基础技能

java面向对象的思想（封装，继承，多态，重载，重写，接口，抽象类）

String不可变?

StringBuffer和StringBuild区别，

常量池

关键字：serializable，wait()

transient有什么作用(序列化)

介绍一下集合

ArrayList和LinkedList,Vector

Hash表的实现原理

Hash表如何解决冲突

HashMap和HashSet的区别

HashMap和HashTable的区别

HashMap的底层原理

HashMap为何初始容量为16

HashMap初始化为18是如何扩容的

HashMap的线程安全问题

ConcurrentHashMap的原理

List Map Set

线程与进程

多线程是什么

java启动线程的方式

线程池原理，有什么好处

线程的状态,可达状态

线程类中的方法sleep(),yield() wait() notify() run() start()

线程安全，如何实现线程安全

集合中的线程安全如何保证

锁

synchronized的可重入怎么实现

synchronized和Lock

死锁的四个条件，如何破坏死锁

关键字：Volatile(保证有序性可见性),final,synchronized

反射

jvm垃圾回收机制

类加载过程

单例模式优缺点

单例模式的实现(饿汉式，懒汉式)

工厂模式

动态代理

装饰者模式

常用的排序及其实现(冒泡，选择，快排，插入)

Linux的命令

Http协议

TCP与UDP

TCP三次握手与四次挥手，

TCP为何不是两次握手，

Tcp建立连接，但客户端出现故障怎么办。

数据库技能

mysql语句，增删改查，能够说出来

mysql查询优化

mysql索引优化(索引失效的情况)

mysql存储过程

mysql如何看是否走索引

数据库事务ACID

数据库事务隔离级别

数据库事务脏读，幻读，不可重复读

Redis的数据类型

Redis主要作用

Redis 缓存穿透(有key但是无value)，缓存击穿（数据库有），缓存雪崩

Redis分布式锁

Redis持久化策略 AOF 与RDB

Redis的数据删除策略（定时删除，定期删除，惰性删除）

Redis的数据逐出策略

Redis的并发竞争key

Redis与数据库双写不一致

JavaEE技能

request作用

session与cookie

Servlet调用过程

谈谈Spring

Spring IOC

Spring AOP原理

Spring Bean的生命周期

Spring Bean DI

Spring 依赖注入的方式

Spring的注解 @Autowried @Resource

Spring MVC执行流程

Spring事务级别默认事务级别

Spring Boot

ORM框架(jdbc template)

mybatis是什么

mybatis的创建过程

mybatis中的#{}与${}（预编译，sql注入)

mybatis 中的like

Spring Boot与Spring Cloud

Spring Cloud的组件

Dubbo与Zookeeper

Dubbo协议

Dubbo注册中心，对象序列化方式

CAS

euraka,zookeeper,consul区别

Zookeeper选举策略(CP)

Eureka选举策略（AP）

Zookeeper分布式锁

rabbitMQ和activeMQ如何在项目中的使用

其他

docker

nginx

nginx的负载均衡策略

JUC 并发编程

Java基础&高级

基本类型常见的面试题：

Q&A:什么是自动装箱与拆箱?

自动装箱就是java编译器在基本数据类型和对应的对象包装类型之间做一个转化，比如int转为Integer,double转为Double.反之为自动拆箱。

Q&A:String是基本的数据类型吗?

String不是节本的数据类型，是final不可修改的类，为了提高效率可以使用StringBuffer类。

Q&A:int 和integer有什么区别?

int是基本类型，integer是包装类型。

integer必须实例化后才能使用，而int变量不需要。

integer默认为null,而int默认为0；

integer实际是对象的引用，当new出一个integer时实际上是生成一个指针指向此对象，而int则是直接存储数据值。

Q&A:以下结果是什么?

```
Integer a= new Integer(3);
Integer b= 3; // 将3 自动装箱为Integer类型
int c=3;
a==b  // false 两个引用没有引用统一对象。
a==c // true a 自动 拆箱为int 再和c 比较。
两个非new生成的integer对象，进行比较时如果两个变量值的区间在-128~127之间则比较为 true,如果不在此区间则为false;
Integer a=100;  => Integer a = Integer.valueOf(100);
Integer b=100;
a==b // true;
Integer a=128;
Integer b=128;
a== b // false;
对于valueOf方法在Integer类中会有一个cache判断:
```

Q&A short s1 = 1; s1 = s1 + 1;有什么错? short s1 = 1; s1 +=1;有什么错?

1) 对于 shorts1=1;s1=s1+1 来说，在s1+1 运算时会自动提升表达式的类型为 int， 那么将int赋予给short类型的变量 s1会出现类型转换错误。
2) 对于 short s1=1;s1+=1 来说 +=是java 语言规定的运算符，java 编译器会对它 进行特殊处理，因此可以正确编译。

Q&A String的不可变性

```
Strings are constant; their values cannot be changed after theyare created.
String buffers support mutable strings.Because String objects are immutable they
can be shared.
    For example:
 String str = "abc";
 与下面的相等;
 *     char data[] = {'a', 'b', 'c'};
 *     String str = new String(data);
```

```
 * Here are some more examples of how strings can be used:
 * <blockquote><pre>
 *     System.out.println("abc");
 *     String cde = "cde";
 *     System.out.println("abc" + cde);
 *     String c = "abc".substring(2,3);
 *     String d = cde.substring(1, 2);
 * </pre></blockquote>
 * <p>
```

Q&A 为何String设计为不可变?

1、运行时常量池的需要,节省内存空间。

比如执行 String s = "abc";执行上述代码时，JVM首先在运行时常量池中查看是否存在String对象“abc”，如果已存在该对象，则不用创建新的String对象“abc”，而是将引用s直接指向运行时常量池中已存在的String对象“abc”；如果不存在该对象，则先在运行时常量池中创建一个新的String对象“abc”，然后将引用s指向运行时常量池中创建的新String对象。这样在运行时常量池中只会创建一个String对象"abc"，这样就节省了内存空间。

2、同步

因为String对象是不可变的，所以是多线程安全的，同一个String实例可以被多个线程共享。这样就不用因为线程安全问题而使用同步。

3、允许String对象缓存hashcode

查看上文JDK1.8中String类源码，可以发现其中有一个字段hash，String类的不可变性保证了hashcode的唯一性，所以可以用hash字段对String对象的hashcode进行缓存，就不需要每次重新计算hashcode。所以Java中String对象经常被用来作为HashMap等容器的键。

4、安全性

如果String对象是可变的，那么会引起很严重的安全问题。比如，数据库的用户名、密码都是以字符串的形式传入来获得数据库的连接，或者在socket编程中，主机名和端口都是以字符串的形式传入。因为String对象是不可变的，所以它的值是不可改变的，否则黑客们可以钻到空子，改变String引用指向的对象的值，造成安全漏洞。

Q&A String s = new String("xyz");创建了几个String Object?二者之间有什么区别?

两个或一个，”xyz”对应一个对象，这个对象放在字符串常量缓冲区，常量”xyz”不管出现多少遍，都是缓冲区中的那一个。New String每写一遍，就创建一个新的对象，它一句那个常量”xyz”对象的内容来创建出一个新String对象。如果以前就用过’xyz’，这句代表就不会创建”xyz”自己了，直接从缓冲区拿。

Q&A 创建对象的方法:

1.使用new关键字。

2.使用Class类中的newInstance方法，newInstance方法调用无参构造方器创建对象。Class.forName.newInstance;

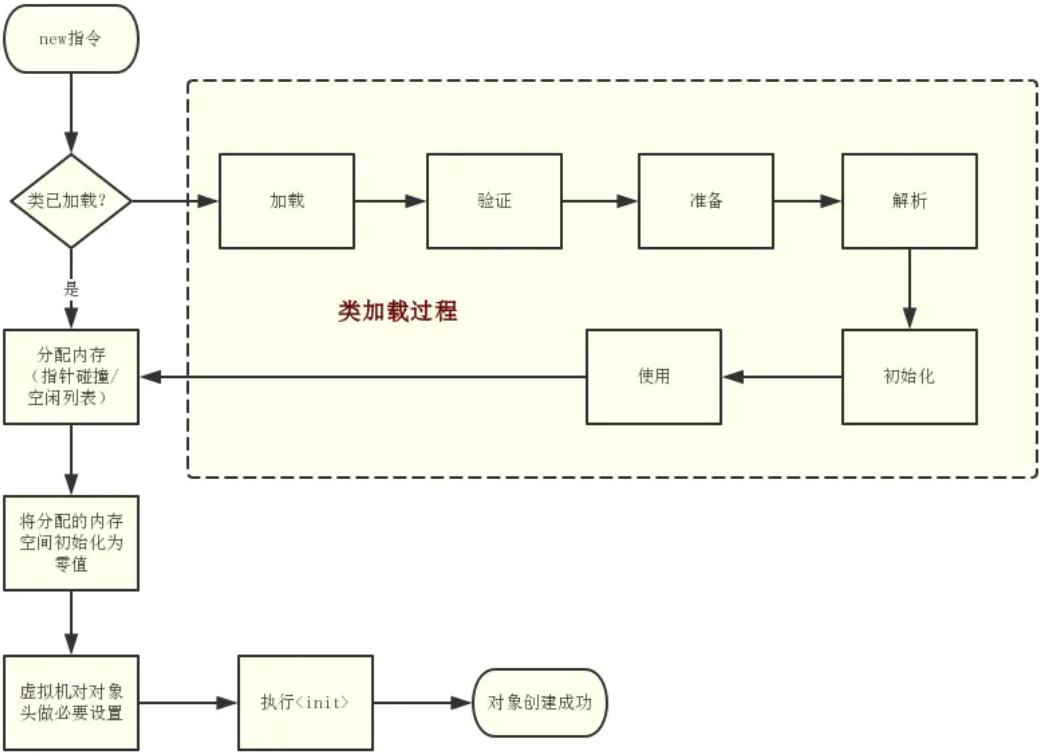
3.使用clone方法、

4.反序列化。

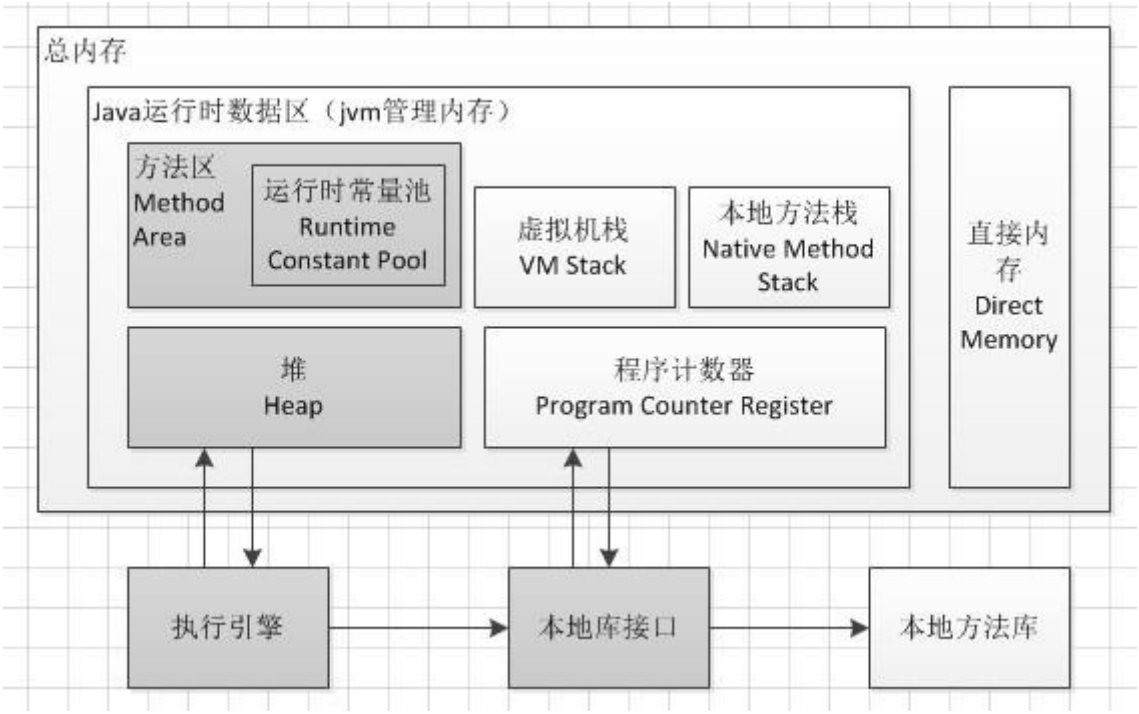
Q&A 对象的产生过程以及存储:

对象的产生:

new将对象存储在堆中，所以用new创建一个对象---特别小的，简单的变量，往往不是很有效。因此对于（基本类型）java不用new来创建这样的变量，而是创建一个并非是引用的“自动”变量。这个变量的值直接存储"值"到堆栈中。



程序创建，运行时对象是如何分配呢？内存是怎样分配呢？



对象产生的时机 类加载，然后进行对象的实例化：

Q&A 什么时候会进行类加载?

1.创建类的实例，也就是new一个对象

2.访问某个类或接口的静态变量，或者对该静态变量赋值

3.调用类的静态方法

4.反射（Class.forName("A")）

5.初始化一个类的子类（会首先初始化子类的父类）

6.JVM启动时标明的启动类，即文件名和类名相同的那个类

new ObjectInitTest（）对象的产生过程

1.JVM会在ObjectInitTest.class文件
2.先加载类级别的成员（静态成员变量 静态块初始化）
3.再加载对象级别的成员（实例成员变量 实例块初始化）

Q&A什么时候进行对象的实例化?

类加载成功（或已加载过）后，就开始进行对象的实例化了。对象的实例化依次进行了如下几个步骤：

1.对象在堆中的内存空间分配

2.初始化零值，这时会将实例变量都赋予零值

3.设置对象头，对象头保存了一些对象在运行期的状态信息，包括类信息地址（知道对象是属于哪个类的）、hashcode（用于hashmap和hashset等hash结构中）、GC分代年龄（可以依此确定对象何时该放入老年代）等

4.init方法执行，这时对变量的实例变量进行初始化

对象初始化的过程也是线程安全的动作。

Q&A StringBuffer StringBulider String 的区别：

StringBuffer线程安全，StringBulider线程不安全，底层实现StringBuffer比StringBulider多一个Synchronized.从源码中可以看得到：

StringBuffer源码分析：

```
    @Override
    public synchronized int length() {
        return count;
    }

    @Override
    public synchronized int capacity() {
        return value.length;
    }

    @Override
    public synchronized void ensureCapacity(int minimumCapacity) {
        if (minimumCapacity > value.length) {
            expandCapacity(minimumCapacity);
        }
    }

    /**
     * @since      1.5
     */
    @Override
    public synchronized void trimToSize() {
        super.trimToSize();
    }
```

```
    /**
     * @throws IndexOutOfBoundsException {@inheritDoc}
     * @see        #length()
     */
    @Override
    public synchronized void setLength(int newLength) {
        toStringCache = null;
        super.setLength(newLength);
    }

    /**
     * @throws IndexOutOfBoundsException {@inheritDoc}
     * @see        #length()
     */
    @Override
    public synchronized char charAt(int index) {
        if ((index < 0) || (index >= count))
            throw new StringIndexOutOfBoundsException(index);
        return value[index];
    }
```

Q&A String 是否可以继承，“+”如何实现的，与StringBuffer区别?

java中通过使用“+”符号串联时实际是使用的StringBuilder实例的appdend()方法来实现的。

Q&A 对于初始化与清理常见的面试题:

Q&A :java 对象加载初始化的顺序?

静态代码块--》非静态代码块-（静态代码块（只有一次），构造函数）-》构造函数，而{}包含的非静态代码块就是和构造方法一样会默认初始化

Q&A: :如何判断一个对象是否要回收?

这就是所谓的对象存活性判断，常用的方法有两种：1.引用计数法； 2.对象可达性分析。由于引用计数法存在互相引用导致无法进行GC的问题，所以目前VM虚拟机多使用对象可达性分析算法。从GC Roots对象计算引用链，能链上的就是存活的。

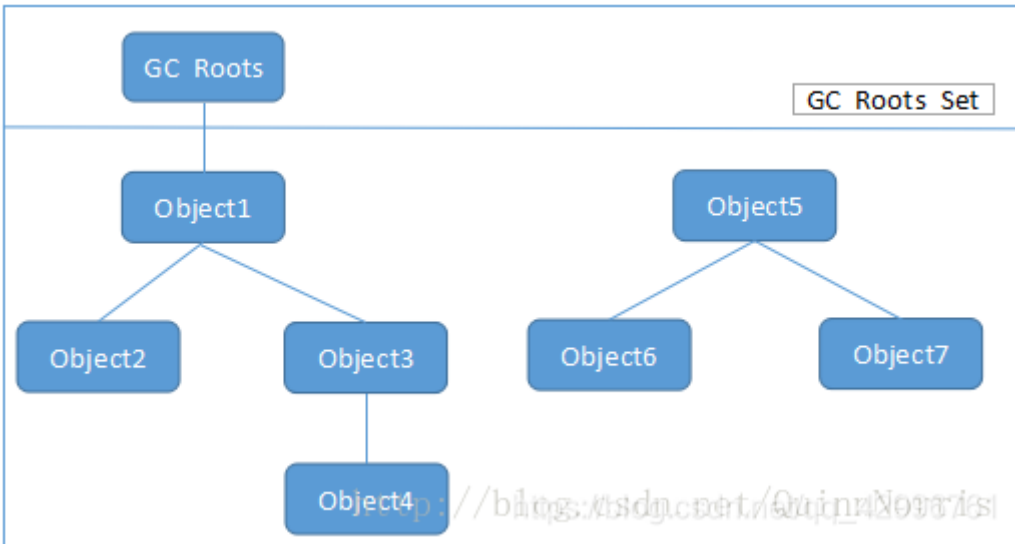
1、引用计数法

引用计数法的逻辑非常简单，但是存在问题，java并不采用这种方式进行对象存活判断。

引用计数法的逻辑是：在堆中存储对象时，在对象头处维护一个counter计数器，如果一个对象增加了一个引用与之相连，则将counter++。如果一个引用关系失效则counter–。如果一个对象的counter变为0，则说明该对象已经被废弃，不处于存活状态。

2、可达性分析算法

在主流的商用程序语言中(Java和C#)，都是使用可达性分析算法判断对象是否存活的。这个算法的基本思路就是通过一系列名为GC Roots的对象作为起始点，从这些节点开始向下搜索，搜索所走过的路径称为引用链(Reference Chain)，当一个对象到GC Roots没有任何引用链相连时，则证明此对象是不可用的，下图对象object5, object6, object7虽然有互相判断，但它们到GC Roots是不可达的，所以它们将会判定为是可回收对象。



参看：https://blog.csdn.net/qq_42996761/article/details/90667725

Q&A: :如何理解java中的垃圾回收机制？为什么需要垃圾回收机制?

java采用分代回收，分为年轻代、老年代、永久代。年轻代又分为E区、S1区、S2区。

到jdk1.8，永久代被元空间取代了。

年轻代都使用复制算法，老年代的收集算法看具体用什么收集器。默认是PS收集器，采用标记-整理算法。

System.gc()即垃圾收集机制是指jvm用于释放那些不再使用的对象所占用的内存。垃圾收集的目的在于清除不再使用的对象。gc通过确定对象是否被活动对象引用来确定是否收集该对象。
如果你向系统申请分配内存进行使用(new)，可是使用完了以后却不归还(delete)，结果你申请到的那块内存你自己也不能再访问,该块已分配出来的内存也无法再使用，随着服务器内存的不断消耗，而无法使用的内存越来越多，系统也不能再次将它分配给需要的程序，产生泄露。一直下去，程序也逐渐无内存使用，就会溢出。

Q&A:java垃圾收集的方式有哪些?

上文已经讲述。

Q&A:java垃圾回收机制的优缺点?

优点：

开发人员无须过多地关心内存管理，而是关注解决具体的业务。虽然内存泄漏在技术上仍然是可能出现的，但不常见。

GC 在管理内存上有很多智能的算法，它们自动在后台运行。与流行的想法相反，这些通常比手动回收更能确定什么时候是执行垃圾回收的最好时机。

缺点：

当垃圾回收发生时将影响程序的性能，显著地降低运行速度甚至将程序停止。所谓 “Stop the world” 就是当垃圾回收发生的时候应用程序的其他任务都将被冻结。对于应用程序的要求来说，这将是不可接收的，虽然 GC 调优可以最小化甚至消除这个影响。

虽然GC有很多方面可以调优，但你不能指定应用程序在何时怎样执行GC

Q&A:java垃圾回收的时间有哪些?

CPU空闲时进行回收、堆内存满了后进行回收、调用System.gc()回收。

Q&A:如果对象置为null，垃圾收集器是否会立即释放占用的内存?

不会。对象回收需要一个过程，这个过程中对象还能复活。而且垃圾回收具有不确定性，指不定什么时候开始回收。

Q&A:什么是GC？为何有GC?

GC就是垃圾收集的意思（Gabage Collection）。我们在开发中会创建很多对象，这些对象一股脑的都扔进了堆里，如果这些对象只增加不减少，那么堆空间很快就会被耗尽。所以我们需要把一些没用的对象清理掉。jvm使用 jstat -gc 12538 5000　即会每5秒一次显示进程号为12538的java进成的GC情况.

Q&A: 方法区如何判断是否需要回收?

方法区主要回收的内容有：废弃常量和无用的类。对于废弃常量也可通过引用的可达性来判断，但是对于无用的类则需要同时满足下面3个条件：
① 该类所有的实例都已经被回收，也就是Java堆中不存在该类的任何实例；
② 加载该类的ClassLoader已经被回收；
③ 该类对应的java.lang.Class对象没有在任何地方被引用，无法在任何地方通过反射访问该类的方法。

Q&A:如果频繁老年代回收怎么分析解决?

老年代频繁回收，一般是Full GC，Full GC 消耗很大，因为在所有用户线程停止的情况下完成回收，而造成频繁 Full GC 的原因可能是，程序存在问题，或者环境存在问题。
对jvm的GC进行必要的监控，操作如下：
1、使用jps命令（或者ps -eaf|grep java）获取到当前的java进程（取得进程id，假如pid为 1234）
2、使用jstat查看GC情况（如：jstat -gc 1234 1000，后面的1000表示每个1000毫米打印一次监控），jstat命令可以参考：https://www.cnblogs.com/yjd_hycf_space/p/7755633.html （此文使用的是jdk8，但是本人亲测jstat在jdk7也是这样的）
jstat -class pid:显示加载class的数量，及所占空间等信息。
jstat -compiler pid:显示VM实时编译的数量等信息。
jstat -gc pid:可以显示gc的信息，查看gc的次数，及时间。其中最后五项，分别是young gc的次数，young gc的时间，full gc的次数，full gc的时间，gc的总时间。
jstat -gccapacity:可以显示，VM内存中三代（young,old,perm）对象的使用和占用大小，如：PGCMN显示的是最小perm的内存使用量，PGCMX显示的是perm的内存最大使用量，PGC是当前新生成的perm内存占用量，PC是但前perm内存占用量。其他的可以根据这个类推， OC是old内纯的占用量。
jstat -gcnew pid:new对象的信息。
jstat -gcnewcapacity pid:new对象的信息及其占用量。
jstat -gcold pid:old对象的信息。
jstat -gcoldcapacity pid:old对象的信息及其占用量。
jstat -gcpermcapacity pid: perm对象的信息及其占用量。
jstat -util pid:统计gc信息统计。
jstat -printcompilation pid:当前VM执行的信息。
除了以上一个参数外，还可以同时加上 两个数字，如：jstat -printcompilation 3024 250 6是每250毫秒打印一次，一共打印6次，还可以加上-h3每三行显示一下标题
3、使用jmap（jmap 是一个可以输出所有内存中对象的工具）导出对象文件。如对于java进程（pid=1234），可以这样：jmap -histo 1234 > a.log 将对象导出到文件，然后通过查看对象内存占用大小，返回去代码里面找问题。
（也可以使用命令，导出对象二进制内容（这样导出内容比jmap -histo多得多，更耗时，对jvm的消耗也更大）：jmap -dump:format=b,file=a.log 1234）

集合常见的面试题：

Q&A HashMap的组成?

jdk 1.7时是数组+链表组成，

jdk1.8时是 数组+ 链表 + 红黑树组成。 链表元素大于等于8时会把链表转为树结构，若桶中链的元素个数小于等于6时，树结构还原成链表。当链表的个数为8左右徘徊时就会生成树转链表，链表转树，效率低下。hasMap的负载因子默认为0.75，2^n是为了散列更加均匀。

Q&A HashMap的key为自定义的类应该怎么办?

如果key为自定义的类应该重写hashcode()和equals()方法。

Q&A HashMap为何线程不安全?

1.在JDK1.7中，当并发执行扩容操作时会造成环形链和数据丢失的情况。
2.在JDK1.8中，在并发执行put操作时会发生数据覆盖的情况。

Q&A HashMap如何保证线程安全?

使用Collections.synchronizedMap()包装一下就可以了，原理就是对所有的修改操作都加上synchronized，保证了线程的安全。

```
Map  map = Collections.synchronizedMap(new HashMap());
```

Q&A HashMap中的key可以为任意类型吗?

不能使用基本类型，HashMap中key是可以为null, 只能存储一个null, 因为计算key的hash值的时候，如果key为null， 则其hash值为0

之所以key不能为基本数据类型，则是因为基本数据类型不能调用其hashcode()方法和equals()方法，进行比较，所以HashMap集合的key只能为引用数据类型，不能为基本数据类型，可以使用基本数据类型的包装类，例如Integer Double等。

Q&A HashMap 和 HashTable 区别?

HashMap和HashTable都实现了Map接口，HashMap允许键和值是null而HashTable不允许键和值是null,HashTable是同步的，而HashMap不是，因此hashMap适用于单线程环境，而HashTable适用于多线程环境。HashMap提供了可供应用迭代的键的集合。

Q&A HashMap和ConCurrentHashMap区别?

hashMap线程不安全，put时在多线程的情况下会形成环而导致循环。

ConCurrentHashMap是线程安全的，采用分段机制，减少锁粒度。

ConCurrentHashMap是线程安全，在jdk1.7时采用Segment+HashEntry的方式进行实现 lock加上Segment上面。1.7 size计算是线采用不加锁的方式。连续计算元素的个数，最多计算3次。

1.8中取而代之是采用Node+CAS +Synchronized来保证并发安全，1.8实现使用一个volatile类型的变量baseCount记录元素的各少数。当插入新数据或删除新数据时，会通过addCount()方法更新baseCount，通过累计baseCount和CounterCell数组中的数量，即可得到元素的总个数。

Q&A ConcurrentHashMap 和 HashTable 区别?

ConcurrentHashMap

- 底层采用分段的数组+链表实现，线程**安全**
- 通过把整个Map分为N个Segment，可以提供相同的线程安全，但是效率提升N倍，默认提升16倍。(读操作不加锁，由于HashEntry的value变量是 volatile的，也能保证读取到最新的值。)
- Hashtable的synchronized是针对整张Hash表的，即每次锁住整张表让线程独占，ConcurrentHashMap允许多个修改操作并发进行，其关键在于使用了锁分离技术
- 有些方法需要跨段，比如size()和containsValue()，它们可能需要锁定整个表而而不仅仅是某个段，这需要按顺序锁定所有段，操作完毕后，又按顺序释放所有段的锁
- 扩容：段内扩容（段内元素超过该段对应Entry数组长度的75%触发扩容，不会对整个Map进行扩容），插入前检测需不需要扩容，有效避免无效扩容

HashTable

- 底层数组+链表实现，无论key还是value都**不能为null**，线程**安全**，实现线程安全的方式是在修改数据时锁住整个HashTable，效率低，ConcurrentHashMap做了相关优化
- 初始size为**11**，扩容：newsize = olesize*2+1
- 计算index的方法：index = (hash & 0x7FFFFFFF) % tab.length

两则的区别：

hashtable线程安全，采用的是线程同步得方法。

- ConcurrentHashMap提供了与Hashtable和SynchronizedMap不同的锁机制。Hashtable中采用的锁机制是一次锁住整个hash表，从而在同一时刻只能由一个线程对其进行操作；而ConcurrentHashMap中则是一次锁住一个桶。
- Java5提供了ConcurrentHashMap，它是HashTable的替代，比HashTable的扩展性更好。

Q&A Linkedhashmap 与 hashmap 的区别?

HashMap

- HashMap 是一个最常用的Map，它根据键的HashCode 值存储数据，根据键可以直接获取它的值，具有很快的访问速度。遍历时，取得数据的顺序是完全随机的。
- HashMap最多只允许一条记录的键为Null；允许多条记录的值为 Null。
- HashMap不支持线程的同步（即任一时刻可以有多个线程同时写HashMap），可能会导致数据的不一致。如果需要同步，可以用 Collections的synchronizedMap方法使HashMap具有同步的能力，或者使用ConcurrentHashMap。
- Hashtable与 HashMap类似，它继承自Dictionary类。不同的是：Hashtable不允许记录的键或者值为空；它支持线程的同步（即任一时刻只有一个线程能写Hashtable），因此也导致了Hashtable在写入时会比较慢。

LinkedHashMap

- **保存插入顺序**：LinkedHashMap保存了记录的插入顺序，在用Iterator遍历LinkedHashMap时，先得到的记录肯定是先插入的。也可以在构造时带参数，按照应用次数排序。
- **速度慢**：在遍历的时候会比HashMap慢，不过有种情况例外：当HashMap容量很大，实际数据较少时，遍历起来可能会比LinkedHashMap慢。因为LinkedHashMap的遍历速度只和实际数据有关，和容量无关，而HashMap的遍历速度和他的容量有关。

Q&A 高性能场景下HashMap性能优化?

1. 考虑初始化大小
2. 减小负载因子
3. String类型的key,不能用==判断，
4. 使用定制版的EnumMap
5. 使用IntObjectHashMap

Q&A hashmap 与 hashset 区别?

HashSet和HashMap之间有很多相似之处。对于HashSet而言，系统采用Hash算法决定集合元素的存储位置，这样可以保证快速存、取集合元素；对于HashMap而言，系统将value当成key的附属，系统根据Hash算法来决定key的存储位置，这样可以保证快速存、取集合key，而value总是紧随key存储。

HashSet的add()方法添加集合元素时实际上转变为调用HashMap的put()方法来添加 key-value对，当新放入 HashMap的Entry 中key 与集合中原有Entry的key 相同（hashCode()返回值相等，通过equals比较也返回true）时，新添加的Entry的value将覆盖原来Entry的value，但key不会有任何改变。因此，如果向HashSet中添加一个已经存在的元素，新添加的集合元素（底层由HashMap的key保存）不会覆盖已有的集合元素

Q&A ArrayList和Vector有何异同点?

相同点：

1. 两者都是基于索引的，都是基于数组的。
2. 两者都维护插入顺序，我们可以根据插入顺序来获取元素。
3. ArrayList 和 Vector 的迭代器实现都是 fail-fast 的。
4. ArrayList 和 Vector 两者允许 null 值，也可以使用索引值对元素进行随机访问。

不同点：

1. Vector 是同步，线程安全，而 ArrayList 非同步，线程不安全。对于 ArrayList，如果 迭代时改变列表，应该使用 CopyOnWriteArrayList。
2. 但是，ArrayList 比 Vector 要快，它因为有同步，不会过载。
3. 在使用上，ArrayList 更加通用，因为 Collections 工具类容易获取同步列表和只读列 表。ArrayList在并发add()可能出现下标越界异常。

Q&A ArrayList 与 LinkedList 区别？

ArrayList 和 LinkedList 都实现了 List 接口，他们有以下的不同点： 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)。 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。 LinkedList 比 ArrayList 更占内存，因为 LinkedList 为每一个节点存储了两个引用，一个指 向前一个元素，一个指向下一个元素。

Q&A 数组(Array)和列表(ArrayList)有什么区别？什么时候应该使用 Array 而不是 ArrayList?

array包含基本类型和对象类型。Arraylist只能包含对象类型。

Arraylist是采用数组实现的，arraylist是可以自动扩容的。比array提供了更多的特性，比如addAll(),removeAll() 等。

Q&A 使用ArrayList的迭代器会出现什么问题？单线程和多线程环境下；

常用的迭代器设计模式，iterator 方法返回一个父类实现的迭代器。 1、迭代器的 hasNext 方法的作用是判断当前位置是否是数组最后一个位置，相等为 false， 否则为 true。 2、迭代器 next 方法用于返回当前的元素，并把指针指向下一个元素，值得注意的是，每次 使用 next 方法的时候，都会判断创建迭代器获取的这个容器的计数器 modCount 是否与此 时 的 不 相 等 ， 不 相 等 说 明 集 合 的 大 小 被 修 改 过 ， 如 果 是 会 抛 出 ConcurrentModificationException 异常，如果相等调用 get 方法返回元素即可

泛型的面试题

Q&A Java中的泛型是什么？使用泛型的好处是什么?

在集合中存储对象并在使用前进行类型转换是多么的不方便。泛型防止了那种情况的发生。它提供了编译期的类型安全，确保你只能把正确类型的对象放入 集合中，避免了在运行时出现ClassCastException

Q&A Java的泛型是如何工作的？什么是类型擦除？

泛型是通过类型擦除来实现的，编译器在编译时擦除了所有类型相关的信息，所以在运行时不存在任何类型相关的信息。例如 `List<String>` 在运行时仅用一个List来表示。这样做的目的，是确保能和Java 5之前的版本开发二进制类库进行兼容。你无法在运行时访问到类型参数，因为编译器已经把泛型类型转换成了原始类型。

Q&A 什么是泛型中的限定通配符和非限定通配符？

限定通配符对类型进行了限制。有两种限定通配符，一种是`<? extends T>`它通过确保类型必须是T的子类来设定类型的上界，另一种是`<? super T>`它通过确保类型必须是T的父类来设定类型的下界。泛型类型必须用限定内的类型来进行初始化，否则会导致编译错误。另一方面`<?>`表 示了非限定通配符，因为`<?>`可以用任意类型来替代

Q&A List<? extends T> 上界 和List <? super T> 下界 之间有什么区别？

上界的list只能get不能add,下届的list只能add不能get

编译器可以支持像上转型，不支持像下转型。

Q&A 你可以把`List<String>`传递给一个接受`List<Object>`参数的方法吗?

因为`List<Object>`可以存储任何类型的对象包括String, Integer等等，而`List<String>`却只能用来存储Strings。

`List<Object> objectList`;

`List<String> stringList`;

objectList = stringList; //compilation error incompatible types

```
public static void main(String[] args) {
        List<String> stringList = new ArrayList<>();
        List<Object> objectList = new ArrayList<>();
        stringList.add("add");
        stringList.add("123");
        objectList.add("123");
        objectList.add("234");
        // 让objectList转为stringList,编译错误stringList必须是接收的List<String>
List<Object>之间不能转换。
        // stringList= objectList; // 编译错误
        // objectList=stringList; // 编译错误
        List<?> list = new ArrayList<>();
        list = stringList;
        System.out.println(list);
        list = objectList;
        System.out.println(list);
        // list.add("sss"); // 编译器不允许这样使用

    }
```

异常常见的面试题：

Q&A java中用来抛出异常的关键字是什么?

throw

Q&A 异常和Error（错误）的区别?

error：是不可捕捉到的，无法采取任何恢复的操作，顶多只能显示错误信息。
Exception ： 表示可恢复的例外，这是可捕捉到的

Q&A 什么是异常?

所谓异常是指程序在运行过程中发生的一些不正常事件。 （如：除0溢出，数组下标越界，所读取的文件不存在）

Q&A 什么类是所有异常类的父类

Throwable类

Q&Ajava 虚拟机能自动处理的异常是什么?

运行异常

Q&A Try-catch-finally的执行过程

1. try{}语句块中放的是要检测的java代码，可能有会抛出异常，也可能会正常执行
2. catch（异常类型）{}块是当java运行时系统接收到try块中所抛出异常对象时，会寻找处理这一异常catch块来进行处理（可以有多个catch块）
3. finally{}不管系统有没有抛出异常都会去执行，一般用来释放资源。除了在之前执行了System.exit（0）

Q&A 常见的异常？你的理解。

常见异常：RuntimeException,IOException,SQLException,ClassNotFoundException

Q&A final, finally, finalize的区别。

1. final用于声明属性，方法和类，分别表示属性不可交变，方法不可覆盖，类不可继承。
2. finally是异常处理语句结构的一部分，表示总是执行。
3. finalize是Object类的一个方法，在垃圾收集器执行的时候会调用被回收对象的此方法，供垃圾收集时的其他资源回收，例如关闭文件等。（**在垃圾回收的时候会调用被回收对象的此方法。**）

Q&A Java 中的异常处理机制的简单原理和应用。

当JAVA程序违反了JAVA的语义规则时，JAVA虚拟机就会将发生的错误表示为一个异常。违反语义规则包括2种情况。一种是JAVA类库内置的语义检查。例如数组下标越界,会引发IndexOutOfBoundsException;访问null的对象时会引发NullPointerException。另一种情况就是JAVA允许程序员扩展这种语义检查，程序员可以创建自己的异常，并自由选择在何时用throw关键字引发异常。所有的异常都是java.lang.Thowable的子类。

Q&A 运行时异常与一般异常有何异同?

Java提供了两类主要的异常:运行时异常runtime exception和一般异常checked exception。checked 异常。对于后者这种异常，JAVA要求程序员对其进行catch。所以，面对这种异常不管我们是否愿意，只能自己去写一大堆catch块去处理可能的异常。

运行时异常我们可以不处理。这样的异常由虚拟机接管。出现运行时异常后，系统会把异常一直往上层抛，一直遇到处理代码。如果不对运行时异常进行处理，那么出现运行时异常之后，要么是线程中止，要么是主程序终止。

Q&A 你平时在项目中是怎样对异常进行处理的。

1. 尽量避免出现runtimeException 。例如对于可能出现空指针的代码，带使用对象之前一定要判断一下该对象是否为空，必要的时候对runtimeException也进行try catch处理。
2. 进行try catch处理的时候要在catch代码块中对异常信息进行记录，通过调用异常类的相关方法获取到异常的相关信息，返回到web端，不仅要给用户良好的用户体验，也要能帮助程序员良好的定位异常出现的位置及原因。

线程的面试题：

Q&A 1.什么是线程?

线程是程序执行运算的最小的基本单位

Q&A线程与进程的区别?

- 进程是系统中正在运行的一个应用程序，表示资源分配的基本单位。又是调度运行的基本单位。

- 一个线程只能属于一个进程，而一个进程可以拥有多个线程。
- 同一进程的所有线程共享该进程的所有资源。同一进程的多个线程共享代码段。

Q&A 如何保证线程安全?

加锁是最简单的直接的方式。synchronized关键字

Q&A 如何使用线程？线程是如何启动的?

- 实现runnable接口，
- 实现Callable接口，
- 继承Thread类，
- 线程启动，重写run方法然后调用start()即可开启一个线程。

Q&A 线程的几种状态?

5种状态：创建，就绪，运行状态，阻塞，死亡

线程创建时new状态，调用start()进入runnable就绪状态，争夺cpu资源后进入running状态，由于某种原因进入

1. 等待阻塞：运行的线程会释放占用的所有资源，jvm会把该线程放入“等待池”进入这个状态后，是不能自动唤醒的，必须依靠其他线程调用notify()或notifyAll()方法才能被唤醒，
2. 同步阻塞：运行的线程在获取对象的同步锁时，若该同步锁被别的线程占用，则JVM会把该线程放入“锁池”中。
3. 其他阻塞：运行的线程执行sleep()或join()方法，或者发出了I/O请求时，JVM会把该线程置为阻塞状态。
   当sleep()状态超时、join()等待线程终止或者超时、或者I/O处理完毕时，线程重新转入就绪状态。
   当线程正常执行结束会进入dead状态（一个未捕获的异常也会使线程终止）

- yield()只是使当前线程重新回到runnable状态
- sleep()会让出cpu，不会释放锁
- join()会让出cpu，释放锁
- wait() 和 notify() 方法与suspend()和 resume()的区别在于wait会释放锁，suspend不会释放锁
- wait() 和 notify()只能运行在Synchronized代码块中，因为wait()需要释放锁，如果不在同步代码块中，就无锁可以释放
- 当线程调用wait()方法后会进入等待队列（进入这个状态会释放所占有的所有资源，与阻塞状态不同），进入这个状态后，是不能自动唤醒的，必须依靠其他线程调用notify()或notifyAll()方法才能被唤醒

Q&A 并发和并行的区别?

并发：同一时段，多个任务都在执行，

并行：单位时间内多个任务同时执行。

Q&A 使用多线程带来什么问题可能?

并发是为了能提高程序的执行效率提高程序运行速度，但是并发编程并不总是能提高程序运行速度的，而且并发编程可能会遇到很多问题，比如：内存泄漏、上下文切换、死锁

Q&A 什么是死锁？如何避免死锁?

死锁:死锁是指两个或两个以上的进程在执行过程中，由于竞争资源或者由于彼此通信而造成的一种阻塞的现象，若无外力作用，它们都将无法推进下去。此时称系统处于死锁状态或系统产生了死锁，这些永远在互相等待的进程称为死锁进程

破坏死锁的四个条件:

1. 互斥条件：该资源任意一个时刻只由一个线程占用。（无法破坏）
2. 请求与保持条件：一个进程因请求资源而阻塞时，对已获得的资源保持不放。（可以使用一次性申请所有资源）
3. 不剥夺条件:线程已获得的资源在未使用完之前不能被其他线程强行剥夺，只有自己使用完毕后才释放资源。（占用部分资源线程去申请其他资源，如果不能申请到就主动释放它占有的资源）
4. 循环等待条件:若干进程之间形成一种头尾相接的循环等待资源关系。（按顺序申请资源。）

Q&A sleep（）和wait()方法的区别?

- sleep方法会让出cpu没有释放锁，wait方法释放了锁。
- 两者都可以暂停线程的执行。
- Wait 通常被用于线程间交互/通信，sleep 通常被用于暂停执行。
- wait() 方法被调用后，线程不会自动苏醒，需要别的线程调用同一个对象上的 notify() 或者 notifyAll() 方法。sleep() 方法执行完成后，线程会自动苏醒。或者可以使用wait(long timeout)超时后线程会自动苏醒。

Q&A 介绍一下Syncronized锁，

synchronized修饰静态方法以及同步代码块的Synchronized用法锁的是类，线程想要执行对应的同步代码就需要或的类锁。synchronized修饰成员方法，线程获取的是调用当前对象实例的对象锁。

Q&A 介绍一下syncronized的锁的升级过程

无锁->偏向锁(jdk15取消了)->轻量锁->重量锁

Q&A 介绍一下Synchronized和lock，

synchronized是java的关键字，当用来修饰一个方法或者代码块的时候，能够保证在同一时刻最多只有一个线程执行该代码。jdk1.5后引入自旋锁，锁粗化，轻量级锁，偏向锁来优化关键字的性能。

Lock是一个接口，Synchronized发生异常时自动释放线程占有的锁，因此不会导致死锁的现象。Lock发生异常时需要通过unLock()去释放锁，则需要在使用finally块中释放锁，Lock可以让等待锁的线程响应中断，而synchronized却不行，synchronized时等待的线程会一直等待。Lock可以知道是否成功获取锁，而synchronized却无法办到。

Q&A 介绍一下volatile

1. volatile修饰的是保障有序性和可见性，比如我们写的代码不一定会按照我们书写的顺序来执行。
2. volatile是Java提供的轻量级的同步机制，比sync的开销要小
3. 被volatile定义的变量,系统每次用到它时都是直接从主存中读取,而不是各个线程的工作内存
4. volatile可以像sync一样保持变量在多线程环境中是实时可见的

可见性：

每个线程都有自己的工作内存，每次线程执行时，会从主存获得变量的拷贝，对变量的操作是在线程的工作内存中进行，不同的线程之间不共享工作内存；对于volatile（sync，final）来说，打破了上述的规则，当线程修改了变量的值，其他线程可以立即知道该变量的改变。而对于普通变量，当一个线程修改了变量，需要将变量写回主存，其他线程从主存中读取变量后才对该线程可见

volatile具有sync的可见性，但是不具备原子性（解决java多线程的执行有序性）。volatile适用于多个变量之间或者某个变量当前值和修改之后值之间没有约束。因此，单独使用volatile还不足以实现计数器，互斥锁等。

在并发编程中谈及到的无非是可见性、有序性及原子性。而这里的Volatile只能够保证前两个性质，对于原子性还是不能保证的，只能通过锁的形式帮助他去解决原子性操作

Q&A ThreadLocal的如何避免内存泄露

Q&A Java中的CAS unsafe方法 CAS的缺陷

Spring全家桶

Spring

Q&A 什么是spring

Q&A Spring包含哪些模块

Q&A 使用spring框架能带来哪些好处

Q&A Spring有几种配置方式

Q&A IOC是什么

Q&A IOC容器初始化过程

Q&A 依赖注入的方式，以及各个之间的区别

Q&A 依赖注入的过程

Q&A 谈你经常用的Spring依赖注入的注解

Q&A Spring 是如何注入集合的

Q&A Spring 是如何注入java properties的

Q&A Spring中注入null和空串可以吗

Q&A Bean的生命周期

Q&A Bean的创建过程

Q&A Bean的作用域的理解

Q&A inner Beans 的理解

Q&A Spring中的单例bean是线程安全的吗

Q&A Spring bean的自动装配的理解

Q&A Spring 有哪几种自动装配模式

Q&A 如何基于注解开启自动装配

Q&A @Autowired和@Resource的区别,@Qualifier

Q&A @Required注解的理解

Q&A BeanFactory ,FactoryBean，ApplicationContext的区别

Q&A Spring中有哪些不同类型的事件

Q&A FileSystemResource和ClassPathResouce的区别

Q&A spring中有哪些设计模式

Q&A 什么是AOP

Q&A AOP的应用场景

Q&A Spring AOP和 AspectJAOP的区别

Q&A Spring AOP中的代理

Q&A Aspect,joinPoint,Advice的理解

Q&A AOP的实现原理

Q&A Spring支持的事务管理器都有哪些

Q&A Spring的事务管理器你更倾向于哪些

Q&A Spring的ORM的理解

Spring MVC

Q&A Spring MVC的组件包涵哪些

Q&A Spring MVC的处理过程

Q&A @Controller,@RequestMapping，@RestController

Q&A Spring MVC的控制器是什么

Q&A WebApplicationContext的理解

Q&A DispatcherServlet的理解

Spring Boot

Q&A 为什么使用springboot

Q&A springboot的核心注解

Q&A spring boot自动配置的原理

Q&A Spring boot的stater,starter-parent

Q&A Spring boot的jar包与普通的jar包的区别

Q&A Spring boot启动顺序

Spring Cloud

Q&A 微服务的理解

Q&A 微服务架构的优势，特点

Q&A 微服务于单体 SOA的区别

Q&A spring Cloud中的Rest的理解

Q&A 服务注册与发现的组件

Q&A 服务注册与发现的原理

Q&A Eureka Server的自我保护机制

Q&A Eureka Client的理解

Q&A Feign的理解

Q&A Hystrix,Hystrix的容错的方式

Q&A Eurake和zookeeper的区别

Q&A 服务雪崩，服务降级，服务容错

JVM

JUC