无论使用何种编程语言，错误处理的一个基本方面是区分编程错误（也称为 bug）和运行时错误。运行时错误可以进一步分为可恢复的运行时错误和不可恢复的运行时错误。不可恢复的运行时错误的一个例子是栈溢出（见第 7 章，内存管理）。当不可恢复的错误发生时，程序通常会立即终止，因此对这类错误发出信号并没有意义。然而，某些错误在一种类型的应用程序中可能被认为是可恢复的，但在其他应用程序中则可能是不可恢复的。

契约式设计指的是调用者和被调用者之间存在的、难以用类型系统表达的规则。

**前置条件**：指定了函数调用者的责任。传递给函数的参数可能存在约束条件。

**后置条件**：指定了函数返回时的责任。

**不变式**：是一个应该始终为真的条件。不变式可以在许多上下文中使用。循环不变式是在每次循环迭代开始时必须为真的条件。

**类不变式**：定义了类对象的有效状态。对于std::vector，size() <= capacity()始终为真，这就是它的一个类不变式。

文本, 信件

AI 生成的内容可能不正确。

因此，定义类不变式通常意味着最终会得到设计良好的类。

文本, 信件

AI 生成的内容可能不正确。

本书选择使用断言（assert()）来检查契约违反情况。虽然断言是一种比较原始的方法，但它简单易用，能够在开发和调试阶段快速发现契约违反的问题，对提高代码质量有很大的帮助。

Static assert 编译时断言**assert()**：这是一个运行时断言

文本, 信件

AI 生成的内容可能不正确。

文本, 信件

AI 生成的内容可能不正确。

bug是编程时错误，是契约前置条件不满足产生的，通常是代码逻辑有问题，可以通过断言来检查

**可恢复的运行时错误**

如果一个函数无法履行其契约中的部分内容（即后置条件），那么就发生了运行时错误，需要将该错误通知到代码中能够处理它并恢复到有效状态的位置。处理可恢复错误的目的是将错误从发生的位置传递到可以恢复有效状态的位置。

强异常安全性可以看作是一种事务。一个函数要么提交所有的状态更改，要么在发生异常的情况下进行完全回滚。

文本, 信件

AI 生成的内容可能不正确。