1. JVM结构原理、GC工作机制是什么？

类加载器（ClassLoader）.执行引擎：负责执行class文件中包含的字节码指令.内存区（也叫运行时数据区）：是在JVM运行的时候操作所分配的内存区

1.标记-清除（Mark-sweep）2.复制（Copying）3.标记-整理（Mark-Compact）

4.分代收集算法

1. [spring中BeanFactory和FactoryBean的区别](https://www.cnblogs.com/xingzc/p/9138256.html)

BeanFactory是个Factory，也就是IOC容器或对象工厂，FactoryBean是个Bean。在Spring中，所有的Bean都是由BeanFactory(也就是IOC容器)来进行管理的。但对FactoryBean而言，这个Bean不是简单的Bean，而是一个能生产或者修饰对象生成的工厂Bean,它的实现与设计模式中的工厂模式和修饰器模式类似

1. Map或者HashMap的存储和实现原理？线程安全吗？如何使其线程安全？有什么其他方案？

数据结构中有数组和链表+红黑树来实现对数据的存储，HashMap线程不安全，使用HashTable或ConcurrentHashMap线程安全（推荐）；或业务实现Synchronized

1. 假设商户表A（id , city ）  ,交易流水表B （aid, amount , time）   这里的time代表交易时间， 请用sql写出查询每个城市每个月的销售业绩

SELECT DATE\_FORMAT( FROM\_UNIXTIME(b.time), '%Y-%m' ) AS mtime, a.city, SUM(b.amount) FROM A AS a JOIN B AS b ON a.id = b.aid GROUP BY a.city, mtime  
group by city

1. hibernate跟Mybatis/ ibatis 的区别，为什么选择？

Hibernate的二级缓存配置在SessionFactory生成的配置文件中进行详细配置，然后再在具体的表-对象映射中配置是那种缓存。MyBatis的二级缓存配置都是在每个具体的表-对象映射中进行详细配置，这样针对不同的表可以自定义不同的缓存机制。并且Mybatis可以在命名空间中共享相同的缓存配置和实例，通过Cache-ref来实现

两者相同点

* Hibernate与MyBatis都可以是通过SessionFactoryBuider由XML配置文件生成SessionFactory，然后由SessionFactory 生成Session，最后由Session来开启执行事务和SQL语句。其中SessionFactoryBuider，SessionFactory，Session的生命周期都是差不多的。
* Hibernate和MyBatis都支持JDBC和JTA事务处理。

## Mybatis优势

MyBatis可以进行更为细致的SQL优化，可以减少查询字段。

MyBatis容易掌握，而Hibernate门槛较高。

## Hibernate优势

Hibernate的DAO层开发比MyBatis简单，Mybatis需要维护SQL和结果映射。

Hibernate对对象的维护和缓存要比MyBatis好，对增删改查的对象的维护要方便。

Hibernate数据库移植性很好，MyBatis的数据库移植性不好，不同的数据库需要写不同SQL。

Hibernate有更好的二级缓存机制，可以使用第三方缓存。MyBatis本身提供的缓存机制不佳。

1. Spring 事务的隔离性，并说说每个隔离性的区别

@Transactional(isolation = Isolation.READ\_UNCOMMITTED)

读取未提交数据(会出现脏读, 不可重复读) 基本不使用

@Transactional(isolation = Isolation.READ\_COMMITTED)

读取已提交数据(会出现不可重复读和幻读)

@Transactional(isolation = Isolation.REPEATABLE\_READ)

可重复读(会出现幻读)

@Transactional(isolation = Isolation.SERIALIZABLE)

串行化

MYSQL: 默认为REPEATABLE\_READ级别

SQLSERVER: 默认为READ\_COMMITTED

脏读 : 一个事务读取到另一事务未提交的更新数据

不可重复读 : 在同一事务中, 多次读取同一数据返回的结果有所不同, 换句话说,

后续读取可以读到另一事务已提交的更新数据. 相反, "可重复读"在同一事务中多次

读取数据时, 能够保证所读数据一样, 也就是后续读取不能读到另一事务已提交的更新数据

幻读 : 一个事务读到另一个事务已提交的insert数据

1. 重写equals为何要重写hashCode

1.使用hashcode方法提前校验，可以避免每一次比对都调用equals方法，提高效率

2.保证是同一个对象，如果重写了equals方法，而没有重写hashcode方法，会出现equals相等的对象，hashcode不相等的情况，重写hashcode方法就是为了避免这种情况的出现

1. 简单说说线程池的原理和实现

1、线程池管理器（ThreadPool）：用于创建并管理线程池，包括 创建线程池，销毁线程池，添加新任务；  
2、工作线程（PoolWorker）：线程池中线程，在没有任务时处于等待状态，可以循环的执行任务；  
3、任务接口（Task）：每个任务必须实现的接口，以供工作线程调度任务的执行，它主要规定了任务的入口，任务执行完后的收尾工作，任务的执行状态等；  
4、任务队列（taskQueue）：用于存放没有处理的任务。提供一种缓冲机制。

1. spring cloud 的核心组件有哪些？简单描述一下各组件的功能

Eureka：服务注册于发现。

Feign：基于动态代理机制，根据注解和选择的机器，拼接请求 url 地址，发起请求。

Ribbon：实现负载均衡，从一个服务的多台机器中选择一台。

Hystrix：提供线程池，不同的服务走不同的线程池，实现了不同服务调用的隔离，避免了服务雪崩的问题。

Zuul：网关管理，由 Zuul 网关转发请求给对应的服务。

1. 写出打印如下图案（菱形）

\*

\* \*

\* \*

\* \*

\* \*

\* \*

\*

//打印给定行数的空心菱形

public static void draw(int size){

if (size % 2 == 0) //如果是偶数行变为奇数

{

size++;

}

//中心点:2\*size -1 ,实现的关键在于找准中心点,下面以打印7行的空心菱形为例进行分析

for (int i = 0; i < size/2+1; i++)//打印菱形上面的4行

{

for (int j = 0; j < 2\*size - 1; j++)//每一行的总长度是2\*size -1

{

if (j == (size -1-2\*i) || j ==(size -1 + 2\*i))//中心点左右两边对称的位置打印 \* 号

{

System.out.print("\*");

}

else

{

System.out.print(" ");

}

}

System.out.println();

}

for (int i = size/2-1; i >= 0; i--)//打印菱形下面的3行

{

for (int j = 0; j < 2\*size - 1; j++)

{

if (j == (size -1-2\*i) || j ==(size -1 + 2\*i))//中心点左右两边对称的位置打印 \* 号

{

System.out.print("\*");

}

else

{

System.out.print(" ");

}

}

System.out.println();

}

}

1. 为什么要使用Redis，Redis有哪些优缺点？Redis如何实现扩容？

Redis支持数据的持久化，可以将内存中的数据保持在磁盘中，重启的时候可以再次加载进行使用。  
Redis不仅仅支zd持简单的key-value类型的数据，同时还提供list，set，zset，hash等数据结构的存储。  
Redis支持数版据的备份，即master-slave模式的数据备份。  
Redis 优势  
性能极高 – Redis能读的速度是110000次/s,写的速度是81000次/s 。  
丰富的数据类型 – Redis支持二进制案例的 Strings, Lists, Hashes, Sets 及 Ordered Sets 数据类型操作。  
原子 – Redis的所有操作都是原子性的，同时Redis还支持对几权个操作全并后的原子性执行。  
丰富的特性 – Redis还支持 publish/subscribe, 通知, key 过期等等特性

扩容搭建一个集群+对Redis的读写分离

1. 为什么要使用Spring，Spring的IOC容器初始化流程是什么？IOC容器实现原理？

第一个过程是Resource资源定位。这个Resouce指的是BeanDefinition的资源定位。这个过程就是容器找数据的过程，就像水桶装水需要先找到水一样。

第二个过程是BeanDefinition的载入过程。这个载入过程是把用户定义好的Bean表示成Ioc容器内部的数据结构，而这个容器内部的数据结构就是BeanDefition。

第三个过程是向IOC容器注册这些BeanDefinition的过程，这个过程就是将前面的BeanDefition保存到HashMap中的过程。

1. 大数据量的导出，有什么方案？怎么优化？

用户在前台查询后点击导出，根据数据量判断是否大于最大导出上限

如果小于等于上限，直接按原有模式，直接同步导出；

如果大于上限，提示文件在生成中，稍后去页面获取并下载；

定时任务处理导出任务，生成文件；

新增查询界面，页面显示文件生成进度，如果生成完成，提供下载链接 允许用户下载；

未避免服务器上文件堆积，定期清理文件，每天清理一次3个月前的数据；

1. 并发包下的lock底层实现、AQS、乐观锁、悲观锁理解

　悲观锁：总是假设最坏的情况，每次去拿数据的时候都认为别人会修改，所以每次在拿数据的时候都会上锁，这样别人想拿这个数据就会阻塞直到它拿到锁。传统的关系型数据库里边就用到了很多这种锁机制，比如行锁，表锁等，读锁，写锁等，都是在做操作之前先上锁。再比如Java里面的同步原语synchronized关键字的实现也是悲观锁。

　　乐观锁：顾名思义，就是很乐观，每次去拿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在此期间别人有没有去更新这个数据，可以使用版本号等机制。乐观锁适用于多读的应用类型，这样可以提高吞吐量，像数据库提供的类似于write\_condition机制，其实都是提供的乐观锁。在Java中java.util.concurrent.atomic包下面的原子变量类就是使用了乐观锁的一种实现方式CAS实现的

1. 内存溢出和内存泄露的区别？如何排查出内存问题？如何监控内存问题？

 系统已经不能再分配出你所需要的空间，比如你需要100M的空间，系统只剩90M了，这就叫内存溢出

意思就是你用资源的时候为他开辟了一段空间，当你用完时忘记释放资源了，这时内存还被占用着，一次没关系，但是内存泄漏次数多了就会导致内存溢出

 1、检查对数据库查询中，是否有一次获得全部数据的查询。一般来说，如果一次取十万条记录到内存，就可能引起内存溢出。这个问题比较隐蔽，在上线前，数据库中数据较少，不容易出问题，上线后，数据库中数据多了，一次查询就有可能引起内存溢出。因此对于数据库查询尽量采用分页的方式查询。

　　　　　 2、检查代码中是否有死循环或递归调用。

　　　　　 3、检查是否有大循环重复产生新对象实体。

　　　　　 4、检查List、MAP等集合对象是否有使用完后，未清除的问题。List、MAP等集合对象会始终存有对对象的 引用，使得这些对象不能被GC回收

1. 启用GC监控的方法：增加java启动参数-verbose:gc  
2. 将GC日志输出到文件：不同JDK设置的参数不同，参考JDK官方文档  
   SUN：-Xloggc:filename （例如：-Xloggc:D:/gc.log）  
   IBM：-Xverbosegc:file=filename 或 -Xverbosegclog:filename  
   HP ：-Xverbosegc=filename

3. 如何设置Java启动参数：有多种方式，以下各举一例  
   Tomcat：在catalina.bat的“set JAVA\_OPTS=%JAVA\_OPTS% ”后设置  
   WebLogic：在startWebLogic.cmd的“%JAVA\_HOME%\bin\java %JAVA\_VM% %MEM\_ARGS% %JAVA\_OPTIONS% ”后设置  
   WebSphere：进入管理控制台，应用服务器->进程定义->Java虚拟机高级定义

4. GC日志的图形分析工具：HP的jtune

1. 新浪微博是如何实现把微博消息推送给订阅者？

对每个用户维护一个缓存，类似每个用户都有个邮箱来保存消息。当用户推送新的消息时，查询其关注者，将每条消息插入到每个关注者的缓存中。因为已经预先将结果取出，之后访问的时候性能非常快。

设计到消息队列、推送、Redis、CDN、集群等等

1. Google是如何在一秒内把搜索结果返回给用户的？

1、用户发出搜索请求。谷歌搜索质量工程师 PatrickRiley 说：在大多数搜索中，你的搜索处于多个并行的控制过程或谷歌实验室的创新项目组过程中，可以说每一个查询请求都会参与一些谷歌的创意实验。

2、谷歌会对用户输入的关键词提供一些建议。  
3、谷歌会用同义词匹配与你的搜索关键词语义相近的查询结果。  
4、生成初步的查询结果，虽然谷歌宣称可以找到成千上万的相关结果，但一般只显示不到1000条，同时查询结果将被进行本地化处理，本土站点在查询结果中优先出现。

1、对查询结果按权威性和 PageRank 进行排序，重复的查询结果被剔除。此时的查询结果已接近最终形态，在这一基础上，有两个进程将分开进行–查询结果优化和为其匹配相应的广告。我们先来看查询结果优化。  
对查询结果进行过滤处理  
  
      2、对查询结果进行过滤处理。包括以下几点：  
           2.1、对通常的查询，谷歌会把相关的专题性垂直搜索结果(比如新闻、购物、视频、书籍、地图等)也加到返回的查询结果中。  
           2.2、个性化，用户访问过的网站在查询结果列表中会更靠上  
           2.3、大量使用锚点的网站有可能被从查询结果中删除  
           2.4、如果网页被其他高 PageRank 的网站引用，则网页的重要性会大大提高。  
           2.5、趋势分析：对搜索流量爆增或有大量新闻的搜索关键词，谷歌会在新的查询结果中增加额外的 PageRank 权值。  
           2.6、同一个域名下的多个网页如果具有相同的 PageRank 会被归为一组。  
           2.7、查询结果最终形成(将与广告一同显示)

1. 12306网站的订票系统如何实现？如何保证票不会被超卖？
2. 如何实现一个秒杀系统？怎么设计？

设计思路：将请求拦截在系统上游，降低下游压力。在一个并发量大，实际需求小的系统中，应当尽量在前端拦截无效流量，降低下游服务器和数据库的压力，不然很可能造成数据库读写锁冲突，甚至导致死锁，最终请求超时。

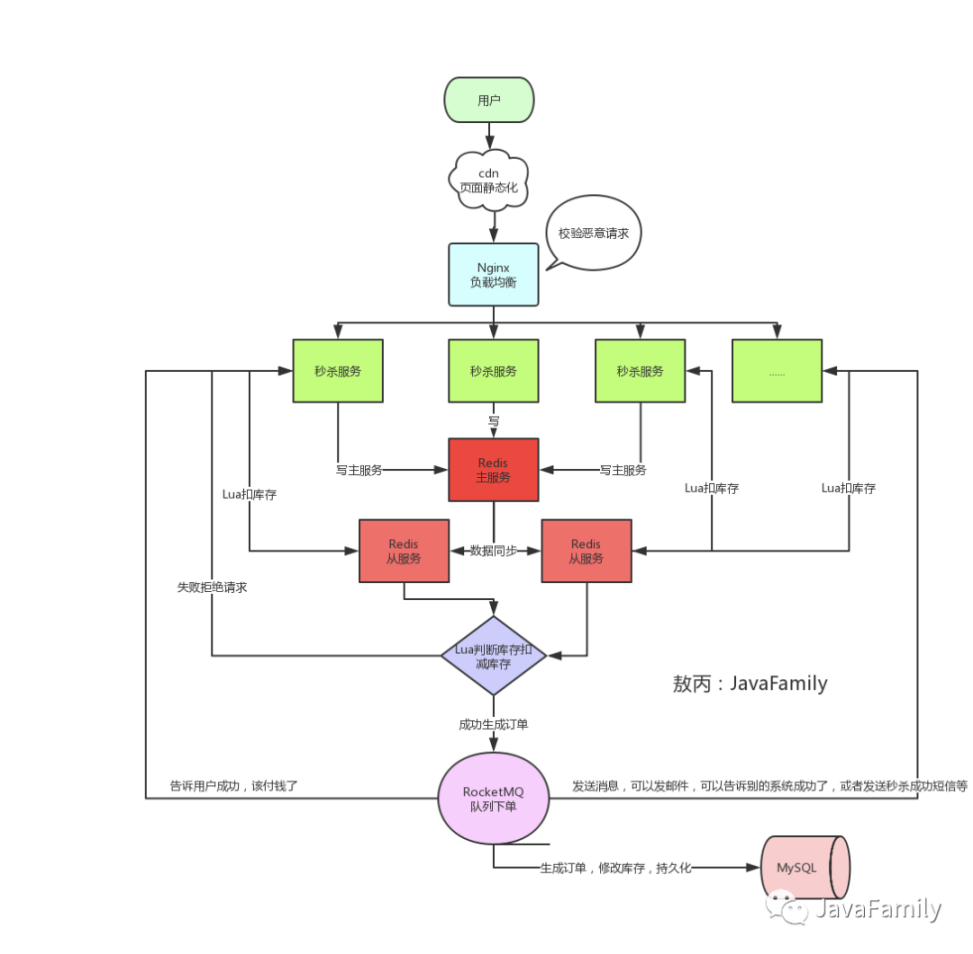
限流：前端直接限流，允许少部分流量流向后端。

削峰：瞬时大流量峰值容易压垮系统，解决这个问题是重中之重。常用的消峰方法有异步处理、缓存和消息中间件等技术。

异步处理：秒杀系统是一个高并发系统，采用异步处理模式可以极大地提高系统并发量，其实异步处理就是削峰的一种实现方式。

内存缓存：秒杀系统最大的瓶颈一般都是数据库读写，由于数据库读写属于磁盘IO，性能很低，如果能够把部分数据或业务逻辑转移到内存缓存，效率会有极大地提升。

消息队列：消息队列可以削峰，将拦截大量并发请求，这也是一个异步处理过程，后台业务根据自己的处理能力，从消息队列中主动的拉取请求消息进行业务处理。

可拓展：当然如果我们想支持更多用户，更大的并发，最好就将系统设计成弹性可拓展的，如果流量来了，拓展机器就好了，像淘宝、京东等双十一活动时会临时增加大量机器应对交易高峰。

1. 如何实现分布式锁？

1.锁没有到期时间。一旦解锁操作失败，锁记录将始终在数据库中，其他线程无法再获取锁。

2.锁是非阻塞的，如果插入失败，数据的插入操作将直接报告错误。未获取锁定的线程不会进入队列。如果要再次获取锁定，它将再次触发锁定操作。

3.锁不是可重入的，同一个线程无法在不释放锁的情况下再次获得锁。

实现分布式锁的步骤

1.获取锁 lock

2.释放锁 unlock

实现分布式锁

1.zookeeper实现分布式锁（推荐）

利用zookeeper的唯一节点特性或者有序临时节点特性获得最小节点作为锁. zookeeper 的实现相对简单，通过curator客户端，已经对锁的操作进行了封装，原理如下

zookeeper的优势

1） 可靠性高、实现简单

2）zookeeper因为临时节点的特性，如果因为其他客户端因为异常和zookeeper连接中断了，那么节点会被删除，意味着锁会被自动释放

3）. zookeeper本身提供了一套很好的集群方案，比较稳定

4）. 释放锁操作，会有watch通知机制，也就是服务器端会主动发送消息给客户端这个锁已经被释放了基于缓存

2.redis实现的分布式锁（推荐使用）

redis中有一个setNx命令，这个命令只有在key不存在的情况下为key设置值。所以可以利用这个特性来实现分布式锁的操作。

加锁 setnx key value

解锁 delete key