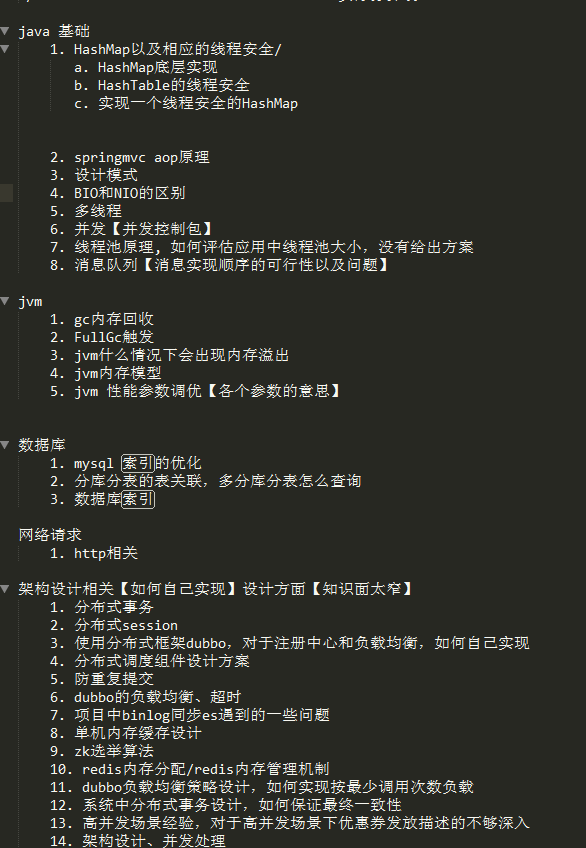
# Java面试题集



## 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>

### 9. 抽象类和接口的区别，类可以继承多个类么，接口可以继承多个接口么,类可以实现多个接口么。

类不能继承多个类，接口不能继承多个接口，类可以实现多个接口。

**抽象类和接口的区别：**

1、抽象类和接口都不能直接实例化，如果要实例化，抽象类变量必须指向实现所有抽象方法的子类对象，接口变量必须指向实现所有接口方法的类对象。

2、抽象类要被子类继承，接口要被类实现。

3、接口只能做方法申明，抽象类中可以做方法申明，也可以做方法实现

4、接口里定义的变量只能是公共的静态的常量，抽象类中的变量是普通变量。

5、抽象类里的抽象方法必须全部被子类所实现，如果子类不能全部实现父类抽象方法，那么该子类只能是抽象类。同样，一个实现接口的时候，如不能全部实现接口方法，那么该类也只能为抽象类。

6、抽象方法只能申明，不能实现。abstract void abc();不能写成abstract void abc(){}。

7、抽象类里可以没有抽象方法

8、如果一个类里有抽象方法，那么这个类只能是抽象类

9、抽象方法要被实现，所以不能是静态的，也不能是私有的。

10、接口可继承接口，并可多继承接口，但类只能单根继承。

## JVM知识

参考：https://blog.csdn.net/u012257955/article/details/70890702

### 什么情况下会发生栈内存溢出。

线程在请求栈深度的时候超过了虚拟机设置的最大栈深度，就会抛出StackOverflowError异常；栈向虚拟机无法申请到更多内存，就会抛出OutOfMemoryError异常。

### 2. JVM的内存结构，Eden和Survivor比例

Jvm内存结构分为**堆内存**，**方法区**，**程序计数器**，**本地方法区**，和**虚拟机栈**。

**堆内存** 是java虚拟机所管理的最大一块内存，被所有线程共享，主要存储对象实例和数组等。

**方法区** 线程共享内存区域，存储被虚拟机加载的类信息，常量，静态变量等数据。

**程序计数器** 标记线程执行字节码的行数，线程独享。

**本地方区** 和虚拟机栈类似区别是本方法栈执行native方法，虚拟机栈执行java方法。

**虚拟机栈** 虚拟机栈是线程私有的，每个方法在执行的同时，都会创建一个栈桢，用于存储局部变量表，操作数栈，动态连接和方法出口等信息的。一个方法从调用到返回的过程就是栈桢入栈到出栈的过程；

Eden 和Survivor 的比例：8:1 两块survivor （To/From） Old

### JVM内存为什么要分成新生代，老年代，持久代。新生代中为什么要分为Eden和Survivor。

参考：https://blog.csdn.net/u011445156/article/details/73441618

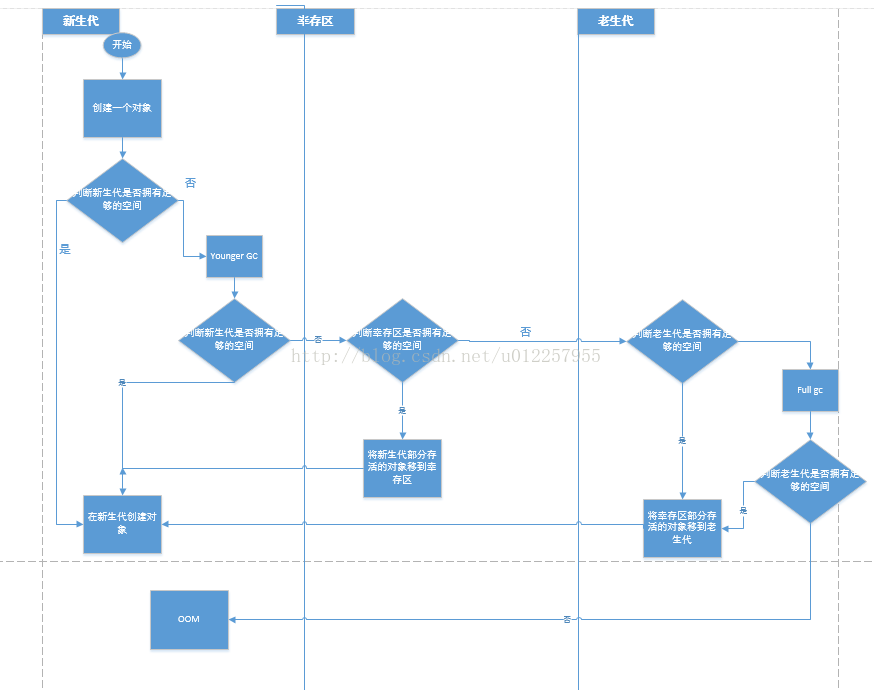
提高GC的效率，因为不是所有的对象都需要频繁的去GC，频繁GC很耗时的。

新生代的GC频繁，老年代相对低，持久代不进行GC。

**为什么新生代分Eden和Survivor** 如果没有分Survivor，Eden区每进行一次minor GC，存活的对象就会被送到老年代。老年代很快就会被填满，老年代一旦填满就会触发Full GC，频繁的FullGC消耗的时间是非常可观的，这一点会影响大型程序的执行速度和响应速度。

**为什么设置两个Survivor区** 整理内存碎片化，保证有一块空的survivor用来整理连续的内存空间。

### JVM 中一次完整的 GC 流程是怎样的，对象如何晋升到老年代，说说你知道的几种主要的 JVM 参数。



GC流程图

对象晋升老生代一共有三个可能：

1.当对象达到成年，经历过15次GC（默认15次，可配置），对象就晋升为老生代

2.大的对象会直接在老生代创建

3.新生代跟幸存区内存不足时，对象可能晋升到老生代

jvm参数：

-Xms 初始化堆大小

-Xmx 堆最大内存

-Xss 栈内存

-XX：PermSize 初始化永久带内存

-XX：MaxPermSize最大永久带内存

### 5. 你知道哪几种垃圾收集器，各自的优缺点，重点讲下 cms 和 G1，包括原理，流程，优缺点。

串行垃圾收集器：收集时间长，停顿时间久

并发垃圾收集器：碎片空间多

CMS：并发标记清除。主要步骤有：初始收集，并发标记，重新标记，并发清除（删除），重置。

G1：主要步骤：初始标记，并发标记，重新标记，复制清除（整理）。

CMS的缺点是对cpu的要求比较高。G1是将内存化成了多块，所有对内段的大小有很大的要求。

CMS是清除，所以会存在很大的内存碎片。G1是整理，所以碎片空间较小。

CMS 初始收集 -> 并发标记 -> 重新标记 -> 并发清除 -> 重置。

G1 初始标记 -> 并发标记 -> 重新标记 -> 复制清除。

### 6. 垃圾回收算法的实现原理。

常用的垃圾回收算法有两种： **引用计数**和**可达性分析**

引用计数是增加一个字段来标识当前的引用次数，引用计数为0的就是可以GC的。但是引用计数不能解决循环引用的问题  
可达性分析：就是通过一系列GC ROOT的对象作为起点，向下搜索，搜索所有没有与当前对象GC ROOT 有引用关系的对象。这些对象就是可以GC的。

### 7. 当出现了内存溢出，你怎么排错。

1.首先控制台查看错误日志

2.然后使用jdk自带的jvisualvm工具查看系统的堆栈日志

3.定位出内存溢出的空间：堆，栈还是永久代（jdk8以后不会出现永久代的内存溢出）。

4.如果是堆内存溢出，看是否创建了超大的对象

5.如果是栈内存溢出，看是否创建了超大的对象，或者产生了死循环。

### 8. JVM 内存模型的相关知识了解多少，比如重排序，内存屏障，happen-before，主内存， 工作内存等。

重排序：jvm虚拟机允许在不影响代码最终结果的情况下，可以乱序执行。

内存屏障：可以阻挡编译器的优化，也可以阻挡处理器的优化

happens-before原则：

1：一个线程的A操作总是在B之前，那多线程的A操作肯定实在B之前。  
2：monitor 再加锁的情况下，持有锁的肯定先执行。  
3：volatile修饰的情况下，写先于读发生  
4：线程启动在一起之前 strat  
5：线程死亡在一切之后 end  
6：线程操作在一切线程中断之前  
7：一个对象构造函数的结束都该对象的finalizer的开始之前  
8：传递性，如果A肯定在B之前，B肯定在C之前，那A肯定是在C之前。

主内存：所有线程共享的内存空间

工作内存：每个线程特有的内存空间

### 9. 简单说说你了解的类加载器，可以打破双亲委派么，怎么打破。

Bootstra ClassLoader（引导类加载器）

Extension ClassLoader（扩展类加载器）

Application ClassLoader（系统加载器）

Customer ClassLoader（自定义类加载器）

双亲委派：双亲委派模式会保证父类加载器先加载类。

### 10. 讲讲 JAVA 的反射机制。

在运行期间，对于任意一个类，都能知道当前类的属性和方法，对于一个对象，都能调用类的方法和属性，这种动态加载机制就是Java反射机制。

### 11. 你们线上应用的 JVM 参数有哪些。

-XX:PermSize=128M  
-XX:MaxPermSize=512m  
-XX:PermSize=128M  
-XX:MaxPermSize=512m  
-Xms512m  
-Xmx1024m  
-XX:PermSize=640m  
-XX:MaxPermSize=1280m  
-XX:NewSize=64m  
-XX:MaxNewSize=256m  
-verbose:gc  
-XX:+PrintGCDetails  
-XX:+PrintGCTimeStamps

### 12. g1 和 cms 区别,吞吐量优先和响应优先的垃圾收集器选择。

Cms是以获取最短回收停顿时间为目标的收集器。基于标记-清除算法实现。比较占用cpu资源，切易造成碎片。

G1是面向服务端的垃圾收集器，是jdk9默认的收集器，基于标记-整理算法实现。可利用多核、多cpu，保留分代，实现可预测停顿，可控。

### 13. 怎么打出线程栈信息。

### 14. 请解释如下 jvm 参数的含义：

-server -Xms512m -Xmx512m -Xss1024K -XX:PermSize=256m -XX:MaxPermSize=512m -XX:MaxTenuringThreshold=20 XX:CMSInitiatingOccupancyFraction=80 -XX:+UseCMSInitiatingOccupancyOnly。

## 操作系统

### 1. Linux 系统下你关注过哪些内核参数，说说你知道的。

### Linux 下 IO 模型有几种，各自的含义是什么。

**同步/异步 和 阻塞/非阻塞**

**同步和异步（消息的通知机制）**

同步sync：发出一个函数调用时，会一直等待，直到方法返回才继续执行后续操作。（几乎Java的使用方法都是同步调用的）

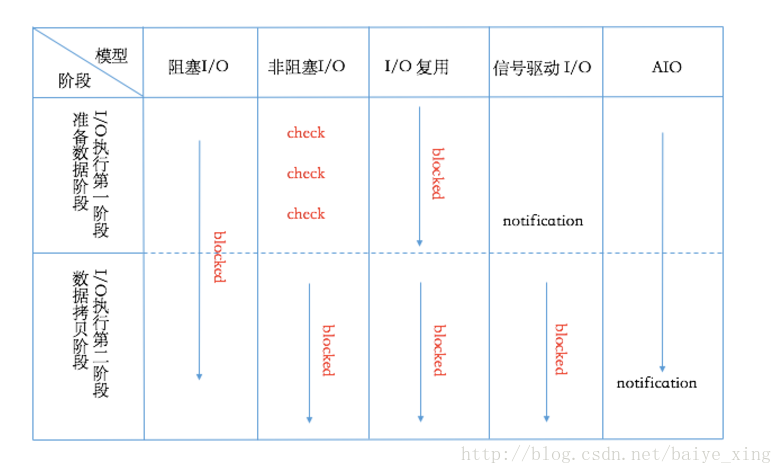
异步Async：发出一个函数调用后，调用者可以在没得到结果之前继续后续操作。当这个调用结束后，一般通过状态、通知或者回调来通知调用者。对于异步来说，调用的返回并不受调用者控制。

**阻塞和非阻塞（程序等待调用结果时的状态）**

阻塞：就是调用结果返回之前，该执行的线程会被挂起，不释放CPU执行权，线程不做其他事情，等待调用结果的返回，返回后才能继续执行。

非阻塞：就是在没有获取调用结果之前，不是一直等待，线程可以往下执行，如果是同步的，通过轮询的方式检查有没有调用结果返回，如果是异步的，就会通知回调。

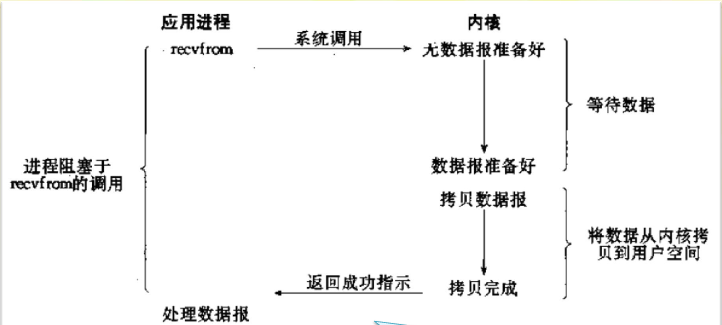
**五种IO模型（）**

****

**（1）阻塞式IO（BIO）**

简介：进程会一直阻塞，直到数据拷贝完成。

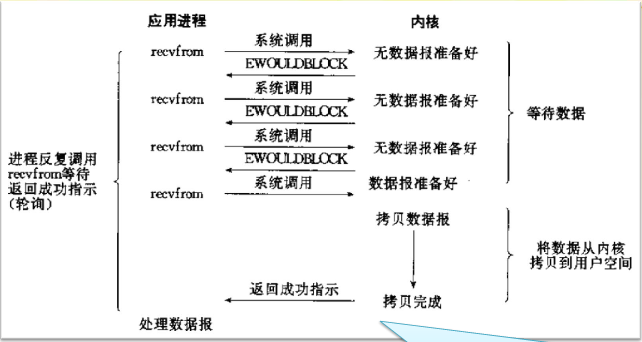
应用程序调用一个IO函数，导致应用程序阻塞，等待数据准备好。如果数据准备好了，从内核拷贝到用户空间。执行完毕后，IO函数会向应用程序返回成功响应，应用程序得到响应后，就不在阻塞，进行后续工作。



**（2）非阻塞式IO（NIO）**

简介：非阻塞IO通过进程反复调用IO函数（多次系统调用，并马上返回）；在数据拷贝过程中，进程是阻塞的。

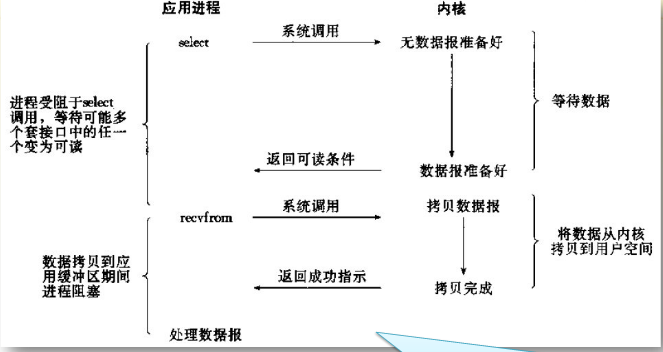
应用程序调用一个IO函数，这个IO操作会从内核中立刻返回。但是这个IO函数具体要执行的事情（写数据）可能还没完成。而对于应用程序，虽然这个IO很快就返回了，但是它并不知道这个IO操作是否真的成功了，为了知道IO操作是否成功，一般有两种策略，一是要应用程序主动轮询，知道数据准备好为止，在这个不断询问的过程中，会大量占用CPU的时间。二是采用IO通知机制，比如IO多路复用或者信号驱动IO。



**（3）IO多路复用模型**

简介：比阻塞IO并没有什么优越性，关键是能实现同时对多个IO端口进行监听。

IO复用模型会用到select、poll、epoll函数，这几个函数也会使进程阻塞，但是和阻塞IO不同的，这两个函数可以同时阻塞多个IO操作，而且可以同时对多个读操作，多个写操作的IO函数进行检测、直到有数据可读或者可写时，才真正调用IO操作函数。

****

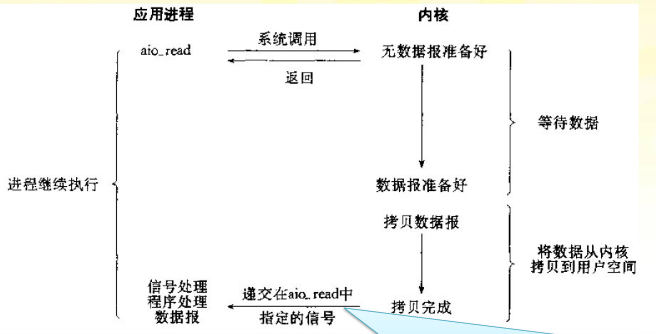
**（4）信号驱动IO模型（signal-driver IO）**

进程预先告知内核，使得当某个Socketfd有events事件发生时，内核使用信号通知相关进程。

**（5）异步IO模型**

简介：告知内核启动某个操作，并让内核在整个操作完成后通知我们；数据拷贝的时候进程无需要阻塞。

当一个异步过程调用发出后，调用者不能立即得到结果，实际处理这个调用函数完成后，通过状态、通知和回调者的输入输出操作。



### 3. epoll 和 poll 有什么区别。

Poll在每次调用都需要拷贝一次fd\_set（或者pollfd），在描述数量大时开销较大。内核需要遍历传入的fd\_set，效率较低。

Epoll主要通过epoll-ctl使每次注册新事件时拷贝一次所有的fd；epoll\_wait只需要轮询一个就绪队列是否为空，这是因为epoll采用了回调机制，将就绪的fd插入到就绪队列，因此不像select和poll中，需要轮询一个fd\_set（很耗CPU）。

### 4. 平时用到哪些 Linux 命令。

Ls ，find，tar，tail，cp，rm，mv，vi，vim，grep，ps，kill等

### 5. 用一行命令查看文件的最后五行。

Tail -n 5 filename

### 6. 用一行命令输出正在运行的 java 进程。

ps -ef|grep java

### 7. 介绍下你理解的操作系统中线程切换过程。

控制权的转换，根据优先级切换上下文（用户，寄存器，系统）   
<http://www.cnblogs.com/kkshaq/p/4544426.html>

### 8. 进程和线程的区别。

进程：是程序运行的实例，是系统进行资源分配和调度的一个独立单位，它包括独立的地址空间，资源以及1个或多个线程。

线程：cpu调度和分配的最小单位，一个轻量级的进程。

### 9. top 命令之后有哪些内容，有什么作用。

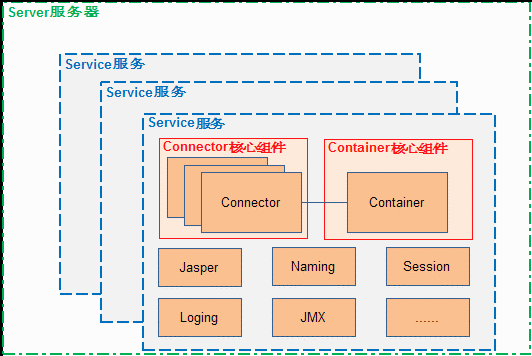
top命令是Linux下常用的性能分析工具，能够实时显示系统中各个进程的资源占用状况，类似于Windows的任务管理器。

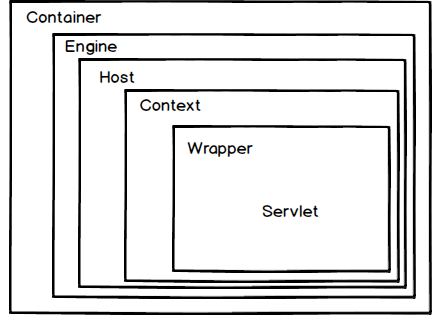
### 10. 线上 CPU 爆高，请问你如何找到问题所在

## 四、开源框架知识

### 1. 简单讲讲tomcat结构，以及其类加载器流程，线程模型等。

#### （1）Tomcat的结构







Tomcat的心脏是两个个组件：Connector和Container。一个Container可以选择多个Connector，多个Connector和一个Container就形成了一个Service。Service可以对外提供服务，而Service服务器控制整个Tomcat的生命周期。

**Tomcat结构：**

Server—多个service

Container：engine🡪host🡪context—>Wrapper

Connector

Logging、Naming、Session、JMX等等。

**Engine 容器**

Engine 容器比较简单，它只定义了一些基本的关联关系

**Host 容器**

Host 是 Engine 的子容器，一个 Host 在 Engine 中代表一个虚拟主机，这个虚拟主机的作用就是运行多个应用，它负责安装和展开这些应用，并且标识这个应用以便能够区分它们。它的子容器通常是 Context，它除了关联子容器外，还有就是保存一个主机应该有的信息。

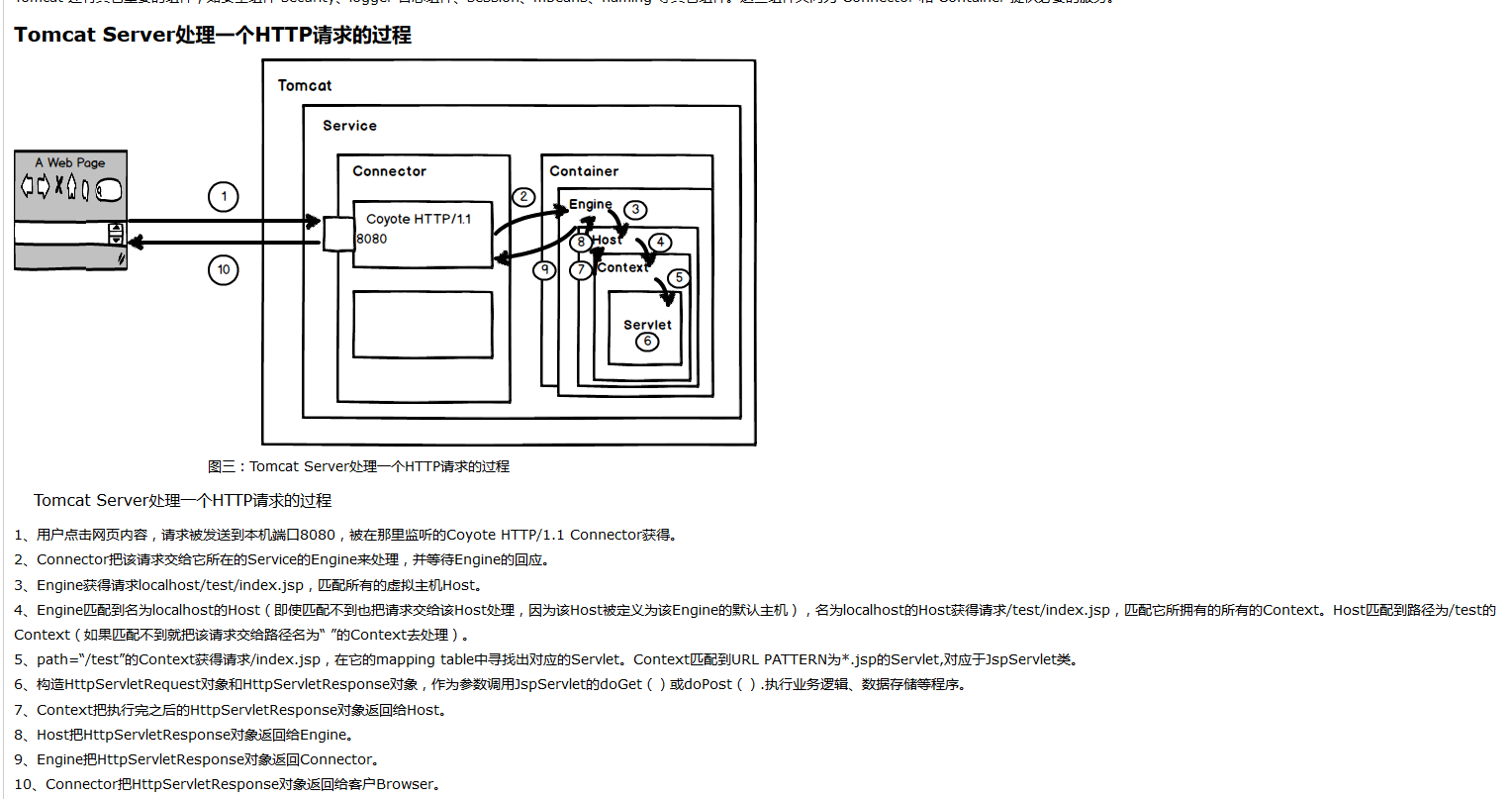
**Context 容器**

Context 代表 Servlet 的 Context，它具备了 Servlet 运行的基本环境，理论上只要有 Context 就能运行 Servlet 了。简单的 Tomcat 可以没有 Engine 和 Host。Context 最重要的功能就是管理它里面的 Servlet 实例，Servlet 实例在 Context 中是以 Wrapper 出现的，还有一点就是 Context 如何才能找到正确的 Servlet 来执行它呢？ Tomcat5 以前是通过一个 Mapper 类来管理的，Tomcat5 以后这个功能被移到了 request 中，在前面的时序图中就可以发现获取子容器都是通过 request 来分配的。

**Wrapper 容器**

Wrapper 代表一个 Servlet，它负责管理一个 Servlet，包括的 Servlet 的装载、初始化、执行以及资源回收。Wrapper 是最底层的容器，它没有子容器了，所以调用它的 addChild 将会报错。

Wrapper 的实现类是 StandardWrapper，StandardWrapper 还实现了拥有一个 Servlet 初始化信息的 ServletConfig，由此看出 StandardWrapper 将直接和 Servlet 的各种信息打交道。



<https://www.cnblogs.com/zhouyuqin/p/5143121.html>

#### （2）Tomcat类加载流程



Tomcat类加载流程：

BootStrap启动类加载

System系统类加载：通常使用CLASSPATH环境变量来初始化。

Common：通用类加载：

Webapp应用类加载：

Bootstrap启动类加载JAVA\_HOME\jre\bin\下的jar包中的类，system系统类的加载，CATALINA\_HOME\conf配置和属性加载。。Common通用类加载CATALINA\_HOME\bin下jar包中的类加载，最后webapp应用类加载，我们自己的项目class文件加载。

#### （3）Tomcat线程模型 BIO 、NIO、AIO

阻塞式IO模型，Tomcat6，Tomcat7支持的线程模型

非阻塞式IO模型，Tomcat7，和Tomcat8支持的线程模型。Tomcat7默认是BIO需要修改配置才能支持NIO。

异步非阻塞IO，Tomcat8支持。

### 2. tomcat 如何调优，涉及哪些参数。

硬件，操作系统，Tomcat版本选择，JDK选择，配置JVM参数，配置connector的线程数量，开启gzip压缩，trimSpaces集群等。

修改Tomcat启动参数

-Xms1024m（JVM堆的初始化大小）

-Xmx1024m（JVM堆的最多能申请的内存）

-Xss512k（设置每个线程的堆栈大小）

-XX:PremSize=16m -XX:MaxPremSize=16m（设置永久代内存初始值和最大值）

-XX:MaxTenuringThreshold=0 （设置垃圾最大年龄）

### 3. 讲讲 Spring 加载流程。

通过listener入口，核心是在AbstractApplicationContext的refresh方法，在此处进行装载bean工厂，bean，创建bean实例，拦截器，后置处理器等。

### 4.Spring AOP 的实现原理。

SpringAOP有两种实现方式：

基于JDK的接口动态代理（Dynamic Proxy）和基于子类化的CGLIB代理。

在使用springAOP时，一般不需要选择具体的实现方式，springAOP能根据上下文环境帮助我们选择一种合适的实现。

参考：<https://www.cnblogs.com/CHENJIAO120/p/7080790.html>

### 5.讲讲 Spring 事务的传播属性。

七种传播属性

事务传播行为

所谓事务的传播行为是指，如果在开始当前之前，一个事务上下文已经存在，此时有若干选项可以指定一个事务方法的执行行为。在TransactionDefinition定义中包括了如下几个表示传播行为的常量：

**Transaction.PROPAGATION\_REQUIRED**：如果当前存在事务，则加入该事务；如果当前没有事务，则创建一个新的事务。

**Transaction.PROPAGATION\_REQUIRED\_NEW**：创建一个新的事务，如果当前存在事务，则把当前事务挂起。

**Transaction.RPOPAGATION\_SUPPORTS**：如果当前存在事务，则加入该事务，如果当前没有事务，则以非事务的方式继续运行。

**Transaction.PROPAGATION\_NOT\_SUPPORTS**：以非事务方式运行，如果当前存在事务，则把当前事务挂起。

**Transaction.PROPAGATION\_NEVER**：以非事务的方式运行，如果当前存在事务，则抛出异常。

**Transaction.PROPAGATION\_MANDATORY**：如果当前存在事务，则加入该事务；如果当前没有事务，则抛出异常。

**Transaction.PROPAGATION\_NESTED**：如果当前存在事务，则创建一个事务做为当前事务的嵌套事务来运行；如果当前没有事务，则创建一个新事务。

### 6. Spring 如何管理事务的。

编程式和声明式

编程式事务，每次事务都需要单独实现，业务量大复杂时，编程式事务时很繁杂的。

声明式事务，在xml文件中配置或使用注解方式声明事务。声明式事务式无侵入的，不会影响业务代码的。

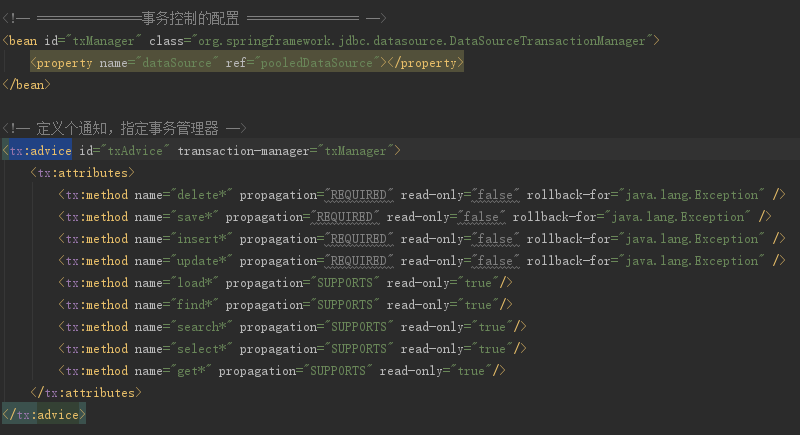
参考：<https://blog.csdn.net/liaohaojian/article/details/70139151>

### 7. Spring 怎么配置事务（具体说出一些关键的 xml 元素）。

<tx:advice>

<tx></tx>

</tx:advice>



### 8. 说说你对 Spring的理解，非单例注入的原理？它的生命周期？循环注入的原理，aop 的实现原理，说说 aop 中的几个术语，它们是怎么相互工作的。

#### （1），对spring的理解。

我的理解就是，spring是一个轻量级的框架，它使用IOC控制反转，将类的创建和依赖关系写在配置文件里面，由配置文件注入，实现了松耦合，其核心就是**beanFactory**，用以构件我们需要的bean。Spring的另一个厉害之处就是AOP，将安全，事务，日志等程序逻辑相对独立的功能抽取出来，利用spring的配置文件将这些功能插入到需要的业务代码中，实现了按照方面编程，提高了代码复用性。

#### （2），非单利注入的原理，生命周期，循环注入的原理。

#### （3），Spring AOP的实现原理

SpringAOP有两种实现方式：

基于JDK的接口动态代理（Dynamic Proxy）和基于子类化的CGLIB代理。

在使用springAOP时，一般不需要选择具体的实现方式，springAOP能根据上下文环境帮助我们选择一种合适的实现。

参考：<https://www.cnblogs.com/CHENJIAO120/p/7080790.html>

#### （4），AOP中的术语，它们是怎么相互工作的。

切面（Aspect）：

连接点（Joinpoint）：

通知（Advice）：

切入点（PointCut）：

引入（Introduction）：

目标对象（Target Object）：

Aop代理（AOP Proxy）：

织入（Weaving）：

### 9. SpringMVC 中 DispatcherServlet 初始化过程。

入口是web.xml中配置的DispatcherServlet，DispatcherServlet继承了HttpServlrtBean，FrameWorkServlet，通过其中的init方法进行初始化装配bean和实例，initServletBean是实际完成上下文工作和bean初始化的方法。

### 10. netty 的线程模型，netty 如何基于 reactor 模型上实现的。

### 11. 为什么选择 netty。

### 12. 什么是 TCP 粘包，拆包。解决方式是什么。

### 13. netty 的 fashwheeltimer 的用法，实现原理，是否出现过调用不够准时，怎么解决。

### 14. netty 的心跳处理在弱网下怎么办。

### 15. netty 的通讯协议是什么样的。

### 16. springmvc 用到的注解，作用是什么，原理。

### 17. springboot 启动机制。

## 五、多线程

### 1. 多线程的几种实现方式，什么是线程安全。

继承Thread类或者实现Runable接口。

线程安全：就是多线程访问时，采用加锁机制，当一个线程访问该类的某个数据时，进行保护，其他线程不能进行访问直到该线程读取完，其他线程才可以使用。不会出现数据不一致或者污染。

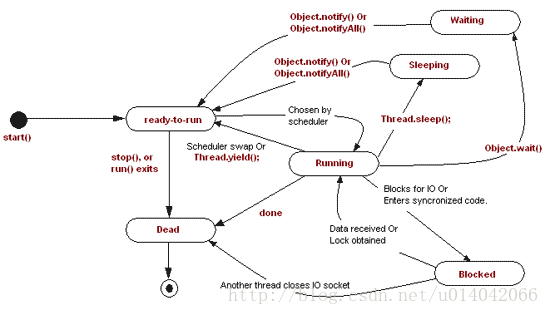
线程不安全：就是不提供数据访问保护，有可能出现多个线程后更改数据造成所得到的数据是脏数据。

### 2. volatile 的原理，作用，能代替锁么。

原理：volatile利用**内存栅栏机制**来保持变量的一致性。

不能代替锁，其只具备数据可见性一致性，不具备原子性。

### 3. 画一个线程的生命周期状态图。



Java线程一共有七个状态：

新建（start()）、可运行（ready-to-run）、运行中（Running）、睡眠（sleep）、阻塞（blocked）、等待（waiting）、死亡（dead）。

### 4. sleep 和 wait 的区别。

Sleep是休眠线程，wait是等待；

Sleep是Thread的静态方法，wait是Object的方法；

Sleep依旧持有锁，并在指定的时间自动唤醒，wait则需要调用notify或者notifyAll释放锁。

### 5. sleep 和 sleep(0)的区别。

Sleep(0) 暂时放弃cpu，让其他线程有机会优先执行。

### 6. Lock 与 Synchronized 的区别。

（1），Lock是一个接口，而synchronized是关键字。

（2），synchronized会自动释放锁，而Lock必须手动释放锁。

（3），Lock可以让等待锁的线程响应中断，而synchronized不会，线程会一直等待下去。

（4），通过Lock可以知道线程有没有拿到锁，而synchronized不能。

（5），Lock能提高多线程读操作的效率。

（6），synchronized能锁住类，方法和代码块，而Lock是块范围的。

首先两者都保持了并发场景下的原子性和可见性，区别则是synchronize的释放锁机制是有其自身控制，且互斥相在某些场景下不符合逻辑，无法进行干预，不可人为中断。

而Lock常用的则有ReentranLock和ReadWiiteLock，添加了类似锁票，定时等候和可中断锁等候的一些特性。此外他还提供了在激烈竞争情况下更佳的性能。

### 7. synchronized 的原理是什么，一般用在什么地方(比如加在静态方法和非静态方法的区别， 静态方法和非静态方法同时执行的时候会有影响吗)，解释以下名词：重排序，自旋锁，偏向锁，轻量级锁，可重入锁，公平锁，非公平锁，乐观锁，悲观锁。

（1）、Synchronized底层是通过监视器的enter和exit实现。

（2）、Synchronized可以加在类、方法、和代码块处。

（3）、Synchronized静态方法属于类，同步静态方法的锁是加在类上；普通方法属于对象，同步非静态方法的锁是加在对象上。（一个是类锁一个是对象锁）

（4）、重排序，自旋锁，偏向锁，轻量级锁，可重入锁，公平锁，非公平锁，乐观锁，悲观锁。

**重排序**：在不改变运行结果前提下，代码可以乱序执行，以提高执行效率。

**自旋锁**：线程在没有取得锁的时候，不被挂起，而转去执行一个空循环，若在若干空循环后，如果线程可以获得锁，则进行执行。若线程依然不能获得锁，才被挂起。

**偏向锁**：它偏向于第一个访问锁的线程，如果在运行过程中，同步锁只有一个线程访问，不存在多线程竞争的情况，则线程不需要触发同步，这种情况下就会给线程加一个偏向锁。如果在运行过程中，遇到了其他线程抢占锁，则持有偏向锁的线程会被挂起，JVM会消除它身上的偏向锁，将锁升级为标准的轻量级锁。

**轻量级锁**：通过CAS来避免进入开销较大的互斥操作，而偏向锁是在无竞争场景下完全消除同步，连CAS也不执行（CAS本身仍旧是一种操作系统同步原语，始终要在JVM与OS之间来回，有一定的开销）

**重量级锁**：重量级锁的加锁、解锁过程和轻量级锁差不多，区别是，竞争失败后，线程阻塞，释放锁后，唤醒阻塞的线程，不使用自旋锁，不会那么消耗CPU，所以重量级锁适合用在同步代码块执行时间长的情况下。

**阻塞锁**：阻塞锁让线程进入阻塞状态进行等待，当获得相应的信号（唤醒或者时间）时，才可以进入线程的准备就绪状态，通过竞争，进入运行状态。

**乐观锁**：乐观锁是一种乐观思想。即认为读多写少，遇到并发的可能性低，每次去拿去数据的时候都认为别人不会修改，所以不会上锁，但是在更新的时候会判断一下再次期间别人有没有去更新这个数据，采在写时先读出当前版本号，然后加锁操作，如果失败则要重新读-比较-写操作。

**悲观锁**：悲观锁就是悲观思想，即认为写多，遇到的并发的可能性高，每次去那数据的时候都认为别人会修改，所以每次在读写数据的时候都会上锁，这样别人想读写就会阻塞直到拿到锁。Java中的悲观锁就是Synchronized，AQS框架下的锁则是先尝试CAS乐观去获取锁，获取不到，才会转锁为悲观锁，入RetreenLock。

### 8. 用过哪些原子类，他们的原理是什么。

AtomicInteger；AtomicLong；AtomicReference；AtomicBoolean；**基于CAS原语**实现，比较并交换、加载连接、条件存储，最坏的情况下是旋转锁。

CAS有三个操作数，一个是内存值V，一个是预期值A，要修改的新值B。当且仅当预期值A和内存值V相同时，才将内存值V修改为新值B，否则什么都不做。

### 9. JUC 下研究过哪些并发工具，讲讲原理。

### 10. 用过线程池吗，如果用过，请说明原理，并说说 newCache 和 newFixed 有什么区别，构造函数的各个参数的含义是什么，比如 coreSize，maxsize 等。

线程池的实现原理：

1.判断**线程池里的核心线程**是否都在执行任务，如果不是（核心线程空闲或者还有核心线程没有被创建），则创建一个新的工作流程来执行任务。如果核心线程都在执行任务，则进入下一个流程。

2.线程池判断工作队列是否已满，如果工作队列没有满，则将新提交的任务存储在这个工作队列里面，如果工作队列满了，则进入下个流程。

3.判断**线程池里的线程**是否都处于工作状态，如果没有，则创建一个新的工作流程来执行任务。如果已经满了，则交给饱和和策略来处理这个任务。

Executors.**newCachedThreadPool**() 初始化一个可以缓存线程的线程池，默认缓存60s，线程池的线程数最大可达Integer.MAX\_VAUE 2147483647，内部使用SynchronousQueue为阻塞队列；

Executors.**newSingleThreadExecutor**(); 初始化一个容量为1的线程池，内部使用LinkedBlockingQueue作为阻塞队列。

Executors.**newFixedThreadPool**(int); 初始化一个指定大小的缓冲池，其中corePoolSize=maiPoolSize，使用Lin得到BlockingQueue阻塞队。

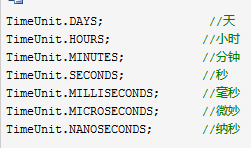
Executors.**newScheduledThreadPool**();初始化的线程池可以在指定时间内周期性的执行所有任务。

**corePoolSize**：线程池的基本大小，没有任务需要执行的时候线程池的大小，并且只有在工作队列满了的情况下才会创建超出这个数量的线程。

**maximumPoolSize**：线程池最大线程数量，表示在线程池中最多能创建的线程数量；

**keepAliveTime**：表示空闲线程最大生存多久被回收。

**unit**：参数keepAliveTime时间单位（7种取值；）



**workQueue**：一个阻塞队列，用来存储等待执行的任务，这个参数的选择和重要，会对现场池的运行产生最大影响，一般来说，这里的阻塞队列有以下几种选择：

ArrayBlockingQueue、LinkedBlockingQueue、SynchronousQueue；

**threadFactory**：线程工厂，主要用来创建线程；

**handler**：表示当拒绝处理任务时的策略，有以下四种；

*ThreadPoolExecutor.AbortPolicy：丢弃任务并抛出RejectExecutorException异常；*

*ThreadPoolExecutor.DiscardPolicy:也是丢弃任务，但是不抛弃异常。*

*ThreadPoolExecutor.DiscardOldestPolicy：丢弃队列最前面的任务，然后重新尝试任务（重复此过程）*

*ThreadPoolExecutor.CallerRunsPolicy：由调用线程处理该任务。*

### 11. 线程池的关闭方式有几种，各自的区别是什么。

Shoutddown ShoutdownNow try Terminate 清空工作队列，终止线程中的各个线程，销毁线程池。

### 12. 假如有一个第三方接口，有很多个线程去调用获取数据，现在规定每秒钟最多有 10 个线程 同时调用它，如何做到。

### 13. spring 的 controller 是单例还是多例，怎么保证并发的安全。

单例。（通过DefaultSingletonBeanRegistry实现单例，通过AsyncTaskExecutor保持安全）

Spring单例模式下用ThreadLocal来切换不同线程之间的参数。用ThreadLocal是为了保证线程安全，单例模式下Spring把每个线程的可能存在线程安全的参数值放入了ThreadLocal，虽然是一个实例在操作，但是不同线程下的数据相互之间都是隔离的，因此运行时创建和销毁bean大大减少了，能减少对内存资源的消耗，而且并发越高优势月明显。

### 14. 用三个线程按顺序循环打印 abc 三个字母，比如 abcabcabc。

### 15. ThreadLocal 用过么，用途是什么，原理是什么，用的时候要注意什么。

ThreadLocal底层使用ThreadLocalMap进行存储键值，每个ThreadLocal类创建一个Map，线程的ID作为Map的key，实例对象作为Map的value，这样就能达到各个线程的值隔离的效果。

ThreadLocal的作用是提供线程内的局部变量，这种变量在线程生命周期内起作用，减少同一个线程内多个函数或者组件之间的一些公共变量的传递的复杂度。

### 16. 如果让你实现一个并发安全的链表，你会怎么做。

Collections.synchronizedList()

ConcurrentLinkedQueue

### 17. 有哪些无锁数据结构，他们实现的原理是什么。

LockFree ，CAS基于JDK的原语实现，例如AtomicInteger、AtomicBoolean等

### 18. 讲讲 java 同步机制的 wait 和 notify。

这两个方法只能在同步代码块中调用，wait会释放掉对象锁，等待notify唤醒。这两个方法都是Object的方法。

### 19. CAS 机制是什么，如何解决 ABA 问题。

CAS机制当中使用了3个基本操作数：内存当中V，旧的预期值A，和要修改的新值B。当要更新一个变量时，只有变量的预期值A和内存值V相等时，才会框内存值V修改为对应的新值B。

### 20. 多线程如果线程挂住了怎么办。

根据具体情况sleep，wait，join等，酌情选择notifyAll，notify进行唤醒线程。

### 21. countdowlatch 和 cyclicbarrier 的内部原理和用法，以及相互之间的差别(比如 countdownlatch 的 await 方法和是怎么实现的)。

### 22. 对 AbstractQueuedSynchronizer 了解多少，讲讲加锁和解锁的流程，独占锁和公 平所加锁有什么不同。

### 23. 使用 synchronized 修饰静态方法和非静态方法有什么区别。

修饰静态方法为类锁，修饰非静态方法为对象锁。

### 24. 简述 ConcurrentLinkedQueue 和 LinkedBlockingQueue 的用处和不同之处。

**LinkedBlockingQueue**是一个基于单向链表的，范围任意的（其实是有界的），FIFO阻塞队列。

**ConcurrentLinkedQueue**是一个基于链接节点的无界线程安全队列，它采用先进先出的规则对节点进行排序，当我们添加一个元素的时候，它会添加到队列的尾部，当我们获取一个元素，它会返回队列头部的元素，它采用“wait-free”算法来实现。

### 25. 导致线程死锁的原因？怎么解除线程死锁。

线程间相互等待资源，而又不释放自身资源，导致无穷等待，就进入死锁。

一般来说出现死锁的条件满足：

1. 互斥条件：一个资源每次只能被一个线程使用；
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。
3. 不剥夺条件：进程已获得的资源，在未使用完成之前不能强行剥夺。
4. 循环等待条件：若干进程之间形成一种头尾连接的循环等待资源关系。

**只要破坏死锁的任意一个条件，死锁问题都能被解决。**

### 26. 非常多个线程（可能是不同机器），相互之间需要等待协调，才能完成某种工作，问怎么设计这种协调方案。

可以使用Executors线程池（问题本质是线程保持顺序执行）

### 27. 用过读写锁吗，原理是什么，一般在什么场景下用。

### 28. 开启多个线程，如果保证顺序执行，有哪几种实现方式，或者如何保证多个线程都执行 完再拿到结果。

### 29. 延迟队列的实现方式，delayQueue 和时间轮算法的异同。