# 重构，第一个案例

本章先介绍一个案例。

## 1.1 起点

如果你发现自己需要为程序添加一个特性，而代码结构使你无法很方便地达成目的，那就先重构那个程序，使特性的添加比较容易进行，然后添加新特性。

## 1.2 重构的第一步

重构的第一步永远相同，为修改的代码建立一组可靠的测试环境，这些测试机制必须有自我检验能力。

## 1.3 分解并重组statement()

提炼与分解冗长函数；

## 1.4 运用多态取代与价格相关的条件逻辑

## 1.5 结语

重构的节奏是测试、小修改、测试、小修改………。

# 重构原则

## 2.1何谓重构

重构：对软件内部结构的一种调整，目的是在不改变可观察行为的前提下，提高其可理解性，降低其修改成本。

重构简单理解就是整理代码，但是重构提供了很多高效的代码整理技术。

与性能优化不同，性能优化也不改变行为，但是性能优化后的代码更加难以理解。

添加新功能->测试->重构->添加新功能->测试->重构………….辩证法。

## 2.2 为何重构

改进软件设计，随着程序修改的进行，会越来越脱离原本的设计，这是累积性的，导致设计越来越难以理解，且重复率增高，影响开发速度；

使程序更容易理解，重构会向更让人理解的方向进行，同时在重构的过程中还可以对程序中不理解的地方进行深入了解，发现设计与隐晦上的一些规则。这是因为代码在重构后，逻辑清晰简洁，不了解的地方自然清晰；

重构能够暴露出原本程序中的BUG；

提高开发速度，设计结构清晰，代码容易理解，添加新功能与改BUG都会很快；良好的设计室快速开发的根本。

## 2.3 何时重构

重构可以随时进行，也有一定的规律，比如已经连续3次做同样的事，就应该重构；

添加新功能时重构，添加新功能需要理解现有代码，此时重构可以理解逻辑并改进，并且使得添加新功能变得更加方便；

修补错误时重构，一眼能够看出BUG；

复审代码时重构，理解代码意图及设计。并做到及时修改。

## 2.4 怎么对经理说

复审时引入重构，不告诉经理，或者直接自己修改，谁也不告诉；重构往往是加入中间层，但是中间层有害有利；间接层的价值：允许逻辑共享、分开解释意图与实现、隔离变化、封装条件逻辑。

## 2.5 重构的难题

重构的难题是不知道重构的局限性在哪里，出现问题比较多的就是数据库，数据库与程序逻辑耦合较多，如果耦合严重影响策程序就需要重构，加一个间隔层；修改接口要慎重，已发布接口很难修改，可以定义新函数，老函数调用新函数；难以重构时则修改程序设计；程序太烂就不要重构了，直接重写吧，项目最后期限到了时，也不要重构了。

## 2.6 重构与设计

重构与设计互补，软件不可能一开始就考虑到所有情况，并设计好，这压力很大，所以先进行一个简单的设计，实行的过程中，随着理解加深，在进行重构，使系统逐步达到灵活性又满足要求。

## 2.7 重构与性能

重构会使软件变慢，但是重构能够让人理解，这样就知道如何优化它，使它变快；编写快速软件的3个方法：时间预算（给每个模块预先分配时间）、持续关注法（任何时候开发都要保持高性能）、性能提升法（优化使系统变慢的部分的代码）；一个构造良好的系统可以让你节约时间来进行调优，让你理解程序逻辑结构知道在哪里调优怎么调优。

## 2.8 重构起源何处

重构起源于smalltalk。

# 代码的坏味道

何时需要重构很重要，这需要经验。

## 3.1 重复代码

重复部分的代码提炼出来，构成独立的个体，或为函数或为类。

## 3.2 过长函数

短函数更好，起一个好名字，只要是注释的语句，就可以提炼出单独的函数。

## 3.3 过大的类

提炼出子类，函数提炼。

## 3.4 过长参数列

放在类内共享。

## 3.5 发散式变化

代码的改动设计到很多方面，根据这些方面拆分类，每一个方面的改动只涉及一个类，更加集中。

## 3.6 霰弹式修改

一种变化引发多个类修改，提取出来。

## 3.7 依恋情节

## 3.8 数据泥团

抽取共同数据到新类。

## 3.9 基本类型偏执

应该总是将相关的小数据提炼成一个单个的类，虽然这个类可能也很基本（节省参数数量）。

## 3.10 switch惊悚现身

使用多态代替switch。

## 3.11 平行继承体系

## 3.12 冗赘类

## 3.13 夸夸其谈未来性

# 构筑测试体系

重构必须要可靠的测试环境。

## 4.1 自测试代码的价值

查找BUG会消耗很多的时间，每次增加一个小功能，都进行测试，如果有BUG就是这次增加的，在测试中对比结果，输出是否OK就可以的，编写测试在开始编程之前，有助于实现接口实现分离，常见的测试是在这个类中写一个main，更好的做法是建立一个独立类用于测试，并加在框架中。

## 4.2 JUnit测试框架

使用它进行测试，可以完成单元测试。

单元测试：测试一部分，假设其他部分正常。

功能测试：软件运行结果。

## 4.3 添加更多测试

不是所有都要进行测试，测试你最担心出错的部分。寻找边界条件，进行测试。

# 重构列表

# 重新组织函数

重构手法：Extract Method，抽取函数，Inline Method，内联函数，完全相反；处理变量、参数以及改进算法。

## 6.1 Extract Method

函数太长或许要注释就要抽取函数，小函数可以增加复用性及容易修改，函数名要与函数体逻辑相符合；做法：将代码贴到新函数，检查新函数中用到的原函数变量，有改变的变量可以用返回值代替，或者封装一下这些临时变量；读取的变量作为参数。

## 6.2 Inline Method

在函数调用点插入这个函数，再移除函数。

## 6.3 Inline Temp内联临时变量

在使用这个临时变量的地方直接使用赋值表达式。临时变量要修改成final类型。

## 6.4 Replace Temp with Query，以查询取代临时变量

将表达式提炼到函数中，将临时变量的所有引用点用这个函数调用代替。适合于只被赋值一次的变量；临时变量要修改成Final的。

## 6.5 Introduce Explaining Variable 引入解释性变量

将复杂表达式的一部分提取出来，给一个名字相符的临时变量。变量是Final的。

## 6.6 Split Temporary Variable，分解临时变量

函数中临时变量多处被赋值，既不是循环变量也不被用于收集结果。每次赋值创造一个独立、对应的临时变量。一个一个起新的名字并修改引用点，注意Final。

## 6.7 Remove Assignment to Parameters 移除对参数的赋值

代码对参数赋值，以一个临时变量取代参数的位置。

## 6.8 Replace Method with Method Object 以函数对象取代函数

大型函数，局部变量使得无法采用Extract Method，将这个函数独立为一个类，局部变量抽取为类属性，方法拆分为多个小的方法。在新类中创建final类型的原函数所在的对象，临时变量做属性，通过这些字段写一个构造函数，新起一个函数名，把原来的代码写到这个新函数中去。

## 6.9 Substitite Algorithm 替换算法

将函数体替换为另一个更清晰的算法。

# 第七章 在对象之间搬移特性

这一章讲述是如何重构类。

## 7.1 Move Method 搬移函数

有函数与所在类之外的另一个类使用的多，在该函数引用的类中建立一个实现同样功能的新函数，将旧函数变成委托函数。这是因为类之间耦合太过的原因。

## 7.2 Move Field 搬移字段

类中的某个字段被另一个类常使用，把字段移到另一个类中，修改源字段的所有用户，令他们改用新字段。

## 7.3 Extract Class 提炼类

某个类做了应该由2个类做的事，建立一个新类，将相关的字段和函数从旧类搬移到新类；类会逐渐变大，根据数据与方法的聚合性拆分类。

## 7.4 Inline Class 将类内联化

与提炼类相反，合并类。

## 7.5 Hide Delegate 隐藏委托关系

建立委托

## 7.6 Remove Middle Man 移除中间人

某个类做了过多的简单委托动作，让上层类直接调用受委托类。

## 7.7 Introduce Foreign Method 引入外加函数

需要哦为服务类增加一个函数，无法修改这个类，在客户类中建立一个函数，并以第一参数形式传入一个服务类实例，在这个函数中实现功能。

## 7.8 Introduce Local Extension 引入本地拓展

为服务类提供额外的函数，无法修改类，建立一个新类，包含这些额外函数，使这个类成为源类的子类或者包装类；子类化工作量少，子类化会拥有2个完全相同的父类对象，包含就会同步修改；建立一个拓展类，子类（父类构造函数构造父类对象），包含（包含类的构造函数，里面有类的属性），在拓展类中加入新特性。

# 重新组织数据

## 8.1 Self Encapsulate Field 自封装字段

直接访问一个字段，与字段的耦合紧密，为字段建立取值/设值函数，以函数的方式访问字段。

## 8.2 Repace Data Value with Object 以对象取代数据值

数据与其他数据或者行为一起时才有用，将这些数据与行为组合成对象。

## 8.3 Change Value to Reference 将值对象改为引用对象

用到了很多相同的值，替换为一个引用对象，节省代码数量。

## 8.4 Change Reference to Value 将引用对象改为值对象

## 8.5 Replace Array with Object 以对象取代数组

有一个数组，其中的元素各自代表不同的内容，用对象取代。

## 8.6 Duplicate Observed Data 复制被监视数据

## 8.16 Replace Subclass with Fields 以字段取代子类

子类之间差别很小，可能只在于返回函数返回的字段上，那么把字段移到超类中变为字段，去掉子类。

# 简化表达式

## 9.1 Decompose Conditional 分解条件表达式

复杂的if-else if-else，从每个分支提炼出独立的函数。

## 9.2 Consolidate Condition Expression 合并条件表达式

一系列条件测试，这些测试的结果均相同，将这些测试合并为一个条件表达式，再将条件表达式提炼成为一个独立函数。

## 9.3 Consolidate Duplicate Conditional Fragments 合并重复的条件片段

在条件表达式的每个分支上有着相同的代码，将重复的代码搬移到条件表达式之外。

## 9.4 Remove Control Flag 移除控制标记

使用break或者continue代替控制标记。

## 9.5 Replace Nested Conditional with Guard Clauses 以卫语句取代嵌套条件表达式

## 9.6 Replace Conditional with Polymorphism以多态取代条件表达式

条件表达式中，每个分支根据对象类型的不同而选择不同的行为，将这个条件表达式的每个分支放进一个子类内的覆写函数中，将原始函数声明为抽象函数。

## 9.7 Introduce Null Object 引入Null对象

# 简化函数调用

## 10.1 Rename Method 函数改名

函数的名称未能揭示函数的用途，改名；命名的办法：将注释变成名称。

## 10.2 Add Parameter 添加参数

某个函数需要从调用端得到更多的消息，将参数封装为对象，让对象带入函数所需要的信息。

## 10.3 Remove Parameter 移除参数

函数本体不再需要某个参数，将该参数除去。

## 10.4 Separate Query from Modifier 将查询函数与修改函数分离

某个函数即返回对象值，又修改对象状态；建立2个不同的函数，其中一个负责查询，另一个负责修改。

## 10.5 Parameterize Method 令函数携带参数

若干函数做了相同的工作，但在函数本体中却包含了不同的值，建立单一函数，以参数表达那些不同的值。

## 10.6 Replace Parameter with Explicit Methods 以明确函数取代参数

函数的参数，根据值不同采取不同的行为，针对每个值及行为建立一个独立函数。

## 10.7 Preserve Whole Object 保持对象完整

从某个对象中取出若干值，将他们作为某一次函数调用时的参数，改为传递整个对象。

## 10.8 Replace Parameter with Methods 以函数取代参数

节省中间的调用环节。

## 10.9 Introduce Parameter Object 引入参数对象

## 10.10 Remove Setting Method 移除设值函数

## 10.11 Hide Method 隐藏函数

有一个函数，从来没被其他类用到，将这个函数修改为private。

## 10.12 Replace Constructor with Factory Method 以工厂函数取代构造函数

## 10.13 封装向下转型

## 10.14 Replace Error Code with Exception以异常取代错误码

## 10.15 Replace Exception with Test 以测试取代异常

# 处理概括关系

## 11.1 Pull Up Field 字段上移

两个子类拥有相同的字段，将该字段移至超类。

## 11.2 Pull Up Method 函数上移

有些函数在各个子类中产生完全相同的结果，将该函数移至超类。

## 11.3 Pull Up Constructor Body 构造函数本体上移

各个子类中拥有一些构造函数，本体一致，在超类中新建一个构造函数，并在子类构造函数中调用它。

## 11.4 Push Down Method 函数下移

超类中某个函数只与部分子类有关，将这个函数移到相关的子类中去。

## 11.5 Push Down Field 字段下移

超类中某个字段只被部分子类用到，将这个字段移到需要它的那些子类中去。

## 11.6 Extract SubClass 提炼子类

## 11.7 Extract SupurClass 提炼超类

## 11.8 Extract Interface 提炼接口

## 11.9 Collapse Hierarchy 折叠继承体系

## 11.10 Form Template Method 塑造模板函数

## 11.11 Replace Inheritance with Delegation 以委托取代继承

## 11.12 Replace Delegation with Inheritance 以继承取代委托

# 大型重构

大型重构是没有确切的步骤的，耗费事件长，为整个团队建立共识。

## 12.1 Tease Apart Inheritance 梳理并分解继承体系

某个继承体系同时承担2项责任，建立2个继承体系，并通过委托关系让其中一个可以调用另一个；

## 12.2 Convert Procedural Design to Objects 将过程化设计转化为对象设计