线程局部存储（TLS--Thread Local Storage）

TLS是一种机制，通过这一机制分配的变量，每个当前线程有一个该变量的实例。在Linux上有两种方式表现TLS机制。

1. __thread变量

它是GCC内置的线程局部存储设施，存取效率可以和全局变量相比。TLS中的变量将一直存在，直到线程终止，届时会自动释放这一变量。例如，linux中errno的定义，每个线程都有自己的一份errno的拷贝，防止了一个线程获取errno时被其他线程干扰。

```
static __thread int value = 0;
```

1.1 关于__thread变量的声明和使用，需要注意以下几点：

- 如果变量声明中使用了关键字static或extern，那么关键字__thread必须紧随其后。
- 和全局或静态变量声明一样，__thread变量可以直接设置一个初始值。
- 可以使用C语言取地址操作符(&)获取__thread变量的地址。

1.2 C++中对__thread变量的使用有额外的限制

- 如果定义__thread变量的时候执行初始化，初始化值必须是一个常量表达式。
- __thread只能修饰POD类型(Plain Old Data)例子：

```
__thread string t_object_1 ("Swift");// 错误，因为不能调用对象的构造函数
__thread string* t_object_2 = new std::string (); // 错误，初始化必须用编译期常量
__thread string* t_object_3 = nullptr;// 正确，但是需要手工初始化并销毁对象

```

1.3 什么是POD类型?

对于C++来说，POD类型包括

- 标量类型(scalar type) ---- 内置数据类型与指针类型

- POD类类型 ----class, struct, union， 且具有以下限制： （a. 不具有用户定义的构造函数、析构函数、拷贝算子、赋值算子 b. 不具有继承关系，因此没有基类 c. 不具有虚函数，所以就没有虚表 d. 非静态数据成员没有私有或保护属性的、没有引用类型的、 没有非POD类类型的(即嵌套类都必须是POD)、没有指针到成员类型 ）

1.4 POD类型用途是什么?

POD类型在源代码兼容于ANSI C时非常重要。POD对象与C语言的对应对象具有共同的一些特性，包括初始化、复制、内存布局、寻址。 C++的new表达式中的对象初始化，POD与non-POD的区别如下图所示：

表达式	POD类型T	non-POD类型T
new T	不初始化	缺省初始化
new T()	总是缺省初始化	
new T(x)	总是调用构造函数初始化	

2. 线程特有数据(thread-specific data)

`POSIX thread`使用`get_threadspecific`和`setthreadspecific`组件来实现这一特性，因此编译要加上动态库`-lpthread`，但是使用这种方式使用起来很繁琐，并且效率很低. 其调用接口如下：

```
1 int pthread_key_create(pthread_key_t * key, void (*destructor)
  (void *));
2 int pthread_setspecific(pthread_key_t key, const void *
  value);
3 void *pthread_getspecific(pthread_key_t key);
4 int pthread_key_delete(pthread_key_t key);
```

2.1 使用线程特有数据需要下面几步：

- A. 创建一个键(`key`)，用以将不同的线程特有数据区分开来。调用函数`pthread_key_create`可创建一个`key`，且只需要在首个调用该函数的线程中创建一次。
- B. 在不同线程中，使用`pthread_setspecific`函数将这个`key`和本线程（调用者线程）中的某个变量的值关联起来，这样就可以做到不同线程使用相同的`key`保存不同的`value`。
- C. 在各线程可通过`pthread_getspecific`函数来取得本线程中`key`对应的值。

`Linux`支持最多1024个`key`，一般是128个，所以通常`key`是够用的，如果一个函数需要多个线程特有数据的值，可以将它们封装为一个结构体，然后仅与一个`key`关联。 注意：`pthread_key_create()`只需在第一个使用这个`key`的线程中调用一次即可