# 代码

<https://github.com/nickliuchao>

## 系统负载与cpu利用率

**系统负载:** 代表单位时间内正在运行或等待的进程或线程数，代表了系统的繁忙程度，

**CPU利用率:** 则代表单位时间内一个线程或进程实时占用CPU的百分比。

我们知道，**一个进程或者线程在运行时，未必都在实时的利用CPU的**。比如，在CPU密集型的情况下，系统的负载未必会高，但CPU的利用率肯定会高，一个线程/进程一直在计算，它对CPU的实时利用率是100%，而系统负载是0.1;

又比如，而对于I/O密集型的程序来说，有可能CPU的利用率不高，但系统的负载却会非常高，这是因为I/O经常引起阻塞，这样导致很多线程/进程被处于阻塞等待状态，处于等待的线程或进程也是属于负载线程/进程的。

## Tps与qps

TPS(transaction per second)是单位时间内处理**事务的数量**，QPS(query per second)是单位时间内**请求的数量**。

TPS代表一个**事务的处理，可以包含了多次请求**。很多公司用QPS作为接口吞吐量的指标，也有很多公司使用TPS作为标准，两者都能表现出系统的吞吐量的大小，

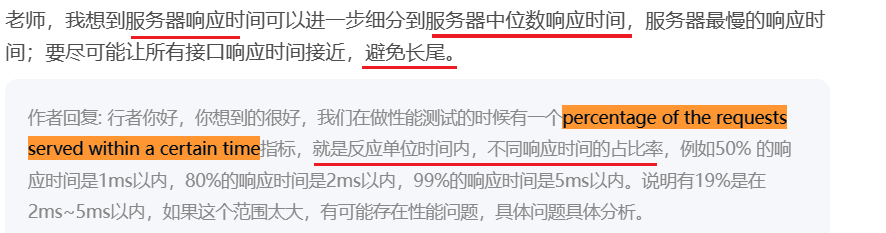
TPS的一次事务代表**一次用户操作到服务器返回结果**，QPS的一次请求代表一个接口的一次请求到服务器返回结果。

当一次用户操作只包含一个请求接口时，TPS和QPS没有区别。

当用户的一次操作包含了多个服务请求时，这个时候TPS作为这次用户操作的性能指标就更具有代表性了。

## percentage of the requests served within a certain time

在特定时间内处理的请求的百分比



# 模块二 · Java编程性能调优

## 03 | 字符串性能优化不容小觑

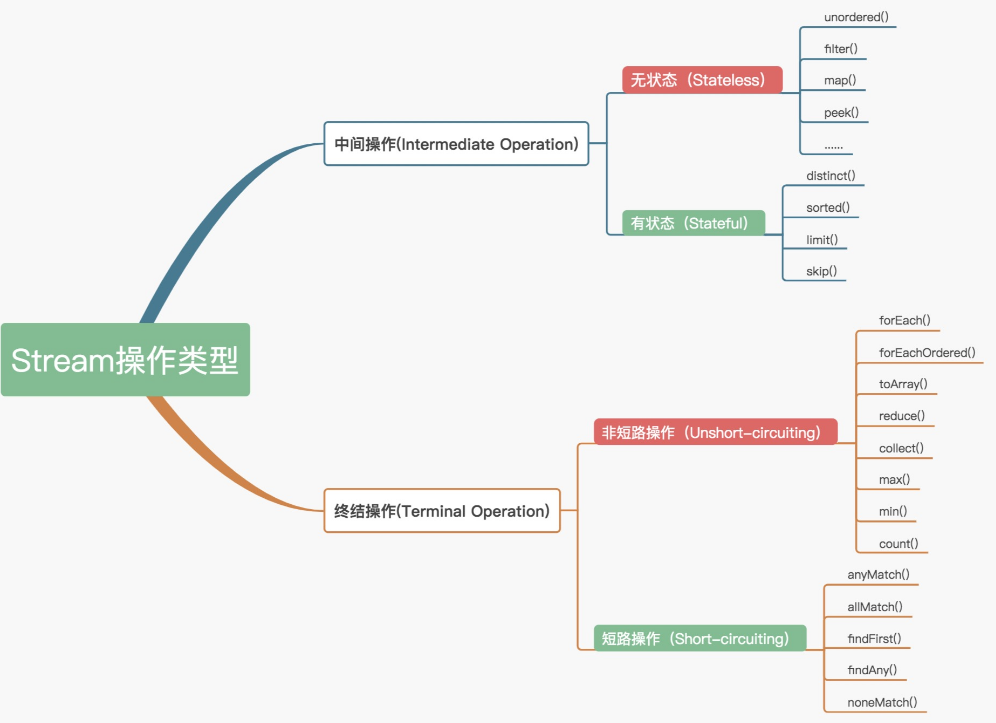
常量池放在堆中，是为了解决之前放在方法区时，由于常量池空间大小有限，存储对象过多导致内存溢出问题。也会存在垃圾回收，但与堆的垃圾回收不一样，这里不会进入老年代，而是直接回收。

 char数组是存放在常量池中，常量是会在编译时生成字面量，在类加载时加载到常量池中。

按照创建顺序来讲，常量“abc”，则会在加载编译时构造常量池时在常量池中创建“abc”字符串对象，而new对象的构造函数是在运行时 引用常量池中的“abc”。还有一个运行时常量池，也就是说，在运行时创建的字符串对象，通过intern方法会在运行时常量池中创建字符串对象。

## 05丨ArrayList还是LinkedList？

## 06丨Stream如何提高遍历集合效率？



## 07丨深入浅出HashMap的设计与优化

HashMap底层是通过数组实现的.

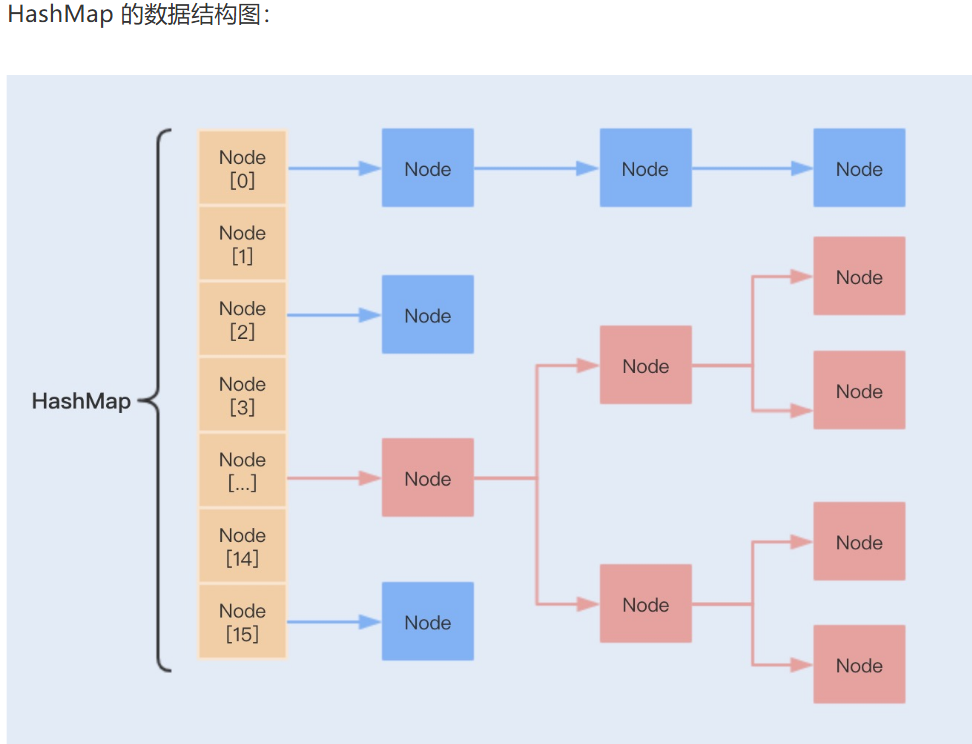
加载因子（loadFactor）

可以理解为容器空间的利用率。

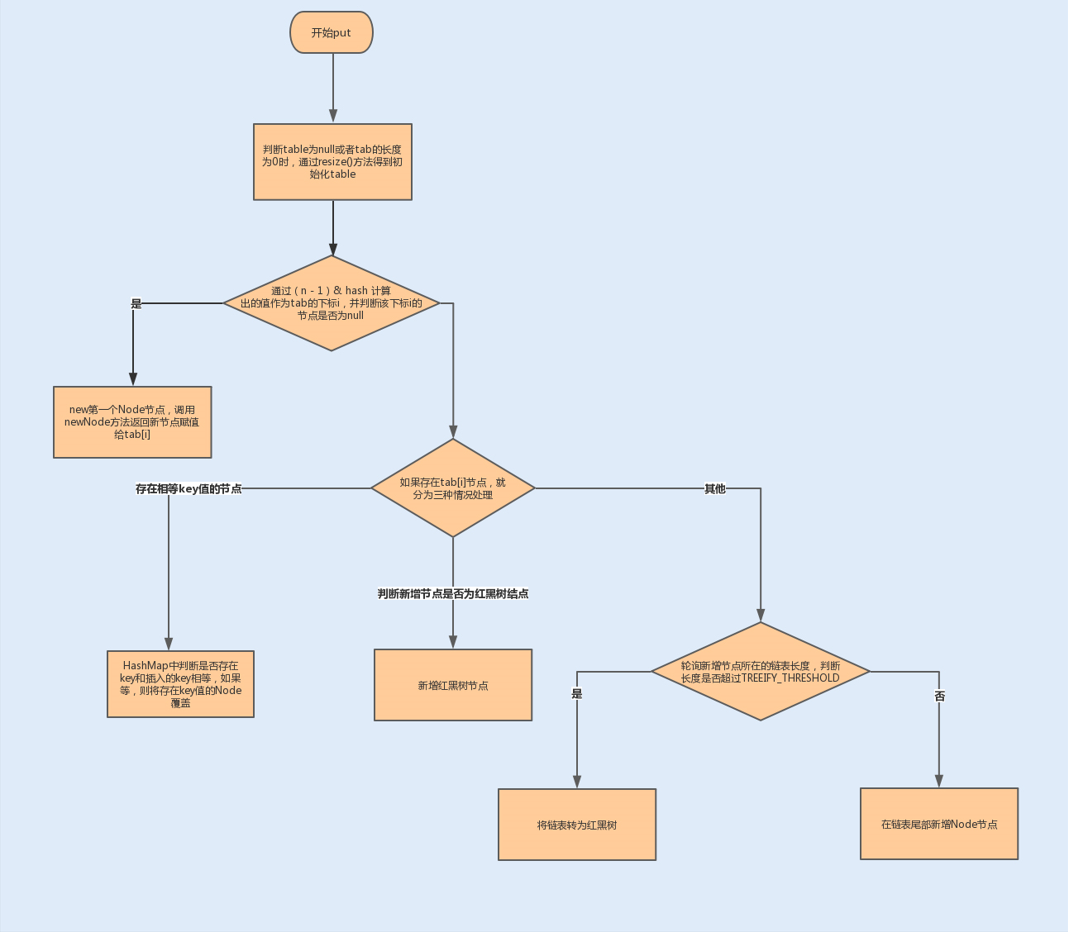
边界值（threshold）

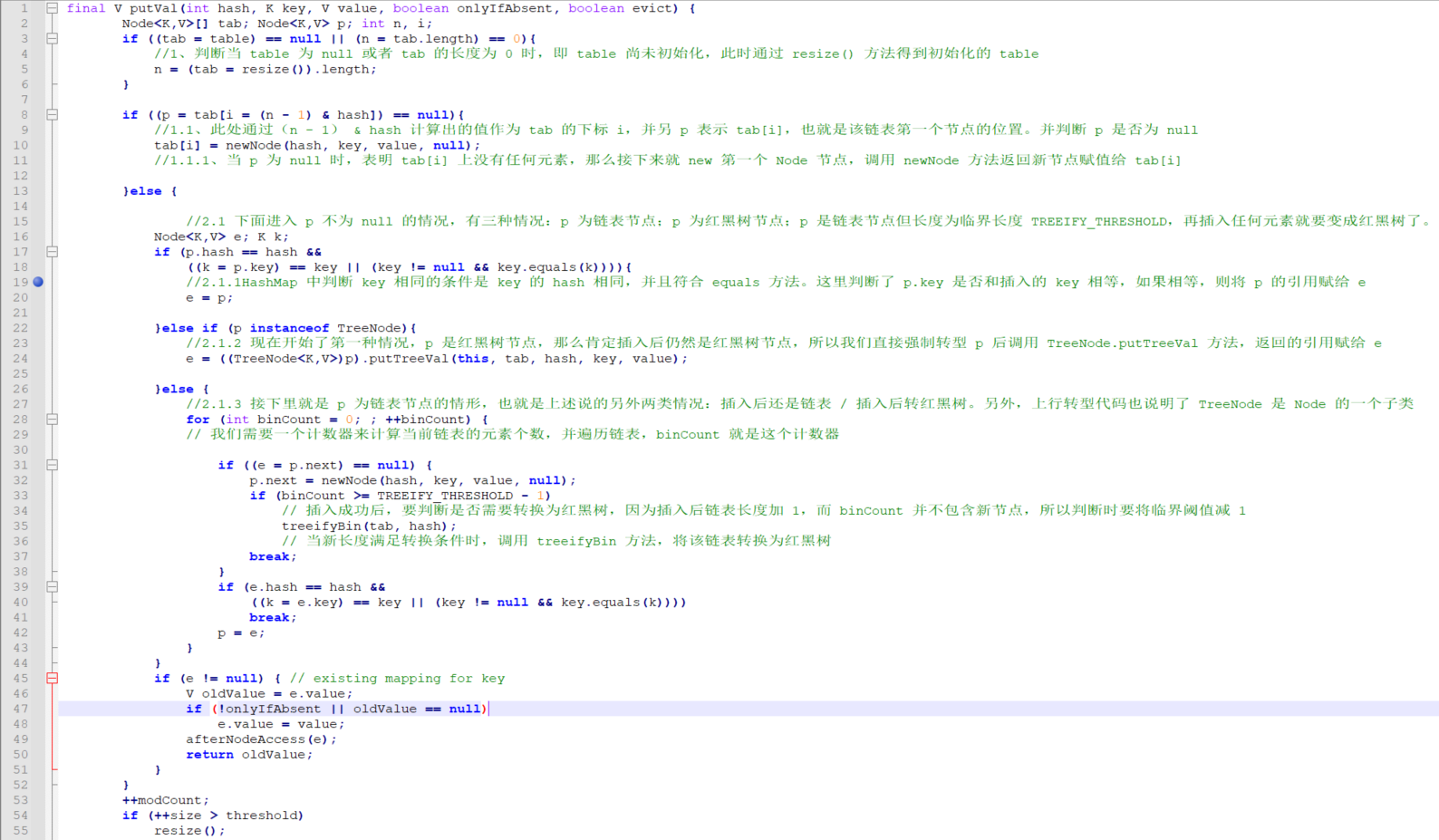
=容器空间大小 \* 加载因子（loadFactor）

表示hashMap的条件的元素数量超过 边界值时，回去进行扩容。



### Put方法逻辑

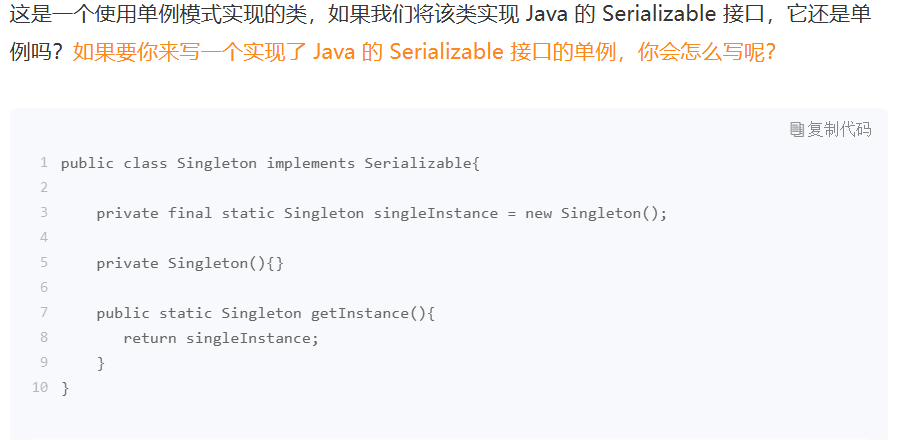






## 08丨网络通信优化之I-O模型

## 09丨网络通信优化之序列化：避免使用Java序列化



**问题：**

可以把类路径上几乎所有实现了 Serializable 接口的对象都实例化。还真不知道怎么写？内部私有类实现，这种可以吗？

**解答：**

导致这个问题的原因是序列化中的readObject会通过反射，调用没有参数的构造方法创建一个新的对象。

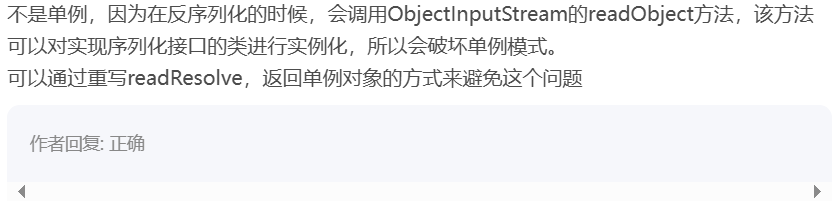
所以我们可以在被序列化类中重写readResolve方法。

private Object readResolve(){

return singleInstance;

}

原因



## 10丨网络通信优化之通信协议：如何优化RPC网络通信？

## 11丨深入了解NIO的优化实现原理