编 程 学 习 笔 记

重构：改善既有代码的设计

Code

作 者 姓 名： 颜佳

目 录

[1 识别代码坏味道 1](#_Toc76293584)

[1.1 神秘命名（Mysterious Name） 1](#_Toc76293585)

[1.2 重复代码（Duplicated Code） 1](#_Toc76293586)

[1.3 过长函数（Long Function） 1](#_Toc76293587)

[1.3.1 需要注释时就要提炼函数，并以其用途命名，即使只有一行代码 1](#_Toc76293588)

[1.3.2 if语句，需分解条件表达式，每个分支都要提炼为函数调用 1](#_Toc76293589)

[1.3.3 switch语句，每个分支都要提炼为函数调用 1](#_Toc76293590)

[1.3.4 for和while循环，循环内代码提炼函数处理 1](#_Toc76293591)

[1.4 过长参数列表（Long Parameter List） 1](#_Toc76293592)

[1.4.1 使用查询取代参数传递 1](#_Toc76293593)

[1.4.2 保持对象完整——直接传递现有的数据结构而不是拆分传递 1](#_Toc76293594)

[1.4.3 引入参数对象——创建类或者结构体 1](#_Toc76293595)

[1.4.4 移除标记参数 1](#_Toc76293596)

[1.5 全局数据（Global Data） 1](#_Toc76293597)

[1.5.1 封装变量——把全局数据用函数封装起来 1](#_Toc76293598)

[1.6 可变数据（Mutable Data） 2](#_Toc76293599)

[1.6.1 封装变量——确保更新操只能通过几个函数来进行 2](#_Toc76293600)

[1.6.2 拆分变量——拆分为不同用途的变量 2](#_Toc76293601)

[1.7 发散式变化（Divergent Change） 2](#_Toc76293602)

[2 重构方法 2](#_Toc76293603)

[2.1 提炼函数（Extract Function） 2](#_Toc76293604)

[2.1.1 动机 2](#_Toc76293605)

[2.1.2 做法 2](#_Toc76293606)

[2.2 内联函数（Inline Function） 3](#_Toc76293607)

[2.2.1 动机 3](#_Toc76293608)

[2.2.2 做法 3](#_Toc76293609)

[2.3 提炼变量（Extract Variable，Introduce Explaining Variable） 3](#_Toc76293610)

[2.3.1 动机 3](#_Toc76293611)

[2.3.2 做法 3](#_Toc76293612)

[2.4 内联变量（Inline Variable，Inline Temp） 3](#_Toc76293613)

[2.4.1 动机 3](#_Toc76293614)

[2.4.2 做法 3](#_Toc76293615)

[2.5 改变函数声明（Change Function Declaration） 4](#_Toc76293616)

[2.5.1 动机 4](#_Toc76293617)

[2.5.2 做法 4](#_Toc76293618)

[2.6 封装变量（Encapsulate Variable） 4](#_Toc76293619)

[2.6.1 动机 4](#_Toc76293620)

[2.6.2 做法 4](#_Toc76293621)

[2.7 变量改名（Rename Variable） 5](#_Toc76293622)

[2.7.1 动机 5](#_Toc76293623)

[2.7.2 做法 5](#_Toc76293624)

[2.8 引入参数对象（Introduce Parameter Object） 5](#_Toc76293625)

[2.8.1 动机 5](#_Toc76293626)

[2.8.2 做法 5](#_Toc76293627)

[2.9 函数组合成类（Combine Functions into Class） 5](#_Toc76293628)

[2.9.1 动机 5](#_Toc76293629)

[2.9.2 做法 5](#_Toc76293630)

[2.10 函数组合成变换（Combine Functions into Transform） 5](#_Toc76293631)

[2.10.1 动机 5](#_Toc76293632)

[2.10.2 做法 5](#_Toc76293633)

[2.11 拆分阶段（Split Phase） 5](#_Toc76293634)

[2.11.1 动机 6](#_Toc76293635)

[2.11.2 做法 6](#_Toc76293636)

[3 简化条件逻辑 6](#_Toc76293637)

[3.1 分解条件表达式（Decompose Conditional） 6](#_Toc76293638)

[3.1.1 动机 6](#_Toc76293639)

[3.1.2 做法 6](#_Toc76293640)

[3.2 分解条件表达式（Decompose Conditional） 6](#_Toc76293641)

[3.2.1 动机 6](#_Toc76293642)

[3.2.2 做法 6](#_Toc76293643)

[3.3 分解条件表达式（Decompose Conditional） 6](#_Toc76293644)

[3.3.1 动机 6](#_Toc76293645)

[3.3.2 做法 6](#_Toc76293646)

[3.4 分解条件表达式（Decompose Conditional） 6](#_Toc76293647)

[3.4.1 动机 6](#_Toc76293648)

[3.4.2 做法 6](#_Toc76293649)

[3.5 分解条件表达式（Decompose Conditional） 6](#_Toc76293650)

[3.5.1 动机 6](#_Toc76293651)

[3.5.2 做法 7](#_Toc76293652)

[3.6 分解条件表达式（Decompose Conditional） 7](#_Toc76293653)

[3.6.1 动机 7](#_Toc76293654)

[3.6.2 做法 7](#_Toc76293655)

[3.7 分解条件表达式（Decompose Conditional） 7](#_Toc76293656)

[3.7.1 动机 7](#_Toc76293657)

[3.7.2 做法 7](#_Toc76293658)

[3.8 分解条件表达式（Decompose Conditional） 7](#_Toc76293659)

[3.8.1 动机 7](#_Toc76293660)

[3.8.2 做法 7](#_Toc76293661)

[3.9 分解条件表达式（Decompose Conditional） 7](#_Toc76293662)

[3.9.1 动机 7](#_Toc76293663)

[3.9.2 做法 7](#_Toc76293664)

[3.10 分解条件表达式（Decompose Conditional） 7](#_Toc76293665)

[3.10.1 动机 7](#_Toc76293666)

[3.10.2 做法 7](#_Toc76293667)

[3.11 分解条件表达式（Decompose Conditional） 7](#_Toc76293668)

[3.11.1 动机 7](#_Toc76293669)

[3.11.2 做法 8](#_Toc76293670)

[3.12 分解条件表达式（Decompose Conditional） 8](#_Toc76293671)

[3.12.1 动机 8](#_Toc76293672)

[3.12.2 做法 8](#_Toc76293673)

[3.13 分解条件表达式（Decompose Conditional） 8](#_Toc76293674)

[3.13.1 动机 8](#_Toc76293675)

[3.13.2 做法 8](#_Toc76293676)

# 识别代码坏味道

## 神秘命名（Mysterious Name）

## 重复代码（Duplicated Code）

## 过长函数（Long Function）

### 需要注释时就要提炼函数，并以其用途命名，即使只有一行代码

### if语句，需分解条件表达式，每个分支都要提炼为函数调用

### switch语句，每个分支都要提炼为函数调用

如果有多个switch语句基于同一个条件进行分支选择，就应该使用以多态取代条件表达式。

### for和while循环，循环内代码提炼函数处理

将循环和循环内的代码提炼到一个独立的函数中。如果发现提炼出的循环很难命名，可能是因为其中做了几件不同的事。如果是这种情况，请勇敢地使用**拆分循环**将其拆分成各自独立的任务。

## 过长参数列表（Long Parameter List）

### 使用查询取代参数传递

### 保持对象完整——直接传递现有的数据结构而不是拆分传递

### 引入参数对象——创建类或者结构体

### 移除标记参数

## 全局数据（Global Data）

包含全局变量，类变量和单例

### 封装变量——把全局数据用函数封装起来

## 可变数据（Mutable Data）

### 封装变量——确保更新操只能通过几个函数来进行

### 拆分变量——拆分为不同用途的变量

## 发散式变化（Divergent Change）

某个模块经常因为不同的原因在不同的方向上发生变化

# 重构方法

## 提炼函数（Extract Function）

### 动机

1. 函数过长。
2. 函数复用。
3. **将意图与实现分开，提炼函数，并以用途命名。**

### 做法

1. 创造一个新函数，根据这个函数的意图来对它命名。
2. 将待提炼的代码从源函数复制到新建的目标函数中。
3. 检查提炼出的代码，看看其中是否引用了作用域限于源函数、在提炼出的新函数中访问不到的变量。若是，以参数的形式将它们传递给新函数。

* 如果这些变量在提炼部分没有重新赋值，直接传入。
* 如果一个变量在提炼部分被重新赋值，把赋值后的变量返回。
* 如果多个变量在提炼部分被重新赋值，先使用其他重构手法（查询取代临时变量，拆分变量）简化变量。

1. 在源函数中，将被提炼代码段替换为对目标函数的调用。

## 内联函数（Inline Function）

### 动机

1. 函数内容和函数名称一样清晰易读，去掉函数直接使用代码；

### 做法

1. 检查函数，确定它不具多态性。
2. 如果该函数属于一个类，并且有子类继承了这个函数，那么就无法内联。
3. 找出这个函数的所有调用点。
4. 将这个函数的所有调用点都替换为函数本体。
5. 每次替换之后，执行测试。
6. 删除该函数的定义。

## 提炼变量（Extract Variable，Introduce Explaining Variable）

### 动机

1. 表达式非常复杂而难以阅读；
2. 确定上下文决定提取变量还是提取函数（在类中）

### 做法

1. 确认**要提炼的表达式没有副作用**。
2. 声明一个不可修改的变量，把你想要提炼的表达式复制一份，以该表达式的结果值给这个变量赋值。
3. 用这个新变量取代原来的表达式。
4. 测试。

## 内联变量（Inline Variable，Inline Temp）

### 动机

1. 变量名与表达式一样清晰易读，去掉变量直接使用表达式；

### 做法

1. 检查确认变量赋值语句的右侧表达式没有副作用。
2. 如果变量没有被声明为**不可修改，先将其变为不可修改**，并执行测试。

这是为了确保该变量只被赋值一次。

1. 找到第一处使用该变量的地方，将其替换为直接使用赋值语句的右侧表达式。
2. 测试。
3. 重复前面两步，逐一替换其他所有使用该变量的地方。
4. 删除该变量的声明点和赋值语句。
5. 测试。

## 改变函数声明（Change Function Declaration）

先写一句注释描述这个函数的用途，再把这句注释变成函数的名字。

### 动机

1. **函数改名；**
2. **修改参数列表**

### 做法

**旧函数作为转发函数，即旧代码调用旧函数，旧函数再调用新函数**

1. 如果想要移除一个参数，需要先确定函数体内没有使用该参数。
2. 修改函数声明，使其成为你期望的状态。
3. 找出所有使用旧的函数声明的地方，将它们改为使用新的函数声明。
4. 测试。

## 封装变量（Encapsulate Variable）

**对于所有可变的数据，只要它的作用域超出单个函数，将其封装起来，只允许通过函数访问**

### 动机

1. 搬移一处被广泛使用的数据，先以函数形式封装所有对该数据的访问；

### 做法

1. 创建封装函数，在其中访问和更新变量值。
2. 执行静态检查。
3. 逐一修改使用该变量的代码，将其改为调用合适的封装函数。每次替换之后，
4. 执行测试。
5. 限制变量的可见性。

## 变量改名（Rename Variable）

### 动机

### 做法

## 引入参数对象（Introduce Parameter Object）

### 动机

1. 数据泥团

### 做法

1. 如果暂时还没有一个合适的数据结构，就创建一个。
2. 使用改变函数声明给原来的函数新增一个参数，类型是新建的数据结构。
3. 调整所有调用者，传入新数据结构的适当实例。每修改一处，执行测试。
4. 用新数据结构中的每项元素，逐一取代参数列表中与之对应的参数项，然后删除原来的参数。

## 函数组合成类（Combine Functions into Class）

### 动机

### 做法

## 函数组合成变换（Combine Functions into Transform）

### 动机

### 做法

## 拆分阶段（Split Phase）

### 动机

### 做法

# 简化条件逻辑

## 分解条件表达式（Decompose Conditional）

### 动机

复杂的条件逻辑

### 做法

对**条件判断和每个条件分支**分别运用提炼函数手法。

1. 将条件判断提炼到一个独立的函数。
2. 将每个条件分支提炼到一个独立的函数中。
3. **用三元运算符重新安排条件语句。**

## 合并条件表达式（Consolidate Conditional Expression）

### 动机

检查条件各不相同，最终行为却一致；

如果认为这些检查的确彼此独立，不应该被视为同一次检查，必要使用本项重构

### 做法

1. 确定这些条件表达式都没有副作用。
2. 使用适当的逻辑运算符，将两个相关条件表达式合并为一个。
3. 顺序执行的条件表达式用逻辑或来合并，嵌套的if语句用逻辑与来合并。
4. 重复前面的合并过程，直到所有相关的条件表达式都合并到一起。
5. 可以考虑对合并后的条件表达式实施提炼函数。

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

### 动机

条件表达式通常有两种风格。第一种风格是：两个条件分支都属于正常行为。第二种风格则是：只有一个条件分支是正常行为，另一个分支则是异常的情况。这两类条件表达式有不同的用途，这一点应该通过代码表现出来。如果两条分支都是正常行为，就应该使用形如if...else...的条件表达式；如果某个条件极其罕见，就应该单独检查该条件，并在该条件为真时立刻从函数中返回。这样的单独检查常常被称为“卫语句”（guard clauses）。

### 做法

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

### 动机

### 做法

## 分解条件表达式（Decompose Conditional）

### 动机

### 做法

## 分解条件表达式（Decompose Conditional）

### 动机

### 做法

## 分解条件表达式（Decompose Conditional）

### 动机

### 做法

## 分解条件表达式（Decompose Conditional）

### 动机

### 做法

## 分解条件表达式（Decompose Conditional）

### 动机

### 做法

# 重构API

## 将查询函数和修改函数分离（Separate Query from Modifier）

### 动机

任何有返回值的函数，都不应该有看得到的副作用

### 做法

1. 复制整个函数，将其作为一个查询来命名。
2. 从新建的查询函数中去掉所有造成副作用的语句。
3. 查找所有调用原函数的地方。如果调用处用到了该函数的返回值，就将其改为调用新建的查询函数，并在下面马上再调用一次原函数。
4. 从原函数中去掉返回值。

## 函数参数化（Parameterize Function）

### 动机

两个函数逻辑非常相似，只有一些字面量值不同，可以将其合并成一个函数，以参数的形式传入不同的值，从而消除重复

### 做法

1. 从一组相似的函数中选择一个。
2. 运用改变函数声明，把需要作为参数传入的字面量添加到参数列表中。
3. 修改该函数所有的调用处，使其在调用时传入该字面量值。
4. 修改函数体，令其使用新传入的参数。每使用一个新参数都要测试。
5. 对于其他与之相似的函数，逐一将其调用处改为调用已经参数化的函数。

## 移除标记参数（Remove Flag Argument）

### 动机

传入参数值影响了函数内部的控制流，这才是标记参数

### 做法

1. 针对参数的每一种可能值，新建一个明确函数。
2. 对于“用字面量值作为参数”的函数调用者，将其改为调用新建的明确函数。

## 保持对象完整（Preserve Whole Object）

### 动机

### 做法

## 以查询取代参数（Replace Parameter with Query）

### 动机

如果调用函数时传入了一个值，而这个值由函数自己来获得也是同样容易。

### 做法

1. 如果有必要，使用提炼函数将参数的计算过程提炼到一个独立的函数中。
2. 将函数体内引用该参数的地方改为调用新建的函数。每次修改后执行测试。
3. 全部替换完成后，使用改变函数声明将该参数去掉。

## 以查询取代参数（Replace Parameter with Query）

如果调用函数时传入了一个值，而这个值由函数自己来获得也是同样容易。

### 动机

### 做法

## 以参数取代查询（Replace Query with Parameter）

如果调用函数时传入了一个值，而这个值由函数自己来获得也是同样容易。

### 动机

### 做法

## 以查询取代参数（Replace Parameter with Query）

如果调用函数时传入了一个值，而这个值由函数自己来获得也是同样容易。

### 动机

### 做法

## 以查询取代参数（Replace Parameter with Query）

如果调用函数时传入了一个值，而这个值由函数自己来获得也是同样容易。

### 动机

### 做法

## 以查询取代参数（Replace Parameter with Query）

如果调用函数时传入了一个值，而这个值由函数自己来获得也是同样容易。

### 动机

### 做法