在多线程开发的过程中可能会有这样的需求，有些变量或者对象在同一个线程中是共享的，在不同的线程中是隔离的，该如何实现呢?

下面的这个例子是演示了在多线程的环境下不同的业务对象使用相同的对象数据时出现了错乱：

```java
public class ThreadSingleton {

    public static Integer data;

    public static void main(String[] args){

        for (int i = 0; i < 2; i++) {
            new Thread(new Runnable() {
                @Override
                public void run() {
                    data = RandomUtils.nextInt();
                    System.out.println("Thread: " + Thread.currentThread().getName() + " , Create Data is " + data);
                    new A().print();
                    new B().print();
                }
            }).start();
        }
    }

    static class A{
        public void print(){
            System.out.println("Class: A , Thread: " + Thread.currentThread().getName() + ", Data : " + data);
            try {
                Thread.sleep(50);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }

    static class B{
        public void print(){
            System.out.println("Class: B , Thread: " + Thread.currentThread().getName() + ", Data : " + data);
            try {
                Thread.sleep(100);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }

}
```

运行结果如下：

```
Thread: Thread-0 , Create Data is 1039570109
Class: A , Thread: Thread-0, Data : 1039570109
Thread: Thread-1 , Create Data is 610962367
Class: A , Thread: Thread-1, Data : 610962367
Class: B , Thread: Thread-0, Data : 610962367
Class: B , Thread: Thread-1, Data : 610962367
```

显然，类A、B在Thread-0和Thread-1中拿到的数据是不一致的。

如何做到一致呢？ 可以考虑用Map实现：

```java
public class ThreadScope {

    public static Map<Thread,Integer> dataMap = new HashMap<Thread, Integer>();

    public static void main(String[] args){

        for (int i = 0; i < 2; i++) {
            new Thread(new Runnable() {
                @Override
                public void run() {
                    Integer data = RandomUtils.nextInt();
                    System.out.println("Thread: " + Thread.currentThread().getName() + " , Create Data is " + data);
                    dataMap.put(Thread.currentThread(),data);
                    new A().print();
                    new B().print();
                }
            }).start();
        }
    }

    static class A{
        public void print(){
            System.out.println("Class: A , Thread: " + Thread.currentThread().getName() + ", Data : " + dataMap.get(Thread.currentThread()));
            try {
                Thread.sleep(50);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }

    static class B{
        public void print(){
            System.out.println("Class: B , Thread: " + Thread.currentThread().getName() + ", Data : " + dataMap.get(Thread.currentThread()));
            try {
                Thread.sleep(100);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }

}
```

运行结果如下：

```
Thread: Thread-0 , Create Data is 1915812090
Class: A , Thread: Thread-0, Data : 1915812090
Thread: Thread-1 , Create Data is 7403431
Class: A , Thread: Thread-1, Data : 7403431
Class: B , Thread: Thread-0, Data : 1915812090
Class: B , Thread: Thread-1, Data : 7403431
```

可以看到，A、B在同一个线程中取到的值是相同的。

可以优雅点实现吗？ 当然可以，就是使用ThreadLocal，顾名思义，线程的本地变量也就是线程范围内的变量，具体实现如下：

```java
public class ThreadLocalTest {

    public static ThreadLocal<Integer> dataMap = new ThreadLocal<Integer>();

    public static void main(String[] args){

        for (int i = 0; i < 2; i++) {
            new Thread(new Runnable() {
                @Override
                public void run() {
                    Integer data = RandomUtils.nextInt();
                    System.out.println("Thread: " + Thread.currentThread().getName() + " , Create Data is " + data);
                    dataMap.set(data);
                    new A().print();
                    new B().print();
                }
            }).start();
        }
    }

    static class A{
        public void print(){
            System.out.println("Class: A , Thread: " + Thread.currentThread().getName() + ", Data : " + dataMap.get());
            try {
                Thread.sleep(50);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }

    static class B{
        public void print(){
            System.out.println("Class: B , Thread: " + Thread.currentThread().getName() + ", Data : " + dataMap.get());
            try {
                Thread.sleep(100);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }

}
```

运行结果如下：

```
Thread: Thread-0 , Create Data is 1927492630
Class: A , Thread: Thread-0, Data : 1927492630
Thread: Thread-1 , Create Data is 865741821
Class: A , Thread: Thread-1, Data : 865741821
Class: B , Thread: Thread-0, Data : 1927492630
Class: B , Thread: Thread-1, Data : 865741821
```

如自己用Map实现的运行结果一样，也达到了我们的目的。

是的，ThreadLocal就是实现了线程内变量的共享，起到的了线程内共享，线程外隔离。

线程结束会释放对应的"Map"中的value吗？ 是的！线程结束后虚拟机会将Thread对应的数据删除掉。

ThreadLocal的应用场景：
1、在Spring的数据库模板、事务管理中都有使用，用来管理数据库资源，给每个线程都分配一份资源，互不干扰，线程结束时资源回收。
2、在Struts2中可以通过ActionContext来获取相应的对象，不管在哪个线程中获取到的都是自己的数据。
3、在同一线程中的数据传递，无需调用传递参数，只需要设置ThreadLocal即可。