# Java相关

## Arraylist与LinkedList默认空间是多少；

ArrayList默认大小是10，但是一开始并没有初始化，扩容：int newCapacity = oldCapacity + (oldCapacity >> 1);

LinkedList基于双链表，默认为０。

## Arraylist与LinkedList区别与各自的优势List 和 Map 区别；

## 谈谈HashMap，哈希表解决hash冲突的方法；

链表法+红黑树

默认大小是16，链表大小>=8变成红黑树，但是只有在超过64的情况下。

## 为什么要重写hashcode()和equals()以及他们之间的区别与关系；

## Object的hashcode()是怎么计算的？

<https://www.zhihu.com/question/29976202>

## 若hashcode方法永远返回1或者一个常量会产生什么结果？

## Java Collections和Arrays的sort方法默认的排序方法是什么；

<https://blog.csdn.net/timheath/article/details/68930482>

Arrays.sort()方法，如果数组长度大于等于286且连续性好的话，就用归并排序，如果大于等于286且连续性不好的话就用双轴快速排序。如果长度小于286且大于等于47的话就用双轴快速排序，如果长度小于47的话就用插入排序。真是有够绕的

## 引用计数法与GC Root可达性分析法区别；

<https://blog.csdn.net/ochangwen/article/details/51406779>

可作为GC Roots对象的包括如下几种：

a.虚拟机栈(栈桢中的本地变量表)中的引用的对象

b.方法区中的类静态属性引用的对象

c.方法区中的常量引用的对象

d.本地方法栈中JNI的引用的对象

## 浅拷贝和深拷贝的区别；

<https://www.cnblogs.com/dolphin0520/p/3700693.html>

<http://www.cnblogs.com/Jtianlin/p/4605477.html>

<https://blog.csdn.net/wangxueming/article/details/52034841>

深拷贝：引用对象的值等信息，复制一份一样的。 |

浅拷贝：只复制引用，另一处修改，你当下的对象也会修改。

## String s="abc"和String s=new String("abc")区别；

<https://blog.csdn.net/akon405/article/details/21865697>

public String(String original) {

this.value = original.value;

this.hash = original.hash;

}

## HashSet方法里面的hashcode存在哪，如果重写equals不重写hashcode会怎么样？

HashSet基于HashMap

## 反射的作用与实现原理；

<https://blog.csdn.net/WildGrasses/article/details/70243342>

<https://www.cnblogs.com/dongguacai/p/6535417.html>

Class：类的对象

Constructor：类的构造方法

Field：类中的属性对象

Method：类中的方法对象

## Java中的回调机制；

<https://blog.csdn.net/xiaanming/article/details/8703708/>

<https://www.cnblogs.com/xrq730/p/6424471.html>

## 模板方法模式；

<https://blog.csdn.net/carson_ho/article/details/54910518>

## 开闭原则说一下；

<https://blog.csdn.net/zhengzhb/article/details/7296944>

一个软件实体如类、模块和函数应该对扩展开放，对修改关闭。

## 发布/订阅使用场景；

<https://blog.csdn.net/fly910905/article/details/78495971>

## KMP算法（一种改进的字符串匹配算法）；

<https://blog.csdn.net/wangbaochu/article/details/50687160>

<https://www.cnblogs.com/dimmacro/p/4460838.html>

<https://blog.csdn.net/luoshengkim/article/details/44724981>

## JMM里边的原子性、可见性、有序性是如何体现出来的，JMM中内存屏障是什么意思，

<http://364434006.iteye.com/blog/1810816>

<https://blog.csdn.net/lkg_vvk/article/details/78980726>

<https://blog.csdn.net/u011080472/article/details/51337422>

<https://blog.csdn.net/zl1zl2zl3/article/details/53693658>

JMM：Java Memory Model(Java内存模型)

可见性：JMM提供了volatile变量定义、final、synchronized块来保证可见性。

原子性：JMM提供保证了访问基本数据类型的原子性（其实在写一个工作内存变量到主内存是分主要两步：store、write），但是实际业务处理场景往往是需要更大的范围的原子性保证，所以模型也提供了synchronized块来保证

有序性：这个概念是相对而言的，如果在本线程内，所有的操作都是有序的，如果在一个线程观察另一个线程，所有的操作都是无序的，前句是“线程内表现为串行行为”，后句是“指令的重排序”和“工作内存和主内存同步延迟”现象，模型提供了volatile和synchronized来保证线程之间操作的有序性。

重排序：在执行程序时为了提高性能，编译器和处理器常常会对指令做重排序(编译器、处理器)，就是因为这些重排序，所以可能会导致多线程程序出现内存可见性问题(数据安全问题)和有序性问题。

# 多线程

## AtomicInteger底层实现原理；

Unsafe/CAS

## synchronized与ReentraLock哪个是公平锁；

synchronized和ReentrantLock的区别

除了synchronized的功能，多了三个高级功能。

等待可中断，公平锁，绑定多个Condition。

1. 等待可中断：在持有锁的线程长时间不释放锁的时候，等待的线程可以选择放弃等待，tryLock(long timeout, TimeUnit unit)

2. 公平锁：按照申请锁的顺序来一次获得锁称为公平锁，synchronized的是非公平锁，ReentrantLock可以通过构造函数实现公平锁。new RenentrantLock(boolean fair)

3. 绑定多个Condition：通过多次newCondition可以获得多个Condition对象，可以简单的实现比较负责的线程同步的功能，通过await(),signal();

## CAS机制会出现什么问题；

<https://www.cnblogs.com/Leo_wl/p/6899716.html>

<https://www.cnblogs.com/winner-0715/p/7270805.html>

ABA

忙等待

总线加锁或者缓存加锁。

总线加锁：总线加锁就是就是使用处理器提供的一个LOCK#信号，当一个处理器在总线上输出此信号时，其他处理器的请求将被阻塞住,那么该处理器可以独占使用共享内存。但是这种处理方式显得有点儿霸道，不厚道，他把CPU和内存之间的通信锁住了，在锁定期间，其他处理器都不能其他内存地址的数据，其开销有点儿大。所以就有了缓存加锁。

缓存加锁：其实针对于上面那种情况我们只需要保证在同一时刻对某个内存地址的操作是原子性的即可。缓存加锁就是缓存在内存区域的数据如果在加锁期间，当它执行锁操作写回内存时，处理器不在输出LOCK#信号，而是修改内部的内存地址，利用缓存一致性协议来保证原子性。缓存一致性机制可以保证同一个内存区域的数据仅能被一个处理器修改，也就是说当CPU1修改缓存行中的i时使用缓存锁定，那么CPU2就不能同时缓存了i的缓存行。

## 用过并发包下边的哪些类；

原子类

锁

同步工具

队列

容器

线程池

Future/Callable

## 一个线程连着调用start两次会出现什么情况？

IllegalThreadStateException

## wait方法能不能被重写，wait能不能被中断；

<https://blog.csdn.net/lingzhm/article/details/44940823>

Final方法

## 线程池的实现？四种线程池？重要参数及原理？任务拒接策略有哪几种？

<https://www.cnblogs.com/sessionbest/articles/8689220.html>

<https://blog.csdn.net/honghailiang888/article/details/51690711>

corePoolSize、maximumPoolSize、keepAliveTime、unit、workQueue、threadFactory、handler

Java通过Executors提供四种线程池，分别为：

newCachedThreadPool创建一个可缓存线程池，如果线程池长度超过处理需要，可灵活回收空闲线程，若无可回收，则新建线程。

newFixedThreadPool 创建一个定长线程池，可控制线程最大并发数，超出的线程会在队列中等待。

newScheduledThreadPool 创建一个定长线程池，支持定时及周期性任务执行。

newSingleThreadExecutor 创建一个单线程化的线程池，它只会用唯一的工作线程来执行任务，保证所有任务按照指定顺序(FIFO, LIFO, 优先级)执行。

handler：表示当拒绝处理任务时的策略，有以下四种取值：

ThreadPoolExecutor.AbortPolicy:丢弃任务并抛出RejectedExecutionException异常。

ThreadPoolExecutor.DiscardPolicy：也是丢弃任务，但是不抛出异常。

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

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

### 线程状态以及API怎么操作会发生这种转换；

<https://www.cnblogs.com/newcj/archive/2011/07/21/2112760.html>

Sleep、Yield

Join、Interrupt

Wait

### 常用的避免死锁方法；

<https://blog.csdn.net/bxyill/article/details/8237339>

<https://blog.csdn.net/li396864285/article/details/51487637>

<https://www.cnblogs.com/flyingeagle/articles/6853167.html>

产生死锁的必要条件：

互斥条件：进程要求对所分配的资源进行排它性控制，即在一段时间内某资源仅为一进程所占用。

请求和保持条件：当进程因请求资源而阻塞时，对已获得的资源保持不放。

不剥夺条件：进程已获得的资源在未使用完之前，不能剥夺，只能在使用完时由自己释放。

环路等待条件：在发生死锁时，必然存在一个进程--资源的环形链。

# JVM

## Minor GC与Full GC分别在什么时候发生？什么时候触发Full GC;

<https://blog.csdn.net/yhyr_ycy/article/details/52566105>

<https://blog.csdn.net/u012500848/article/details/51355404>

Minor GC触发条件：当Eden区满时，触发Minor GC。

Full GC触发条件：

（1）调用System.gc时，系统建议执行Full GC，但是不必然执行

（2）老年代空间不足

（3）方法去空间不足

（4）通过Minor GC后进入老年代的平均大小大于老年代的可用内存

（5）由Eden区、From Space区向To Space区复制时，对象大小大于To Space可用内存，则把该对象转存到老年代，且老年代的可用内存小于该对象大小

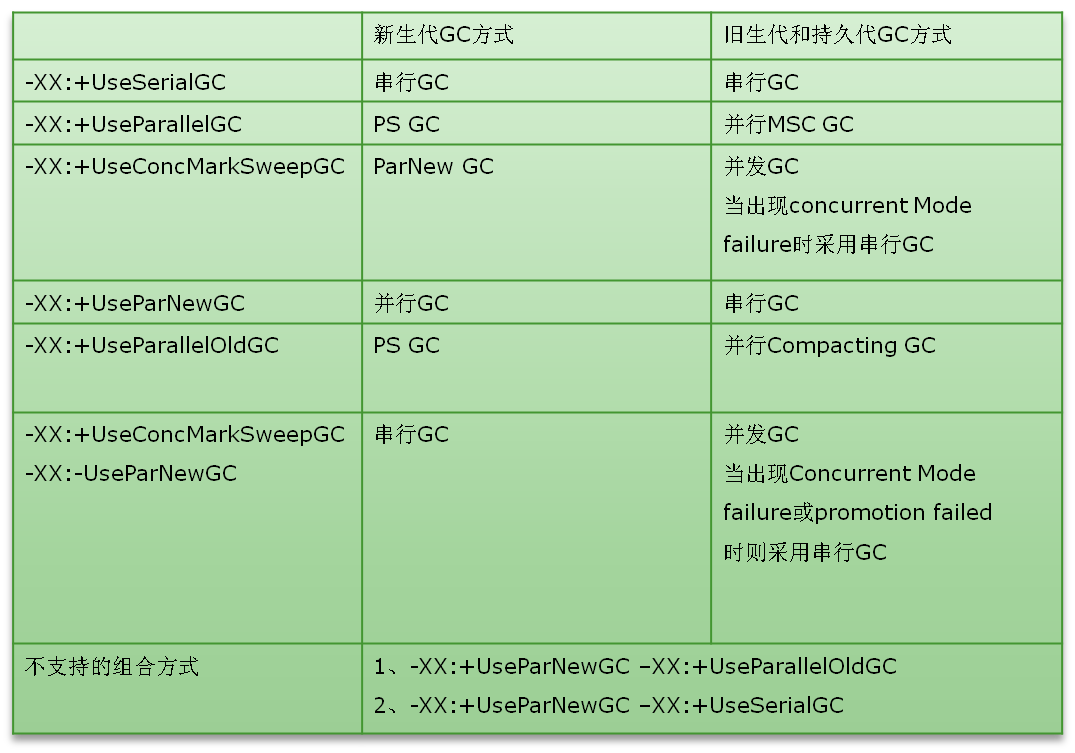
## GC收集器有哪些？CMS收集器与G1收集器的特点。

<https://blog.csdn.net/linhu007/article/details/48897597>

<https://www.jianshu.com/p/bdd6f03923d1>

CMS：老年代

G1：不区分



## Java在什么时候会出现内存泄漏；

<https://blog.csdn.net/sinat_36246371/article/details/54585741>

<https://blog.csdn.net/anxpp/article/details/51325838>

长生命周期的引用了很多短生命周期

## Java中的大对象如何进行存储；

## rt.jar被什么类加载器加载，什么时间加载；

<http://www.importnew.com/21470.html>

<https://www.cnblogs.com/ITtangtang/p/3978102.html>

1、启动类加载器（Bootstrap ClassLoader）：负责加载 JAVA\_HOME\lib 目录中的，或通过-Xbootclasspath参数指定路径中的，且被虚拟机认可（按文件名识别，如rt.jar）的类。

2、扩展类加载器（Extension ClassLoader）：负责加载 JAVA\_HOME\lib\ext 目录中的，或通过java.ext.dirs系统变量指定路径中的类库。

3、应用程序类加载器（Application ClassLoader）：负责加载用户路径（classpath）上的类库。

## 自己写的类被什么加载，什么时间加载；

## 自己写的两个不同的类是被同一个类加载器加载的吗？为什么？

## 为什么新生代内存需要有两个Survivor区？

<https://blog.csdn.net/antony9118/article/details/51425581>

<https://segmentfault.com/q/1010000006886669?_ea=1165966>

<https://segmentfault.com/a/1190000004926898#articleHeader21>

<https://blog.csdn.net/chenleixing/article/details/46706039>

复制算法、内存碎片、性能

## 几种常用的内存调试工具：jmap、jstack、jconsole；

jps:查看虚拟机进程的状况，如进程ID。

jmap: 用于生成堆转储快照文件（某一时刻的）。

jhat：对生成的堆转储快照文件进行分析。

jstack:用来生成线程快照（某一时刻的）。生成线程快照的主要目的是定位线程长时停顿的原因（如死锁，死循环，等待I/O 等），通过查看各个线程的调用堆栈，就可以知道没有响应的线程在后台做了什么或者等待什么资源。

jstat:虚拟机统计信息监视工具。如显示垃圾收集的情况，内存使用的情况。

jconsole:主要是内存监控和线程监控。内存监控：可以显示内存的使用情况。线程监控：遇到线程停顿时，可以使用这个功能。

## 类加载的五个过程：加载、验证、准备、解析、初始化；

1) 加载：根据查找路径找到相应的class文件，然后导入。类的加载方式分为

隐式加载和显示加载两种。隐式加载指的是程序在使用new关键词创建对象时，会隐式的调用类的加载器把对应的类加载到jvm中。显示加载指的是通过直接调用class.forName（）方法来把所需的类加载到jvm中。

2) 检查：检查夹加载的class文件的正确性。

3) 准备；给类中的静态变量分配内存空间。

4) 解析：虚拟机将常量池中的符号引用替换成直接引用的过程。符号引用就理解为一个标示，而在直接引用直接指向内存中的地址。

5) 初始化：对静态变量和静态代码块执行初始化工作。

## G1停顿吗，CMS回收步骤，CMS为什么会停顿，停顿时间；

G1运作步骤：

<https://blog.csdn.net/qqqqq1993qqqqq/article/details/71882733>

1. 初始标记；2、并发标记；3、最终标记；4、筛选回收

可预测的停顿：这是G1相对于CMS的另一个大优势，降低停顿时间是G1和ＣＭＳ共同的关注点，但Ｇ１除了追求低停顿外，还能建立可预测的停顿时间模型，能让使用者明确指定在一个长度为M毫秒的时间片段内，

-XX:MaxGCPauseMillis=200

CMS

<https://blog.csdn.net/mawming/article/details/51967472>

1. 初始标记

2. 并发标记

3. 重新标记 这个阶段会暂停虚拟机，收集器线程扫描在CMS堆中剩余的对象。扫描从"跟对象"开始向下追溯，并处理对象关联。

4. 并发清理

内存碎片的整理

## 栈主要存的数据是什么，堆呢？

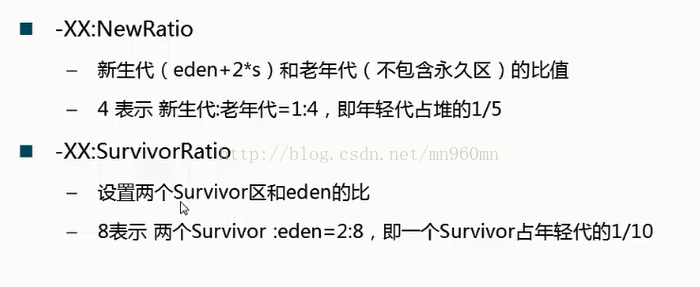
<https://www.cnblogs.com/duzhentong/p/7816545.html>

栈：程序计数器(PC)、线程栈、本地方法栈

## 堆分为哪几块，比如说新生代老生代，那么新生代又分为什么？

新生代（Eden、２个Survivor）

老年代



## 软引用和弱引用的使用场景（软引用可以实现缓存，弱引用可以用来在回调函数中防止内存泄露）；

SoftReference

WeakReference

# 数据库

## 数据库索引，什么是全文索引，全文索引中的倒排索引是什么原理；

<https://blog.csdn.net/softmanfly/article/details/51376886>

非结构化数据又一种叫法叫全文数据。

由于从字符串到文件的映射是文件到字符串映射的反向过程，于是保存这种信息的索引称为反向索引 。

每个字符串都指向包含此字符串的文档(Document)链表，此文档链表称为倒排表 (Posting List)。

## 数据库最佳左前缀原则是什么？

最左前缀原则：顾名思义是最左优先，以最左边的为起点任何连续的索引都能匹配上，

注：如果第一个字段是范围查询需要单独建一个索引

注：在创建多列索引时，要根据业务需求，where子句中使用最频繁的一列放在最左边。

当创建(a,b,c)复合索引时，想要索引生效的话,只能使用 a和a,b和a,b,c三种组合

## 数据库的三大范式；

<https://blog.csdn.net/zymx14/article/details/69789326>

第一范式（1NF）：确保每一列的原子性

第二范式:非键字段必须依赖于键字段

第三范式：在1NF基础上，除了主键以外的其它列都不传递依赖于主键列，或者说： 任何非主属性不依赖于其它非主属性（在2NF基础上消除传递依赖）

## 悲观锁和乐观锁的原理和应用场景；

悲观锁：写多读少

乐观锁：读多写少

## 左连接、右连接、内连接、外连接、交叉连接、笛卡儿积等；

<https://blog.csdn.net/basycia/article/details/52205916>

<https://blog.csdn.net/yl2isoft/article/details/17205413>

## 一般情况下数据库宕机了如何进行恢复（什么是Write Ahead Log机制，什么是Double Write机制，什么是Check Point）；

<https://www.cnblogs.com/ohuang/p/5807543.html>

HBase的Write Ahead Log (WAL)提供了一种高并发、持久化的日志保存与回放机制。每一个业务数据的写入操作（PUT / DELETE）执行前，都会记账在WAL中。

如果出现HBase服务器宕机，则可以从WAL中回放执行之前没有完成的操作。

<https://yq.aliyun.com/articles/80423>

<http://blog.itpub.net/29018063/viewspace-2086098/>

分开2次写

## 什么是redo日志、什么是undo日志；

<https://blog.csdn.net/kobejayandy/article/details/50885693>

Undo日志记录某数据被修改前的值，可以用来在事务失败时进行rollback；Redo日志记录某数据块被修改后的值，可以用来恢复未写入data file的已成功事务更新的数据。

## 数据库中的隔离性是怎样实现的；原子性、一致性、持久性又是如何实现的；

<https://blog.csdn.net/when_less_is_more/article/details/70183327>

<https://www.zhihu.com/question/30272728>

Redo/Undo日志

锁

## 什么是组合索引，组合索引什么时候会失效；

<https://blog.csdn.net/guoxingege/article/details/51034387>

最佳左前缀原则

## 关系型数据库和非关系型数据库区别；

<https://www.cnblogs.com/alilcu/p/8068259.html>

<https://blog.csdn.net/longxingzhiwen/article/details/53896702>

## 数据库死锁如何解决；

<https://blog.csdn.net/qq_16681169/article/details/74784193>

## MySQL并发情况下怎么解决（通过事务、隔离级别、锁）；

<https://www.cnblogs.com/leefreeman/p/8286550.html>

## MySQL中的MVCC机制是什么意思，根据具体场景，MVCC是否有问题；

<https://blog.csdn.net/whoamiyang/article/details/51901888>

<https://www.cnblogs.com/chenpingzhao/p/5065316.html>

InnoDB的MVCC,是通过在每行记录后面保存两个隐藏的列来实现的,这两个列，分别保存了这个行的创建时间，一个保存的是行的删除时间。这里存储的并不是实际的时间值,而是系统版本号(可以理解为事务的ID)，没开始一个新的事务，系统版本号就会自动递增，事务开始时刻的系统版本号会作为事务的ID.

## MySQL数据库的隔离级别，以及如何解决幻读；

<https://www.cnblogs.com/hebao0514/p/5494588.html>

MVCC（多版本并发控制）

# 缓存服务器

## Redis中zSet跳跃表问题；

<https://blog.csdn.net/acceptedxukai/article/details/17333673>

<https://blog.csdn.net/universe_ant/article/details/51134020>

<https://www.cnblogs.com/George1994/p/7635731.html>

<https://blog.csdn.net/u014427196/article/details/52454462>

插入：找到层数

删除：找到位置然后删除，如果没有节点则删除层数

## Redis的set的应用场合？

<https://blog.csdn.net/qq_19943157/article/details/50495925>

<https://www.cnblogs.com/mrhgw/p/6278619.html>

redis常用五种数据类型:string,hash,list,set,zset(sorted set).

## Redis高级特性了解吗？

<https://www.cnblogs.com/logo-fox/p/6473142.html>

RDB/AOF

PUB/SUB

## Redis的pipeline有什么用处？

<https://segmentfault.com/a/1190000011440752>

multi命令代表事务的开始,exec命令代表事务结束,他们之间的命令是原子顺序执行的.

## Redis集群宕机如何处理，怎么样进行数据的迁移；

<https://blog.csdn.net/sanwenyublog/article/details/53453085>

<https://www.cnblogs.com/chenmh/p/5578376.html>

## Redis的集群方案；

<https://blog.csdn.net/sanpo/article/details/52839044>

<https://blog.csdn.net/u010963948/article/details/78961685>

客户端分片

Twemproxy

Codis

Redis官方集群

Redis sentinel

## Redis原子操作怎么用比较好；

<https://segmentfault.com/q/1010000006636934>

## Redis过期策略是怎么实现的呢？

<https://blog.csdn.net/xiangnan129/article/details/54928672>

<https://www.cnblogs.com/xuliangxing/p/7151812.html>

<https://www.jianshu.com/p/8aa619933ebb>

过期策略通常有以下三种：

定时过期：每个设置过期时间的key都需要创建一个定时器，到过期时间就会立即清除。该策略可以立即清除过期的数据，对内存很友好；但是会占用大量的CPU资源去处理过期的数据，从而影响缓存的响应时间和吞吐量。

惰性过期：只有当访问一个key时，才会判断该key是否已过期，过期则清除。该策略可以最大化地节省CPU资源，却对内存非常不友好。极端情况可能出现大量的过期key没有再次被访问，从而不会被清除，占用大量内存。

定期过期：每隔一定的时间，会扫描一定数量的数据库的expires字典中一定数量的key，并清除其中已过期的key。该策略是前两者的一个折中方案。通过调整定时扫描的时间间隔和每次扫描的限定耗时，可以在不同情况下使得CPU和内存资源达到最优的平衡效果。

(expires字典会保存所有设置了过期时间的key的过期时间数据，其中，key是指向键空间中的某个键的指针，value是该键的毫秒精度的UNIX时间戳表示的过期时间。键空间是指该Redis集群中保存的所有键。)

Redis中同时使用了惰性过期和定期过期两种过期策略。

Redis的内存淘汰策略是指在Redis的用于缓存的内存不足时，怎么处理需要新写入且需要申请额外空间的数据。

noeviction：当内存不足以容纳新写入数据时，新写入操作会报错。

allkeys-lru：当内存不足以容纳新写入数据时，在键空间中，移除最近最少使用的key。

allkeys-random：当内存不足以容纳新写入数据时，在键空间中，随机移除某个key。

volatile-lru：当内存不足以容纳新写入数据时，在设置了过期时间的键空间中，移除最近最少使用的key。

volatile-random：当内存不足以容纳新写入数据时，在设置了过期时间的键空间中，随机移除某个key。

volatile-ttl：当内存不足以容纳新写入数据时，在设置了过期时间的键空间中，有更早过期时间的key优先移除。

# SSM相关

## Spring中@Autowired和@Resource注解的区别？

<https://blog.csdn.net/wangzuojia001/article/details/54312074>

<https://blog.csdn.net/littleskey/article/details/52595485>

1、@Autowired

由Spring提供，只按照byType注入

2、@Resource

由J2EE提供，默认按照byName自动注入

@Resource有两个重要的属性：name和type

## Spring声明一个 bean 如何对其进行个性化定制；

<https://blog.csdn.net/u013361010/article/details/52830508>

<bean id="" class="" autowire="" autowire-candidate=""

primary="" lazy-init="" depends-on="" factory-bean="" factory-method="" parent="" abstract="" destroy-method="" init-method="" name="" scope=""/>

<https://www.cnblogs.com/zhanglei93/p/6231882.html>

<https://www.cnblogs.com/redcool/p/6258973.html>

5种作用域：Spring 3中为Bean定义了5中作用域，分别为singleton（单例）、prototype（原型）、request、session和global session

## MyBatis有什么优势；

## MyBatis如何做事务管理；

<https://www.cnblogs.com/xrq730/p/5448656.html>



## Mybatis缓存

<https://www.cnblogs.com/xdp-gacl/p/4270403.html>

MyBatis 同样提供了一级缓存和二级缓存的支持

一级缓存: 基于PerpetualCache 的 HashMap本地缓存，其存储作用域为 Session，当 Session flush 或 close 之后，该Session中的所有 Cache 就将清空。

　　2. 二级缓存与一级缓存其机制相同，默认也是采用 PerpetualCache，HashMap存储，不同在于其存储作用域为 Mapper(Namespace)，并且可自定义存储源，如 Ehcache。

3. 对于缓存数据更新机制，当某一个作用域(一级缓存Session/二级缓存Namespaces)的进行了 C/U/D 操作后，默认该作用域下所有 select 中的缓存将被clear。

<cache

eviction="FIFO" <!--回收策略为先进先出-->

flushInterval="60000" <!--自动刷新时间60s-->

size="512" <!--最多缓存512个引用对象-->

readOnly="true"/> <!--只读-->

# 操作系统

## Linux静态链接和动态链接；

<https://blog.csdn.net/lwhsyit/article/details/2860964>

<https://blog.csdn.net/shreck66/article/details/49583057>

\*.a

\*.so

## 什么是IO多路复用模型（select、poll、epoll）；

<https://www.cnblogs.com/zingp/p/6863170.html>

## Linux中的grep管道用处？Linux的常用命令？

<https://www.cnblogs.com/luntai/p/6232824.html>

grep命令是一种强大的文本搜索工具，它能使用正则表达式搜索文本，并把匹 配的行打印出来。grep全称是Global Regular Expression Print，表示全局正则表达式版本，它的使用权限是所有用户。

## 操作系统中虚拟地址、逻辑地址、线性地址、物理地址的概念及区别；

<https://blog.csdn.net/yusiguyuan/article/details/9664887>

<https://www.cnblogs.com/bhlsheji/p/4868964.html>

<https://www.cnblogs.com/zengkefu/p/5452792.html>

逻辑地址（Logical Address）是指由程序产生的与段相关的偏移地址部分。

线性地址（Linear Address）（虚拟地址） 是逻辑地址到物理地址变换之间的中间层。程序代码会产生逻辑地址，或者说是段中的偏移地址，加上相应段的基地址就生成了一个线性地址。如果启用了分页机制，那么线性地址可以再经变换以产生一个物理地址。若没有启用分页机制，那么线性地址直接就是物理地址。

## 内存的页面置换算法；

<https://www.cnblogs.com/yongheng20/p/4967213.html>

## 进程调度算法，操作系统是如何调度进程的；

<https://blog.csdn.net/fuzhongmin05/article/details/55802925>

时间片轮转法

## 父子进程、孤儿进程、僵死进程等概念；

<https://blog.csdn.net/qq_32437319/article/details/53467089>

<https://www.cnblogs.com/Anker/p/3271773.html>

Fork

== 0说明是子进程

> 0 说明是父进程，返回的是子进程的PID

-1　失败

正常情况下，父进程会收到两个返回值一个是 exit code 也是 SIGCHLD 信号与 reason for termination 之后，父进程会使用 wait(&status) 系统调用以获取子进程的退出状态，然后内核就可以从内存中释放已结束的子进程的 PCB；而如若父进程没有这么做的话，子进程的 PCB 就会一直驻留在内存中，一直留在系统中做为僵尸进程（Zombie）。

而另外如果父进程结束（非正常的结束），未能即使收回子进程，子进程仍在运行，这样的子进程称之为孤儿进程。

## fork进程时的操作；

<https://www.cnblogs.com/wuchanming/p/4495479.html>

父返子，子返0，fork出错返-1

## kill用法，某个进程杀不掉的原因（僵死进程；进入内核态，忽略kill信号）；

<https://blog.csdn.net/aquester/article/details/23339645>

<https://www.cnblogs.com/xubiao/p/6497390.html>

<https://blog.csdn.net/u012349696/article/details/52250640>

<https://blog.csdn.net/lemontree1945/article/details/79169178>

找到父进程

ps -ef | grep ocs | grep -v grep | cut -c 9-15 | xargs kill -s 9

kill -kill pid

## 系统管理命令（如查看内存使用、网络情况）；

查看内存

<https://www.cnblogs.com/sunny3096/p/7289705.html>

<https://blog.csdn.net/CSDN_duomaomao/article/details/77877108>

一、Linux下的任务管理器top：

top

二、内核及发行版：

uname -a

uname -sr

cat /etc/issue

lsb\_release -a

三、CPU硬件：

cat /proc/cpuinfo

四、内存情况：

cat /proc/meminfo

五、磁盘、Swap情况：

fdisk -l

free -h

六、网络情况：

ip a

ifconfig

netstat -tln

netstat -an

ss -l

ss -ta

ss -s

route -n

traceroute www.baidu.cn

nslookup www.baidu.cn

lsof -i:22

## find命令、awk使用；

<http://blog.itpub.net/25724608/viewspace-1049709/>

<http://www.cnblogs.com/hushaojun/p/4851485.html>

<https://www.cnblogs.com/wt645631686/p/6868706.html>

find pathname -options [-print -exec -ok ...]

awk 数据流处理工具

查找出日志文件中的每一列

awk '{print $0}' log.log

查找出文件中的第一列和第七列

awk '{print %$1 "\t"$7}' log.log

~匹配192.168.10.2的ip地址统计。!~为不匹配

cat file | awk '$0 !~ /192.168.10.2/' | grep 'php' |wc -|

## Linux下排查某个死循环的线程；

<https://blog.csdn.net/zb872676223/article/details/48135757>

首先找到进程ID

看看能不能GDB调试

# 网络相关

## 数据链路层是做什么的?

<https://blog.csdn.net/moshenglv/article/details/55667192>

物理层负责将比特流转换成信号，并传递给线缆。既不知道信号会发往何处，也不在意某一位是否会在某处丢失。

数据链路层提供了一种能够正确且稳定地传输比特流的结构，把数据传输到目的地。

数据链路层在意的是，数据是从哪个端口来的，应该送到哪个端口。

（这个端口是指物理交换机上的物理端口，也就是接口）

## 数据链路层的流量控制？

<https://blog.csdn.net/baidu_35534327/article/details/53542020>

## 网络模型的分层、IP和Mac地址在那个层、TCP和HTTP分别在那个层；

<https://blog.csdn.net/hhcrazy12345/article/details/46682223>

OSI模型分为七层。

TCP/IP模型分为四层：应用层（Application）、传输层（Host-to-Host Transport）、互联网层(Internet)、网络接口层(Network Interface)。



## TCP滑动窗口；

<https://blog.csdn.net/wdscq1234/article/details/52444277>

<https://blog.csdn.net/lishanmin11/article/details/77092652>

<https://blog.csdn.net/q1007729991/article/details/70142341>

## TCP为什么可靠；

<https://blog.csdn.net/tengfei_scut/article/details/69360744>

TCP 连接的每一端都必须设有两个窗口——一个发送窗口和一个接收窗口。TCP 的可靠传输机制用字节的序号进行控制。TCP 所有的确认都是基于序号而不是基于报文段。

发送过的数据未收到确认之前必须保留，以便超时重传时使用。发送窗口没收到确认不动，和收到新的确认后前移。

发送缓存用来暂时存放： 发送应用程序传送给发送方 TCP 准备发送的数据；TCP 已发送出但尚未收到确认的数据。

接收缓存用来暂时存放：按序到达的、但尚未被接收应用程序读取的数据； 不按序到达的数据。

##### TCP的同传，拆包与组装包是什么意思；

## Https和Http有什么区别；

## Http 为什么是无状态的；

<https://blog.csdn.net/wu1991924/article/details/8548051>

<https://www.cnblogs.com/Jadie/p/6877392.html>

## TCP三次握手，为什么不是三次，为什么不是四次；

<https://blog.csdn.net/jacklittlepig/article/details/79043578>

## TCP的拥塞控制、流量控制详细说明？

<https://blog.csdn.net/kzq_qmi/article/details/46940463>

发送窗口和接收窗口

拥塞控制：整体网络

## Http1.0和Http2.0的区别；

<https://blog.csdn.net/linsongbin1/article/details/54980801/>

## 两个不同ip地址的计算机之间如何通信；

## 地址解析协议ARP；

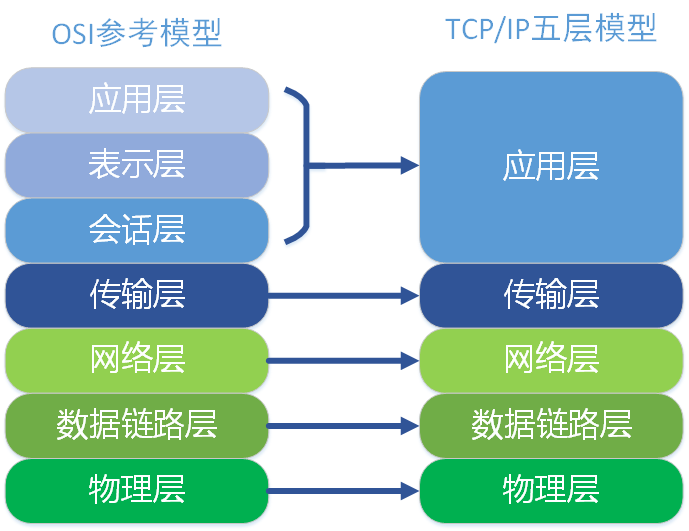
<https://www.cnblogs.com/csguo/p/7542944.html>

地址解析协议

广播

## OSI七层模型分别对应着五层模型的哪一部分；

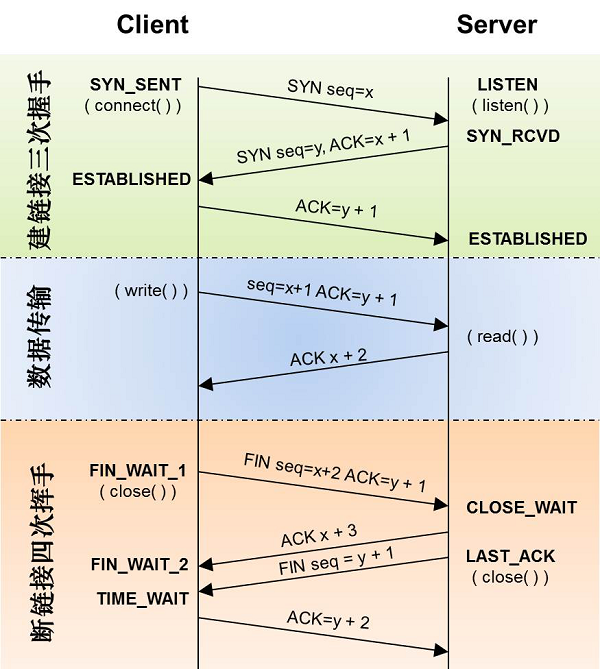
<https://www.cnblogs.com/qishui/p/5428938.html>



## TCP三次握手数据丢失了怎么办？那如果后面又找到了呢？

<https://blog.csdn.net/qq_26222859/article/details/60955713>

<https://blog.csdn.net/changyourmind/article/details/53127100>



# 分布式相关

## 消息队列使用的场景介绍和作用（应用耦合、异步消息、流量削锋等）；

日志处理

异步

## 如何解决消息队列丢失消息和重复消费问题；

<https://blog.csdn.net/yeweiouyang/article/details/74943278>

<http://ju.outofmemory.cn/entry/106889>

<https://www.cnblogs.com/MrZhangL/articles/6004358.html>

点对点、发布订阅

一个消费者

## Kafka使用过吗，什么是幂等性？怎么保证一致性，持久化怎么做，分区partition的理解，LEO是什么意思，如何保证多个partition之间数据一致性的（ISR机制），为什么Kafka可以这么快（基于磁盘的顺序读写）；

<https://blog.csdn.net/tomisaboy/article/details/47109131>

<https://blog.csdn.net/tangdong3415/article/details/53432166>

<https://blog.csdn.net/lizhitao/article/details/51718185>

<https://blog.csdn.net/qq_37502106/article/details/80271800>

一个topic 可以配置几个partition，produce发送的消息分发到不同的partition中，consumer接受数据的时候是按照group来接受，kafka确保每个partition只能同一个group中的同一个consumer消费，如果想要重复消费，那么需要其他的组来消费。

Topic在逻辑上可以被认为是一个queue，每条消费都必须指定它的Topic，可以简单理解为必须指明把这条消息放进哪个queue里。为了使得Kafka的吞吐率可以线性提高，物理上把Topic分成一个或多个Partition，每个Partition在物理上对应一个文件夹，该文件夹下存储这个Partition的所有消息和索引文件

LEO（log end offset）、 ISR（in-sync replicas）

## 异步队列怎么实现；

<https://blog.csdn.net/lzq_csdn_th/article/details/51945408>

## 你项目的并发是多少？怎么解决高并发问题？单机情况下Tomcat的并发大概是多少，MySQL的并发大致是多少？

<https://www.cnblogs.com/linjiqin/p/4430269.html>

<https://blog.csdn.net/q576709166/article/details/27179193>

## 什么是C10K问题；

<https://my.oschina.net/xianggao/blog/664275>

## 高并发情况下怎么办；

<https://www.cnblogs.com/lr393993507/p/5909804.html>

## 分布式理论，什么是CAP理论，什么是Base理论，什么是Paxos理论；

BASE是Basically Available（基本可用）、Soft state（软状态）和Eventually consistent（最终一致性）三个短语的缩写

## 分布式协议的选举算法；

<https://www.cnblogs.com/smartloli/p/7190360.html>

## 说一下你对微服务的理解，与SOA的区别；

<https://www.cnblogs.com/WeiGe/p/5658237.html>

微服务：微型组件

## Dubbo的基本原理，RPC，支持哪些通信方式，服务的调用过程；

<https://blog.csdn.net/TVwR8OfV0P/article/details/78212370?locationNum=4&fps=1>

<https://www.cnblogs.com/1201x/p/6482638.html>

<https://www.cnblogs.com/yangzhilong/p/6121551.html>

dubbo共支持如下几种通信协议：

dubbo://

rmi://

hessian://

http://

webservice://

thrift://

memcached://

redis://

## Dubbo如果有一个服务挂掉了怎么办；

## 分布式事务，操作两个表不在一个库，如何保证一致性。

## 分布式系统中，每台机器如何产生一个唯一的随机值；

<https://www.cnblogs.com/flystar32/p/uuid.html>

## 系统的量级、pv、uv等；

PV(Page View)页面浏览量

UV(Unique Visitor)独立用户

## 什么是Hash一致性算法？分布式缓存的一致性，服务器如何扩容（哈希环）；

## 正向代理、反向代理；

## 什么是客户端负载均衡策略、什么是服务器端负载均衡策略；

<https://blog.csdn.net/u012702547/article/details/77893030>

客户端负载均衡和服务端负载均衡最大的区别在于服务清单所存储的位置。在客户端负载均衡中，所有的客户端节点都有一份自己要访问的服务端清单，这些清单统统都是从Eureka服务注册中心获取的。在Spring Cloud中我们如果想要使用客户端负载均衡，方法很简单，开启@LoadBalanced注解即可，这样客户端在发起请求的时候会先自行选择一个服务端，向该服务端发起请求，从而实现负载均衡。

## 如何优化Tomcat，常见的优化方式有哪些；

<https://blog.csdn.net/qq_16216221/article/details/46529957>

<https://blog.csdn.net/huahuagongzi99999/article/details/52044378>

<https://blog.csdn.net/wangyonglin1123/article/details/50986524>

<https://www.cnblogs.com/xuwc/p/8523681.html>

线程数、线程池

JVM

Tomcat Connector三种运行模式（BIO, NIO, APR）

## Nginx的Master和Worker，Nginx是如何处理请求的；

<https://blog.csdn.net/yankai0219/article/details/8220695>

<http://tengine.taobao.org/book/chapter_02.html#id12>

# 系统设计相关

## 如何防止表单重复提交（Token令牌环等方式）；

<https://blog.csdn.net/myhuashengmi/article/details/52694077>

<https://blog.csdn.net/cor_twi/article/details/48596537>

js禁掉提交按钮

重定向转到其他页面

## 有一个url白名单，需要使用正则表达式进行过滤，但是url量级很大，大概亿级，那么如何优化正则表达式？如何优化亿级的url匹配呢？

<https://blog.csdn.net/t_1007/article/details/52293475>

<http://blog.jobbole.com/111934/>

<http://chuansong.me/n/2212976>

<https://blog.csdn.net/s44359487yad/article/details/80133942>

<http://cdn-qcloud.jianshu.io/p/5448f130b94d>

处理海量数据，不外乎

分而治之/hash映射 + hash统计 + 堆/快速/归并排序

双层桶划分

Bloom filter/Bitmap；

Trie树/数据库/倒排索引；

外排序；

分布式处理之Hadoop/Mapreduce。

## 常见的Nginx负载均衡策略；已有两台Nginx服务器了，倘若这时候再增加一台服务器，采用什么负载均衡算法比较好？

<https://blog.csdn.net/andong154564667/article/details/52121765>

<http://outofmemory.cn/code-snippet/3074/nginx-load-junheng-configuration-jizhong-strategy>

nginx 的 upstream目前支持 4 种方式的分配

1、轮询（默认）

每个请求按时间顺序逐一分配到不同的后端服务器，如果后端服务器down掉，能自动剔除。

2、weight

指定轮询几率，weight和访问比率成正比，用于后端服务器性能不均的情况。

3、ip\_hash

每个请求按访问ip的hash结果分配，这样每个访客固定访问一个后端服务器，可以解决session的问题。

4、fair（第三方）

按后端服务器的响应时间来分配请求，响应时间短的优先分配。

5、url\_hash（第三方）

## 扫描二维码登录的过程解析；

<https://blog.csdn.net/qq_30656761/article/details/52587997>

<https://www.cnblogs.com/otherside/p/8331778.html>

<https://blog.csdn.net/qq_20124743/article/details/78905220>

## 如何设计一个生成唯一UUID的算法？

## 实现一个负载均衡的算法，服务器资源分配为70%、20%、10%；

<https://blog.csdn.net/zgwangbo/article/details/51533657>

<https://blog.csdn.net/u014660247/article/details/78195313>

<https://www.cnblogs.com/xiangzhong/p/5014493.html>

随机

轮询

## 有三个线程T1 T2 T3，如何保证他们按顺序执行；

<https://blog.csdn.net/landebug/article/details/48650479>

<https://blog.csdn.net/answer100answer/article/details/77658097>

<https://blog.csdn.net/difffate/article/details/63684290>

Thread.join

3个CountDownLatch

利用共享变量，状态为1执行１

## 三个线程循环输出ABCABCABC....

# 安全相关

## 什么是XSS攻击，XSS攻击的一般表现形式有哪些？如何防止XSS攻击；

<https://blog.csdn.net/qq_21956483/article/details/54377947>

<

>

‘

&

/

HtmlUtils.htmlEscape(value);