1.可读性的重要性

编程有很大一部分时间是在阅读代码，不仅要阅读自己的代码，而且要阅读别人的代码。因此，可读性良好的代码能够大大提高编程效率。

可读性良好的代码往往会让代码架构更好，因为程序员更愿意去修改这部分代码，而且也更容易修改。

只有在核心领域为了效率才可以放弃可读性，否则可读性是第一位。

2.用名字表达代码含义

一些比较有表达力的单词：

单词 可替代单词

send deliver、dispatch、announce、distribute、route

find search、extract、locate、recover

start launch、create、begin、open

make create、set up、build、generate、compose、add、new

使用 i、j、k 作为循环迭代器的名字过于简单，user\_i、member\_i 这种名字会更有表达力。因为循环层次越多，代码越难理解，有表达力的迭代器名字可读性会更高。

为名字添加形容词等信息能让名字更具有表达力，但是名字也会变长。名字长短的准则是：作用域越大，名字越长。因此只有在短作用域才能使用一些简单名字。

3.名字不能带来歧义

起完名字要思考一下别人会对这个名字有何解读，会不会误解了原本想表达的含义。

布尔相关的命名加上 is、can、should、has 等前缀。

用 min、max 表示数量范围；

用 first、last 表示访问空间的包含范围；

begin、end 表示访问空间的排除范围，即 end 不包含尾部。

四、良好的代码风格

适当的空行和缩进。

排列整齐的注释：

int a = 1; // 注释

int b = 11; // 注释

int c = 111; // 注释

语句顺序不能随意，比如与 html 表单相关联的变量的赋值应该和表单在 html 中的顺序一致。

五、为何编写注释

阅读代码首先会注意到注释，如果注释没太大作用，那么就会浪费代码阅读的时间。那些能直接看出含义的代码不需要写注释，特别是并不需要为每个方法都加上注释，比如那些简单的 getter 和 setter 方法，为这些方法写注释反而让代码可读性更差。

不能因为有注释就随便起个名字，而是争取起个好名字而不写注释。

可以用注释来记录采用当前解决办法的思考过程，从而让读者更容易理解代码。

注释用来提醒一些特殊情况。

用 TODO 等做标记：

标记 用法

TODO 待做

FIXME 待修复

HACK 粗糙的解决方案

XXX 危险！这里有重要的问题

六、如何编写注释

尽量简洁明了：

// The first String is student's name

// The Second Integer is student's score

Map<String, Integer> scoreMap = new HashMap<>();

// Student's name -> Student's score

Map<String, Integer> scoreMap = new HashMap<>();

添加测试用例来说明：

// ...

// Example: add(1, 2), return 3

int add(int x, int y) {

return x + y;

}

使用专业名词来缩短概念上的解释，比如用设计模式名来说明代码。

七、提高控制流的可读性

条件表达式中，左侧是变量，右侧是常数。比如下面第一个语句正确：

if (len < 10)

if (10 > len)

只有在逻辑简单的情况下使用 ? : 三目运算符来使代码更紧凑，否则应该拆分成 if / else；

do / while 的条件放在后面，不够简单明了，并且会有一些迷惑的地方，最好使用 while 来代替。

如果只有一个 goto 目标，那么 goto 尚且还能接受，但是过于复杂的 goto 会让代码可读性特别差，应该避免使用 goto。

在嵌套的循环中，用一些 return 语句往往能减少嵌套的层数。

八、拆分长表达式

长表达式的可读性很差，可以引入一些解释变量从而拆分表达式：

if line.split(':')[0].strip() == "root":

...

username = line.split(':')[0].strip()

if username == "root":

...

使用摩根定理简化一些逻辑表达式：

if (!a && !b) {

...

}

if (!(a || b)) {

...

}