2.5 标识线程

线程标识类型是 `std::thread::id` ，可以通过两种方式进行检索。第一种，可以通过调用 `std::thread` 对象的成员函数 `get_id()` 来直接获取。如果 `std::thread` 对象没有与任何执行线程相关联， `get_id()` 将返回 `std::thread::type` 默认构造值，这个值表示“无线程”。第二种，当前线程中调用 `std::this_thread::get_id()` (这个函数定义在 `<thread>` 头文件中)也可以获得线程标识。

`std::thread::id` 对象可以自由的拷贝和对比,因为标识符就可以复用。如果两个对象的 `std::thread::id` 相等，那它们就是同一个线程，或者都“无线程”。如果不等，那么就代表了两个不同线程，或者一个有线程，另一没有线程。

线程库不会限制你去检查线程标识是否一样， `std::thread::id` 类型对象提供相当丰富的对比操作；比如，提供为不同的值进行排序。这意味着允许程序员将其当做为容器的键值，做排序，或做其他方式的比较。按默认顺序比较不同值的 `std::thread::id` ，所以这个行为可预见的：当 `a<b` ， `b<c` 时，得 `a<c` ，等等。标准库也提供 `std::hash<std::thread::id>` 容器，所以 `std::thread::id` 也可以作为无序容器的键值。

`std::thread::id` 实例常用作检测线程是否需要进行一些操作，比如：当用线程来分割一项工作(如清单2.8)，主线程可能要做一些与其他线程不同的工作。这种情况下，启动其他线程前，它可以将自己的线程ID通过 `std::this_thread::get_id()` 得到，并进行存储。就是算法核心部分(所有线程都一样的),每个线程都要检查一下，其拥有的线程ID是否与初始线程的ID相同。

```
std::thread::id master_thread;
void some_core_part_of_algorithm()
{
  if(std::this_thread::get_id()==master_thread)
  {
    do_master_thread_work();
  }
  do_common_work();
}
```

另外，当前线程的 `std::thread::id` 将存储到一个数据结构中。之后在这个结构体中对当前线程的ID与存储的线程ID做对比，来决定操作是被“允许”，还是“需要”(permitted/required)。

同样，作为线程和本地存储不适配的替代方案，线程ID在容器中可作为键值。例如，容器可以存储其掌控下每个线程的信息，或在多个线程中互传信息。

`std::thread::id` 可以作为一个线程的通用标识符，当标识符只与语义相关(比如，数组的索引)时，就需要这个方案了。也可以使用输出流(`std::cout`)来记录一个 `std::thread::id` 对象的值。

```
std::cout<<std::this_thread::get_id();
```

具体的输出结果是严格依赖于具体实现的，C++标准的唯一要求就是要保证ID比较结果相等的线程，必须有相同的输出。