# 第11章 **错误处理（黑体 ，三号）**

本章要点：理解rust错误处理概念与机制，通过例子学习如何运用错误处理应对rust实践中的挑战。（黑体，小五，300字以内）

本章导图：

前面几章，我们学习了 Rust 语言的一些基本概念及使用方法，运用这些知识可以构建软件。但需要指出的是错误是软件中不可避免的情况，所以 Rust 有一些处理错误的特性。本章我们将理解 Rust 错误处理概念，学习如何运用错误处理机制来使你的程序更加健壮。

## 11.1 Rust错误处理概述（节标题宋体，四号）

**1. Rust错误处理的概念**

不同的语言对于错误(error)，异常(exception)有不同的定义。在Rust 语言中，程序遇到的非正常情况统称为错误 (error)，Rust中不使用exception这个概念。在Rust 官方文档中，将错误分为可恢复错误和不可恢复错误两大类。在有些资料中（例如 mastering rust…）,又增加了fault error这一类错误。该3种错误如下：

**可恢复错误(Recoverable errors)**：指的是用户、环境和程序交互时可预期的异常情况。例如文件未找到，网络通信故障等。对于这一类错误，我们往往希望向用户报告这个问题并尝试修复这个异常情况。

Rust 使用基于类型的错误处理方式来处理这类错误。

**不可恢复错误(unrecoverable errors)**: 指的是破坏程序contracts 或者 invariants的异常行为，例如下标越界(index out of bounds)或是除以0的操作(divide by zero)。对于这一类错误，我们往往想立刻停止程序。

Rust 使用被称为panic的机制来处理这类错误。

**致命性错误(fault errors)**: 指的是立即让程序终止运行的异常情况。这种情况包括内存不足和堆栈溢出等。

在Java中，这一类错误用Error类表示，程序不应该处理这类错误。同样，Rust不会处理这一类错误。

**2.多种语言的错误处理机制的对比**

总的来说，错误处理机制的两种主流范式是返回代码（return codes）和异常（exceptions）。

C语言采用返回代码范式，通过函数的返回值来表示是否出现错误。例如C标准库的fopen函数使用返回值0/-1来表示文件成功打开与否。这种错误处理方式简单，灵活，没有额外的开销，但是会造成一些问题。其一是返回值的错误检查不是强制性的，有可能被开发者疏忽。其二是错误处理代码与功能代码交织在一起，容易造成逻辑混乱。

相比之下，C++，Java，Python等高级语言引入了引入了异常处理机制(excpetions)。异常处理机制利用栈回退(Stack Unwind) 或栈回溯(Stack Backtrack)机制自动处理异常。同时在语言层面引入了专门的语法例如try-catch来实现异常处理逻辑，实现了处理错误逻辑和功能逻辑的分离。但它仍然有不足，比较突出的是这种处理机制开销较大。

Rust对于可恢复的错误采用了C语言的返回代码机制，而没有选择异常处理机制，这是由于后者所带来的较大开销与Rust 零运行时成本(zero runtime costs)哲学相违背。另一方面，Rust改进了C 语言的返回代码机制，使用了基于类型的错误处理机制。在Rust中，函数的返回值采用适当的错误类型，例如Option，Result，并支持用户自定义错误类型。这和Haskell中的Maybe，Either类似。这种设计的好处是可以在编译阶段通过类型检查确保没有错误被忽略处理。

同时，对于不可恢复的错误，Rust提供了被称为panic的机制，这是一种“fail fast”机制，用作最后的处理措施。