# Refactoring \_学习笔记

## 第二章：重构原则

### 什么是重构？

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

重构的要点是：

重构之后的软件功能一如既往。

### 为什么要重构？？？

#### 消除重复代码

消除重复代码

“

完成同样一件事，设计不良的程序往往需要更多的代码，这常常是因为代码在不同的地方使用相同的语句做同样的事。

因此改进设计的一个重要方向就是消除重复代码。这个动作的重要性在于方便未来的修改。

然而代码量减少将使未来可能的程序修改动作容易的多。代码愈多，正确的修改就愈困难，因为有更多代码需要理解。

你在这儿做了点修改，系统却不如预期那样工作，是因为有更多代码需要理解，你在这儿做了点修改，系统却不能如预期那样工作，是因为

你没有修改另一处-----那儿的代码做着几乎完全一样的事情，只是所处环境略有不同。如果消除重复代码，你就可以确定所有事物和行为在代码中只表述一次，这正是优秀设计的根本。

”

#### 重构让代码更易读

“准确地说出你要做什么”

#### 良好的设计

“拥有良好的设计才能做到快速开发。良好设计师维持软件开发速度的根本”

“事不过三，三则重构”

#### “重构让添加新功能更简单”

“

重构的另一个原动力是：代码的设计无法帮助我轻松添加我所需要的特性。然后我对自己说：“如果用某种方式来设计，添加特性会简单得多。”

这种情况下我不会因为自己过去的错误而懊恼----我用重构来弥补它。

之所以这么做，部分原因是为了让未来增加新特性时能够更轻松一些，但最主要的原因还是：

我发现这是最快捷的途径。重构是一个快速流畅的过程，一旦完成重构，新特性的添加就会更迅速、更流畅。

”

#### 程序的两面性价值

程序的两面性价值： “今天可以为你做什么”和“明天可以为你做什么”。

#### 增加间接层

#### 重构接口

该如何面对那些必须修改“已发布接口”的重构手法？？？

：

“

简言之，如果重构手法改变了已发布接口，你必须同时维护新旧两个接口，

知道所有用户都有时间对这个变化做出反应。幸运的是，这不太困难，你通常都有办法把事情组织好，让旧接口继续工作。

请尽量这么做：让旧接口调用新接口。当你要修改某个函数名称时，请留下旧函数，让它调用新函数。千万不要复制函数实现。

那会让你陷入重复代码的泥潭中难以自拔。你还应该使用Java提供的deprecation(不建议使用)设施，将旧接口标记为deprecated.

这样一来，你的调用者就会注意到它了。

”

#### 把“大块头软件”重构为封装良好的小型组件

“

将“大块头软件”重构为封装良好的小型组件，然后你就可以逐一对组件做出“重构或重建”的决定。

”

#### 重构与性能

如果你想优化性能，那么你首先需要准确找出性能热点。

“

关于性能，一件很有趣的事情是：如果你对大多数程序进行分析，就会发现它把大半时间都耗费在一小段代码身上。

如果你一视同仁地优化所有代码，90%的优化工作都是白费劲的，因为被你优化的代码大多很少被执行。

你花时间做优化是为了让程序运行的更快，但如果因为缺乏对程序的清楚认识而花费时间，那些时间就都是被浪费掉了。

”

“

在性能优化阶段，你首先应该用一个度量工具来监控程序的运行，让它告诉你程序中哪些地方大量消耗时间和空间。

这样你就可以找出性能热点所在的一小段代码。然后你应该集中关注这些性能热点，并使用持续关注法中的优化手段来、

优化它们。由于你把注意力都集中在热点上，较少的工作量便可显现较好的成果。即便如此你还是必须保持谨慎。

和重构一样，你应该小幅度进行修改。每走一步都需要编译、测试、再次度量。如果没能提高性能，就应该撤销此次修改。

你应该继续这个“发现热点、去除热点”的过程，直到获得客户满意的性能为止。

”