1. 你好，C++的并发世界

\*并发：单个系统里同时执行多个独立地任务，而非顺序的进行一些活动。

\*分离关注点（SOC） Separation of concerns

+ 一是分层，二是面向接口编程

\*native\_handle()：直接操作底层实现

1. 线程管理

\*detach过的线程为 守护线程

\*发后即忘（fire and forget）的任务使用到守护进程方式（detach函数）。

\* void process\_big\_object(std::unique\_ptr<big\_object>);

std::unique\_ptr<big\_object> p(new big\_object);

p->prepare\_data(42);

std::thread t(process\_big\_object, std::move(p));

\*scoped\_thread 用法

1. class scoped\_thread
2. {
3. std::thread t;
4. public:
5. explicit scoped\_thread(std::thread t\_): // 1
6. t(std::move(t\_))
7. {
8. if(!t.joinable()) // 2
9. throw std::logic\_error(“No thread”);
10. }
11. ~scoped\_thread()
12. {
13. t.join(); // 3
14. }
15. scoped\_thread(scoped\_thread const&)=delete;
16. scoped\_thread& operator=(scoped\_thread const&)=delete;
17. };
18. struct func; // 定义在清单2.1中
19. void f()
20. {
21. int some\_local\_state;
22. scoped\_thread t(std::thread(func(some\_local\_state))); // 4
23. do\_something\_in\_current\_thread();
24. } // 5

\*std::thread::hardware\_concurrency() 返回 cpu核心数量

\*std::this\_thread::get\_id() ; t1.get\_id(); 返回线程id

1. 线程间共享数据