Thread Local Storage

线程局部存储，Thread Local Storage，TLS。

从名字上也可以看出，所谓线程局部存储，是指存放在该区域中的变量有两个含义：

- 存放在该区域中的变量是全局变量，所有线程都可以访问
- 虽然看上去所有线程访问的都是同一个变量，但该全局变量独属于一个线程，一个线程对此变量的修改对其他线程不可见。

说了这么多还是没懂有没有？没关系，接下来看完这两段代码还不懂你来打我。

第一段代码：

```
int a = 1; // 全局变量
void print_a() {
    cout<<a<<endl;
}
void run() {
    ++a;
    print_a();
}
void main() {
    thread t1(run);
    t1.join();
    thread t2(run);
    t2.join();
}
```

上述代码是用C++11写的，我来讲解下这段代码是什么意思。

- 首先我们创建了一个全局变量a，初始值为1
- 其次我们创建了两个线程，每个线程对变量a加1
- 线程的join函数表示该线程运行完毕后才继续运行接下来的代码

那么这段代码的运行起来会打印什么呢？

全局变量a的初始值为1，第一个线程加1后a变为2，因此会打印2；第二个线程再次加1后a变为3，因此会打印3，让我们来看一下运行结果：

```
2
3
```

看来我们分析的没错，全局变量在两个线程分别加1后最终变为3。

接下来我们对变量a的定义稍作修改，其它代码不做改动：

```
__thread int a = 1; // 线程局部存储
```

我们看到全局变量a前面加了一个__thread关键词用来修饰，也就是说我们告诉编译器把变量a放在线程局部存储中，那这会对程序带来哪些改变呢？

简单运行一下就知道了：

```
2
2
```

和你想的一样吗？有的同学可能会大吃一惊，为什么我们明明对变量a加了两次，但第二次运行为什么还是打印2而不是3呢？

想一想这是为什么。

原来，这就是线程局部存储的作用所在，线程t1对变量a的修改不会影响到线程t2，线程t1在将变量a加到1后变为2，但对于线程t2来说此时变量a依然是1，因此加1后依然是2。

因此，**线程局部存储可以让你使用一个独属于线程的全局变量。**也就是说，虽然该变量可以被所有线程访问，但该变量在每个线程中都有一个副本，一个线程对改变量的修改不会影响到其它线程。