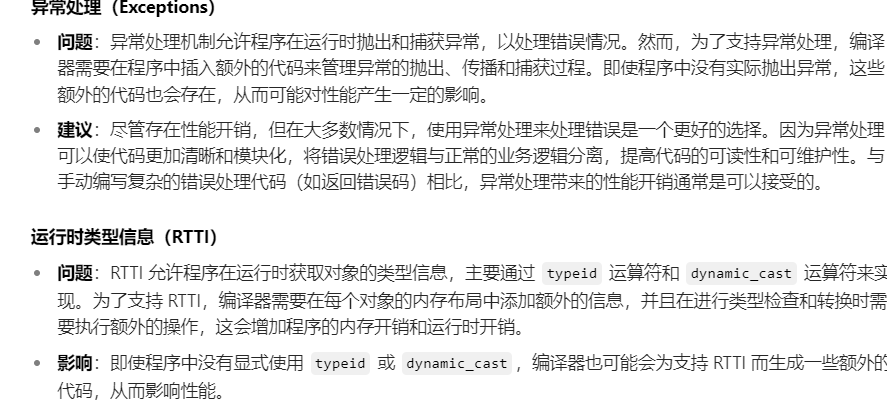
许多现代 C++ 语言特性可以看作是在基本 C 功能之上的抽象。



值语义

要使用引用语义，我们需要显式地使用引用或指针。

常量正确性

对象所有权

除非在非常罕见的情况下，C++程序员应该将内存处理留给容器和智能指针，永远不必依赖手动内存处理。

如果程序员在C++中最小化共享所有权，那么产生的代码更容易使用，也更难滥用，因为它可以强制类的用户按照预期使用它。

C++中的确定性销毁

使用C++引用避免空对象

C++的缺点

2 C++基本技巧

自动类型推导以及在声明函数和变量时如何使用auto关键字。

移动语义和五的规则和零的规则。

错误处理和合同。虽然这些主题并没有呈现任何可以被认为是现代C++的东西，但异常和契约都是当今C++中备受争议的领域。

使用lambda表达式创建函数对象，这是C++11最重要的特性之一。

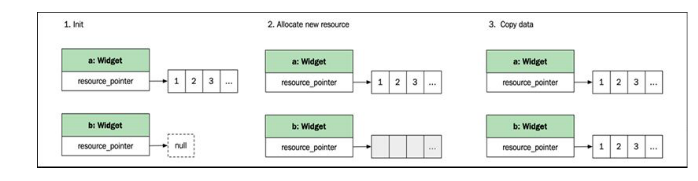
应用程序

AI 生成的内容可能不正确。

虽然使用 auto& 可以推导出常量引用，但为了代码的清晰性和可读性，建议始终使用 const auto& 来明确表示常量引用，而只使用 auto& 来表示可变引用。这样可以让代码的意图更加明确，减少误解。

只有在函数模板中，当 T 是该函数模板的模板参数时，auto&& 和 T&& 才是转发引用。

指针的常量传播



图示

AI 生成的内容可能不正确。

**源对象的状态**

移动操作后，源对象被重置，但仍处于有效状态。这意味着我们可以安全地销毁源对象或对其进行赋值操作。然而，编译器不会自动重置源对象，我们需要在移动构造函数中手动实现这个重置逻辑。

移动操作主要适用于对象拥有动态分配的资源（如堆内存）的情况。如果对象的所有数据都存储在栈上，那么复制操作通常是更简单和高效的选择，因为栈上的数据复制开销很小。

C++中的一个基本概念是类应该完全处理它的资源。这意味着当类被复制、移动、复制赋值、移动赋值或析构时，该类应确保其资源得到相应的处理。实现这五个功能的必要性通常被称为五个规则。

从函数中出来，或者有std：：move（）。

一般来说，具名变量是左值，会进行复制操作；临时对象是右值，会进行移动操作。但使用 std::move 可以将左值转换为右值，前提是该左值不是 const 的。

只要声明了这五个特殊成员函数（复制构造函数、复制赋值运算符、移动构造函数、移动赋值运算符、析构函数）中的任意一个，编译器就不会自动生成其他几个了。

图形用户界面, 应用程序

AI 生成的内容可能不正确。

图形用户界面, 文本, 应用程序, 电子邮件

AI 生成的内容可能不正确。

**零规则与代码库划分**

如果一个类需要显式编写特殊成员函数，说明这个类可能有一些复杂的资源管理逻辑。将这样的类放在代码库的库部分是一个好的实践，因为库代码通常更适合处理复杂的资源管理和底层细节。应用代码可以依赖这些库类，而无需关心特殊成员函数的实现。

为了让编译器能够进行更多的优化，提高程序的性能，在不需要自定义析构函数逻辑时，应该优先使用默认析构函数（= default）或者不声明析构函数。避免使用空析构函数，因为它会阻止编译器应用一些潜在的优化。

文本, 信件

AI 生成的内容可能不正确。

图形用户界面, 文本, 应用程序, 电子邮件

AI 生成的内容可能不正确。

return std::move(x); // 不要这样做，这会阻止返回值优化（RVO）

图形用户界面, 文本, 应用程序, 电子邮件

AI 生成的内容可能不正确。

