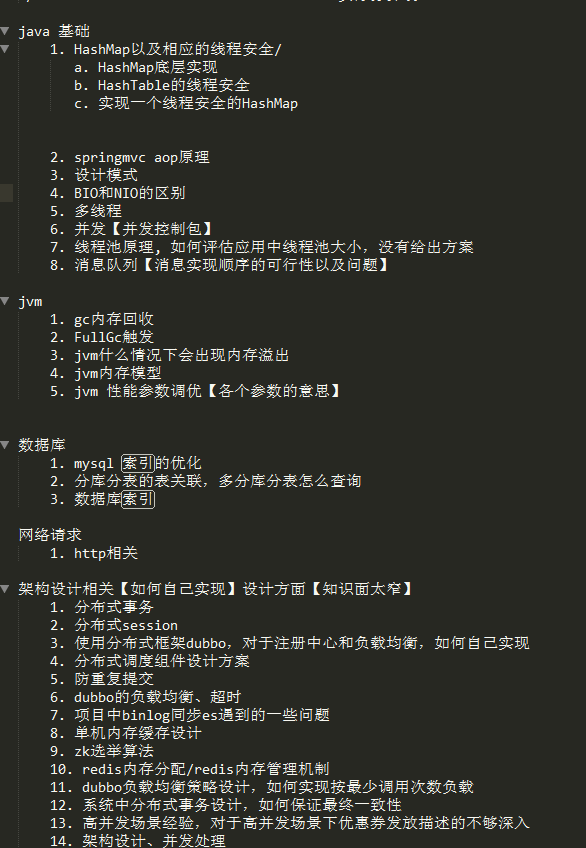
# Java面试题集



## 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

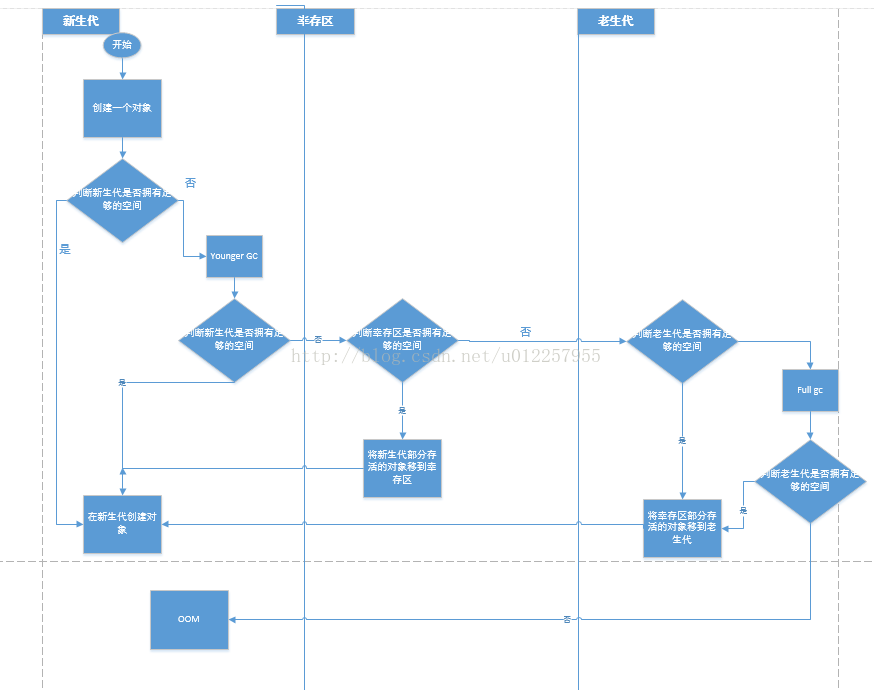
提高GC的效率，因为不是所有的对象都需要频繁的去GC，频繁GC很耗时的。

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）

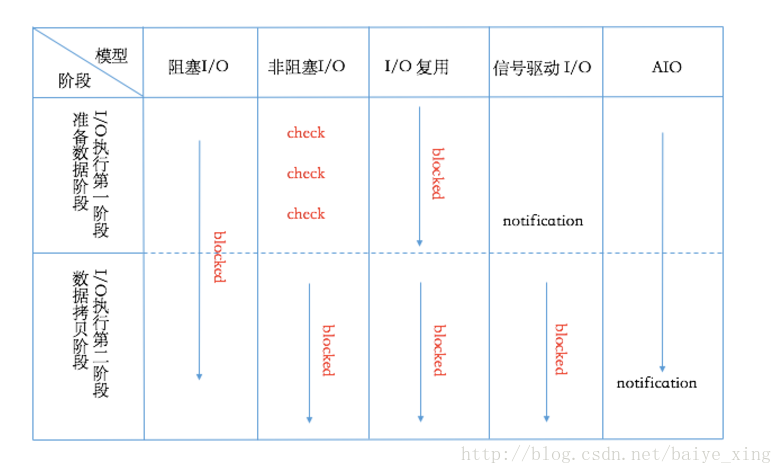
异步Async：发出一个函数调用后，调用者可以在没得到结果之前继续后续操作。当这个调用结束后，一般通过状态、通知或者回调来通知调用者。对于异步来说，调用的返回并不受调用者控制。

**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。

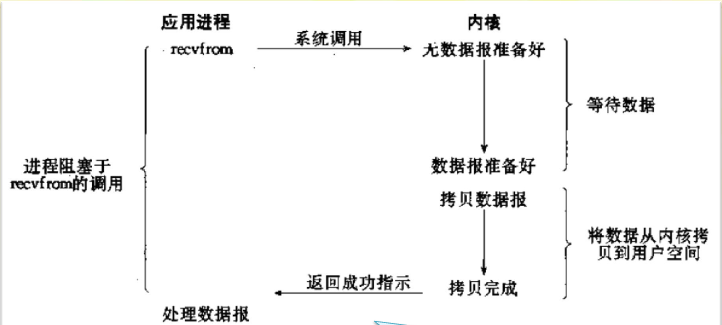
**五种IO模型（）**

****

**（1）阻塞式IO（BIO）**

简介：进程会一直阻塞，直到数据拷贝完成。

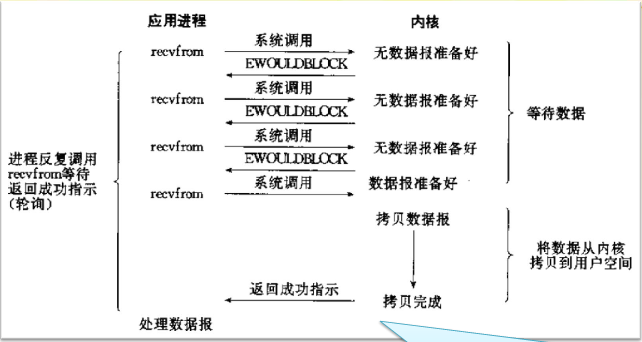
应用程序调用一个IO函数，导致应用程序阻塞，等待数据准备好。如果数据准备好了，从内核拷贝到用户空间。执行完毕后，IO函数会向应用程序返回成功响应，应用程序得到响应后，就不在阻塞，进行后续工作。



**（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。

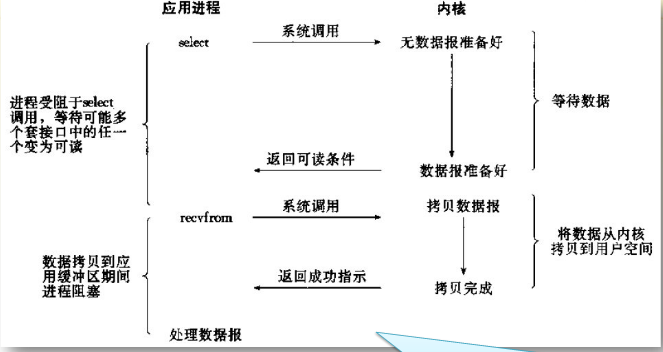
应用程序调用一个IO函数，这个IO操作会从内核中立刻返回。但是这个IO函数具体要执行的事情（写数据）可能还没完成。而对于应用程序，虽然这个IO很快就返回了，但是它并不知道这个IO操作是否真的成功了，为了知道IO操作是否成功，一般有两种策略，一是要应用程序主动轮询，知道数据准备好为止，在这个不断询问的过程中，会大量占用CPU的时间。二是采用IO通知机制，比如IO多路复用或者信号驱动IO。



**（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。

****

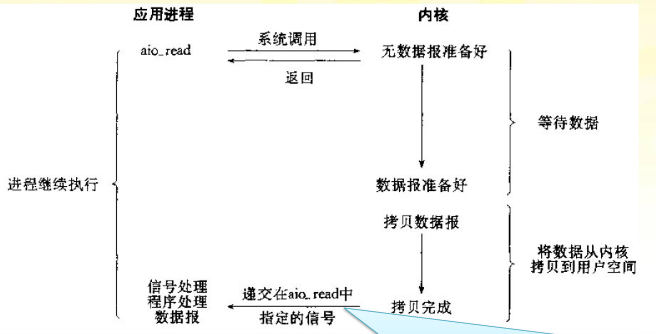
**（4）信号驱动IO模型（signal-driver IO）**

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。

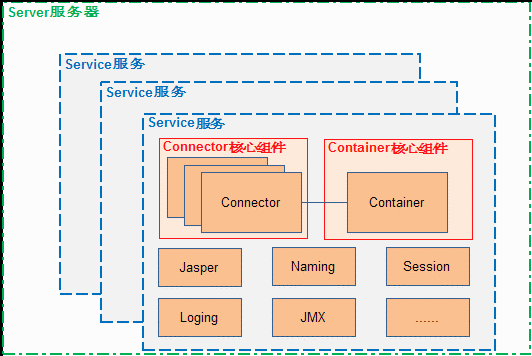
top命令是Linux下常用的性能分析工具，能够实时显示系统中各个进程的资源占用状况，类似于Windows的任务管理器。

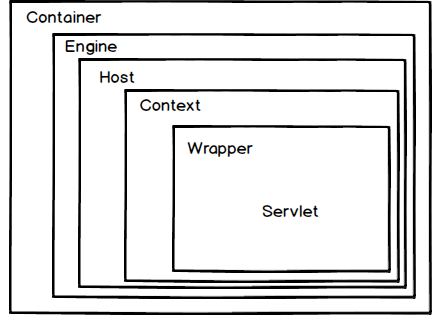
### 10. 线上 CPU 爆高，请问你如何找到问题所在

## 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。

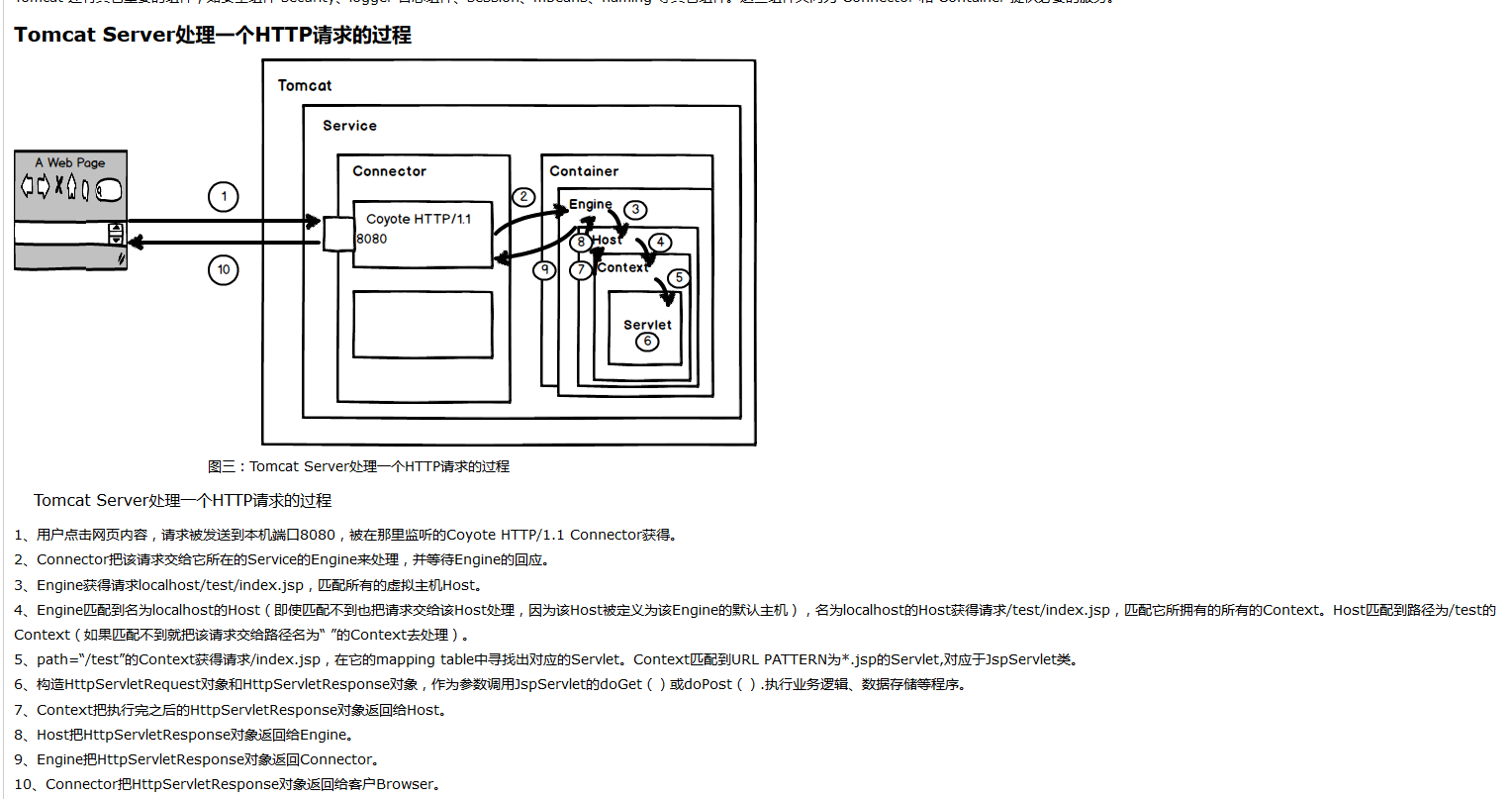
**Context 容器**

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。

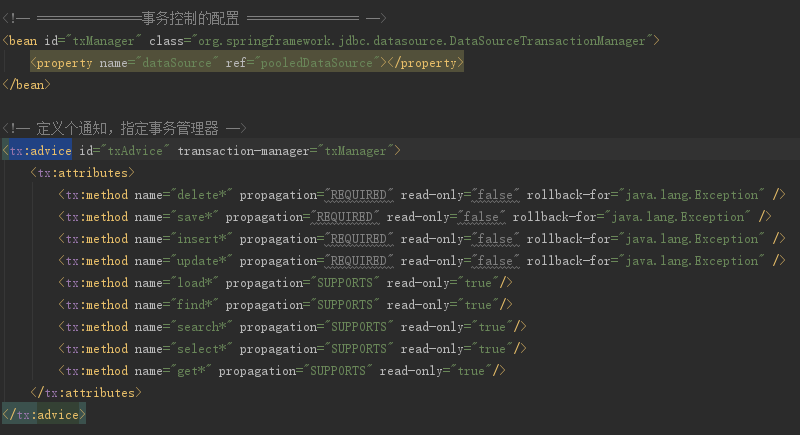
参考：<https://blog.csdn.net/liaohaojian/article/details/70139151>

### 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。

### 18.Springmvc 工作原理。



1. 用户发送请求至前端控制器**DispatchServlet**。
2. DispatchServlet收到请求调用**HandlerMapping**处理映射器。
3. HandlerMapping找到具体的处理器（用过xml配置，注解进行查找），生成处理器对象以及拦截器（如果有则生成）并返回给DispatchServlet。
4. DispatchServlet调用**handlerAdapter**调用具体的处理器（**Controller**，也叫后端控制器）
5. Controller执行完成后返回**ModelAndView**
6. HandlerAdapter将Controller执行结果ModelAndView返回给DispatchServlet。
7. DispatchServlet将ModleAndView传给**ViewReslover**视图处理器，解析后返回具体的View。
8. DispatchServlet根据View进行视图渲染（数据和视图的绑定）
9. 最后DispatchServlet将视图返回给用户。

## 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。

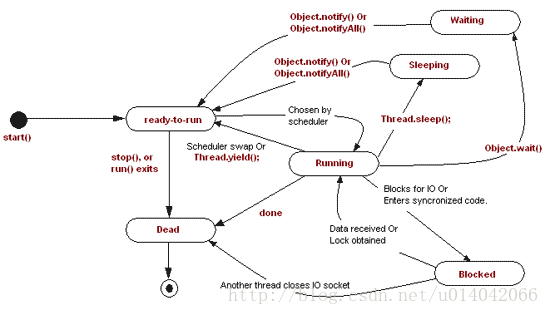
线程不安全：就是不提供数据访问保护，有可能出现多个线程后更改数据造成所得到的数据是脏数据。

### 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。

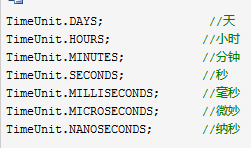
Executors.**newScheduledThreadPool**();初始化的线程池可以在指定时间内周期性的执行所有任务。

**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。

## TCP 与 HTTP

### 1.http1.0 和 http1.1 有什么区别。

### 2.TCP三次握手和四次挥手的流程，为什么断开连接要4次,如果握手只有两次，会出现什么。

3.TIME\_WAIT 和 CLOSE\_WAIT 的区别。

1. 说说你知道的几种 HTTP 响应码，比如 200, 302, 404。
2. 当你用浏览器打开一个链接的时候，计算机做了哪些工作步骤。

6. TCP/IP 如何保证可靠性，说说 TCP 头的结构。

7. 如何避免浏览器缓存。

8. 如何理解 HTTP 协议的无状态性。

9. 简述 Http 请求 get 和 post 的区别以及数据包格式。

10. HTTP 有哪些 method 11. 简述 HTTP 请求的报文格式。

12. HTTP 的长连接是什么意思。

13. HTTPS 的加密方式是什么，讲讲整个加密解密流程。

14. Http 和 https 的三次握手有什么区别。

15. 什么是分块传送。

16. Session 和 cookie 的区别。

## 七、架构设计与分布式

### 1. 用 java 自己实现一个 LRU。

### 2. 分布式集群下如何做到唯一序列号。

参考：<http://www.cnblogs.com/haoxinyue/p/5208136.html>

1，数据库自增长序列或字段

2，UUID：常见的方式，可以利用数据库也可以利用程序生产，一般来说全局唯一。

3，变种的UUID，可以使用UUID to Int64的方法。

4，Redis生成ID，主要依赖Redis单线程，所以也可以生成全局唯一的ID。可以用Redis的原子操作INCR和INCRBY来实现。可以使用Redis集群来提高吞吐量。

5，Twitter的snowflake算法。Snowflake是Twitter开源的分布式ID生成算法，结果是一个Long型的ID。其核心思想是，使用41bit作为毫秒数，10bit作为机器的ID（5bit是数据中心，5bit机器的ID），12bit作为毫秒内的流水号（意味每个节点在没毫秒可以产生4096个ID），最后还有一个符号位，永远是0。

6，利用zookeeper生成唯一ID。依赖咋弄的数据版本生成序列好。

7，MongoDB的ObjectId。MongoDB的ObjectID和snowflake算法类锁。它设计成请量型的，不同的机器都能用全局唯一的同种方法方便生成它。

### 3. 设计一个秒杀系统，30 分钟没付款就自动关闭交易。

分流-限流-异步-公平性（只能参加一次）-用户体验（第几位，多少分钟，一抢完）

容错队列。Redis队列mysql

30分钟关闭交易，可以借助redis的发布订阅机制，或者mq的消息队列死信队列，比如RabbitMQ。

### 4. 如何使用 redis 和 zookeeper 实现分布式锁？有什么区别优缺点，会有什么问题，分别适用什么场景。（延伸：如果知道 redlock，讲讲他的算法实现，争议在哪里）

首先分布式锁实现，常见的有数据库锁（表记录），缓存锁，基于ZK（临时有序的节点可以实现）三种。

Redis适用于对性能要求比较高的场景。Redis可以每秒执行1owci，内网延迟不超过1ms，缺点是数据放于内存，宕机后锁丢失。

锁无法释放，适用zk可以有效的解决锁无法释放的问题，因为在创建锁的时候，客户端会在zk中创建一个临时节点，一旦客户端获取到锁后突然挂掉（Session断开连接），那么这个临时节点就会自动删除，其他客户端就可以再次获取锁。

### 5. 如果有人恶意创建非法连接，怎么解决。

可以使用Filter过滤处理。

### 6. 分布式事务的原理，优缺点，如何使用分布式事务，2pc 3pc 的区别，解决了哪些问题， 还有哪些问题没解决，如何解决，你自己项目里涉及到分布式事务是怎么处理的。

两阶段提交协议，**请求阶段**，协调者将通知事务参与者准备提交或者回滚事务，参与者收到通知并知协调者自己的状态能否提交事务。**提交阶段**，协调者基于第一阶段所有参与者投票结果进行决策，提交或者取消事务，当且仅当所有参与者都同意提交事务，协调者才通知所有参与者提交事务，否则通知所有参与者取消事务。

优点是可以管理多机事务，拥有无线扩展缺点是易用性难，承担延时风险JTA，Atomiks等

### 7. 什么是一致性 hash。

一致性hash是一种分布式hash实现算法。满足平衡性，单调性，分散性和负载。

### 8. 什么是 restful，讲讲你理解的 restful。

REST指的是一组架构约束条件和原则，满足这些约束条件和原则的应用程序或设计就是RESTful。

### 9. 如何设计一个良好的 API。

### 10. 如何设计建立和保持 100w 的长连接。

服务器内核调优（tcp，文件数），客户端调优，框架选择（netty）。

### 11. 解释什么是 MESI 协议(缓存一致性)。

MESI缓存四种缓存段状态的，M（Modified）已修改的缓存段，它属于脏段，E（Exclusive）独占缓存段，是主存内容保持一致的一份拷贝。S（Share）共享缓存段，是主存的一份拷贝，只能读，不能写。I（Invalid）失效缓存段，要么不存在内存中了要么已经过时了。

### 12. 说说你知道的几种 HASH 算法，简单的也可以。

哈希（Hash）算法，即散列函数。它是单向密码体制，即它是一个从明文到密文的不可逆的映射，只有加密过程，没有解密过程。同时，哈希函数可以将任意长度的输入经过变化以后得到固定的输出。

MD4 MD5 SHA

### 13. 什么是 paxos 算法，什么是 zab 协议。

Zab协议数Zookeeper原子广播协议的简称。

这个zab协议主要包括消息广播和崩溃恢复两个过程，进一步可以分为三个阶段，分别是：

发现Discovery

同步Synchronization

广播Broadcast

组成ZAB协议的每一个分布式进程，都会循环执行这三个阶段，将这样一个循环称为一个主进程周期。

### 14. 一个在线文档系统，文档可以被编辑，如何防止多人同时对同一份文档进行编辑更新。

点击编辑是可以让Redis加锁setNX完成后expire一下。也可以通过版本号进行控制。

### 15. 线上系统突然变得异常缓慢，你如何查找问题。

逐个排查（网络、磁盘、内存、CPU），数据库，日志，中间件等监控工具排查。

### 16. 说说你平时用到的设计模式。

单例，代理，工厂，装饰，观察者模式等待。

### 17. Dubbo 的原理，有看过源码么，数据怎么流转的，怎么实现集群，负载均衡，服务注册和发现，重试转发，快速失败的策略是怎样的。

18. 一次 RPC 请求的流程是什么。

19. 自己实现过 rpc 么，原理可以简单讲讲。Rpc 要解决什么问题。

20. 异步模式的用途和意义。

21. 编程中自己都怎么考虑一些设计原则的，比如开闭原则，以及在工作中的应用。

22. 设计一个社交网站中的“私信”功能，要求高并发、可扩展等等。 画一下架构图。

23. MVC 模式，即常见的 MVC 框架。

24. 聊下曾经参与设计的服务器架构并画图，谈谈遇到的问题，怎么解决的。

25. 应用服务器怎么监控性能，各种方式的区别。

26. 如何设计一套高并发支付方案，架构如何设计。

27. 如何实现负载均衡，有哪些算法可以实现。

### 28. Zookeeper 的用途，选举的原理是什么。

分布式协调者，半数+1

29. Zookeeper watch 机制原理。

### 30. Mybatis 的底层实现原理。

31. 请思考一个方案，实现分布式环境下的 countDownLatch。

32. 后台系统怎么防止请求重复提交。

33. 描述一个服务从发布到被消费的详细过程。

34. 讲讲你理解的服务治理。

35. 如何做到接口的幂等性。

36. 如何做限流策略，令牌桶和漏斗算法的使用场景。

37. 什么叫数据一致性，你怎么理解数据一致性。

38. 分布式服务调用方，不依赖服务提供方的话，怎么处理服务方挂掉后，大量无效资源请 求的浪费，如果只是服务提供方吞吐不高的时候该怎么做，如果服务挂了，那么一会重 启，该怎么做到最小的资源浪费，流量半开的实现机制是什么。

39. dubbo 的泛化调用怎么实现的，如果是你，你会怎么做。

40. 远程调用会有超时现象，如果做到优雅的控制，JDK 自带的超时机制有哪些，怎么实现的。

## 八、数据库知识

### 数据库隔离级别有哪些，各自的含义是什么，MYSQL 默认的隔离级别是是什么。

四个级别分别对于 脏读、不可重复读、幻读。

**Read uncommitted 读未提交 可能会出现脏读**

**Read committed 读提交**（大多数数据库的默认级别就是Read Committed，Sql Server ，Oracle）

**Repeateble read 重复读**

**Serializable 序列化** 是最高的事务隔离级别，同时代价也是花费最高，性能很低，一般很少用。该事务级别下，事务顺序执行，可以避免脏读，不可重复读和幻读。

### 2. 什么是幻读。

事务在插入已经检查过不存在的记录时，会发现数据已经存在了，之前的检查就像幻影一样。

### 3. MYSQL 有哪些存储引擎，各自优缺点。

InnoDB ：具有提交、回滚和崩溃恢复能力的事务安全。相对myISAM效率第一项，并会占用更多的磁盘空间以保留数据和索引。

MyISAM：不支持事务和外键，优点，访问速度块

Memory：内存存储，数据放在内存中，访问极快，但是断电数据丢失。

### 4. 高并发下，如何做到安全的修改同一行数据。

使用悲观锁，悲观锁的本质就是当前只有一个线程执行操作，结束了唤醒其他线程进行处理。也可以用redis锁，拿到锁的操作数据。

### 5. 乐观锁和悲观锁是什么，INNODB 的标准行级锁有哪 2 种，解释其含义。

乐观锁就是乐观的认为每次修改都不会冲突，只在提交的时候去检查，悲观锁设定每次修改都会发生冲突，持有排他锁。行锁分为共享锁和排他锁两种，共享锁又称读锁，排他锁又称写锁。

### 6. SQL 优化的一般步骤是什么，怎么看执行计划，如何理解其中各个字段的含义。

查看慢日志，定位慢查询，查看慢程序执行计划，根据执行计划确认优化方案。

### 7. 数据库会死锁吗，举一个死锁的例子，mysql 怎么解决死锁。

会，系统资源不足，进程运行推进的顺序不合适，资源分配不当等都会产生死锁。

Mysql解决死锁的办法：

（1）、重启数据库。

（2）、杀掉抢占资源的进程。

Select \* From INFORMATRION\_SCHEMA.INNODB\_TRX;

Kill trx\_mysql\_thread\_id;

### 8. MYsql 的索引原理，索引的类型有哪些，如何创建合理的索引，索引如何优化。

**索引的原理**：通过复杂的算法，提高数据查询性能的优化手段。从磁盘的IO到内存IO的转变。

普通索引

唯一索引

主键索引

组合索引

全文索引

### 9. 聚集索引和非聚集索引的区别。

“聚集索引” 就是索引和记录紧密联系在一起。

“非聚集索引” 索引文件和数据文件分开存储，索引文件的叶子页只保存了主键，要定位记录还要去查询相应的数据块。

### 10. select for update 是什么含义，会锁表还是锁行或是其他。

Mysql事务以及加锁机制：参考<http://lhc1986.iteye.com/blog/1462856>

显示的加锁：

SELECT .... LOCK IN SHARE MODE (加共享锁)

SELECT .....FOR UPDATE(加排他锁)

属于行锁。

### 11. 为什么要用 Btree 实现，它是怎么分裂的，什么时候分裂，为什么是平衡的。

加快查找速度。

### 12. 数据库的 ACID 是什么。

A：Atomic，原子性，事务要么都提交，要么都失败，不能只提交一部分。

C：Consistent，一致性，事务开始以及结束后，数据的一致性约束没有破坏。

I：Isolation，隔离性，并发事务间相互隔离，互不干扰。

D：Durability，持久性，已经提交的事务对数据库所做的更新必须是永久保持。即使程序发生崩溃，也不能被回滚或数据丢失。

### 13. 某个表有近千万数据，CRUD 比较慢，如何优化。

参考：<https://blog.csdn.net/diquren/article/details/50606808>

大多数应用都是读多写少的，优化Read的性能基本就决定了应用的性能。

（1）、创建索引。

（2）、配置缓存。在配置文件中所做query\_cache\_size。

（3）、slow\_query\_log分析。

（4）、分库分表

（5）、Explain Select查询

（6）、当只有一行数据时使用LIMIT 1

（7）、避免使用SELECT \*

（8）、永远为每张表设置一个ID

（9）、使用ENUM而不是VARCHAR

（10）、字段尽量不用NULL

（5）、读写分离

（6）、硬件升级

### 14. Mysql 怎么优化 table scan 的。

避免在Where子句中对字段进行is null的判断

避免在Where子句中使用!= 或<>操作符，这些操作符会让引擎放弃索引，使用全表扫描。

避免在Where子句中使用or连接条件

In 和not in 使用谨慎

Where子句中对字段进行表达式操作 num/2=xx

在Where子句中对字段进行函数操作

### 15. 如何写 sql 能够有效的使用到复合索引。

Select table tableName where a=xx and b =xx 或者and b=xx like‘xxx%’

### 16. mysql 中 in 和 exists 区别。

In是外表和内表做hash连接，exists是对外表做loop循环在对内表进行查询。

Exists只返回True或False，不返回UNKNOWN

In当遇到NULL的情况会返回UNKOWN

### 17. 数据库自增主键可能的问题。

在分库分表时可能会生成重复主键，利用自增比例达到唯一自增1,2,3等。

### 18. MVCC 的含义，如何实现的。

MVCC：将数据库的行锁与行的多个版本结合起来，只需要很小的开销，就可以实现非锁定读，从而大大提高数据库系统的并发性能。

实现原理：

*MVCC保存某个时间点上的数据快照。一个事务内，看到的是同一个版本的快照，数据一致。不同事务在同一时间点看到的数据会不一致，因为他们得到的数据版本不一样。InnoDB在每行记录存在额外的隐藏字段，其中一列存储行被更新的版本号,另外一列存储行被删除的版本号。每当一个事务开始的时候,innodb都会给这个事务分配一个递增的版本号,所以版本号也可以被认为是事务号.对于每一个”查询”语句,innodb都会把这个查询语句的版本号同这个查询语句遇到的行的版本号进行对比,然后结合*[*不同的事务隔离等级*](http://www.phpddt.com/db/mysql-isolation-level.html)*,来决定是否返回该行。当隔离级别是REPEATABLE READ时，这种策略下，select、delete、 insert、 update语句如何操作：   
1) SELECT 对于select语句，只有同时满足了下面两个条件的行，才能被返回:  
    •行的被修改版本号小于或者等于该事务号  
    •行的被删除版本号要么没有被定义,要么大于事务的版本号:行的删除版本号如果没有被定义,说明该行没有被删除过;如果删除版本号大于当前事务的事务号,说明该行是被该事务后面启动的事务删除的,由于是repeatable read隔离等级,后开始的事务对数据的影响不应该被先开始的事务看见,所以该行应该被返回.  
2) INSERT 对新插入的行,行的更新版本被修改为该事务的事务号  
3) DELETE 对于删除,innodb直接把该行的被删除版本号设置为当前的事务号,相当于标记为删除,而不是实际删除  
4) UPDATE 在更新行的时候,innodb会把原来的行复制一份到回滚段中,并把当前的事务号作为该行的更新版本*

**在每一行数据中额外保存两个隐藏字段：当前行创建的版本号和删除时的版本号（可能为空）。每个事务又有自己的版本号，这样事务内执行CRUD操作时，就通过版本号的比较来达到数据版本控制的目的。**

### 19. 你做过的项目里遇到分库分表了吗，怎么做的，有用到中间件么，比如 sharding jdbc 等, 他们的原理知道么。

分库分表参考：<https://www.cnblogs.com/zwt1990/p/6762135.html>

### 20. MYSQL 的主从延迟怎么解决。

**1. 主从同步的延迟的原因**

我们知道，一个服务器开放Ｎ个链接给客户端来连接的，这样有会有大并发的更新操作, 但是从服务器的里面读取binlog 的线程仅有一个，当某个SQL在从服务器上执行的时间稍长或者由于某个SQL要进行锁表就会导致，主服务器的SQL大量积压，未被同步到从服务器里。这就导致了主从不一致， 也就是主从延迟。

**2. 主从同步延迟的解决办法**

实际上主从同步延迟根本没有什么一招制敌的办法，因为所有的SQL必须都要在从服务器里面执行一遍，但是主服务器如果不断的有更新操作源源不断的写入，那么一旦有延迟产生，那么延迟加重的可能性就会原来越大。当然我们可以做一些缓解的措施。

a. 我们知道因为主服务器要负责更新操作， 他对安全性的要求比从服务器高， 所有有些设置可以修改，比如sync\_binlog=1，innodb\_flush\_log\_at\_trx\_commit = 1 之类的设置，而slave则不需要这么高的数据安全，完全可以讲sync\_binlog设置为0或者关闭binlog，**innodb\_flushlog， innodb\_flush\_log\_at\_trx\_commit 也 可以设置为0来提高sql的执行效率 这个能很大程度上提高效率**。另外就是使用比主库更好的硬件设备作为slave。

b. 就是把，一台从服务器当度作为备份使用，而不提供查询，那边他的负载下来了，执行relay log 里面的SQL效率自然就高了。

c. 增加从服务器喽，这个目的还是分散读的压力，从而降低服务器负载。

**3. 判断主从延迟的方法**

MySQL提供了从服务器状态命令，可以通过 show slave status 进行查看，  比如可以看看Seconds\_Behind\_Master参数的值来判断，是否有发生主从延时。  
其值有这么几种：  
NULL - 表示io\_thread或是sql\_thread有任何一个发生故障，也就是该线程的Running状态是No,而非Yes.  
0 - 该值为零，是我们极为渴望看到的情况，表示主从复制状态正常。

## 九、消息队列

### 1. 消息队列的使用场景。

异步处理，应用解耦（带来数据的一致性问题，采用最终一致性），流量削峰，日志处理，延时任务。

### 2. 消息的重发，补充策略。

编码参考：<https://my.oschina.net/alexgaoyh/blog/268504>

### 3. 如何保证消息的有序性。

### 4. 用过哪些 MQ，和其他 mq 比较有什么优缺点，MQ 的连接是线程安全的吗，你们公司的 MQ 服务架构怎样的。

RabbitMQ，

### 5. MQ 系统的数据如何保证不丢失。

基本都是对数据进行持久化，多盘存储。

### 6. rabbitmq 如何实现集群高可用。

参考部署：<https://blog.csdn.net/super_rd/article/details/70856909>

Rabbit模式大概分为三种：单主机模式、普通集群模式、镜像集群模式。

（1）、单主机模式：RabbitMQ服务运行在单独的一台主机中，通常生产环境不使用该模式，性能有限，并且如果服务宕机将完全不可用。

（2）、普通集群模式：

该模式下的队列中消息实体只存在其中一个节点，集群其他节点仅有相同的元数据，即队列结构。该模式下如果出现一个节点故障后其他节点无法获取到故障节点中还未消费的消息。如果消息做了持久化，那么得等该节点恢复后，然后才能被消费；如果没有做持久化，那么消息就会永远的丢失。

（3）、镜像集群模式：

该模式下会把队列结构和消息都进行同步，属于RabbitMQ的HA方案，但是同步消息会降低系统性能，如果镜像队列数量过多，加入大量的消息，集群内部的网络宽度会被这种同步大量消耗，所以这种模式用于可靠性要求较高的场合中。

### 7. kafka 吞吐量高的原因。

Kafka主要使用一下几个方式实现超高吞吐率

**顺序读写**：

Kafka的消息不断追加到文件，这个特性使kafka可以充分利用磁盘的顺序读写性能，顺序读写不需要磁盘磁头的寻道时间，只需要很少的扇区旋转时间，所以速度远快于随机读写。

**零拷贝**：

跳过用户缓冲区的拷贝，直接是内存和磁盘的映射，数据不在复制到用户态缓冲区。系统上下文切换减少2次，可以提升一倍的性能。

**文件分段**：

Kafka的队列topic被分为多个区partition，每个partition又分为多个段segment，所以一个队列中的消息实际上是保存在N多个片段文件中。通过分段的方式，每次文件操作都是对一个小文件的操作，非常轻便，同时也增加了并行处理能力。

**批量发送**：

Kafka运行批量发送消息，现将消息缓存在内存中，然后一次请求批量发送出去。

这种策略大大减少了服务端的I/O次数。

**数据压缩**：

Kafka还支持对消息集合进行压缩，Producer可以通过GZIP或者Snappy格式对消息集合进行压缩，压缩的好处就是减少传输的数据量，减少对网络传输的压力。

### 8. kafka 和其他消息队列的区别，kafka 主从同步怎么实现。

Rabbitmq遵循AMQP协议，有erlang语言开发，用在实时的对消息可靠性要求比较高的信息传递上。

Kafka主要用于处理活跃的流式数据，大数据量的数据处理上。

### 9. 利用 mq 怎么实现最终一致性。

### 10. 使用 kafka 有没有遇到什么问题，怎么解决的。

### 11. MQ 有可能发生重复消费，如何避免，如何做到幂等。

生成一个业务全局唯一的ID，比如支付ID，订单ID，帖子ID等。

（1）对于同一个业务场景，全局唯一

（2）由业务消息发送方生成，业务相关，对MQ透明。

（3）由业务消息消费方负责，以保证幂等。

### 12. MQ 的消息延迟了怎么处理，消息可以设置过期时间么，过期了你们一般怎么处理。

可以设置过期，一般直接删除。

## 十、缓存

### 1. 常见的缓存策略有哪些，如何做到缓存(比如redis)与DB 里的数据一致性，你们项目中用到了什么缓存系统，如何设计的。

Redis 或者 MemCache

**不管是先写库，再删除缓存；还是先删缓存，再写库，都有可能出现数据不一致的情况**  
因为写和读是并发的，没法保证顺序，如果删了缓存，还没有来得及写库，另一个线程就来读取，发现缓存为空，则去数据库中读取数据写入缓存，此时缓存中为脏数据。如果先写了库，再删除缓存前，写库的线程宕机了，没有删除掉缓存，则也会出现数据不一致情况。   
如果是redis集群，或者主从模式，写主读从，由于redis复制存在一定的时间延迟，也有可能导致数据不一致。

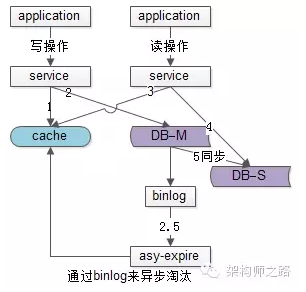
优化思路：

1. 双删 + 超时

在写库前后都进行redis.del（key）操作，并且设置合理的超时时间，这样最差的情况也是在超时时间内存在不一致，当然这种情况是极其少见的，可能的原因是服务器宕机。这种情况能够满足绝大数需求。当然这种策略要考虑redis和数据库主从同步的耗时，所以第二次删除前最后休眠一定时间，这样毫无疑问增加了请求的耗时。

1. 异步淘汰缓存

通过binlog的方式，异步淘汰缓存。



好处：业务代码侵入性低，将缓存与数据库不一致的时间尽量缩小。

### 2. 如何防止缓存击穿和雪崩。

**参考：**[**https://blog.csdn.net/zeb\_perfect/article/details/54135506**](https://blog.csdn.net/zeb_perfect/article/details/54135506)

**缓存击穿：**

缓存穿透是指查询一个一定不存在的数据，由于缓存是不命中时被动写的，并且出于容错考虑，如果从存储层查不到数据则不写入缓存，这将导致这个不存在的数据每次请求都要到存储层去查询，失去了缓存的意义。在流量大时，可能DB就挂掉了，要是有人利用不存在的key频繁攻击我们的应用，这就是漏洞。

采用布隆过滤器，将所有可能存在的数据哈希存到一个足够大的bitmap中，一个不存在的数据就会被bitmap拦截掉，从而避免对底层存储系统的查询压力。

**缓存雪崩**：

缓存在某一时刻大面积失效，请求全部积压到DB，DB瞬时压力过重雪崩。

缓存失效时的雪崩效应对底层系统的冲击非常可怕。大多数系统设计者考虑用加锁或者队列的方式保证缓存的单线 程（进程）写，从而避免失效时大量的并发请求落到底层存储系统上。这里分享一个简单方案就时讲缓存失效时间分散开，比如我们可以在原有的失效时间基础上增加一个随机值，比如1-5分钟随机，这样每一个缓存的过期时间的重复率就会降低，就很难引发集体失效的事件。

**缓存击穿**

对于一些设置了过期时间的key，如果这些key可能会在某些时间点被超高并发地访问，是一种非常“热点”的数据。这个时候，需要考虑一个问题：缓存被“击穿”的问题，这个和缓存雪崩的区别在于这里针对某一key缓存，前者则是很多key。

缓存在某个时间点过期的时候，恰好在这个时间点对这个Key有大量的并发请求过来，这些请求发现缓存过期一般都会从后端DB加载数据并回设到缓存，这个时候大并发的请求可能会瞬间把后端DB压垮。

解决方案

1.使用互斥锁(mutex key)

业界比较常用的做法，是使用mutex。简单地来说，就是在缓存失效的时候（判断拿出来的值为空），不是立即去load db，而是先使用缓存工具的某些带成功操作返回值的操作（比如Redis的SETNX或者Memcache的ADD）去set一个mutex key，当操作返回成功时，再进行load db的操作并回设缓存；否则，就重试整个get缓存的方法。

SETNX，是「SET if Not eXists」的缩写，也就是只有不存在的时候才设置，可以利用它来实现锁的效果。在redis2.6.1之前版本未实现setnx的过期时间，所以这里给出两种版本代码参考：

2. "提前"使用互斥锁(mutex key)：

在value内部设置1个超时值(timeout1), timeout1比实际的memcache timeout(timeout2)小。当从cache读取到timeout1发现它已经过期时候，马上延长timeout1并重新设置到cache。然后再从数据库加载数据并设置到cache中。伪代码如下：

3. "永远不过期"：

这里的“永远不过期”包含两层意思：

(1) 从redis上看，确实没有设置过期时间，这就保证了，不会出现热点key过期问题，也就是“物理”不过期。

(2) 从功能上看，如果不过期，那不就成静态的了吗？所以我们把过期时间存在key对应的value里，如果发现要过期了，通过一个后台的异步线程进行缓存的构建，也就是“逻辑”过期

4. 资源保护：

采用netflix的hystrix，可以做资源的隔离保护主线程池，如果把这个应用到缓存的构建也未尝不可。

### 3. 缓存数据过期后的更新如何设计。

### 4. redis 的 list 结构相关的操作。

#### [1]Redis的数据结构以及操作

##### String

set my 35

setex my 30 60 **[将值 value 关联到 key ，并将 key 的过期时间设为 seconds (以秒为单位)。]**

setnx my 12 **[只有在 key 不存在时设置 key 的值。]**

get my

incr my

incrby my 4

decr my

decrby my 10

##### List

Lpush mykey kaka

Rpush mykey kakarer

Lindex mykey 0

Lset mykey 0 kala0

Lrange mykey 0 10

Llen mykey （获取数组长度）

Lpop mykey （移出并获取列表的第一个元素）

Rpop mykey （移出并获取列表的最后一个元素）

##### Hash

增 **hset** myhash name 小小

hset myhash age 22

**hmset** myhash name 小小 age 23 descM 牛逼的石头人 skill VT2连

Hsetnx myhash name 小小

删 **hdel** myhash name

**Del** myhash

改 hset myhash

**Hincrby** myhash age 12

查 **hget** myhash name

**Hget** myhash age

**hgetall** myhash

**hkeys** myhash

**hvals** myhash

##### Set

增 sadd color read green blue gray

Sadd color2 pink

删

改

查 smembers color （查询color全部数据）

Srandmember color （随机从color中获取一个值）

Scard color（返回color的值数量 集合的长度）

Sismember color red（验证red是否存在）

集合的操作

Sdiff color color2（返回color一种与color2中不同的集合）

Sdiffstore color color2

Sinter color color2（返回color 和color2的交集）

Sunion color1 color2（取color1和color2的并集）

##### SortedSet

### 5. Redis 的数据结构都有哪些。

String List hash set sortedSet

### 6. Redis 的使用要注意什么，讲讲持久化方式，内存设置，集群的应用和优劣势，淘汰策略等。

持久化方式：RDB时间点快照，AOF记录服务器执行的所有写操作命令，并在启动服务器时，通过重新执行这些命令来还原数据集。

内存设置：maxmemory userd\_memory

3.0采用Cluster方式

Redis集群相对单机在功能上存在一些限制， 需要开发人员提前了解，   
在使用时做好规避。 限制如下：   
1） key批量操作支持有限。 如mset、 mget， 目前只支持具有相同slot值的   
key执行批量操作。 对于映射为不同slot值的key由于执行mget、 mget等操作可   
能存在于多个节点上因此不被支持。

2） key事务操作支持有限。 同理只支持多key在同一节点上的事务操   
作， 当多个key分布在不同的节点上时无法使用事务功能。   
3） key作为数据分区的最小粒度， 因此不能将一个大的键值对象如   
hash、 list等映射到不同的节点。

4） 不支持多数据库空间。 单机下的Redis可以支持16个数据库， 集群模   
式下只能使用一个数据库空间， 即db0。

5） 复制结构只支持一层， 从节点只能复制主节点， 不支持嵌套树状复   
制结构。 Redis Cluster是Redis的分布式解决方案， 在3.0版本正式推出， 有效地解   
决了Redis分布式方面的需求。 当遇到单机内存、 并发、 流量等瓶颈时， 可   
以采用Cluster架构方案达到负载均衡的目的。 之前， Redis分布式方案一般   
有两种：   
客户端分区方案， 优点是分区逻辑可控， 缺点是需要自己处理数据路   
由、 高可用、 故障转移等问题。   
·代理方案， 优点是简化客户端分布式逻辑和升级维护便利， 缺点是加   
重架构部署复杂度和性能损耗。   
现在官方为我们提供了专有的集群方案： Redis Cluster， 它非常优雅地   
解决了Redis集群方面的问题， 因此理解应用好Redis Cluster将极大地解放我   
们使用分布式Redis的工作量， 同时它也是学习分布式存储的绝佳案例。

### 7. redis2 和 redis3 的区别，redis3 内部通讯机制。

集群方式区别：redis3采用Cluster，2采用客户端分区方案和代理方案。通信过程说明：

1. 集群中每个节点都会单独开辟一个TCP通道，用于节点之间彼此通信，通信信号端口在基础端口上加10000.
2. 每个节点在固定周期内通过特点规则选择几个节点发送ping消息。
3. 接收到ping消息的节点用pong消息作为响应。

### 8. 当前 redis 集群有哪些玩法，各自优缺点，场景。

主从哨兵模式、Cluster

9. Memcache 的原理，哪些数据适合放在缓存中。

### 10. redis 和 memcached 的内存管理的区别。

Redis单个key存储比memcached大，memecached单个key只能保持512k的数据

Redis支持的数据类型更多，memecahed很单一。

Redis支持数据备份，持久化。Memecahed不支持持久化。

### 11. Redis 的并发竞争问题如何解决，了解 Redis 事务的 CAS 操作吗。

**Redis的并发竞争问题。**

Redis为单进程单线程模式，采用队列模式将并发访问变为串行访问。Redis本身没有锁的概念，Redis对于多个客户端连接并不存在竞争，但是Jedis客户端对Redis进行并发访问时就会发生连接超时、数据转换错误、阻塞、客户端关闭连接等问题，这些问题均是由客户端连接混乱造成。一般有2种解决办法：

1. 客户端角度：对连接进行池化，保证正常有序的与Redis进行通讯。同时客户端读写Redis操作JVM锁Synchronized
2. 服务端角度，使用setnx实现锁。

**Redis事务CAS操作**

Redis作为NoSQL数据库也同样提供了事务机制。在Redis中，MULTI/EXEC/DISCARD/WATCH这四个命令是我们实现事务的基石

1. 在事务中的所有命令都将会被串行化的顺序执行，事务执行期间，Redis不会再为其它客户端的请求提供任何服务，从而保证了事物中的所有命令被原子的执行   
2. 和关系型数据库中的事务相比，在Redis事务中如果有某一条命令执行失败，其后的命令仍然会被继续执行   
3. 我们可以通过MULTI命令开启一个事务，有关系型数据库开发经验的人可以将其理解为”BEGIN TRANSACTION”语句。在该语句之后执行的命令都将被视为事务之内的操作，最后我们可以通过执行EXEC/DISCARD命令来提交/回滚该事务内的所有操作。这两个Redis命令可被视为等同于关系型数据库中的COMMIT/ROLLBACK语句。   
4. 在事务开启之前，如果客户端与服务器之间出现通讯故障并导致网络断开，其后所有待执行的语句都将不会被服务器执行。然而如果网络中断事件是发生在客户端执行EXEC命令之后，那么该事务中的所有命令都会被服务器执行   
5. 当使用Append-Only模式时，Redis会通过调用系统函数write将该事务内的所有写操作在本次调用中全部写入磁盘。然而如果在写入的过程中出现系统崩溃，如电源故障导致的宕机，那么此时也许只有部分数据被写入到磁盘，而另外一部分数据却已经丢失。Redis服务器会在重新启动时执行一系列必要的一致性检测，一旦发现类似问题，就会立即退出并给出相应的错误提示。此时，我们就要充分利用Redis工具包中提供的redis-check-aof工具，该工具可以帮助我们定位到数据不一致的错误，并将已经写入的部分数据进行回滚。修复之后我们就可以再次重新启动Redis服务器了

### 12. Redis 的选举算法和流程是怎样的。

### 13. redis 的持久化的机制，aof 和 rdb 的区别。

RDB 定时快照方式(snapshot)： 定时备份，可能会丢失数据   
AOF 基于语句追加方式 只追加写操作   
AOF 持久化和 RDB 持久化的最主要区别在于，前者记录了数据的变更，而后者是保存了数据本身

### 14. redis 的集群怎么同步的数据的。

redis replication

redis-migrate-tool等方式

### 15. 知道哪些 redis 的优化操作。

### 16. Reids 的主从复制机制原理。

Redis主从复制可根据是否是全量分为全量同步和增量同步。

**（1）、全量同步**

Redis全量复制一般发生在slave初始化阶段，这时slave需要将master上的所有数据都复制一份。

1. 从服务器连接主服务器，发送SYNC命令；
2. 主服务器接受到SYNC命名后，开始执行BGSAVE命令生成RDB文件并使用缓冲区记录此后执行的所有写命令；
3. 主服务器BGSAVE执行完后，向所有从服务器发送快照文件，并在发送期间继续记录被执行的写命令；
4. 从服务器收到快照文件后丢弃所有旧数据，载入收到的快照；
5. 主服务器快照发送完毕后开始向从服务器发送缓冲区中的写命令；
6. 从服务器完成快照的载入，开始接受命令请求，并执行来自主服务器缓冲区的写命令。

**（2）、增量同步**

Redis增量复制是指Slave初始化后开始正常工作时主服务器发生写的操作同步接受到从服务器的过程。增量复制的过程主要是主服务器每执行一个写命令就向服务器发送相同的写命令，从服务器接收并执行收到的写命令。

**（3）、Redis主从同步策略**

主从刚刚连接的时候，进行全量同步；全量同步结束后，进行增量同步。当然，如果有需要，slave在任何时候都可以发起全量同步。Redis策略，无论如何，首先会尝试进行增量同步，如果不成功，要求从机进行全量同步。

**注意点**

如果多个Salve断线了，需要重启的时候，因为只要Slave启动，就会发送sync请求和主机全量同步，当多个同步出现的时候，可能会导致Master IO剧增宕机。

### 17. Redis 的线程模型是什么。

Redis基于Reactor模式开发自己的网络事件处理器：这个处理器被称为文件事件处理器（file event handler）

***文件事件处理器使用I/O多路复用（multiplexing）程序来同时监听多个套连接，并根据套连接字目前执行的任务来为套连接字关联不同的事件处理器。***

***当被监听的套接字准备好执行连接应答（accept）、读取（read）、写入（write）、关闭（close）等操作时，与操作相对应的文件事件就会产生，这时文件处理器就会调用套连接字之前关联好的事件处理器来处理这些事件。***

### 18. 请思考一个方案，设计一个可以控制缓存总体大小的自动适应的本地缓存。

### 19. 如何看待缓存的使用（本地缓存，集中式缓存），简述本地缓存和集中式缓存和优缺点。 本地缓存在并发使用时的注意事项。

## 十一、Mybitis

### 1.#{} 和 ${}的区别

#{} 是预编译处理，${}是字符替换

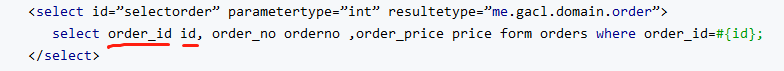
Mybitis在处理#{}时，会将sql中的#{}替换为?号，调用PrepareStatement的set方法来赋值；

Mybitis在处理${}时，就是把${}替换成变量的值

使用#{}可以有效的防止SQL注入，提高系统安全性。

### 2.当实体类中属性名和表中的字段名不一样，怎么办？

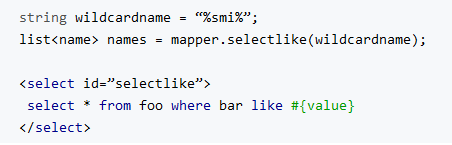
第一种：可以通过查询的sql语句中定义的字段别名，让字段名和实体类的属性名一致。



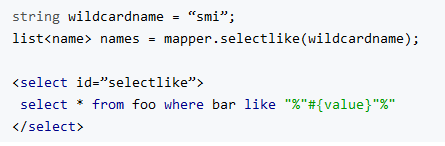
第二种：通过<resultMap>来映射字段名和实体属性名一一对应的关系。

### 3.模糊查询like语句怎么写？

第一种：在Java代码中添加sql通配符。



第二种：在sql语句中连接通配符，会引起sql注入。



### 4.通常一个Xml映射文件，都会写一个Dao接口与之对应，请问这个Dao接口的工作原理是什么？Dao接口里的方法，参数不同时，方法能重载吗？

Dao接口，就是人们常说的Mapper接口，接口的权限名，就是映射文件的namespace的值，接口的方法名，就是映射文件中MapperStatemnt的id，接口的方法内的参数，就是传递给sql的参数。Mapper接口是没有实现类的，当调用接口方法时，接口全限名+方法名拼接字符串作为key值，可以唯一定位一个MapperStatement，举例：com.mybitis3.mapper.StudentDao.findStudentById，可以唯一找到namespace为com.mybitis3.mppers.StudentDao下面id=findStudentById的MappedStatement。在mybitis中，每一个<select> <insert> <update> <delete> 标签，都会被解析为一个MappedStatement对象。

Dao接口里的方法，是不能重载的，因为全限名+方法名的保持和寻找策略。

Dao接口的工作原理是JDK动态代理，Mybitis运行时会使用JDK动态代理为Dao接口生成Proxy对象，代理对象Proxy会拦截接口方法，转而执行MappedStatement所代表的sql，然后将sql执行结果返回。

### 5.Mybitis是如何进行分页的？分页插件的原理是什么？

Mybitis使用RowBounds对象进行分页，它是针对ResultSet结果集进行内存分页，可以在sql内直接书写带有物理分页参数来完成物理分页功能，也可以使用分页插件来完成物理分页。

分页插件的基本原理是使用mybitis提供的插件接口，实现自定义插件，在插件的拦截方法内拦截待执行的sql，然后重写sql，根据dialect方言，添加对应的物理分页语句和物理分页参数。

### 6.Mybatis是如何将sql执行结果封装为目标对象返回的？都有那些映射形式？

第一种：通过<resultMap>标签，逐一定义列名和对象属性之间的映射关系。

第二种是使用sql列的别名功能，将列名书写为对象属性名。

有了列名和属性的映射关系后，mybitis通过反射创建对象，同时使用反射给对象的属性逐一赋值并返回，那些找不到映射关系的属性，是无法完成赋值的。

### 7.如何执行批量插入？



### 8.如何获取自动生成的主键键值？

Insert 方法返回一个int值-这个值代表插入的行数。而自动主键生成的键值在insert方法执行完后可以被传入到参数对象中。

在insert中设置usegeneratedkeys=”true” keyproperty=”id”

### 9.在mapper中如何传递多个参数？

第一种：Dao层的函数定义多个参数 对于xml用#{}或者${}获取参数

第二种：使用@Pararm注解（推荐封装称为一个mao，作为单个参数传递给mapper）

### 10.Mybitis动态sql是做什么的？都有那些动态sql？能简述一下动态sql的执行原理？

Mybitis动态sql可以让我们在xml映射文件内，以标签的形式动态编写动态sql，完成逻辑判断或者动态拼接sql的功能。

Mybatis提供了9中动态sql标签：trim | where | set | foreach | if | choose | when | otherwise |bind。

其执行原理为，使用OGNL从sql参数对象中计算表达式的值，根据表达式的值动态拼接sql，以此来完成动态sql的功能。

### 11.Mybatis的Xml映射文件中，不同的Xml映射文件，id是否可以重复？

不同的xml映射文件，如果配置了namespace，那么id可以重复；如果没有配置namespace，那么id不能重复；

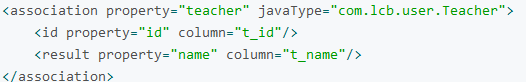
毕竟namespace不是必须的，只是最近实践而已。

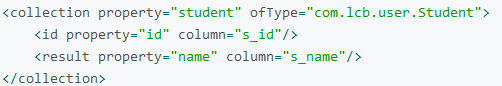
原因就是namespace+id是作为Map<String,MappedStatement>的key使用的，如果没有namespace，就剩下id，那么id重复会导致数据相互覆盖。有了namespace，自然id就可以重复，namespace不同，namespace+id自然也就不同。

### 12.为什么说Mybitis是半自动ORM映射工具？它与全自动的区别在哪里？

Hibernate属于全自动ORM映射工具，使用Hibernate查询对象或者关联集合对象时，可以根据对象关系模型直接获取，所以它是全自动的。而mybitis在查询关联集合对象时，需要手动编写sql来完成，所以，称为半自动ORM映射工具。

### 13.一对一、一对多的关联查询。





### 14.简单介绍下你对mybitis的理解？

mybitis配置

SQLMapConfig.xml，此文件作为mybitis的全局配置文件，配置mybitis的运行环境等信息。

mapper.xml文件即sql映射文件，文件中配置了操作数据库的sql语句。此文件需要在SqlMapConfig.xml中加载。

通过mybitis环境等配置信息构件SqlSessionFactory即会话工厂。

由会话工厂创建sqlSession即会话，操作数据库需要通过sqlsession进行。

Mybitis底层自定义了Executor执行器接口操作数据库，Executor接口有两个实现，一个是基本执行器、一个是缓存执行器。

MappedStatement也是Mybatis一个底层封装对象，它包装了mybitis配置信息及sql映射信息等。Mapperxml文件中一个sql对应一个MappedStatemt对象，sql的id就是MappedStatemt的id。

MappedStatement对sql执行输入参数进行定义，包括hashMap、基本类型、pojo，Executor通过MappedStatement在执行sql前将输入的java对象映射至sql中，输入参数映射就是jdbc编程中对preparedStatement设置参数。

MappedStatement对sql执行输出结果进行定义，包括hashMap、基本数据类型、pojo，Executor通过MappedStatement在执行sql后将输出结果映射到java对象中，输出结果映射工厂就相当于jdbc编程中对结果的解析处理过程。

## 十二、一面

### 说说maven如何进行多模块的？maven的依赖？

通过pom.xml 中配置modules中的module

### 说说你项目中的使用Spring boot的好处？如何使配置更低，低在哪？比如？

去除了大量的xml配置文件。

简化复杂的依赖管理。

配合各种starter使用，基本上可以做到自动化配置。

快速启动容器。

独立spring应用程序，嵌入式的Tomcat，jetty容器，无需外部的war包。

### Springboot运行原理

参考：<https://blog.csdn.net/xiaobing_122613/article/details/54943448>

通过@SpringBootApplication这个组合注解，包含@Configuration、@EnableAutoConfiguration、@ComponentScan，它的核心功能有@EnableAutoConfiguration这个注解提供。@EnableAutoConfiguration注解的关键是@Inport注解的导入功能和@AutoConfigurationPackage会将主类所做包的使用组件扫描到spring容器中。

EnableAutoConfigurationInportSelector（导入开启自动配置导入的选择器）使用SpringFactorieslLoader.LorderFactoryNames方法来扫描具有META-INFO/spring.factories文件的jar包，Spring-boot-autoconfiguration-xxxx.jar包就有一个Spring.factories文件，这个文件声明了有那些需要自动配置。

@Configuration注解是一个配置类，当满足在类路径下容器没有bean时就会自动被装配；

### SpringBoot自动配置的原理

Springboot自动配置主要通过@EnableAutoConfiguration，

### 说说你项目中如何前端和后端使如何连接在一起的？前后端分离如何分离？

### 说说你项目中的架构？自己用过分布式架构吗？谈谈你对分布式架构的理解？

Dubbo+zookeeper 分布式 springcloud做分布式。

将传统的集中式的项目按一定划分为多个功能微服务，各个服务提供单一职责，订单、账单、用户都只管自己的。。。。

### 你项目中使用了restful风格吗？怎么实现的？实现restful风格的好处？

### 说说Spring的Aop的底层原理？它和Aspect有什么区别？你用过Aop吗？它主要用在什么地方？解决了什么问题？

SpringAOP底层通过JDK的动态代理或者子类的Cglib代理实现。

### 说说Spring在整个项目中的作用？它和SpringMVC是什么关系？

自动注入管理bean的实例化、作用域、以及生命周期。和依赖关系，都由容器来管理，还能提供一些面向切面的编程等。SpringMVC是spring一个子模块。SpringMVC需要依赖spring容器。

## 十三、二面

### 说说消息队列的机制?其底层原理是什么？用到哪里？怎么用的？用消息队列有什么好处？怎么保证负载均衡？

### 说说在哪里要用到多线程安全？如何使多线程安全？synchronized和volatile关键字的区别？使用的场景分别是？

### 垃圾回收分代收集算法？为什么会有两个Survivor区？new一个对象会保存在哪里？

### 垃圾回收为什么会停顿？哪些对象可能作为GCRoots?

### G1回收器和其他回收器有什么区别？

### 说说你是如何监控线程的？Linux的ps进程命令是什么？

Ps命令用于报告当前系统的进程状态；

### 说说mysql表的拆分，读写分离，负载均衡，主从复制的原理？唯一索引和非唯一索引的区别？主键索引和唯一索引的区别？

**垂直拆分**---优点：可以降低并发情况下对表的锁定，缺点：对于表单来说随着数据记录增多，读写压力增大。

**水平分割**—原理通过hash算法，将一张表分为N多页，并通过一个新的表记录每页的位置。

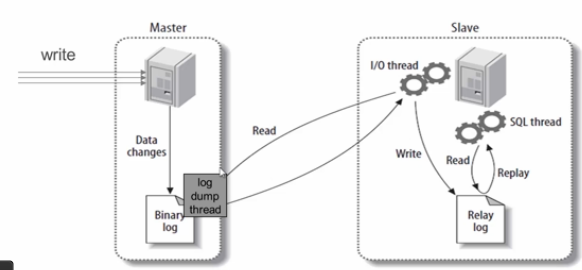
**Mysql主从复制原理：**

从库两个线程，一个I/O线程，一个SQL线程。

Io线程去请求主库的binlog，并将得到的binlog日志写到relay log（中继文件）文件中；

主库会生成一个log dump线程，用来给从库IO线程传binlog；

SQL线程，会读取relay log日志文件，并解析成具体的操作，来实现主从的操作一致，而最终数据一致；



**唯一索引和非唯一索引的区别**：

唯一索引要求字段唯一，否则无法插入数据。

非唯一索引比唯一索引的数据访问速度更快。

### 用过中间件没？如何在线程池里使用中间件？为啥要用线程池？使用线程池的好处是？ 线程池的原理？线程池的核心线程数如何设置？为啥要这样设置？

### 说说观察者模式？在哪些情况下用到了观察者模式？为啥要用？

### 一个Controller调用两个Service，这两Service又都分别调用两个Dao，问其中用到了几个数据库连接池的连接？数据库连接池是线程安全的吗？

你知道什么是泛型吗？解释一下概念和应用场景

### 说说jvm的内存模型？说说强弱软虚引用？

说说线程的状态？产生死锁怎么办？如何进行多线程调优？