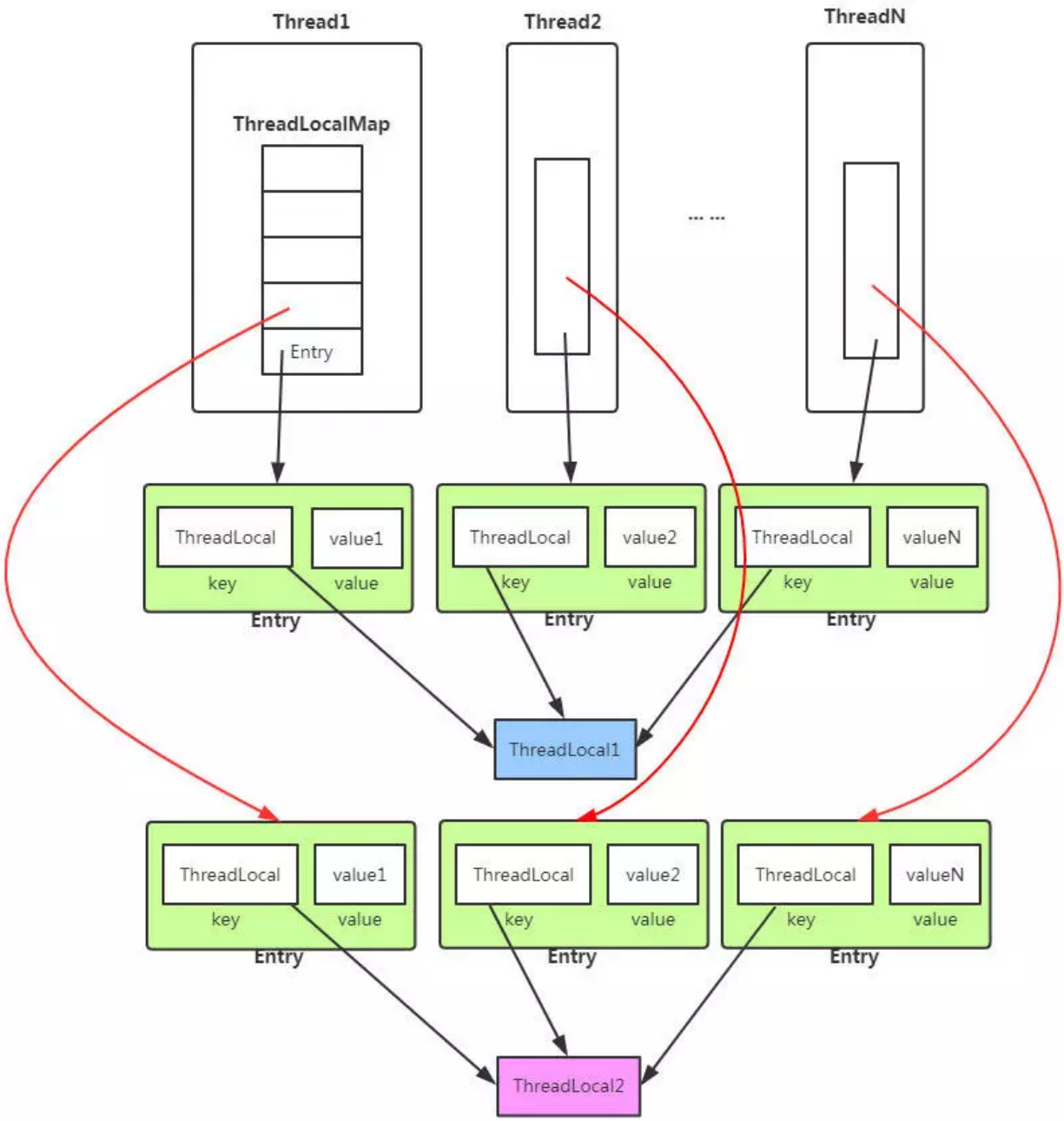
ThreadLocal

- 定义：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。
- 结构：



- 原理：

1. get():用于获取当前线程的副本变量值

```
public T get() {
    // 1. 获取当前线程的ThreadLocalMap对象->
    Thread t = Thread.currentThread();
    ThreadLocalMap map = getMap(t);

    if (map != null) {
```

```
        // 2. 从map中获取线程存储的 k-v entry节点
        ThreadLocalMap.Entry e = map.getEntry(this);
        if (e != null)
            // 3. 从entry节点获取value返回
            return (T)e.value;
    }
    // 4. map为空时默认返回初始值null,可以重写initialValue设置不同的初始值->
    // 注意空指针异常
    return setInitialValue();
}

ThreadLocalMap getMap(Thread t) {
    // 1.1 返回一个ThreadLocalMap对象->
    return t.threadLocals;
}

static class ThreadLocalMap {
        // 1.2 ThreadLocalMap是threadLocal类里的一个静态内部类
        static class Entry extends WeakReference<ThreadLocal<?>> {
            Object value;
            Entry(ThreadLocal<?> k, Object v) {
                // 1.3 键弱引用，值强引用
                super(k);
                value = v;
            }
        }
        // 补充：ThreadLocalMap结构非常简单，采用线性探测的方式解决冲突
        // 所以建议每个线程只存一个变量，减少冲突
        ...
}

private T setInitialValue() {
    // 4.1 initialValue默认返回null->
    T value = initialValue();
    Thread t = Thread.currentThread();
    ThreadLocalMap map = getMap(t);
    if (map != null)
        // 4.3 map不为空就set
        map.set(this, value);
    else
        // 4.4 map为空就create->
        createMap(t, value);
    return value;
}

protected T initialValue() {
    // 4.2 返回空->
    return null;
}

void createMap(Thread t, T firstValue) {
        // 4.5 new ThreadLocalMap->
        t.threadLocals = new ThreadLocalMap(this, firstValue);
```

```
}
```

2. set():用于保存当前线程的副本变量值

```
public void set(T value) {
    // 1. 获取当前线程的ThreadLocalMap对象
    Thread t = Thread.currentThread();
    ThreadLocalMap map = getMap(t);
    if (map != null)
        // 2. map不为空，更新value
        map.set(this, value);
    else
        // 3. map为空，new ThreadLocalMap
        createMap(t, value);
}
void createMap(Thread t, T firstValue) {
        t.threadLocals = new ThreadLocalMap(this, firstValue);
}
```

3. initial():为当前线程初始副本变量值

4. remove():移除当前线程的副本变量值

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

- ThreadLocal与内存泄漏

  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。

```
static class ThreadLocalMap {
        // 1.2 ThreadLocalMap是threadLocal类里的一个静态内部类
        static class Entry extends WeakReference<ThreadLocal<?>> {
            Object value;
            Entry(ThreadLocal<?> k, Object v) {
                // 1.3 键弱引用，值强引用
                // 键（键是个ThreadLocal）使用弱引用的原因在于，当没有强引用指向ThreadLocal变量时，ThreadLocal可以被回收
                // 但是，ThreadLocalMap 维护 ThreadLocal 变量与具体实例的映射，当 ThreadLocal 变量被回收后，该映射的键变为 null，该 Entry 无法被移除。从而使得实例被该 Entry 引用而无法被回收造成内存泄漏。
                super(k);
                value = v;
            }
        }
        ...
        private void set(ThreadLocal<?> key, Object value) {
            // 针对Entry无法被回收造成内存泄漏的问题
            // ThreadLocalMap 的 set 方法中
```

```
            Entry[] tab = table;
            int len = tab.length;
            int i = key.threadLocalHashCode & (len-1);

            for (Entry e = tab[i];
                 e != null;
                 e = tab[i = nextIndex(i, len)]) {
                ThreadLocal<?> k = e.get();

                if (k == key) {
                    e.value = value;
                    return;
                }

                if (k == null) {
                    // 通过 replaceStaleEntry 方法将所有键为 null 的 Entry 的值设置为
null，从而使得该值可被回收。
                    replaceStaleEntry(key, value, i);
                    return;
                }
            }

            tab[i] = new Entry(key, value);
            int sz = ++size;
            if (!cleanSomeSlots(i, sz) && sz >= threshold)
                // 另外，会在 rehash 方法中通过 expungeStaleEntry 方法将键和值为 null 的
Entry 设置为 null 从而使得该 Entry 可被回收。
                rehash();
            // 通过这种方式，ThreadLocal 可防止内存泄漏。
        }
        ...
}
```

关于key为弱引用的设计：

1. key 使用强引用：由于每个线程访问某 ThreadLocal 变量后，都会在自己的 Map 内维护该 ThreadLocal 变量与具体实例的映射，如果不删除这些引用（映射），则这些 ThreadLocal 不能被回收，可能会造成内存泄漏。
2. key 使用弱引用：ThreadLocalMap 维护 ThreadLocal 变量与具体实例的映射，当 ThreadLocal 变量被回收后，该映射的键变为 null，该 Entry 无法被移除。从而使得实例被该 Entry 引用而无法被回收造成内存泄漏。
3. 针对key 使用弱引用而导致的内存泄漏，ThreadLocalMap 的 set 方法通过调用 replaceStaleEntry 方法，回收键为 null 的 Entry 对象的值（即为具体实例）以及 Entry 对象本身，从而防止内存泄漏。