### **1. 多线程编程基础知识**

* ****线程****：线程是程序执行流的最小单元，一个进程可以包含多个线程，这些线程共享进程的资源。
* ****互斥锁****：当多个线程需要访问共享资源时，为了避免数据冲突，需要使用互斥锁来保证一次只有一个线程可以访问该资源。
* ****条件变量****：用于线程间的协调，一个线程可以等待某个条件成立（另一个线程改变某些状态），当条件满足时，等待的线程将被唤醒。

### **2. std::thread**

std::thread是C++11标准库中提供的类，用于表示一个线程对象。通过创建std::thread对象并传入一个可调用对象（如函数、lambda表达式或函数对象），可以启动一个新的线程。

#include <thread>

#include <iostream>

void threadFunction()

{ for (int i = 0; i < 10; ++i)

{ std::cout << "Thread function: " << i << std::endl; }

}

int main()

{ std::thread t(threadFunction); // 创建新线程并开始执行 t.join(); // 等待线程结束 return 0; }

### 3. std::mutex 和 std::lock\_guard

std::mutex是一个互斥对象，用于保护共享数据，防止多个线程同时访问。而std::lock\_guard是一个RAII风格的互斥锁，它在构造时锁定互斥锁，在析构时自动解锁互斥锁，可以简化互斥锁的使用。

#include <thread>

#include <mutex>

#include <iostream>

std::mutex mtx; // 全局互斥锁

int shared\_data = 0; // 共享数据

void increment()

{ std::lock\_guard<std::mutex> lock(mtx); // 在作用域内自动锁定和解锁互斥锁 ++shared\_data; // 对共享数据加1 }

int main()

{ std::thread t1(increment); // 启动两个线程同时访问共享数据

std::thread t2(increment); t1.join(); t2.join(); // 等待两个线程结束

std::cout << "Shared data: " << shared\_data << std::endl; // 输出共享数据值，应为2 return 0; }

### 4. std::condition\_variable

std::condition\_variable用于线程间的协调，允许一个或多个线程等待某个条件成立。当条件满足时，等待的线程将被唤醒。

### 5. 线程池

线程池是一种管理线程的技术，可以预先创建一组线程并保存在内存中，需要时从线程池中取出线程进行处理，处理完毕后将线程归还给线程池。这样可以避免频繁地创建和销毁线程，提高程序的性能。在C++中，可以使用第三方库来实现线程池的功能。例如，可以使用Boost.Thread库中的Boost.Thread.Pool来实现线程池。