**Java ThreadLocal**

* [Creating a ThreadLocal](http://tutorials.jenkov.com/java-concurrency/threadlocal.html#creating-a-threadlocal)
* [Accessing a ThreadLocal](http://tutorials.jenkov.com/java-concurrency/threadlocal.html#accessing-a-threadlocal)
* [Generic ThreadLocal](http://tutorials.jenkov.com/java-concurrency/threadlocal.html#generic-threadlocal)
* [Initial ThreadLocal Value](http://tutorials.jenkov.com/java-concurrency/threadlocal.html#initial-threadlocal-value)
* [Full ThreadLocal Example](http://tutorials.jenkov.com/java-concurrency/threadlocal.html#full-threadlocal-example)
* [InheritableThreadLocal](http://tutorials.jenkov.com/java-concurrency/threadlocal.html#inheritableThreadLocal)

Java中的ThreadLocal类使得你可以创建只能被同一个线程读取和写入的变量。因此，即使两个线程执行同一段代码，代码中有一个引用指向一个ThreadLocal变量，那么两个线程也不能看见彼此的ThreadLocal变量。

**Creating a ThreadLocal**

这是一个代码示例，演示如何创建一个ThreadLocal变量：

private ThreadLocal myThreadLocal = new ThreadLocal();

正如您所看到的，实例化一个新的ThreadLocal对象。每线程只需要一次。即使不同的线程执行访问ThreadLocal的相同代码，每个线程将只看到自己的ThreadLocal实例。即使两个不同的线程在同一ThreadLocal对象上设置不同的值，它们也不能看到彼此的值。

**Accessing a ThreadLocal**

一旦创建了一个ThreadLocal对象，可以设置值来存储其中，如下所示：

myThreadLocal.set("A thread local value");

读取存储在一个ThreadLocal对象中的值像这样：

String threadLocalValue = (String) myThreadLocal.get();

get()方法返回一个Object，而set()方法将一个Object作为参数。

**Generic ThreadLocal**

您可以创建一个泛型的ThreadLocal对象，这样就不必对get()返回的值进行类型化。这里是一个泛型的ThreadLocal例子：

private ThreadLocal<String> myThreadLocal = new ThreadLocal<String>();

现在只能在ThreadLocal实例中存储字符串。此外，您不再需要类型转化从ThreadLocal获得的值：

myThreadLocal.set("Hello ThreadLocal");

String threadLocalValue = myThreadLocal.get();

**Initial ThreadLocal Value**

由于在ThreadLocal对象上设置的值仅对设置该值的线程可见，所以没有线程可以使用set()(对所有线程可见)，在ThreadLocal上设置初始值。

相反，您可以通过继承ThreadLocal并重写initialValue()方法来为ThreadLocal对象指定初始值。就像这样：

private ThreadLocal myThreadLocal = new ThreadLocal<String>() {

@Override protected String initialValue() {

return "This is the initial value";

}

};

现在所有的线程在调用get()时都会看到相同的初始值，在调用set()之前。

**Full ThreadLocal Example**

这是一个完整的可运行的Java ThreadLocal例子：

public class ThreadLocalExample {

public static class MyRunnable implements Runnable {

private ThreadLocal<Integer> threadLocal =

new ThreadLocal<Integer>();

@Override

public void run() {

threadLocal.set( (int) (Math.random() \* 100D) );

try {

Thread.sleep(2000);

} catch (InterruptedException e) {

}

System.out.println(threadLocal.get());

}

}

public static void main(String[] args) {

MyRunnable sharedRunnableInstance = new MyRunnable();

Thread thread1 = new Thread(sharedRunnableInstance);

Thread thread2 = new Thread(sharedRunnableInstance);

thread1.start();

thread2.start();

thread1.join(); //wait for thread 1 to terminate

thread2.join(); //wait for thread 2 to terminate

}

}

此示例创建一个单一的MyRunnable实例，该实例传递给两个不同的线程。两个线程都执行run()方法，因此在ThreadLocal实例上设置不同的值。如果对set()调用的访问已经同步，并且它不是ThreadLocal对象，那么第二个线程将覆盖第一个线程设置的值。

但是，由于它是ThreadLocal对象，所以这两个线程不能看到彼此的值。因此，他们设置和获得不同的值。

**InheritableThreadLocal**

InheritableThreadLocal类是ThreadLocal的子类。与每个线程在ThreadLocal中具有自己的值不同，InheritableThreadLocal授予线程和由该线程创建的所有子线程对值的访问权限。