#### String和StringBuffer、StringBuilder的区别

可变性

String类中使用字符数组保存字符串，private

final char value[]，所以string对象是不可变的。StringBuilder与StringBuffer都继承自AbstractStringBuilder类，在AbstractStringBuilder中也是使用字符数组保存字符串，char[]

value，这两种对象都是可变的。

线程安全性

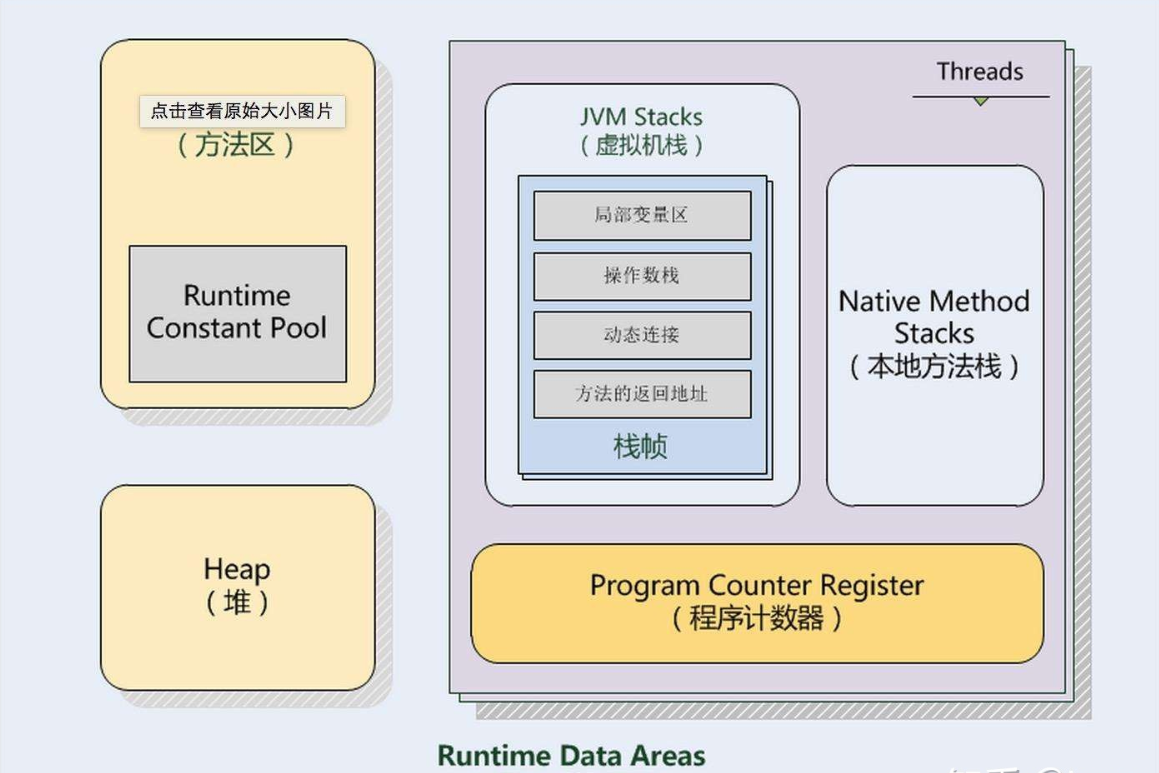
String中的对象是不可变的，也就可以理解为常量，线程安全。StringBuffer对方法加了同步锁或者对调用的方法加了同步锁，所以是线程安全的。StringBuilder并没有对方法进行加同步锁，所以是非线程安全的。

性能

StringBuffer相比StringBuilder加了锁，所以性能比StringBuilder差

#### 运行时数据区

根据 JVM 规范，JVM 内存共分为虚拟机栈、堆、方法区、程序计数器、本地方法栈五个部分。



#### HashMap和HashTable区别

1）.HashTable的方法前面都有synchronized来同步，是线程安全的；HashMap未经同步，是非线程安全的。

2）.HashTable不允许null值(key和value都不可以) ；HashMap允许null值(key和value都可以)。

3）.HashTable有一个contains(Object

value)功能和containsValue(Object

value)功能一样。

4）.HashTable使用Enumeration进行遍历；HashMap使用Iterator进行遍历。

5）.HashTable中hash数组默认大小是11，增加的方式是old\*2+1；HashMap中hash数组的默认大小是16，而且一定是2的指数。

6）.哈希值的使用不同，HashTable直接使用对象的hashCode； HashMap重新计算hash值，而且用与代替求模。

#### ArrayList和vector区别

ArrayList和Vector都实现了List接口，都是通过数组实现的。

Vector是线程安全的，而ArrayList是非线程安全的。

List第一次创建的时候，会有一个初始大小，随着不断向List中增加元素，当List 认为容量不够的时候就会进行扩容。Vector缺省情况下自动增长原来一倍的数组长度，ArrayList增长原来的50%。

#### ArrayList和LinkedList区别及使用场景

区别

ArrayList底层是用数组实现的，可以认为ArrayList是一个可改变大小的数组。随着越来越多的元素被添加到ArrayList中，其规模是动态增加的。

LinkedList底层是通过双向链表实现的， LinkedList和ArrayList相比，增删的速度较快。但是查询和修改值的速度较慢。同时，LinkedList还实现了Queue接口，所以他还提供了offer(),

peek(), poll()等方法。

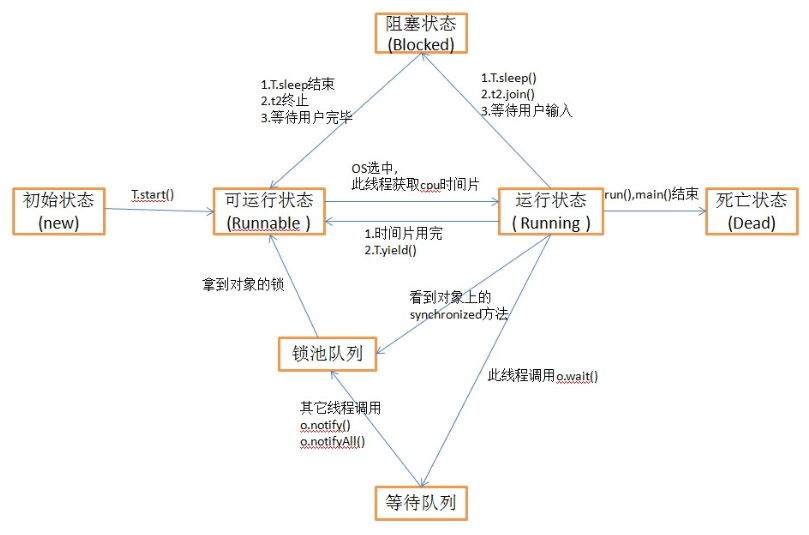
使用场景

LinkedList更适合从中间插入或者删除（链表的特性）。

ArrayList更适合检索和在末尾插入或删除（数组的特性）。

#### 线程的状态转换

线程一共有如图所示的5种状态



这里记录一下等待队列的状态切换

调用obj的wait(), notify()方法前，必须获得obj锁，也就是必须写在synchronized(obj) 代码段内。

与等待队列相关的步骤和图

线程1获取对象A的锁，正在使用对象A。

线程1调用对象A的wait()方法。

线程1释放对象A的锁，并马上进入等待队列。

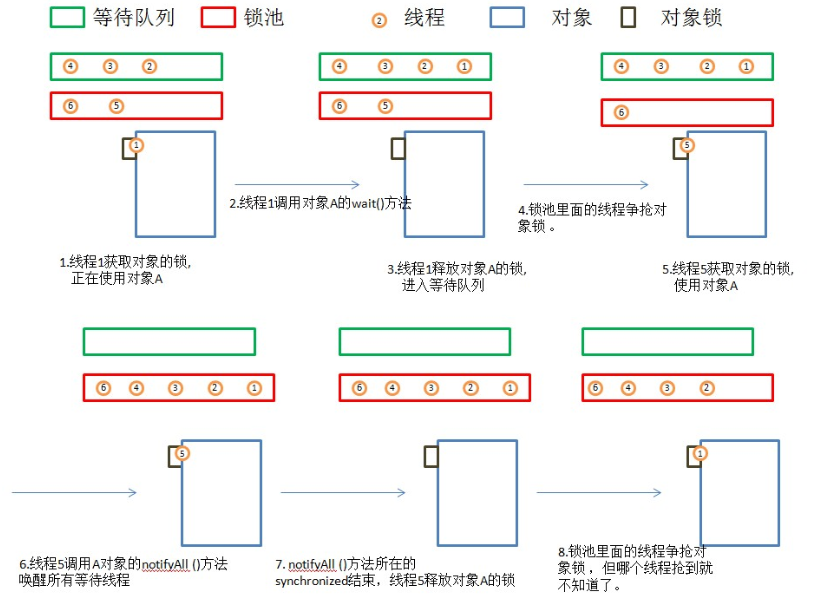
锁池里面的对象争抢对象A的锁。

线程5获得对象A的锁，这时候进入的是可运行状态，被系统分配了时间片，才进入运行状态，进入synchronized块，使用对象A。

线程5调用对象A的notifyAll()方法，唤醒所有线程，所有线程进入锁池。||||| 线程5调用对象A的notify()方法，唤醒一个线程，不知道会唤醒谁，被唤醒的那个线程进入锁池。

notifyAll()方法所在的synchronized结束，线程5释放对象A的锁。

锁池里面的线程争抢对象锁，但线程1什么时候能抢到就不知道了。||||| 原本锁池+第6步被唤醒的线程一起争抢对象锁。



#### sleep和wait的区别(考察的方向是是否会释放锁)

sleep()方法是Thread类中方法，而wait()方法是Object类中的方法。

sleep()方法导致了程序暂停执行指定的时间，让出cpu该其他线程，但是他的监控状态依然保持者，当指定的时间到了又会自动恢复运行状态，在调用sleep()方法的过程中，线程不会释放对象锁。而当调用wait()方法的时候，线程会放弃对象锁，进入等待此对象的等待锁定池，只有针对此对象调用notify()方法后本线程才进入对象锁定池准备。

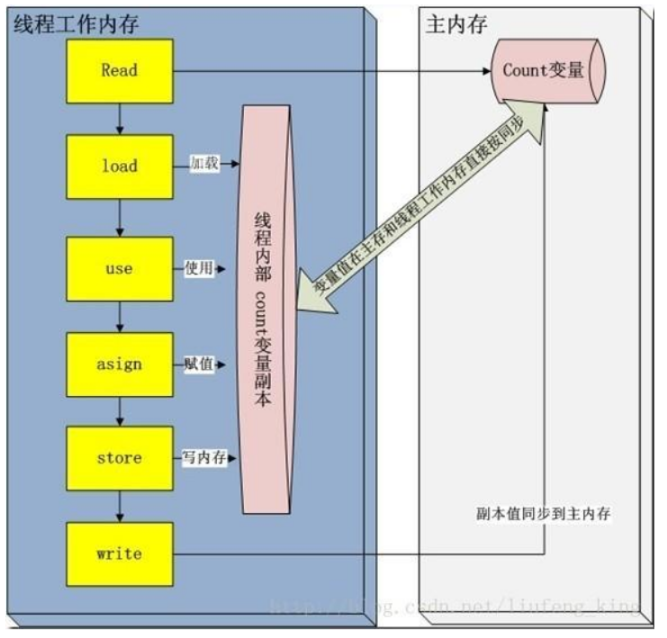
volatile关键字的理解

1. 阻止指令重排序

指令重排序可能导致异常，比如双检锁单例中如果没有保证指令顺序，有可能导致引用了空的对象，抛出空指针异常

1. volatile无法保证原子性

每一个线程运行时都有一个线程栈，线程栈保存了线程运行时候变量值信息。当线程访问某一个对象时候值的时候，首先通过对象的引用找到对应在堆内存的变量的值，然后把堆内存变量的具体值load到线程本地内存中，建立一个变量副本，之后线程就不再和对象在堆内存变量值有任何关系，而是直接修改副本变量的值，在修改完之后的某一个时刻（线程退出之前），自动把线程变量副本的值回写到对象在堆中变量。这样在堆中的对象的值就产生变化了。由于volatile不具有原子性，所以存在两个线程都将volatile对象存入线程副本变量，然后重复写到主内存，导致最终的结果并不正确。



read and load 从主存复制变量到当前工作内存

use and assign 执行代码，改变共享变量值

store and write 用工作内存数据刷新主存相关内容

其中use and

assign 可以多次出现，但是这一些操作并不是原子性，也就是在read load之后，如果主内存count变量发生修改之后，线程工作内存中的值由于已经加载，不会产生对应的变化，所以计算出来的结果会和预期不一样。

相对应的代码示例，由于无法保证原子性，最终结果和预期并不相同



