## 1ThreadLocal 是什么？有哪些使用场景？

当执行set方法时，ThreadLocal首先会获取当前线程对象，然后获取当前线程的ThreadLocalMap对 象。再以当前ThreadLocal对象为key，将值存储进ThreadLocalMap对象中。 get方法执行过程类似。ThreadLocal首先会获取当前线程对象，然后获取当前线程的ThreadLocalMap 对象。再以当前ThreadLocal对象为key，获取对应的value。 由于每一条线程均含有各自私有的ThreadLocalMap容器，这些容器相互独立互不影响，因此不会存在 线程安全性问题，从而也无需使用同步机制来保证多条线程访问容器的互斥性。

ThreadLocal 为每个使用该变量的线程提供独立的变量副本，所以每一个线程都可以独立地改变自己的副本，而不会影响其它线程所对应的副本。

使用场景：

1、在进行对象跨层传递的时候，使用ThreadLocal可以避免多次传递，打破层次间的约束。

2、线程间数据隔离

3、进行事务操作，用于存储线程事务信息。

4、数据库连接，Session会话管理。

详细请看以下链接 解释十分完美

<https://blog.csdn.net/weixin_44050144/article/details/113061884>

比较以上两种情况，我们就会发现，内存泄漏的发生跟ThreadLocalMap中的key是否使用弱引用是没有关系的。那么内存泄漏的的真正原因是什么呢？

​细心的同学会发现，在以上两种内存泄漏的情况中，都有两个前提：

1 没有手动删除这个Entry

2 CurrentThread依然运行

**ThreadLocal内存泄漏的根源是**：由于ThreadLocalMap的生命周期跟Thread一样长，如果没有手动删除对应key就会导致内存泄漏。

## 2 HashMap和HashTable区别

HashMap方法没有synchronized修饰，线程非安全，HashTable线程安全； （2）HashMap允许key和value为null，而HashTable不允许

## 3Hashmap：

理解其底层数据结构

线程不安全

默认容器容量是16，负载因子是0.75,16\*0.75=12，也就是说，当容量达到了12的时候就会进行扩容操作。预估 HashMap 的大小最好，尽量的减少扩容带来的性能损耗

真实数据是存在entry中

* Key :就是写入时的键。
* value :自然就是值。
* next:开始的时候就提到 HashMap 是由数组和链表组成，所以这个 next 就是用于实现链表结构。
* hash:存放的是当前 key 的 hashcode。

hashMap 多线程的话再扩容的时候会造成环形链表，读的时候会造成死循环，cup占用率就会高。

concurrentHashMap,线程安全，读写快，底层实现是一种以空间换时间的结构，创建的时候直接分了16个segment，每个segment实际上存储的还是哈希表，写入的时候先找到对应的segment，然后对segment加锁，写完，解锁。锁segment的时候其他segment还可以继续工作

## 4线程

五种状态

yield（）执行后线程直接进入就绪状态，马上释放了cpu的执行权，但是依然保留了cpu的执行资格， 所以有可能cpu下次进行线程调度还会让这个线程获取到执行权继续执行

join（）执行后线程进入阻塞状态，例如在线程B中调用线程A的join（），那线程B会进入到阻塞队 列，直到线程A结束或中断线程

jvm内存模型

堆是Java虚拟机所管理的内存中最大的一块，是所有线程共享的一块内存区域

栈是每个线程独有的，保存其运行状态和局部自动变量的。栈在线程开始的时候初始化，每个线程的栈 互相独立，因此，栈是线程安全的。

Java内存模型

每个线程也有自己的独立工作内存

## 5并发、并行、串行的区别

串行在时间上不可能发生重叠，前一个任务没搞定，下一个任务就只能等着 并行在时间上是重叠的，两个任务在同一时刻互不干扰的同时执行。 并发允许两个任务彼此干扰。统一时间点、只有一个任务运行，交替执行

## 6并发的三大特性

原子性，

可见性， 当多个线程访问同一个变量时，一个线程修改了这个变量的值，其他线程能够立即看得到修改的值

有序性 对于那些改变顺序之后不会对最终结果造成影响的代码，虚拟机不一定会按 照我们写的代码的顺序来执行，有可能将他们重排序。

例子 i++线程不安全

synchronized关键字同时满足以上三种特性，但是volatile关键字不满足原子性。

## 7线程池中线程复用原理

线程池将线程和任务进行解耦，线程是线程，任务是任务，摆脱了之前通过 Thread 创建线程时的 一个线程必须对应一个任务的限制。

其核心原理在于线程池对 Thread 进行了封装，并不是每次执行任务都会调用 Thread.start() 来创建新线程