# **耦合**

该组中的所有气味都会导致类之间的过度耦合，或者显示如果耦合被过度委派取代会发生什么。

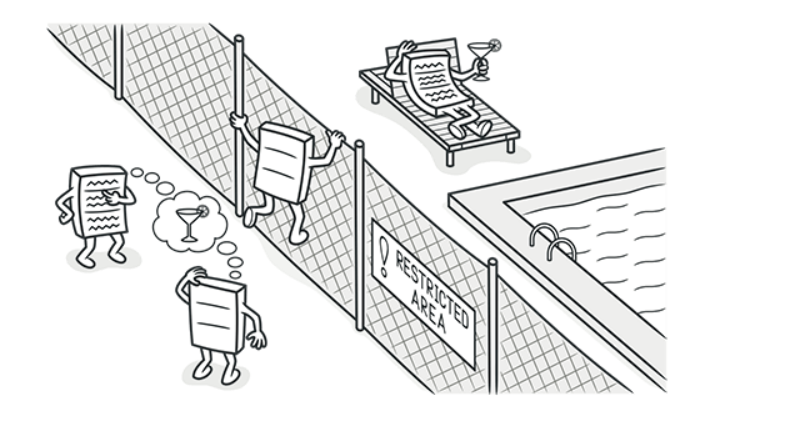
# **功能嫉妒**

### **体征和症状**

一个方法访问另一个对象的数据多于访问其自己的数据。

### **问题的原因**

将字段移动到数据类后，可能会出现这种气味。如果是这种情况，您可能还希望将数据操作移动到此类。



### **治疗**

作为一项基本规则，如果两个事件同时发生，您应该将它们放在同一个地方。通常，使用此数据的数据和函数会一起更改（尽管可能存在例外情况）。

如果某个方法应移动到其他位置，请使用**[Move 方法](https://refactoring.guru/move-method)**。

如果只有一个方法的一部分访问另一个对象的数据，请使用**[Extract 方法](https://refactoring.guru/extract-method)**移动相关部件。

如果某个方法使用来自其他几个类的函数，请首先确定哪个类包含所使用的大部分数据。然后将该方法与其他数据一起放在此类中。或者，使用**[Extract 方法](https://refactoring.guru/extract-method)**将方法拆分为多个部分，这些部分可以放置在不同类的不同位置。



### **收益**

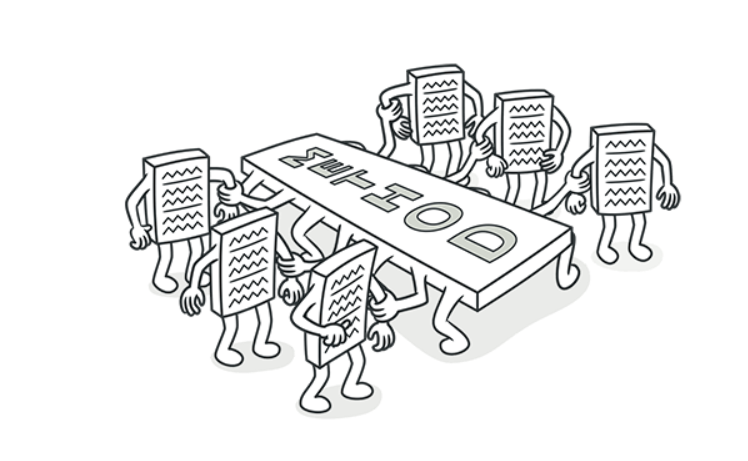
减少代码重复（如果数据处理的代码放在中心位置）。

更好的代码组织结构（处理数据的方法紧挨着实际数据）。

# **不恰当的亲密关系**

### **体征和症状**

一个类使用另一个类的内部字段和方法。



### **问题的原因**

密切关注那些花太多时间在一起的类。好的类应该尽可能少地了解彼此。这样的类更易于维护和重用。

### **治疗**

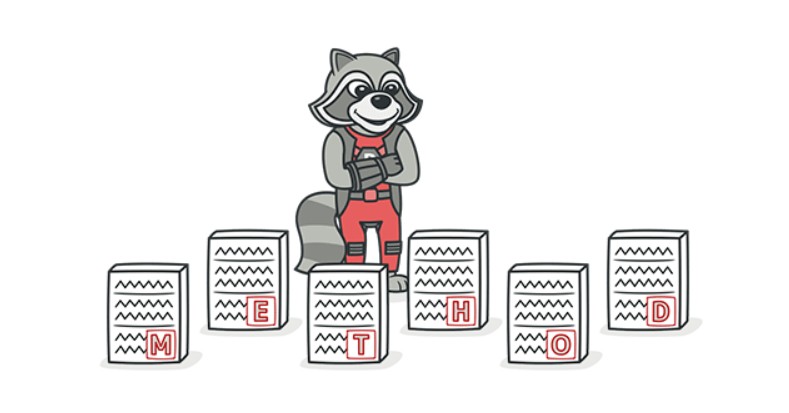


最简单的解决方案是使用**["移动方法"](https://refactoring.guru/move-method)**和**["移动字段"](https://refactoring.guru/move-field)**将一个类的各个部分移动到使用这些部分的类中。但是，只有当第一类真的不需要这些部件时，这才有效。

另一种解决方案是在**[类上使用"提取类](https://refactoring.guru/extract-class)["和"隐藏委托](https://refactoring.guru/hide-delegate)**"，以使代码关系成为"官方"。

如果这些类是相互依赖的，则应使用**["将双向关联更改为单向](https://refactoring.guru/change-bidirectional-association-to-unidirectional)**"。

如果这种"亲密关系"介于子类和超类之间，请考虑**[用继承 替换委派](https://refactoring.guru/replace-delegation-with-inheritance)**。



### **收益**

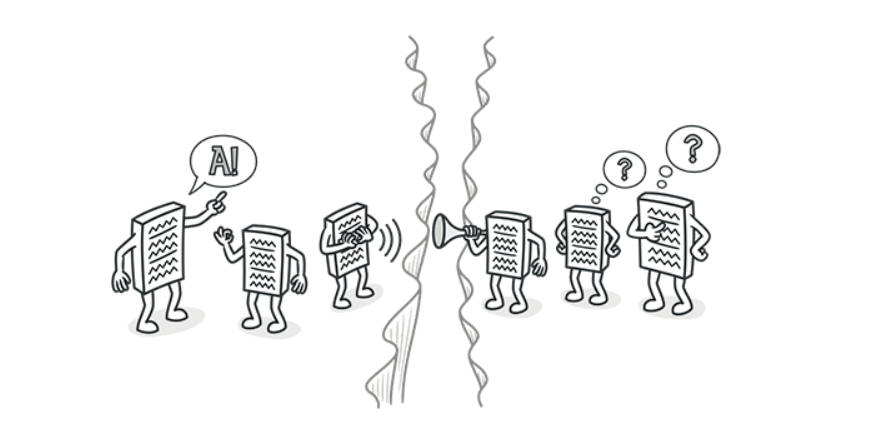
### 改进了代码组织结构。

简化支持和代码重用。

# **消息链**

### **体征和症状**

在代码中，您会看到一系列类似于$a->b()->c()->d()



### **问题的原因**

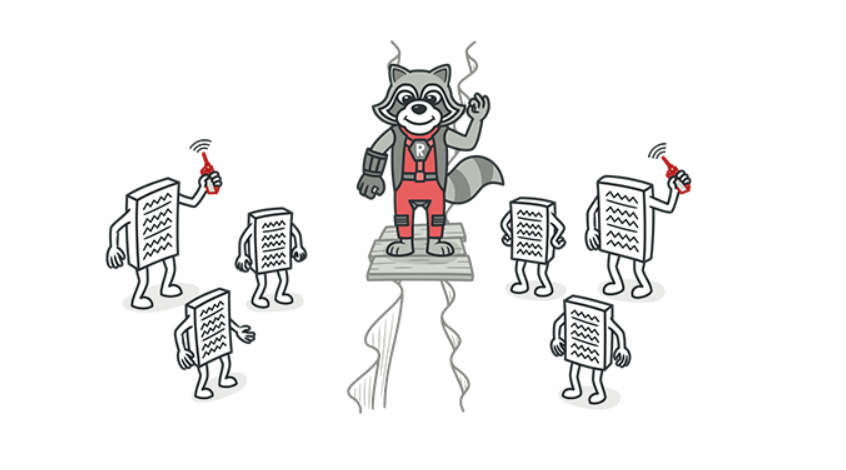
当客户端请求另一个对象，该对象请求另一个对象时，将发生消息链，依此类推。这些链意味着客户端依赖于类结构上的导航。这些关系中的任何更改都需要修改客户端。



### **治疗**

要删除消息链，请使用**[隐藏委托](https://refactoring.guru/hide-delegate)**。

有时，最好考虑一下使用最终对象的原因。也许使用**[Extract方法](https://refactoring.guru/extract-method)**来实现此功能并将其移动到链的开头是有意义的**[移动方法](https://refactoring.guru/move-method)**。



### **收益**

减少链类之间的依赖关系。

减少臃肿的代码量。

# **中间人**

### **体征和症状**

如果一个类只执行一个操作，将工作委托给另一个类，为什么它存在？



### **问题的原因**

这种气味可能是过度热衷于消除**[消息链](https://refactoring.guru/smells/message-chains)**的结果。

在其他情况下，它可能是一个类的有用工作逐渐移动到其他类的结果。除委托之外不执行任何其他操作。

### **治疗**

如果方法的大多数类委托给另一个类，**[则"删除中间人"](https://refactoring.guru/remove-middle-man)**是有序的。



### **收益**

更轻便的代码。