多线程编程是指在一个程序中使用多个线程并行执行不同的任务，以提高程序的效率和响应性。多线程编程需要注意线程的创建、管理、同步和通信等问题，以避免数据竞争、死锁、内存泄漏等错误。

C++11标准库提供了一些类和函数来支持多线程编程，其中主要有以下几个：

std::thread：表示一个线程对象，可以通过构造函数传入一个可调用对象来启动一个线程，也可以通过join或detach方法来等待或分离一个线程。std::thread还提供了一些静态成员函数，如hardware\_concurrency来获取系统支持的并发线程数，以及this\_thread命名空间下的一些函数，如get\_id、yield、sleep\_for、sleep\_until等，用于操作当前线程。

·std::mutex：表示一个互斥量对象，用于保护共享数据不被多个线程同时访问。std::mutex提供了lock和unlock方法来加锁和解锁一个互斥量，以及try\_lock方法来尝试加锁一个互斥量。std::mutex还有一些派生类，如std::recursive\_mutex（允许同一线程多次加锁）、std::timed\_mutex（允许在指定时间内尝试加锁）、std::recursive\_timed\_mutex（允许同一线程多次在指定时间内尝试加锁）等，以及std::shared\_mutex（C++17）和std::shared\_timed\_mutex（C++14）（允许多个线程共享读锁，但只有一个线程独占写锁）。

·std::lock\_guard：表示一个锁管理器对象，用于实现RAII风格的互斥量管理。std::lock\_guard在构造函数中接收一个互斥量对象并自动加锁，在析构函数中自动解锁。std::lock\_guard可以保证在异常或提前返回的情况下，互斥量能够被正确地解锁。std::lock\_guard不支持移动语义，也不提供解锁操作，因此不能用于实现条件变量等需要手动解锁的场景。

·std::condition\_variable：表示一个条件变量对象，用于实现线程间的同步和通信。std::condition\_variable可以让一个或多个线程在满足某个条件之前阻塞等待，直到另一个线程通知它们继续执行。std::condition\_variable提供了wait、wait\_for、wait\_until等方法来让线程等待一个条件，以及notify\_one和notify\_all方法来通知一个或所有等待的线程。std::condition\_variable需要和std::unique\_lock配合使用，因为它需要在等待时解锁互斥量，而在被通知时重新加锁互斥量。std::condition\_variable\_any是一个通用的条件变量类，可以和任何满足Lockable要求的锁类型配合使用，但效率可能低于std::condition\_variable。

线程池：表示一个管理一组工作线程的资源池，用于执行异步的任务。线程池可以避免频繁地创建和销毁线程，提高线程的复用性和可控性。C++标准库没有提供线程池的实现，但提供了一些支持异步任务的类和函数，如std::future、std::promise、std::packaged\_task、std::async等，可以用于实现自定义的线程池。另外，也可以使用第三方的线程池库，如boost::threadpool[1](https://en.cppreference.com/w/cpp/thread/mutex" \t "https://copilot.microsoft.com/_blank)、ctpl[2](https://www.geeksforgeeks.org/std-mutex-in-cpp/" \t "https://copilot.microsoft.com/_blank)等。