**ThreadLocal**

在多线程编程中通常解决线程安全的问题我们会利用synchronzed或者lock控制线程对临界区资源的同步顺序从而解决线程安全的问题，但是这种加锁的方式会让未获取到锁的线程进行阻塞等待，很显然这种方式的时间效率并不是很好。**线程安全问题的核心在于多个线程会对同一个临界区共享资源进行操作，那么，如果每个线程都使用自己的“共享资源”，各自使用各自的，又互相不影响到彼此即让多个线程间达到隔离的状态，这样就不会出现线程安全的问题。事实上，这就是一种“空间换时间”的方案，**每个线程都会都拥有自己的“共享资源”无疑内存会大很多，但是由于不需要同步也就减少了线程可能存在的阻塞等待的情况从而提高的时间效率。

虽然ThreadLocal并不在java.util.concurrent包中而在java.lang包中，但我更倾向于把它当作是一种并发容器（虽然真正存放数据的是ThreadLoclMap）进行归类。**从ThreadLocal这个类名可以顾名思义的进行理解，表示线程的“本地变量”，即每个线程都拥有该变量副本，达到人手一份的效果，各用各的这样就可以避免共享资源的竞争。**

ThreadLocal和线程同步机制都是为了解决多线程中相同变量的访问冲突问题。

而ThreadLocal则从另一个角度来解决多线程的并发访问。ThreadLocal会为每一个线程提供一个独立的变量副本，从而隔离了多个线程对数据的访问冲突。因为每一个线程都拥有自己的变量副本，从而也就没有必要对该变量进行同步了。ThreadLocal提供了线程安全的共享对象，在编写多线程代码时，可以把不安全的变量封装进ThreadLocal。

概括起来说，对于多线程资源共享的问题，同步机制采用了“以时间换空间”的方式，而ThreadLocal采用了“以空间换时间”的方式。前者仅提供一份变量，让不同的线程排队访问，而后者为每一个线程都提供了一份变量，因此可以同时访问而互不影响。

**最常见的ThreadLocal使用场景为 用来解决 数据库连接、Session管理等。**

ThreadLocal是如何做到为每一个线程维护变量的副本的呢？其实实现的思路很简单：**在ThreadLocal类中有一个Map，用于存储每一个线程的变量副本，Map中元素的键为线程对象，而值对应线程的变量副本。**

总的来说，**ThreadLocal 适用于每个线程需要自己独立的实例且该实例需要在多个方法中被使用，也即变量在线程间隔离而在方法或类间共享的场景**。

概括起来说，对于多线程资源共享的问题，同步机制采用了“以时间换空间”的方式，而ThreadLocal采用了“以空间换时间”的方式。前者仅提供一份变量，让不同的线程排队访问，而后者为每一个线程都提供了一份变量，因此可以同时访问而互不影响。

**ThreadLocal为变量在每个线程中都创建了一个副本，那么每个线程可以访问自己内部的副本变量。**

因为ThreadLocal在每个线程中对该变量会创建一个副本，即每个线程内部都会有一个该变量，且在线程内部任何地方都可以使用，线程之间互不影响，这样一来就不存在线程安全问题，也不会严重影响程序执行性能。

ThreadLocal翻译成中文比较准确的叫法应该是：线程局部变量。

　　这个玩意有什么用处，或者说为什么要有这么一个东东？先解释一下，在并发编程的时候，成员变量如果不做任何处理其实是线程不安全的，各个线程都在操作同一个变量，显然是不行的，并且我们也知道volatile这个关键字也是不能保证线程安全的。那么在有一种情况之下，我们需要满足这样一个条件：变量是同一个，但是每个线程都使用同一个初始值，也就是使用同一个变量的一个新的副本。这种情况之下ThreadLocal就非常使用，比如说DAO的数据库连接，我们知道DAO是单例的，那么他的属性Connection就不是一个线程安全的变量。而我们每个线程都需要使用他，并且各自使用各自的。这种情况，ThreadLocal就比较好的解决了这个问题。

那么实现机制是如何的呢？

1. 每个Thread对象内部都维护了一个ThreadLocalMap这样一个ThreadLocal的Map，可以存放若干个ThreadLocal。
2. 2、当我们在调用get()方法的时候，先获取当前线程，然后获取到当前线程的ThreadLocalMap对象，如果非空，那么取出ThreadLocal的value，否则进行初始化，初始化就是将initialValue的值set到ThreadLocal中。

3、当我们调用set()方法的时候，很常规，就是将值设置进ThreadLocal中。

　　4、总结：当我们调用get方法的时候，其实每个当前线程中都有一个ThreadLocal。每次获取或者设置都是对该ThreadLocal进行的操作，是与其他线程分开的。

　　5、应用场景**：当很多线程需要多次使用同一个对象，并且需要该对象具有相同初始化值的时候最适合使用ThreadLocal**。

　　6、其实说再多也不如看一下源码来得清晰。如果要看源码，其中涉及到一个WeakReference和一个Map，这两个地方需要了解下，这两个东西分别是a.Java的弱引用，也就是GC的时候会销毁该引用所包裹(引用)的对象，这个threadLocal作为key可能被销毁，但是只要我们定义成他的类不卸载，tl这个强引用就始终引用着这个ThreadLocal的，永远不会被gc掉。b.和HashMap差不多。

　　事实上，从本质来讲，就是每个线程都维护了一个map，而这个map的key就是threadLocal，而值就是我们set的那个值，每次线程在get的时候，都从自己的变量中取值，既然从自己的变量中取值，那肯定就不存在线程安全问题，总体来讲，ThreadLocal这个变量的状态根本没有发生变化，他仅仅是充当一个key的角色，另外提供给每一个线程一个初始值。如果允许的话，我们自己就能实现一个这样的功能，只不过恰好JDK就已经帮我们做了这个事情。