《代码整洁之道》

【美】Robert C. Martin著

韩磊 译

人民邮电出版社

2017-07-22：开始~Page11

2017-07-26：Page12~Page28

2017-07-27：Page29~Page35

2017-07-29：Page36~Page

影响代码质量的因素：架构、项目管理、代码质量等

大约在1951年，一种名为“全员生成维护”（Total Productive Maintenance， TPM）的质量保证手段在日本出现。TPM的主要支柱之一是所谓的5S原则体系。

5S哲学包括以下概念：

* 整理（Seiri），或谓组织：搞清楚事物之所在——通过切当地命名之类的手段——至关重要
* 整顿（Seiton），或谓整齐：物皆有其位，而后物尽归其位
* 清楚（Seiso），或谓清洁：对于那种四处遗弃的带注释的代码以及反映过往或期望的无注释代码，应除之而后快。
* 清洁（Seiketsu），或谓标准化：在开发组内使用一贯的代码风格和实践手段。
* 身美（Shitsuke），或谓纪律（自律）：在实践中贯彻规程，并时时体现于个人工作上，而且要乐于改进。

# 序

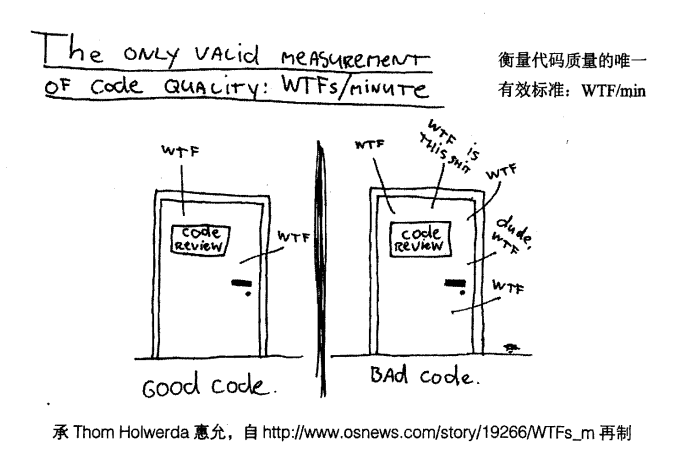
我们过早地放弃在代码上的工作，并不是因为它业已完成，而是因为我们的价值体系关注外在表现甚于关注要交付之物的本质。

# 代码猴子与童子军军规

代码猴子：低水平编码者

童子军军规：离开时要比发现时更整洁

# 前言



批注：好搞笑啊，哈哈哈哈~

如何保证我们在那道正确的门后做补救工作？答案是：技艺（craftsmanship）。

习艺之要有二：知和行。你应当习得有关原则、模式和实践的知识，穷尽应知之事，并且要对其了如指掌，通过刻苦实践掌握它。

学写整洁代码很难。它可不止于要求你掌握原则和模式。你得在这上面花功夫。你须自行实践，且体验自己的失败。你须观察他人的实践与失败。你须看看别人是怎样蹒跚学步，再转头研究他们的路数。你须看看别人是如何绞尽脑汁做出决策，又是如何为错误决策付出代价。

阅读本书要多用心思，要非常用功。如何用功？阅读代码——大量代码。而且你要去琢磨某段代码好在什么地方、坏在什么地方。

# 整洁代码

## 1.3 混乱的代价

### 1.3.5 什么是整洁代码

我喜欢优雅和高效的代码。代码逻辑应当直截了当，叫缺陷难以隐藏；尽量减少依赖关系，使之便于维护；依赖某种分层战略完善错误处理代码；性能调至最优，省得引诱别人做没规矩的优化，搞出一堆混乱来。整洁的代码只做好一件事。

**Bjarne Stroustrup，C++语言发明者，C++ Programming Language（中译版《C++程序设计语言》）一书作者**

破窗理论：窗户破损了的建筑让人觉得似乎无人照管。于是别人也再不关心。他们放任窗户继续破损。最终自己也参与破坏活动，在外墙上涂鸦，任垃圾堆积。一扇破损的窗户开辟了大厦走向倾颓的道路。

整洁的代码简单直接。整洁的代码如同优美的散文。整洁的代码从不隐藏设计者的意图，充满了干净利落的抽象和直截了当的控制语句。

**Grady Booch，Object Oriented Analysis and Design with Applications（中译版《面向对象分析与设计》）一书作者**

整洁的代码应可由作者之外的开发者阅读和增补。它应该有单元测试和验收测试。它使用有意义的命名。它只提供一种而非多种做一件事的途径。它只有尽量少的依赖关系，而且要明确地定义和提供清晰、尽量少的API。代码应通过其字面表达含义，因为不同的语言导致并非所有必需信息均可通过代码自身清晰表达。

**“老大” Dave Thomas，OTI公司创始人，Eclipse战略**

代码应在字面上表达其含义。这一观点源自Knuth的“字面编程”（literate programming）。结论就是应当用人类可读的方式来写代码。

……

## 1.4 思想流派

要通过多元思维来思考问题，取各家之长。

## 1.5 我们是作者

读与写花费时间的比例超过10:1。写新代码时，我们一直在读旧代码。不读周边代码的话就没法写代码。编写代码的难度，取决于读周边代码的难度。要想干得快，要想早点做完，要想轻松写代码，先让代码易读吧。

## 1.6 童子军军规

让营地比你来时更干净。

## 1.7 前传与原则

本书可认为是Agile Software Development: Principles, Patterns, and Practices（中译版《敏捷软件开发：原则、模式与实践》，简称PPP）的“前传”。

本书引用的设计原则有：单一权责原则（Single Responsibility Principle, SRP）、开放闭合原则（Open Closed Principle, OCP）和依赖倒置原则（Dependency Inversion Principle, DIP）等。

## 1.8 小结

本书展示了号程序员的思维过程，还有他们使用的技巧、技术和工具。

# 第2章 有意义的命名

## 2.2 名副其实

变量、函数或类的名称应该已经答复了所有的大问题。它该告诉你，它为什么会存在，它做什么事，应该怎么用。如果名称需要注释来补充，那就不算是名副其实。

反例：

int d; //消逝的时间，以日计

正例：

int elapsedTimeInDays;

int daysSinceCreation;

int daysSinceModificaiton;

int fileAgeInDays;

反例：

public List<int[]> getThem(){

List<int[]> list1 = new ArrayList<int[]>();

for (int[] x : theList)

if(x[0] == 4)

list1.add(x);

return list1;

}

正例：

public List<int[]> getFlaggedCells(){

List<int[]> flaggedCells = new ArrayList<int[]>();

for (int[] cell : gameBoard)

if (cell[STATUS\_VALUE] == FLAGGED)

flaggedCells.add(cell);

return flaggedCells;

}

【批注】反例中并不清楚代码具体在做什么事情，而正例中讲的是在开发一种扫雷游戏。盘面名称由theList改为gameBoard，0下标改为STATUS\_VALUE，4改为FLAGGED表示“已标记”。

更进一步：

public List<Cell> getFlaggedCells(){

List<Cell> flaggedCells = new ArrayList<Cell>();

for (Cell cell : gameBoard)

if (cell.isFlagged())

flaggedCells.add(cell);

return flaggedCells;

}

【批注】int数组表示单元格，改为用一个类来表示。

## 2.3 避免误导