本章涵盖

ϒ 启动线程，以及指定在新线程上运行的代码的各种方法

ϒ 等待线程完成与让它继续运行

ϒ 唯一标识线程

在最简单的情况下，该任务是一个普通的、普通的 void 返回函数，不带任何参数。 该函数在其自己的线程上运行，直到返回，然后线程停止。 在另一个极端，任务可能是一个函数对象，它接受附加参数并执行一系列独立操作，这些操作在运行时通过某种消息传递系统指定，并且线程仅在收到信号时才停止，同样通过 某种消息系统。

在这种情况下，提供的函数对象被复制到属于新创建的执行线程的存储中，并从那里调用。 因此，副本的行为必须与原始版本相同，否则结果可能不是预期的。

**函数对象（Function Object）** 是C++中的一个重要概念，它是一个可以像函数一样被调用的对象。函数对象通常是通过重载 operator() 运算符来实现的，因此也被称为 **仿函数（Functor）**。

图形用户界面, 文本, 应用程序, 电子邮件

描述已自动生成

图形用户界面

低可信度描述已自动生成

这段话强调了在使用 **std::thread** 时，必须在 std::thread 对象销毁之前明确决定是 **等待线程完成（join）** 还是 **让线程独立运行（detach）**。如果不做决定，程序会在 std::thread 对象销毁时调用 std::terminate() 终止

join() 是一种简单且强力的技术——要么等待线程完成，要么不等待。 如果您需要对等待线程进行更细粒度的控制，例如检查线程是否完成，或者仅等待一段时间，那么您必须使用替代机制，例如条件变量和 future，我们使用这些机制 将在第 4 章中介绍。

用 join() 的行为还会清除与线程关联的任何存储，因此 std::thread 对象不再与现已完成的线程关联； 它不与任何线程关联。

如果不做决定，程序会在 std::thread 对象销毁时调用 std::terminate() 终止。特别是在异常情况下，必须确保线程被正确管理，以避免资源泄漏或未定义行为。.

**1. std::thread 对象的生命周期**

* **std::thread 对象** 是C++中用于管理线程的类对象。
* 当 std::thread 对象被销毁时（例如，超出作用域或被显式销毁），如果线程既没有 join() 也没有 detach()，程序会调用 std::terminate() 终止。

**RAII 机制**：

* thread\_guard 的构造函数接受一个 std::thread 对象的引用，并将其保存。
* 在析构函数中，检查线程是否可 join（即线程是否仍在运行），如果是，则调用 join() 等待线程结束。
* 这样可以确保线程不会因为异常或提前退出而被遗忘
* **分离线程的特点**：
  + **独立运行**：分离线程与主线程无关，主线程不需要等待它结束。
  + **无法 join**：一旦线程被分离，就不能再调用 join() 等待它结束。
  + **资源自动回收**：当分离线程执行完毕后，其资源会自动释放。

1. **std::thread 构造函数**：
   * std::thread 的构造函数接受一个可调用对象（如函数、函数对象或 lambda 表达式）以及该可调用对象的参数。
   * 语法为：

cpp

复制

std::thread t(Callable, arg1, arg2, ...);

其中：

* + - Callable 是可调用对象。
    - arg1, arg2, ... 是传递给 Callable 的参数。

1. **std::ref 的使用场景**：
   * 当需要将对象的引用传递给函数或线程时，可以使用 std::ref。
   * 例如，std::thread 的构造函数默认会复制参数，但如果希望传递引用，就需要使用 std::ref。

**std::ref 不会延长临时对象的生命周期**：

* 如果 data 是一个临时对象（例如 widget\_data()），使用 std::ref(data) 会导致未定义行为，因为临时对象会在表达式结束后被销毁，而引用包装器仍然持有它的引用。
* **std::thread 的构造函数**：
  + std::thread 的构造函数可以接受成员函数指针和对象指针，并在线程中调用该成员函数。
  + 语法：

cpp

复制

std::thread t(&Class::MemberFunction, &object, arg1, arg2, ...);

其中：

* + - &Class::MemberFunction 是成员函数指针。
    - &object 是对象指针。
    - arg1, arg2, ... 是传递给成员函数的参数。

**std::thread 的所有权语义**

1. **独占资源**：
   * 每个 std::thread 对象管理一个线程的执行资源。
   * 与 std::unique\_ptr 类似，std::thread 独占其所管理的线程资源。
2. **不可复制**：
   * std::thread 不能被复制，因为复制会导致多个 std::thread 对象管理同一个线程资源，这会导致未定义行为。
   * 如果尝试复制 std::thread，会导致编译错误。
3. **可移动**：
   * std::thread 支持移动语义，可以通过移动构造函数或移动赋值运算符将线程资源的所有权从一个 std::thread 对象转移到另一个。
   * 移动后，源 std::thread 对象会变为无效状态（即不再关联任何线程）。
4. **线程资源的释放**：
   * 如果 std::thread 对象在赋值或移动赋值时仍然管理着一个线程（即 joinable() 返回 true），则会调用 std::terminate()，导致程序终止。
   * 因此，在赋值或移动赋值之前，必须确保 std::thread 对象不再管理任何线程（即调用 join() 或 detach()）。
5. **scoped\_thread 的构造函数**：
   * 接受一个 std::thread 对象，并通过 std::move 接管其所有权。
   * 如果传入的 std::thread 对象不可 join（即 !t.joinable()），则抛出 std::logic\_error 异常。
6. **scoped\_thread 的析构函数**：
   * 在析构函数中调用 join()，确保线程在 scoped\_thread 对象销毁时完成。
   * 这种方式避免了线程资源泄漏。
7. **禁止拷贝**：
   * 删除拷贝构造函数和拷贝赋值运算符，确保 scoped\_thread 对象不能被复制。
   * 这是为了避免多个 scoped\_thread 对象管理同一个线程资源。
8. **什么是范围 for 循环？**
   * 范围 for 循环（Range-based for loop）是 C++11 引入的一种简化遍历容器（如 std::vector、std::list 等）的语法。
   * 它的语法如下：

cpp

复制

for (auto& element : container) {

// 使用 element

}

* + 1. container 是一个容器（如 std::vector）。
    2. element 是容器中的每个元素的引用。

**关键点**

1. **std::thread::hardware\_concurrency()**：
   * 返回硬件支持的并发线程数（如 CPU 核心数）。
   * 用于确定合理的线程数量，避免创建过多线程导致性能下降。
2. **任务分配**：
   * 将任务均匀分配给多个线程，每个线程处理一部分数据。
   * 主线程也参与计算，避免浪费资源。
3. **结果汇总**：
   * 每个线程计算部分和，最后将所有部分和累加，得到最终结果。
4. **异常处理**：
   * 代码假设不会抛出异常，但在实际应用中需要考虑异常处理（如线程创建失败）。

**std::thread::id 的作用**

1. **线程标识符**：
   * std::thread::id 是用于标识线程的唯一标识符。
   * 每个 std::thread 对象都有一个关联的 std::thread::id，用于标识其管理的线程。
2. **获取线程标识符**：
   * 通过 std::thread 对象的 get\_id() 成员函数获取：

cpp

复制

std::thread::id thread\_id = thread.get\_id();

* + 通过 std::this\_thread::get\_id() 获取当前线程的标识符：

cpp

复制

std::thread::id current\_thread\_id = std::this\_thread::get\_id();

**std::thread::id 的特性**

1. **可复制和可比较**：
   * std::thread::id 对象可以自由复制和比较。
   * 如果两个 std::thread::id 对象相等，则表示它们代表同一个线程，或者都表示“没有线程”。
   * 如果两个 std::thread::id 对象不相等，则表示它们代表不同的线程，或者一个代表线程而另一个表示“没有线程”。
2. **比较操作**：
   * std::thread::id 支持完整的比较运算符（==, !=, <, <=, >, >=）。
   * 这些比较运算符为所有不同的 std::thread::id 值提供了全序关系，因此可以用于排序或作为关联容器的键。
3. **哈希支持**：
   * std::thread::id 支持哈希操作，因此可以用作无序关联容器（如 std::unordered\_map）的键。

我介绍了使用 C++ 标准库进行线程管理的基础知识：启动线程、等待它们完成，以及不等待它们完成，因为您希望它们在后台运行。 我们还了解了如何在启动线程时将参数传递到线程函数中，如何将管理线程的职责从代码的一部分转移到另一部分，以及如何使用线程组来划分工作。 最后，我们讨论了识别线程，以便将数据或行为与不方便通过其他方式关联的特定线程相关联。 尽管您可以使用每个操作单独数据的完全独立的线程执行大量操作，但有时需要在线程运行时在线程之间共享数据。 第 3 章讨论了有关在线程之间直接共享数据的问题，第 4 章涵盖了有关有或没有共享数据的同步操作的更一般问题。