A.8 线程本地变量

线程本地变量允许程序中的每个线程都有一个独立的实例拷贝。可以使用 `thread_local` 关键字来对这样的变量进行声明。命名空间内的变量，静态成员变量，以及本地变量都可以声明成线程本地变量，为了在线程运行前对这些数据进行存储操作：

```
thread_local int x;  // 命名空间内的线程本地变量

class X
{
  static thread_local std::string s;  // 线程本地的静态成员变量
};
static thread_local std::string X::s;  // 这里需要添加X::s

void foo()
{
  thread_local std::vector<int> v;  // 一般线程本地变量
}
```

由命名空间或静态数据成员构成的线程本地变量，需要在线程单元对其进行使用**前**进行构建。有些实现中，会将对线程本地变量的初始化过程，放在线程中去做；还有一些可能会在其他时间点做初始化，在一些有依赖的组合中，根据具体情况来进行决定。将没有构造好的线程本地变量传递给线程单元使用，不能保证它们会在线程中进行构造。这样就可以动态加载带有线程本地变量的模块——变量首先需要在一个给定的线程中进行构造，之后其他线程就可以通过动态加载模块对线程本地变量进行引用。

函数中声明的线程本地变量，需要使用一个给定线程进行初始化(通过第一波控制流将这些声明传递给指定线程)。如果函数没有被指定线程调用，那么这个函数中声明的线程本地变量就不会构造。本地静态变量也是同样的情况，除非其单独的应用于每一个线程。

静态变量与线程本地变量会共享一些属性——它们可以做进一步的初始化(比如，动态初始化)；如果在构造线程本地变量时抛出异常， `srd::terminate()` 就会将程序终止。

析构函数会在构造线程本地变量的那个线程返回时调用，析构顺序是构造的逆顺序。当初始化顺序没有指定时，确定析构函数和这些变量是否有相互依存关系就尤为重要了。当线程本地变量的析构函数抛出异常时， `std::terminate()` 会被调用，将程序终止。

当线程调用 `std::exit()` 或从main()函数返回(等价于调用 `std::exit()` 作为main()的“返回值”)时，线程本地变量也会为了这个线程进行销毁。应用退出时还有线程在运行，对于这些线程来说，线程本地变量的析构函数就没有被调用。

虽然，线程本地变量在不同线程上有不同的地址，不过还是可以获取指向这些变量的一般指针。指针会在线程中，通过获取地址的方式，引用相应的对象。当引用被销毁的对象时，会出现未定义行为，所以在向其他线程传递线程本地变量指针时，就需要保证指向对象所在的线程结束后，不能对相应的指针进行解引用。