ThreadLocal

01 ThreadLocal是什么?

ThreadLocal

ThreadLocal 是一个线程的本地变量，也就是说这个变量是线程独有的，随着线程的消亡而消亡，是不能与其他线程共享的，这样可以避免资源竞争带来的多线程的问题。（典型的空间换时间的做法）

本质

ThreadLocal 用一种存储变量与线程绑定的方式，在每个线程中用自己的 ThreadLocalMap 安全隔离变量。因为每个线程拥有自己唯一的 ThreadLocalMap ，所以 ThreadLocalMap 是天然线程安全的。

使用场景

可以根据资源是需要多线程之间共享的还是单线程内部共享的，来确定他们的使用场景。

为每个线程创建一个独立的数据库连接。

属性和构造方法

对于变量 HASH_INCREMENT = 1640531527，这个值常用于在散列中增加哈希值。它是为了让哈希码能均匀的分布在2的N次方的数组里。（暂时不讨论具体的散列过程）

对于变量 nextHashCode ，使用并发包中的原子类，它由 static 修饰，所以是一个共享变量。

对于变量 threadLocalHashCode，每当初始化 ThreadLocal 实例时，都会在 nextHashCode 变量的基础上使用 CAS 操作增加。

```
public class ThreadLocal<T> {
    /** ThreadLocals rely on per-thread linear-probe hash maps attached ...*/
    private final int threadLocalHashCode = nextHashCode();

    /** The next hash code to be given out. Updated atomically. Starts at ...*/
    private static AtomicInteger nextHashCode =
        new AtomicInteger();

    /** The difference between successively generated hash codes - turns ...*/
    private static final int HASH_INCREMENT = 0x61c88647;

    /** Returns the next hash code. ...*/
    private static int nextHashCode() { return nextHashCode.getAndAdd(HASH_INCREMENT); }

    /** Returns the current thread's "initial value" for this ...*/
    protected T initialValue() { return null; }

    /** Creates a thread local variable. The initial value of the variable is ...*/
    public static <S> ThreadLocal<S> withInitial(Supplier<? extends S> supplier) {...}

    /** Creates a thread local variable. ...*/
    public ThreadLocal() {
    }
```

常用方法

通过查看set、get、remove方法，可以发现每个线程中都有一个ThreadLocalMap数据结构。

它只是相当于一个工具包，提供了操作该容器的方法，如get、set、remove等。

```
public void set(T value) {
    Thread t = Thread.currentThread();
    ThreadLocalMap map = getMap(t);
    if (map != null) {
        map.set(this, value);
    } else {
        createMap(t, value);
    }
}
```

```
public T get() {
    Thread t = Thread.currentThread();
    ThreadLocalMap map = getMap(t);
    if (map != null) {
        ThreadLocalMap.Entry e = map.getEntry( key: this);
        if (e != null) {
            /unchecked/
            T result = (T)e.value;
            return result;
        }
    }
    return setInitialValue();
}
```

```
public void remove() {
    ThreadLocalMap m = getMap(Thread.currentThread());
    if (m != null) {
        m.remove( key: this);
    }
}
```

其他方法

通过查看createMap、getMap方法，可以发现执行set方法时，其值是保存在当前线程的threadLocals变量中。当执行get方法中，是从当前线程的threadLocals变量获取。

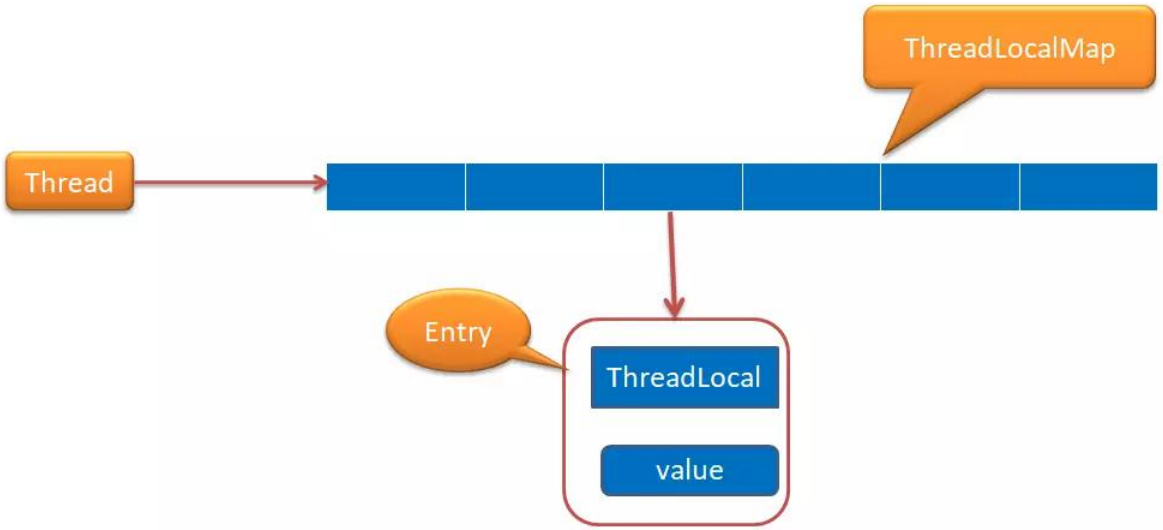
所以可以看出，ThreadLocalMap 会在初始化时与线程绑定，每个线程中用自己唯一的ThreadLocalMap 安全隔离变量。

```
/** Get the map associated with a ThreadLocal. Overridden in ...*/
ThreadLocalMap getMap(Thread t) {
    return t.threadLocals;
}

/** Create the map associated with a ThreadLocal. Overridden in ...*/
void createMap(Thread t, T firstValue) {
    t.threadLocals = new ThreadLocalMap( firstKey: this, firstValue);
}
```

02 Thread、ThreadLocal、ThreadLocalMap之间的关系

Thread、ThreadLocal、ThreadLocalMap之间的关系



ThreadLocal 并不维护 ThreadLocalMap，并不是一个存储数据的容器，它只是相当于一个工具包，提供了操作该容器的方法，如 get、set、remove 等方法。

ThreadLocal 的内部类 ThreadLocalMap 才是存储数据的容器，并且该容器由 Thread 维护。

每一个 Thread 对象均含有一个 ThreadLocalMap 类型的成员变量 threadLocals ，它存储本线程中所有ThreadLocal对象及其对应的值。

03 ThreadLocalMap是什么?

ThreadLocalMap

从名字上看，可以猜到它也是一个类似 HashMap 的数据结构，但是在 ThreadLocalMap 中，并没实现 Map 接口。

属性和构造方法

时机：在调用 ThreadLocal 的 set 方法时，若 ThreadLoalMap 不存在，则会初始化 ThreadLoalMap。

过程：ThreadLoalMap 初始化时，创建一个大小为16的 Entry 数组，Entry 对象用来保存每一个 key-value 键值对，key 为 ThreadLocal 对象。并且设置一个临界阀值，用来判断是否需要扩容。

是不是很神奇，通过 ThreadLocal 对象的 set 方法，结果把 ThreadLocal 对象自己当做 key ，放进了ThreadLoalMap 中。

```
static class ThreadLocalMap {

    /** The entries in this hash map extend WeakReference, using ...*/
    static class Entry extends WeakReference<ThreadLocal<?>> {...}

    /** The initial capacity -- MUST be a power of two. ...*/
    private static final int INITIAL_CAPACITY = 16;

    /** The table, resized as necessary. ...*/
    private Entry[] table;

    /** The number of entries in the table. ...*/
    private int size = 0;

    /** The next size value at which to resize. ...*/
    private int threshold; // Default to 0

    /** Set the resize threshold to maintain at worst a 2/3 load factor. ...*/
    private void setThreshold(int len) { threshold = len * 2 / 3; }

    /** Increment i modulo len. ...*/
    private static int nextIndex(int i, int len) { return ((i + 1 < len) ? i + 1 : 0); }

    /** Decrement i modulo len. ...*/
    private static int prevIndex(int i, int len) { return ((i - 1 >= 0) ? i - 1 : len - 1); }

    /** Construct a new map initially containing (firstKey, firstValue). ...*/
    ThreadLocalMap(ThreadLocal<?> firstKey, Object firstValue) {
        table = new Entry[INITIAL_CAPACITY];
        int i = firstKey.threadLocalHashCode & (INITIAL_CAPACITY - 1);
        table[i] = new Entry(firstKey, firstValue);
        size = 1;
        setThreshold(INITIAL_CAPACITY);
    }
```

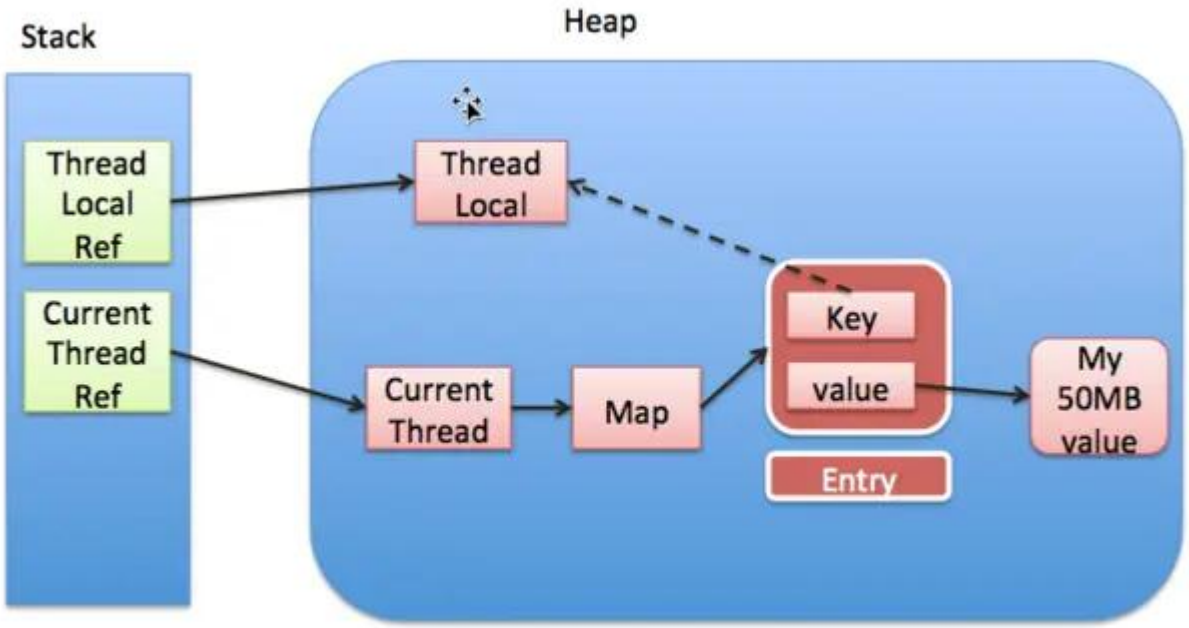
Entry

```java
/** The entries in this hash map extend WeakReference, using ...*/
static class Entry extends WeakReference<ThreadLocal<?>> {
    /** The value associated with this ThreadLocal. */
    Object value;

    Entry(ThreadLocal<?> k, Object v) {
        super(k);
        value = v;
    }
}
```

Entry 是一个以 ThreadLocal 为 key ，Object 为 value 的键值对。

注意，threadLocal 是弱引用。因为 Entry 继承了 WeakReference ，在 Entry 的构造方法中，调用了 super(k) 方法就会将 threadLocal 实例包装成一个 WeakReferenece 。



为什么使用弱引用?

内存泄露问题

ThreadLocal 在没有外部强引用时，发生 gc 时会被回收，如果创建 ThreadLocal 的线程一直持续运行，那么这个 Entry 对象中的 value 就有可能一直得不到回收，发生内存泄露。

若线程执行结束后会被销毁，则相关的引用、实例对象等都会被回收。在实际使用中，一般会使用线程池去维护我们的线程，达到线程复用。线程不会销毁，所以threadLocal内存泄漏就值得我们关注。

Hash冲突

Hash冲突

在理想状态下，哈希函数可以将关键字均匀的分散到数组的不同位置，不会出现两个关键字散列值相同（假设关键字数量小于数组的大小）的情况。

在实际使用中，经常会出现多个关键字散列值相同的情况，我们将这种情况称为散列冲突。

解决Hash冲突

为了解决散列冲突，主要采用两种方式： 分离链表法和开放定址法。

为什么使用开放定址法?

散列值分散的十分均匀，很少会出现冲突。

经常需要清除无用的对象，使用纯数组更加方便。

清理过期数据

在源码中针对这种 key 为 null 的 Entry 称之为 "stale entry"。

在 set、get、remove 操作过程中，有 replaceStaleEntry、expungeStaleEntry、cleanSomeSlots 等方法负责清理过期数据。

扩容问题

时机

在 set 过程中，若无过期数据，且有效数据容量超过阀值。

过程

扫描整个数组，清理所有过期数据。
当前大小大于等于数组容量的1/2时，开始扩容。
创建两倍容量大小的新数组。
遍历旧数组(只考虑 Entry 存在的情况)，若 key 过期，则将 value 设置为 null 。若 key 有效，则插入新表中。
重新设置阀值、大小、引用关系等。

Public方法

分析源码

感谢