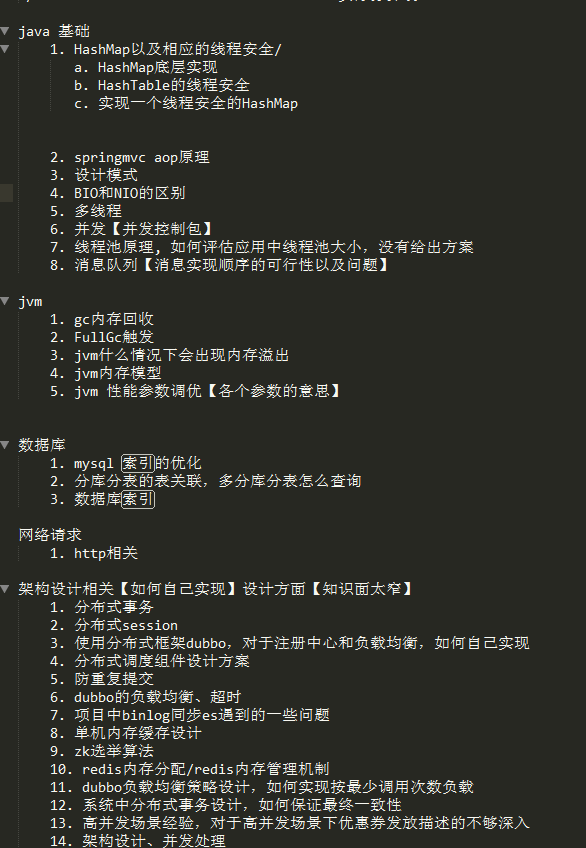
# Java面试题集



## Java基础

### JAVA 中的几种基本数据类型是什么，各自占用多少字节。

char 16

byte 8

short 16

int 32

long 64

boolean 1

float 32

double 64

### String 类能被继承吗，为什么。

String类型不能被继承，因为String为final修饰的类，final修饰的类不能被继承。

### 3. String，Stringbuffer，StringBuilder 的区别。

String字符串常量（final修饰，不可被继承），String是常量，当创建之后既不可被改变。（可以通过StringBuffer和StringBuilder创建String对象）

StringBuffer字符串变量线程安全，也是final类别的，不允许被继承，其中的绝大多数方法都进行了同步处理，包括常用的Append方法都是synchronize修饰。其toString方法会将对象缓存，以减少元素复制开销。

public synchronized String toString() {   
if (toStringCache == null) {   
toStringCache = Arrays.copyOfRange(value, 0, count);   
}   
return new String(toStringCache, true);   
}

StringBuilder字符串变量线程不安全，JDK1.5出现。与StringBuffer一样都继承和实现了同样的接口类方法，方法除了没被synchronize修饰外基本一致。不同之处在于最后toString的时候，会直返回一个新对象。New String();

public String toString() {   
// Create a copy, don’t share the array   
return new String(value, 0, count);   
}

### 4.ArrayList 和 LinkedList 有什么区别。

1、ArrayList是基于索引的数据接口，它的底层是数组。它可以以O(1)时间复杂度对元素进行随机访问。与此对应，LinkedList是以元素列表的形式存储它的数据，每一个元素都和它的前一个和后一个元素链接在一起，在这种情况下，查找某个元素的时间复杂度是O(n)。   
2、相对于ArrayList，LinkedList的插入，添加，删除操作速度更快，因为当元素被添加到集合任意位置的时候，不需要像数组那样重新计算大小或者是更新索引。   
3、LinkedList比ArrayList更占内存

### 5. 讲讲类的实例化顺序，比如父类静态数据，构造函数，字段，子类静态数据，构造函数， 字段，当 new 的时候，他们的执行顺序。

父类的静态属性

父类的静态代码块

子类类静态属性

子类静态代码块

父类成员变量

父类构造方法

子类成员变量

子类构造方法

### 6. 用过哪些 Map 类，都有什么区别，HashMap 是线程安全的吗,并发下使用的 Map 是什么， 他们内部原理分别是什么，比如存储方式，hashcode，扩容，默认容量等。

HashMap HashTable ConcurrentHashMap TreeMap LinkedHashMap

#### HashMap的实现原理

1、基于哈希表（数组+链表+二叉树（红黑树））JDK1.8

2、默认加载因子为0.75，默认数组大小是16（1 << 4）

3、把对象存储到哈希表中，如何存储？

把key对象通过hash()方法计算hash值，然后用这个hash值对数组长度取余数（默认16），来决定对该key的对象在数组中的位置index，

当这个位置有多个对象时，以链表结构存储，JDK1.8后，当链表长度大于8时，链表将转换为红黑树结构存储。

（这样的目的是为了取值更快，存储数据量越大，性能表现越明显。）

4、扩容原理：当数组的容量超过了负载因子，你们表示该数组需要扩容，如何扩容？

扩充算法是：当前数组容量的<<1（相对应乘2），扩大一倍，扩容次数过多，会影响性能，每次扩容表示哈希表重新散列（重新计算每个对象的存储位置），

我们在开发中尽量要减少扩充次数带来的性能问题。

5、线程不安全，适合在单线程中使用。

### 7. JAVA8 的 ConcurrentHashMap 为什么放弃了分段锁，有什么问题吗，如果你来设计， 你如何设计。

### 8. 有没有有顺序的 Map 实现类，如果有，他们是怎么保证有序的。

SortedMap<K,V>