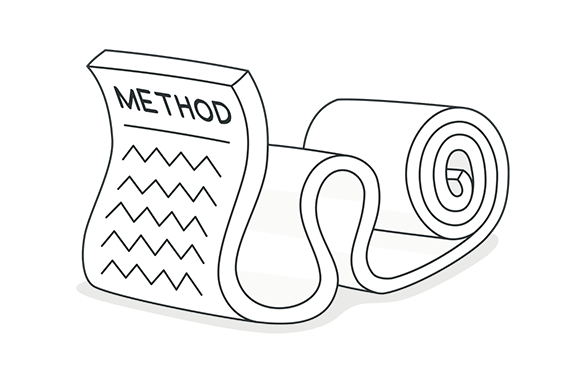
# 臃肿

臃肿是指代码、方法和类，它们已经增加到难以处理的巨大比例。通常这些气味不会立即出现，而是随着程序的发展而随着时间的推移而积累（尤其是当没有人努力消除它们时）。

# **长方法**

### **体征和症状**

一个方法包含太多的代码行。一般来说，任何超过十行的方法都可以提出质疑。



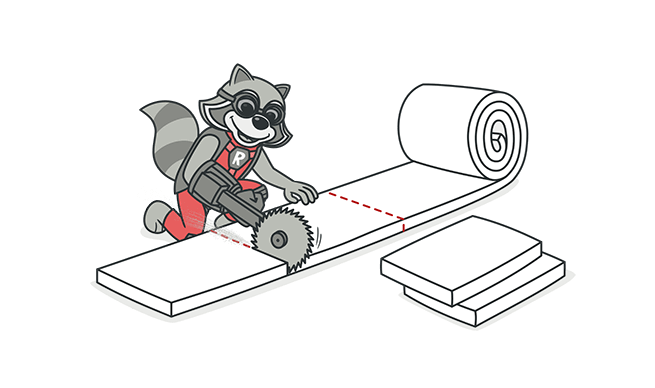
### **问题的原因**

就像“加利福尼亚酒店”一样，总是在方法中添加一些东西，但从来没有删除任何东西。由于编写代码比阅读代码更容易，因此这种“气味”在方法变成丑陋的、巨大的野兽之前不会引起注意。

从心理上讲，创建新方法通常比添加到现有方法更难：“这里只有两行代码，仅仅为此创建一个完整的方法是很浪费的……”因此我们给现有方法添加了一行，然后又一行，催生了一堆乱七八糟的“意大利面条”代码。

### **治疗**

根据经验，如果您觉得需要对方法中的某些内容进行注释，您应该使用此代码并将其放入新方法中。如果需要解释，即使是单行也可以而且应该拆分为单独的方法。如果该方法具有描述性名称，则不应该让其他人需要查看代码才能了解它的作用。



要减少方法体的长度，请使用**抽象方法**。

如果局部变量和参数干扰提取方法，请使用**以查询代替临时变量**、**引入参数对象**或**保存整个对象**等方法。

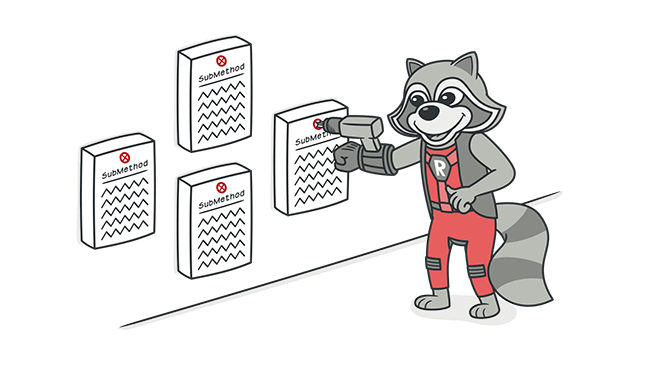
如果前面的方法都没有帮助，请尝试通过**用方法对象替换方法**将整个方法移动到单独的对象。

条件运算符和循环是可以将代码移至单独方法的一个很好的线索。对于条件，使用**分解条件表达式**。如果有循环，请尝试**抽象方法**。

### **益处**

在所有类型的面向对象代码中，具有短方法的类寿命最长。方法或功能越长，就越难理解和维护它。

此外，长方法为不需要的重复代码提供了完美的藏身之处。



### **表现**

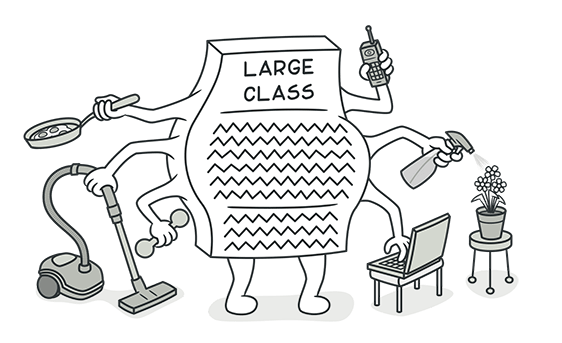
方法数量的增加是否会像许多人所说的那样损害性能？在几乎所有情况下，影响都微不足道，甚至不值得担心。

此外，既然您拥有清晰易懂的代码，您就更有可能找到真正有效的方法来重构代码并在需要时获得真正的性能提升。

# **巨大的类**

### **体征和症状**

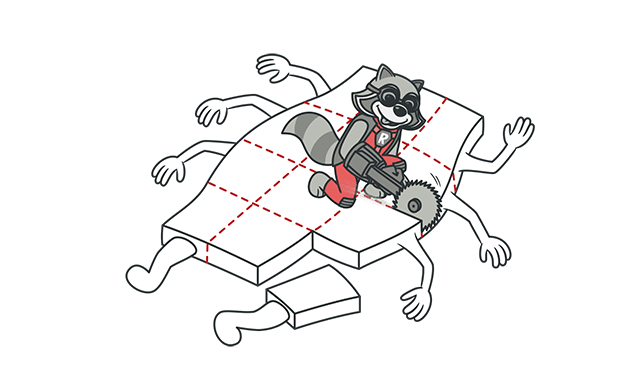
一个类包含许多字段/方法/代码行。



### **问题的原因**

课程通常从小开始。但随着时间的推移，随着程序的发展，它们会变得臃肿。

与长方法的情况一样，程序员通常会发现在现有类中放置新功能比为该功能创建新类更容易。



### **治疗**

当一个类戴着太多（功能性）帽子时，请考虑将其分开：

如果可以将大型类的部分行为拆分为单独的组件，则**[提取类](https://refactoring.guru/extract-class)**[会有所帮助](https://refactoring.guru/extract-class)**[。](https://refactoring.guru/extract-class)**

如果大类的部分行为可以以不同的方式实现或在极少数情况下使用，则**[提取子类](https://refactoring.guru/extract-subclass)**[会有所帮助](https://refactoring.guru/extract-subclass)**[。](https://refactoring.guru/extract-subclass)**

如果需要列出客户端可以使用的操作和行为，**[提取接口](https://refactoring.guru/extract-interface)**[会有所帮助](https://refactoring.guru/extract-interface)**[。](https://refactoring.guru/extract-interface)**

如果一个大类负责图形界面，您可能会尝试将它的一些数据和行为移动到一个单独的域对象中。这样做时，可能需要将某些数据的副本存储在两个位置并保持数据一致。**复制被监视的数据** 提供了一种方法来做到这一点。



### **益处**

这些类的重构使开发人员无需记住类的大量属性。

在许多情况下，将大类拆分为多个部分可以避免代码和功能的重复。

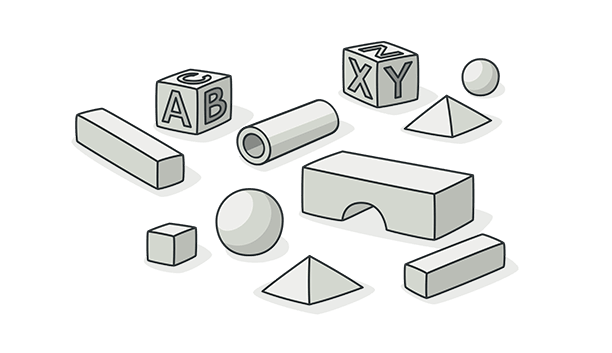
# **基本型别偏执**

### **体征和症状**

在简单任务中使用基本型别而不是对象（例如货币、范围、电话号码的特殊字符串等）

使用常量来编码信息（例如USER\_ADMIN\_ROLE = 1用于引用具有管理员权限的用户的常量。）

使用字符串常量作为数据数组中使用的字段名称。



### **问题的原因**

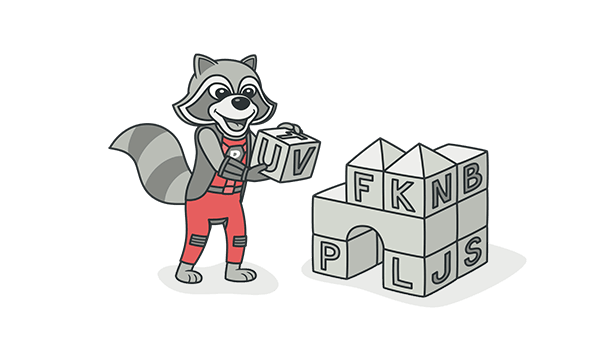
像大多数其他气味一样，基本型别的痴迷是在虚弱的时刻产生的。“只是一个存储一些数据的字段！” 程序员说。创建一个原始字段比创建一个全新的类要容易得多，对吧？就这样完成了。然后需要另一个字段并以相同的方式创建。瞧，这个类变得庞大而笨拙。

基本实体通常用于“模拟”类型。因此，您拥有一组数字或字符串，而不是单独的数据类型，这些数字或字符串构成了某些实体的允许值列表。然后通过常量为这些特定的数字和字符串赋予易于理解的名称，这就是它们广泛传播的原因。

原始字段使用不佳的另一个例子是现场模拟。该类包含大量不同数据和字符串常量（在类中指定）用作获取此数据的数组索引。

### **治疗**

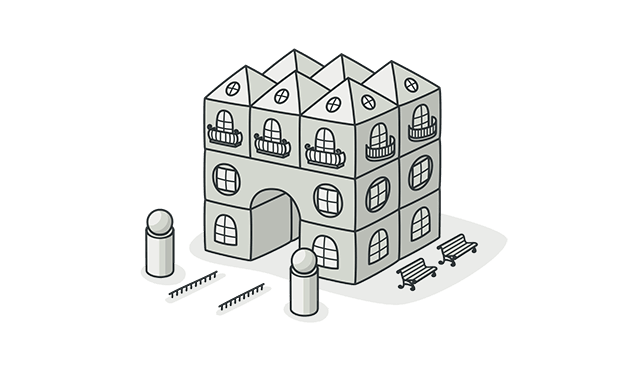
如果您有大量的原始字段，则可以将其中一些在逻辑上分组到它们自己的类中。更好的是，将与此数据相关的行为也移动到类中。对于此任务，请尝试**[将数据值替换为对象](https://refactoring.guru/replace-data-value-with-object)**。



如果原始字段的值用于方法参数，请使用**引入参数对象**或**保存整个对象**。

在变量中编码复杂数据时，请使用**用类替换类型代码**、**用子类替换类型代码**或**用状态/策略替换类型代码**。

如果变量中有数组，请使用**用对象替换数组**。



### **清偿**

由于使用对象而不是原语，代码变得更加灵活。

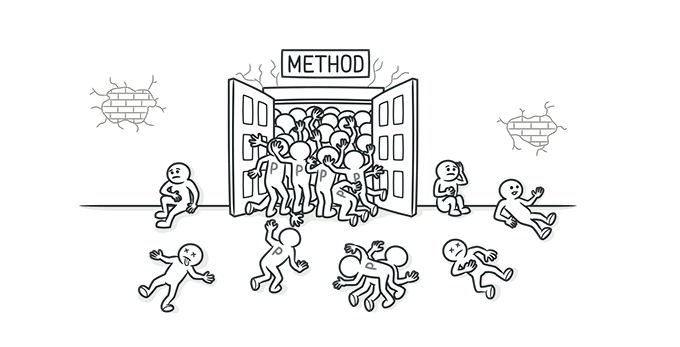
更好的可理解性和代码组织。对特定数据的操作都在同一个地方。无需再猜测所有这些奇怪常量的存在的原因以及它们为何存在于数组中。

更容易找到重复代码。

# **长参数列表**

### **体征和症状**

一个方法的参数超过三个。



### **问题的原因**

在将几种类型的算法合并到一个方法中之后，可能会出现一长串参数。可能已经创建了一个长列表来控制将运行哪个算法以及如何运行。

长参数列表也可能是努力使类彼此更加独立的副产品。例如，用于创建方法中所需特定对象的代码从方法移至调用方法的代码，但创建的对象作为参数传递给方法。这样原来的类就不再知道对象之间的关系了，依赖也减少了。但是如果创建了几个这样的对象，每个对象都需要自己的参数，这意味着更长的参数列表。

这样的列表很难以理解，随着它们的增长，它们变得矛盾且难以使用。一个方法可以使用它自己对象的数据，而不是一长串参数。如果当前对象不包含所有必要的数据，则可以将另一个对象（它将获取必要的数据）作为方法参数传递。

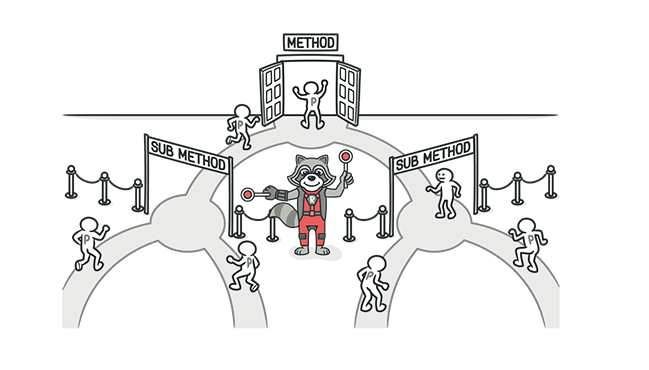
### **治疗**

检查传递给参数的值。如果某些参数只是另一个对象的方法调用的结果，请使用**用方法调用替换参数**。这个对象可以放在自己类的字段中，也可以作为方法参数传递。

不是将从另一个对象接收的一组数据作为参数传递，而是通过使用**保存整个对象**将对象本身传递给方法。

如果有多个不相关的数据元素，有时您可以通过**引入参数对象**将它们合并为单个参数对象。

益处



更易读、更短的代码。

重构可能会揭示以前未被注意到的重复代码。

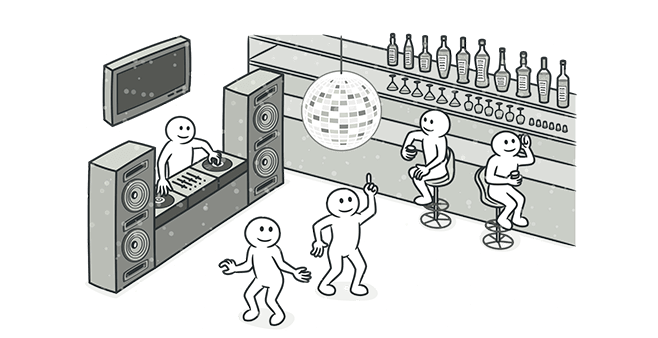
### **何时忽略**

如果这样做会导致类之间不需要的依赖关系，请不要删除参数。

# **数据块**

### **体征和症状**

有时代码的不同部分包含相同的变量组（例如用于连接数据库的参数）。这些团块应该变成它们自己的类。



### **问题的原因**

通常这些数据组是由于糟糕的程序结构或“复制粘贴编程”造成的。

如果要确定某些数据是否为数据块，只需删除其中一个数据值，然后查看其他值是否仍然有意义。如果不是这种情况，这是一个很好的迹象，表明这组变量应该组合成一个对象。

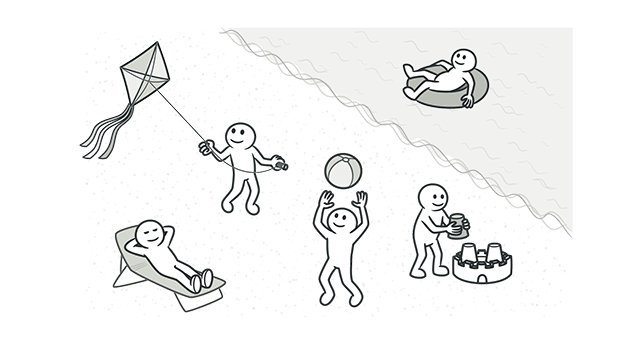
### **治疗**

如果重复数据包含类的字段，请使用**[提取类](https://refactoring.guru/extract-class)**将字段移动到它们自己的类。

如果在方法的参数中传递了相同的数据块，请使用**引入参数对象**将它们设置为一个类。

如果某些数据传递给其他方法，请考虑将整个数据对象传递给该方法，而不仅仅是单个字段。**[保留整个对象](https://refactoring.guru/preserve-whole-object)**将对此有所帮助。

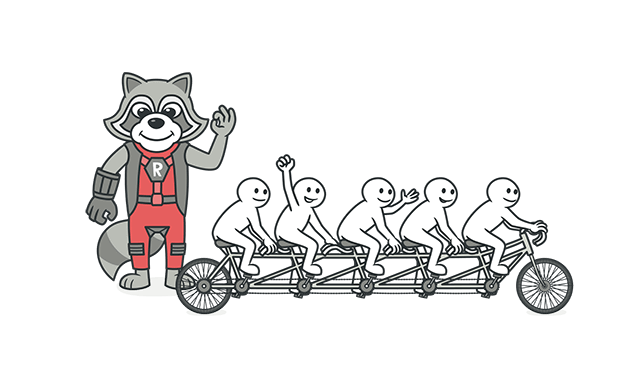
查看这些字段使用的代码。将此代码移至数据类可能是个好主意。



### **益处**

提高对代码的理解和组织。对特定数据的操作现在集中在一个地方，而不是在整个代码中随意进行。

减少代码大小。



### **何时忽略**

在方法的参数中传递整个对象，而不是仅传递其值（原始类型），可能会在两个类之间产生不希望的依赖关系。