多线程编程的基础知识

多线程指的是在同一个进程中执行多个线程。同一个进程内多个线程之间可以共享代码段、数据段、打开的文件等资源，但每个线程各自都有一套独立的寄存器和栈，这样可以确保线程的控制流是相对独立的。进程是一个在内存中运行的应用程序。每个进程都有自己独立的一块内存空间，一个进程可以有多个线程。线程是进程中的一个执行任务，负责当前进程中程序的执行。

std::thread是 C++ 标准库当中最基本的多线程实现方式。可以使用 thread 对象查看和管理应用程序中的执行线程。使用thread默认构造函数创建的对象不与任何执行线程相关联。使用thread可调用对象构造的对象将创建一个新的执行线程，并调用该中的可调用对象thread。Thread 对象可以移动，但不能复制。

例程：

#include <iostream>

#include <thread>

void show() {

std::cout << "thread example\n";

}

int main() {

std::thread t(show);

t.join();

return 0;

}

std:：mutex互斥量对作用域的手动上锁与解锁。std::lock\_guard是C++标准库提供的互斥锁RAII封装工具，用于实现互斥访问。lock\_guard 是不可移动的，即不能拷贝、赋值、移动，只能通过构造函数初始化和析构函数销毁。

例程：

#include <iostream>

#include <thread>

#include <mutex>

using namespace std;

int g\_num = 0;

mutex g\_mutex;

void thread1(){

g\_mutex.lock();

g\_num = 50;

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

cout << "thread1:" << g\_num << endl;

g\_mutex.unlock();

}

void thread2(){

lock\_guard<mutex>lg(g\_mutex);

g\_num = 100;

for (int j = 0; j < 10; ++j)

cout << "thread2:" << g\_num << endl;

}

int main(){

thread t1(thread1);

thread t2(thread2);

t1.join();

t2.join();

return 0;

}

std::condition\_variable条件变量是利用线程间共享的全局变量进行同步的一种机制，主要包括两个动作：一个线程等待条件变量的条件成立而挂起;另一个线程使条件成立（给出条件成立信号）。为了防止竞争，条件变量总是和一个互斥量结合在一起使用。

例程：  
#include <iostream>

#include <thread>

#include <mutex>

#include <condition\_variable>

std::mutex mtx;

std::condition\_variable cv;

int cargo = 0;

bool shipment\_available(){

return cargo != 0;

}

// 消费者线程.

void consume(int n){

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

std::unique\_lock <std::mutex> lck(mtx);

cv.wait(lck, shipment\_available);

std::cout << cargo << '\n';

cargo = 0;

}

}

int main()

{

std::thread consumer\_thread(consume, 10); // 消费者线程

// 主线程为生产者线程, 生产 10 个物品

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

while (shipment\_available())

std::this\_thread::yield();

std::unique\_lock <std::mutex> lck(mtx);

cargo = i + 1;

cv.notify\_one();

}

consumer\_thread.join();

return 0;

}

线程池是一种并发编程的机制，用于管理和复用线程。它包含一组预先创建的线程，这些线程在需要执行任务时被重复使用，而不是为每个任务都创建一个新的线程。线程池的主要目标是提高并发性能、降低线程创建和销毁的开销，以及有效管理系统资源。