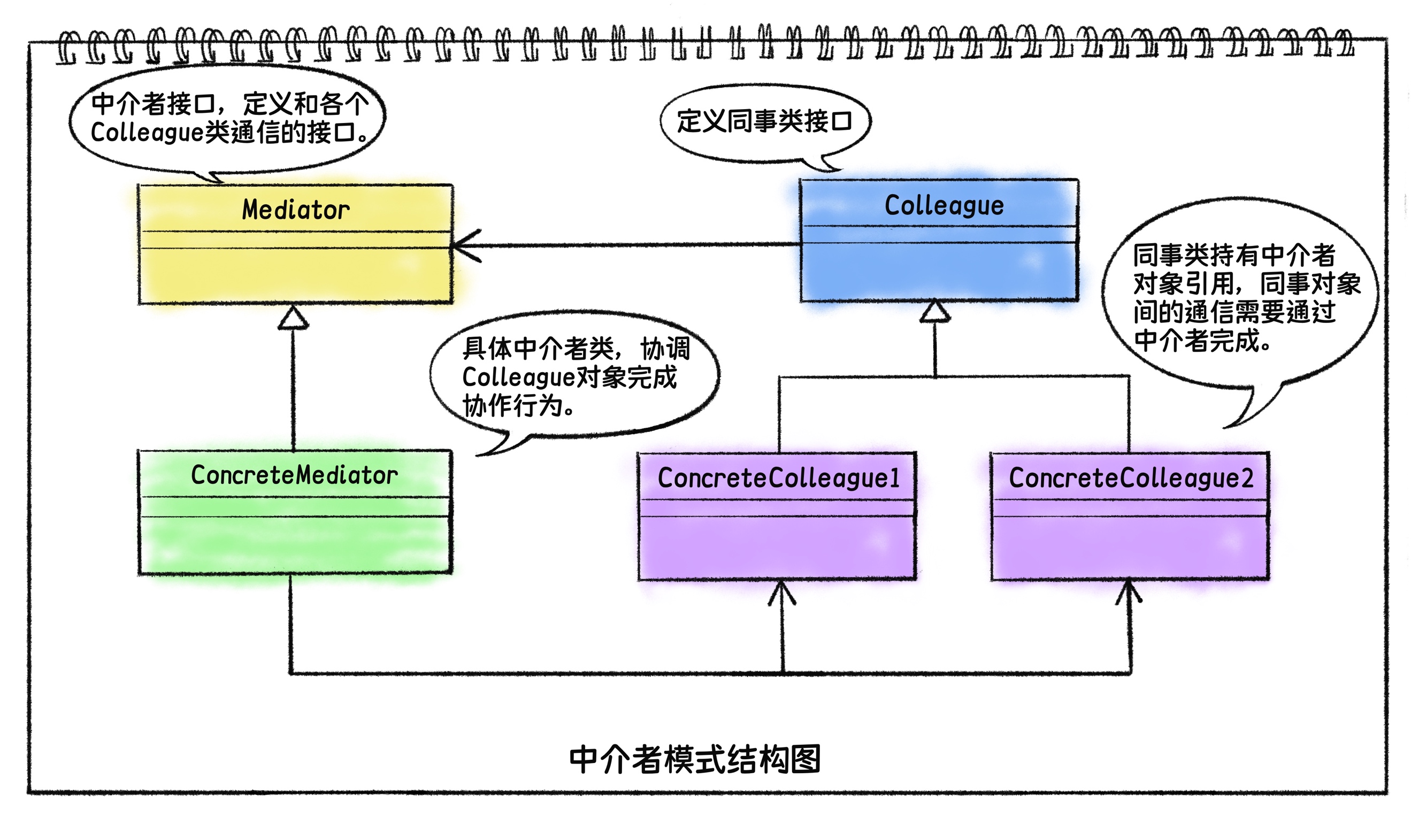
程序输出如下，符合预期。

租房人：我要交租金  
房主：已收到租金  
租房人：我要修理自来水管道  
物业公司：自来水管道已经修好  
房主：我要交物业费  
物业公司：已收到物业费，我给您开发票  
物业公司：我发布了消息，内容是"11月5日修理自来水管道，8：00-9：00停水"  
租房人：已收到消息"11月5日修理自来水管道，8：00-9：00停水"  
房主：已收到消息"11月5日修理自来水管道，8：00-9：00停水"

熊小猫：有了中介后，房屋相关方之间不再直接耦合，各方都和中介对话，中介再找到匹配的相关方来处理业务。这就是中介者模式，下面我们来详细分析。

## 20.4 中介者模式的优缺点及适用场景

熊小猫：我们先看看中介者模式的结构图。



Mediator是中介者接口，如果不需要扩展中介者，也可以不定义该接口。我们的练习中就没有定义该接口。ConcreteMediator是具体的中介者，协调各Colleague对象，协作完成业务操作。它对应练习中的EstateMediator。我们可以把一起配合完成工作的相关方看做“同事”。Colleague类是同事接口。在我们的例子中，各个同事类的差异很大，因此没有定义该接口。ConcreteColleague是具体同事类，实现自己的职责。它对应练习中的HouseOwner、Tenant等类。

中介者模式主要有以下两个优点。

1. 将各个Colleague类解耦。每个Colleague都有自己独特的职责。复杂业务，需要由多个Colleague相互配合才能完成。这会导致各个Colleague之间的依赖呈现出网状的多对多关系。Mediator将各个Colleague的职责集中在自己身上，从而让各个Colleague只需要依赖Mediator。
2. 对协作行为进行抽象。房屋中介对租房期间需要协调的行为进行了抽象、整合。无论在业务层面还是设计层面，中介者都产生于更高层面的抽象和提炼。Mediator在协调Colleague完成业务时，也可以加入自己的处理。例如发布通知的操作中，Mediator可以结合观察者模式，实现订阅/发布。

兔小白：中介者者模式确实优点明显，但是我看到一个问题。Mediator成为了协作的中心，所有Colleague都要围绕它来工作。随着系统的发展，它很可能成为一个无所不能的庞然大物，变得难以维护。

熊小猫：中心化确实是中介者模式的致命伤。所以在使用中介者模式的时候更要谨慎。千万不要觉得几个类之间的依赖关系比较复杂，就引入中介者。类之间存在依赖关系太正常不过。一个类和其他类没有任何关联，那么它也就失去了存在的价值。我们看看适合使用中介者模式的场景都具备哪些特征。

1. 同事类归属于同一块业务。处理同一块业务，才能称之为同事类。同一块业务也为中介者限定了业务范围，避免中介者无序扩张。但要注意，即使同一块业务，也要注意业务扩张的问题，不要让中介者增长失控。
2. 同事类属于同一类型。例如界面中的窗口、按钮、输入框等类都属于窗口组件类，可以通过中介者协调工作。同一类型组件的关联度更高，每个组件的变化，都可能引起其他组件的变化。 “同一类型”为中介者划清了协调职责的边界，不会让中介者变得过于庞大。
3. 同事类之间呈现网状依赖关系。如果依赖关系非常清晰，比如层次型的的依赖关系，那么即使依赖关系非常多，也不需要使用中介者模式优化。当类之间的依赖关系呈现网状，抓不住重点，也分不出层次的时候，才适合使用中介者模式。

兔小白：当多方依赖，陷入混乱，就是采用中介者模式的时候了。对了，你还没说不收中介费是怎么回事呢？为什么中介提供的服务更多，反而不收中介费了？

熊小猫：哈哈，房租也是钱，中介费也是钱，既然现在租房人只和中介打交道，把中介费匀到房租里不就行了？中介提供价值，那么收费也是合情合理。这就像我们做程序设计，虽然增加中介者，需要付出更多的开发成本，但也会带来很大收益。

兔小白：你说的有道理，下午我就心甘情愿地去交中介费。