【软件构造】第七章第三节 断言和防御性编程

第七章第三节 断言和防御性编程

本节：第2种技术——断言、防御式编程

Outline

- 断言
  - 什么是断言
  - 断言的应用场景
- 防御式编程（不是考点，不加叙述）

Notes:

断言

【什么是断言】

- 作用：允许程序在运行时检查自己，测试有关程序逻辑的假设，如前置条件、后置条件、内部不变量、表示不变量、控制流不变量等
- 目的： 为了在开发阶段调试程序、尽快避免错误
- 使用阶段：
  - 断言主要用于开发阶段，避免引入和帮助发现bug
  - 实际运行阶段， 不再使用断言
  - 软件发布阶段，禁用断言避免影响性能。

【应用场景】

- 输入参数或输出参数的取值处于预期范围
- 子程序开始执行（结束）时，文件或流处于打开（关闭）状态
- 子程序开始执行（结束）时，文件或流的读写位置处于开头（结尾）
- 文件或流已打开
- 输入变量的值没有被子程序修改
- 指针非空
- 传入子程序的数组至少能容纳X个元素
- 表已初始化，存储着真实的数据
- 子程序开始（结束）时，容器空（满）
- 一个高度优化过的子程序与一个缓慢的子程序，结果一致
- 断言只在开发阶段被编译到目标代码中，而在生成代码时不编译进去。使用断言的指导建议：
- 用错误处理代码来处理预期会发生的状况，断言不行！
- 避免把需要执行的代码放入断言中（如果未编译断言呢？）
- 用断言来注解并验证前条件和后条件

- 对于高健壮性的代码，应该先用断言，再处理错误

【注意】

- 编译时加入-ea(enable assertion)选项运行断言，-da(disable assertion)关闭断言
- 条件语句或开关没有涵盖所有可能的情况，最好使用断言来阻止非法事件
- 可以在预计正常情况下程序不会到达的地方放置断言：assert false
- 断言有代价，需慎用，一般用于验证正确性，处理绝不应该发生的情况
- 不能作为公共方法的检查，也不能有边界效应

【断言和异常的对比】

- 用异常处理技术来处理你“希望发生”的不正常情况
- 用断言来处理“不希望发生”的情况；断言的方式处理一定是发生了错误
- 不要把业务逻辑（执行代码）放到断言里面去处理
- 参数检查通常是方法发布的规范（或契约）的一部分，无论断言是启用还是禁用，都必须遵守这些规范。
  - 如果参数来自于外部（不受自己控制），使用异常处理
  - 如果来自于自己所写的其他代码，可以使用断言来帮助发现错误（例如postcondition就需要）

防御性编程（不是考点，不加叙述）

- 防御式编程的主要思想就是，子程序不应该因为传入错误数据而被破坏。其核心想法就是要承认程序都会有问题，都需要被修改。
- 处理外来垃圾的方法：检查所有来自外部的数据值，检查子程序的输入参数值，决定如何处理数据。
- 防御式编码的最佳方式就是一开始代码中引入错误，使用迭代式设计、编码前先写伪代码、写代码前先写测试用例、底层设计检查等活动都可以防止。