# 概述

## 简介

本文档主要记录学习JDK1.8中java.lang包下的源码。阅读目的主要是学习比较常用的并且经常被用作面试的的class，提升自己的基础知识。

# 根目录

根目录就是指java.lang包下的类，不包含其子包。该目录下需要掌握的是这几类：

* Object，超级父类，理清其各个方法的作用
* 包装类，Java内置类型对应的包装类（包括Void），掌握包装类中的Cache。
* String，StringBuilder，StringBuffer
* Thread，Process
* Class，ClassLoader

## 超级父类Object

Object类主要掌握类中包含的各个方法。详见java object类.doc文档。

## 包装类

包装类是指Java内部基础类型的封装，各个基础类型对应的包装类如下表：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 字节数 | 基础类型 | 包装类 | 是否有缓存 | 缓存范围 |
| 1 | boolean | Boolean | 有 | 两个枚举值 |
| byte | Byte | 有 | [-128,127] |
| 2 | char | Character | 有 | [0,127] |
| short | Short | 有 | [-128,127] |
| 4 | int | Integer | 有 | [-128,127] |
| float | Float | 无 |  |
| 8 | long | Long | 有 | [-128,127] |
| double | Double | 无 |  |

包装类对象内存复用发生在装箱的时候，装箱调用的是valueOf方法。

所有的包装类都继承了Comparable接口，因此都是可比较的。调用compareTo则可比较大小。

## String，StringBuilder，StringBuffer

这三个类需要对比学习，通过对比理清各个类的特点及使用场景。

|  |  |  |  |
| --- | --- | --- | --- |
| 特点 | String | StringBuilder | StringBuffer |
| final类 | 是 | 是 | 是 |
| Comparable接口 | 是 | 否 | 否 |
| synchronized操作 | 否 | 否 | 是 |
| 属性 final | 是 | 否 | 否 |

## Thread线程

参考：<https://blog.csdn.net/pange1991/article/details/53860651/>

参考：Thread类定义中的State枚举类

### 状态列表

Java线程一共有6个状态，分别如下表所示。这6中状态只针对Java线程，而不是操作系统线程的状态。各个状态之间的转换详见1.3.2。

|  |  |
| --- | --- |
| 状态 | 描述 |
| New | 当一个线程被创建当还未调用start方法时，此时线程处于New状态 |
| Runnable | 此状态表示的是线程在java虚拟机中的状态，处于该状态的线程，从操作系统层面来看可能正在运行，也可能正在等待cpu，包括两种情况，一种是Ready，一种是Running。当线程的start方法被调用后，线程将位于可运行线程池中等待被调度，获取CPU，此时处于Ready状态；当线程获取到CPU时，线程处于Running状态 |
| Blocked | 线程如果未获取到锁将被阻塞，此时线程将处于Blocked状态 |
| Waiting | 线程在执行过程中如果调用了Object.wait（）；Thread.join（）；LockSupport.park（）方法时，线程将进入Waiting状态。 |
| Timed\_Waiting | 线程在执行过程中如果调用了Thread.sleep()；Object.wait(long)；Thread.join（long）;LockSupport. parkNanos();LockSupport.parkUntil()方法时，线程将进入Timed\_Waiting状态 |
| Terminated | 线程完成执行之后将进入种终止状态 |

### 状态机

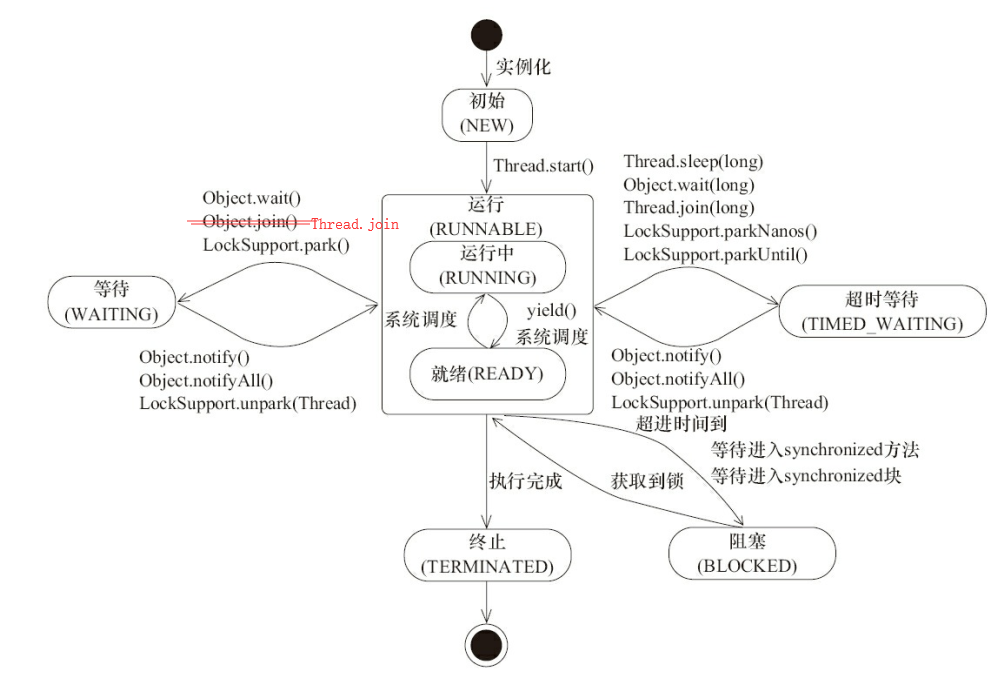


图- 8 Java线程状态机

### yield和sleep

* yield方法使得当前线程让出CPU，同时进入同等优先级的Runnable队列，重新参与CPU获取；
* sleep方法使当前线程休眠指定时间，但是不会释放锁，待时间到了之后才进入同等优先级的Runnable队列；

## ThreadLocal

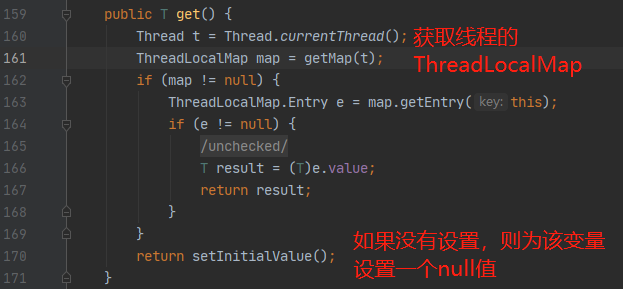
顾名思义，ThreadLocal是将一个对象的值存放在Thread本地空间，其实质是将ThreadLocal变量的引用作为key，生成的值作为value，封装为Entry存放在Thread内的ThreadLocalMap（底层采用的是数组来存放）中。如此各个Thread在获取该ThreadLocal变量的值是都是从本地线程空间的Map中获取，各个线程引用的值互相独立，其示意图可参考如下。

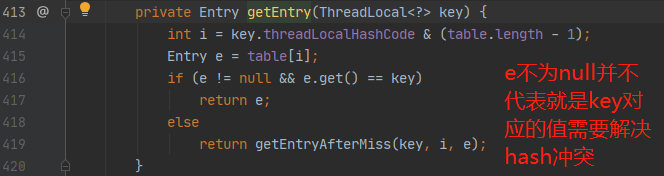


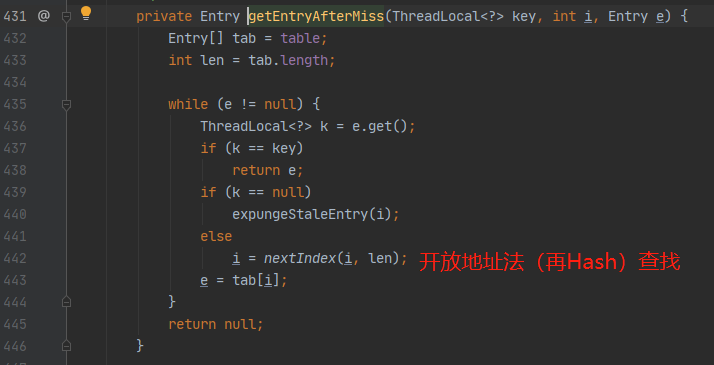
因此对ThreadLocal变量的核心操作主要是Map的get和set操作。ThreadLocalMap采用开放地址法解决hash冲突。

### get操作

以上图为例，当线程A执行a.get（）操作时，首先需要获取当前线程中的ThreadLocalMap，如果Map不为空，则计算a在该map中的index，然后从index处查直到找到该key对应的值，或者得知该key没有值，则在map中给该key设置“零值”。

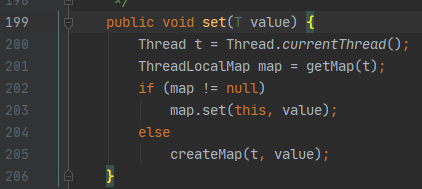


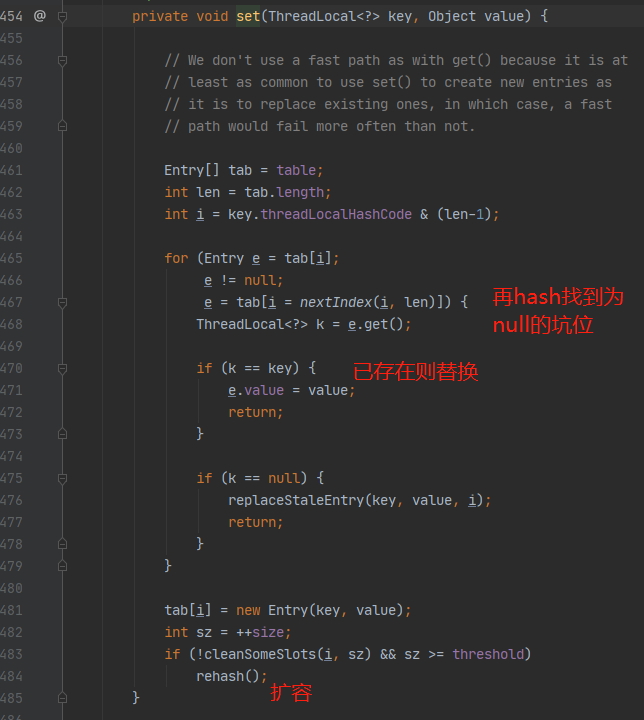




### set操作

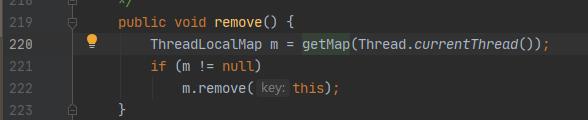
set操作首先也是获取当前线程及其ThreadLocalMap，判断map是否已经创建，没创建就创建map并将值存入map，已创建则根据key的threadlocalhashcode（threadlocalhashcode是一个AtomicInteger类型的全局变量，每定义一个ThreadLocal变量，该变量的threadlocalhashcode将自增一个常量（不是1））计算index并存入value，最后根据size是否超过Threshold判断是否需要扩容。

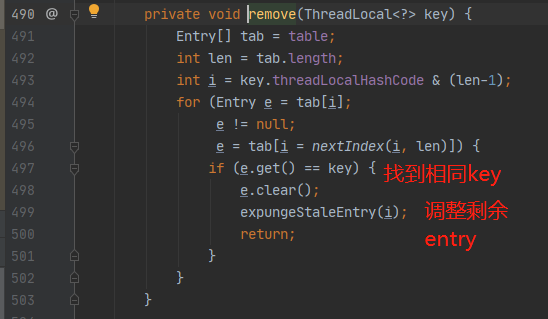


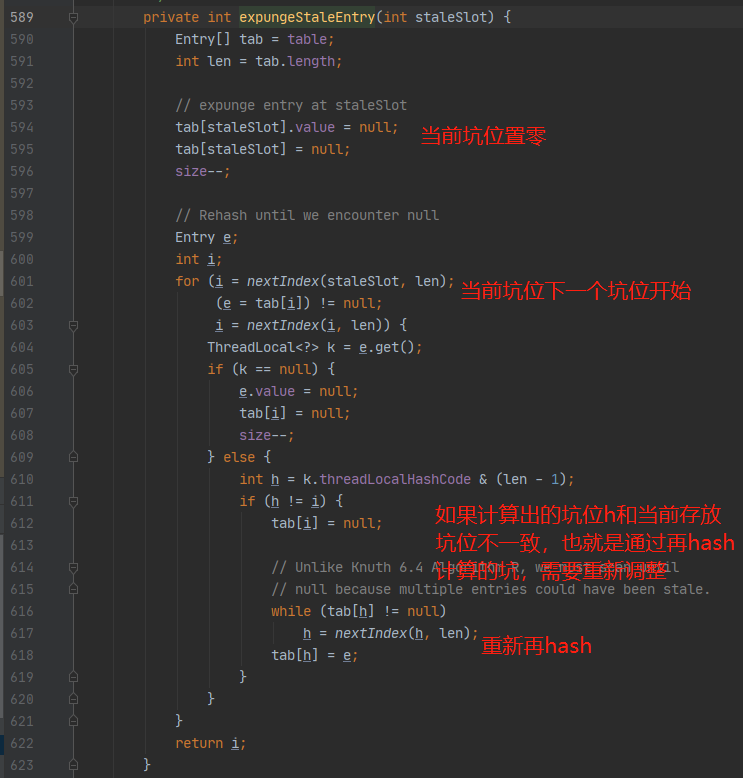


### remove操作

remove操作的思路也是先获取线程及其ThreadLocalMap，根据ThreadLocalHashCode计算index，开放地址法（再hash法）定位找到相同key（记为new\_index）并设置为null，最后还有一个非常重要的步骤就是重新从new\_index开始采用再hash法复位其他entry。







## Class和ClassLoader

Class.forName和ClassLoader.loadClass的区别是前者除了将class文件加载到内存外，还需要对类做链接和初始化操作，而后者默认是只加载到内存，不做链接和初始化操作。