1. std::thread:std::thread是C++标准库中用于创建新线程的类。可以将函数或可调用对象传递给std::thread来开始一个新的线程。

cpp`#include <thread>

void my\_function() {

// 线程执行的代码

}

int main() {

std::thread t(my\_function); // 创建一个新线程，执行my\_function

t.join(); // 等待线程结束

return 0;

}`

1. std::mutex:std::mutex是一个互斥量，用于保护共享数据，防止同时被多个线程访问。使用std::lock()和std::unlock()方法手动锁定和解锁互斥量可能会导致死锁，可以使用std::lock\_guard或std::unique\_lock。

cpp`#include <mutex>

std::mutex mtx; // 全局互斥量

void my\_function() {

mtx.lock(); // 锁定互斥量

// 访问共享数据

mtx.unlock(); // 解锁互斥量

}`

1. std::lock\_guard:std::lock\_guard是一个RAII包装器，用于自动管理互斥量的锁定和解锁。当std::lock\_guard对象被创建时，它会自动锁定互斥量，当对象被销毁时，它会自动解锁互斥量。

cpp`#include <mutex>

std::mutex mtx; // 全局互斥量

void my\_function() {

std::lock\_guard<std::mutex> lock(mtx); // 自动锁定互斥量

// 访问共享数据

// 当lock离开作用域时，它将被销毁，并自动解锁mtx

}`

1. std::condition\_variable:std::condition\_variable是一个条件变量，允许线程等待某个条件成立。它通常与std::mutex一起使用。可以使用std::condition\_variable::wait()方法让线程等待，该方法会阻塞当前线程，直到另一个线程调用相应的std::condition\_variable::notify\_one()或std::condition\_variable::notify\_all()方法。

cpp`#include <condition\_variable>

std::mutex mtx; // 全局互斥量

std::condition\_variable cv; // 全局条件变量

bool ready = false; // 全局布尔变量，表示条件是否成立

void my\_function() {

std::unique\_lock<std::mutex> lock(mtx); // 锁定互斥量

cv.wait(lock, []{ return ready; }); // 等待条件成立（即ready为true）

// 当条件成立时，cv.wait()返回，并自动解锁mtx

}`