# JavaSE

https://www.cnblogs.com/chanshuyi/p/4998776.html

## **HashMap的put 方法是怎么实现的,get方法是怎么实现的?**

HashMap的底层源码实现put方法：

现根据key的hashCode（计算hash值的方法:int hash = hash(key.hashCode())，此方法加了高位运算，以防止hash冲突）重新计算hash值，然后再根据该hash值得到这个元素在数组中的位置（得到该hash值所对应table中索引的方法：int i = indexFor(hash,table.length)）（即下标）。如果该位置上没有元素，就直接将该元素放到此数组中的该位置上。若该位置上已经存放了其他元素了，则在该位置上的元素将以链表的形式存放，新加入的放在链表头，最先加入的放在链表尾。（把元素放到该索引位置上的方法：addEntry(hash,key,value,i)）。

HashMap的底层源码实现get方法：

从HashMap中get元素时，首先根据key的hashCode()计算key的hash值（int hash = hash(key.hashCode())），找到数组中对应位置的某一元素（table[i] = indexFor(hash,table.length)），然后通过key的equals方法在对应位置的链表中找到需要的元素的值。

## **hashMap和HashSet的区别**

HashSet底层调用的是HashMap，区别如下：

|  |  |
| --- | --- |
| \*HashMap\* | \*HashSet\* |
| HashMap实现了Map接口 | HashSet实现了Set接口 |
| HashMap储存键值对 | HashSet仅仅存储对象 |
| 使用put()方法将元素放入map中 | 使用add()方法将元素放入set中 |
| HashMap中使用键对象来计算hashcode值 | HashSet使用成员对象来计算hashcode值，对于两个对象来说hashcode可能相同，所以equals()方法用来判断对象的相等性，如果两个对象不同的话，那么返回false |
| HashMap比较快，因为是使用唯一的键来获取对象 | HashSet较HashMap来说比较慢 |

## **HashMap 的数据结构；线程安全么？为什么？**

　答：不安全，可能造成死循环，具体表现链表的循环指向；应该使用ConcurrentHashMap。

## **hashMap 实现原理**

HashMap的主干是一个Entry数组。Entry是HashMap的基本组成单元，每一个Entry包含一个key-value键值对。HashMap由数组+链表组成的，数组是HashMap的主体，链表则是主要为了解决哈希冲突而存在的，如果定位到的数组位置不含链表（当前entry的next指向null）,那么对于查找，添加等操作很快，仅需一次寻址即可；如果定位到的数组包含链表，对于添加操作，其时间复杂度为O(n)，首先遍历链表，存在即覆盖，否则新增；对于查找操作来讲，仍需遍历链表，然后通过key对象的equals方法逐一比对查找。所以，性能考虑，HashMap中的链表出现越少，性能才会越好。

## JDK1.8 HashMap有哪些优化

HashMap实际上是一个“链表散列”的数据结构即数组和链表的结构 HashMap就是一个链表.

Node是HashMap的一个内部类，实现了Map.Entry接口，本质是就是一个映射(键值对)。

有时两个key会定位到相同的位置，表示发生了Hash碰撞。当然Hash算法计算结果越分散均匀，Hash碰撞的概率就越小，map的存取效率就会越高。

HashMap类中有一个非常重要的字段，就是 Node[] table，即哈希桶数组，明显它是一个Node的数组。如果哈希桶数组很大，即使较差的Hash算法也会比较分散，如果哈希桶数组数组很小，即使好的Hash算法也会出现较多碰撞，所以就需要在空间成本和时间成本之间权衡，其实就是在根据实际情况确定哈希桶数组的大小，并在此基础上设计好的hash算法减少Hash碰撞。那么通过什么方式来控制map使得Hash碰撞的概率又小，哈希桶数组（Node[] table）占用空间又少呢？答案就是好的Hash算法和扩容机制。

如果哈希桶数组很大，即使较差的Hash算法也会比较分散，如果哈希桶数组数组很小，即使好的Hash算法也会出现较多碰撞，所以就需要在空间成本和时间成本之间权衡，其实就是在根据实际情况确定哈希桶数组的大小，并在此基础上设计好的hash算法减少Hash碰撞。

这里存在一个问题，即使负载因子和Hash算法设计的再合理，也免不了会出现拉链过长的情况，一旦出现拉链过长，则会严重影响HashMap的性能。于是，在JDK1.8版本中，对数据结构做了进一步的优化，引入了红黑树。而当链表长度太长（默认超过8）时，链表就转换为红黑树，利用红黑树快速增删改查的特点提高HashMap的性能，其中会用到红黑树的插入、删除、查找等算法

## Arraylist 和 LinkedList有什么区别 数据结构有哪些？

ArrayList查询快!LinkedList增删快!

ArrayList底层是一个数组,所以查询快,LinkedList底层是一个链表,所以增删快.

## ArrayList 怎么去重

声明2个ArrayList，分别为listA与listB ，listA为待去重list ，listB 保存去重之后数据 。遍历listA ，然后判断listB中是否包含各个元素，若不包含，把此元素加入到listB中。

利用set集合进行去重

## Set集合去重之后会存在哪些问题

经set去重后，列表会变成乱序，和原列表比较的话，会显示两者不等所以set去重后不能直接比较 再次排一次序就可以了，两个就相等了。

## HashMap和HashTable和ConcurrentMap的区别和联系

HashMap，Hashtable，ConcurrentHashMap的区别

　　a、HashMap是非线程安全的，HashTable是线程安全的。

　　b、HashMap的键和值都允许有null值存在，而HashTable则不行。

　　c、因为线程安全的问题，HashMap效率比HashTable的要高。

　　HashMap：它根据键的hashCode值存储数据，大多数情况下可以直接定位到它的值，因而具有很快的访问速度，但遍历顺序却是不确定的。 HashMap最多只允许一条记录的键为null，允许多条记录的值为null。HashMap非线程安全，即任一时刻可以有多个线程同时写HashMap，可能会导致数据的不一致。如果需要满足线程安全，可以用 Collections的synchronizedMap方法使HashMap具有线程安全的能力，或者使用ConcurrentHashMap。

　　 Hashtable：Hashtable是遗留类，很多映射的常用功能与HashMap类似，不同的是它承自Dictionary类，并且是线程安全的，任一时间只有一个线程能写Hashtable，并发性不如ConcurrentHashMap，因为ConcurrentHashMap引入了分段锁.

## **解决hash碰撞**

1、开防定址法

2、再哈希法

3、链地址法（Java的hashmap的解决办法就是这个）

4、建立一个公共溢出区

## **Java中的队列都有哪些？**

阻塞队列：ArrayBlockingQueue、LinkedBlockingQueue、SynchronousQueue等

普通队列：LinkedList、ArrayDequeue

非阻塞队列：[ConcurrentLinkedQueue](mk:@MSITStore:E:\\各期视频\\小码课堂\\基础班课程\\B大纲\\api&docs\\JDK_API_1_6_zh_CN.CHM::/java/util/../../java/util/concurrent/ConcurrentLinkedQueue.html" \o "java.util.concurrent 中的类)

## 多线程的线程安全问题怎么解决?有哪几种方法?synchronized和Lock都是互斥锁的原理,除了这个,还有什么java代码实现的方案可以解决线程安全的问题?

线程安全问题解决方案：synchronized代码块、synchronized方法、Lock锁

其他解决线程安全问题：synchronized和lock都是悲观锁，还有乐观锁解决线程安全。乐观锁假设不会发生并发冲突，每次不加锁而是假设没有冲突而去完成某项操作，只在提交操作时检查是否违反数据完整性。如果因为冲突失败就重试，直到成功为止。乐观锁的实现有CAS算法。

## 对线程池的了解?线程池的创建方式?

创建方法有两种：

方式1：直接new ThreadPoolExecutor，Executor接口表示线程池，而ThreadPoolExecutor 为Executor接口最常用的实现类。

方式2：使用Executors线程池工具类创建。创建方式又有以下几种：

|  |  |
| --- | --- |
| 方法名 | 方法描述 |
| newFixedThreadPool(int nThreads) | （常用）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。 |
| newWorkStealingPool(int parallelism) | 创建持有足够线程的线程池来支持给定的并行级别，并通过使用多个队列，减少竞争，它需要穿一个并行级别的参数，如果不传，则被设定为默认的CPU数量。  ForkJoinPool支持大任务分解成小任务的线程池 |
| newSingleThreadExecutor() | （常用）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 |
| newCachedThreadPool() | （常用）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。线程可以重复利用执行任务。默认为60s未使用就被终止和移除 |
| newSingleThreadScheduledExecutor() | 创建一个单线程执行程序，它可安排在给定延迟后运行命令或者定期地执行。 |
| newScheduledThreadPool(int corePoolSize) | （常用）创建一个定长线程池，支持定时及周期性任务执行。 |

## 关于多线程中实现线程安全的几种方式

Synchronized 同步代码块 Lock 和UnLock

Lock和Synchronized区别 :

1. lock是一个接口，而synchronized是java的一个关键字，synchronized是内置的语言实现；

2. synchronized在发生异常时候会自动释放占有的锁，因此不会出现死锁；而lock发生异常时候，不会主动释放占有的锁，必须手动unlock来释放锁，可能引起死锁的发生。（所以最好将同步代码块用try catch包起来，finally中写入unlock，避免死锁的发生。）

3. lock等待锁过程中可以用interrupt来终端等待，而synchronized只能等待锁的释放，不能响应中断；

4. lock可以通过trylock来知道有没有获取锁，而synchronized不能；

5. Lock可以提高多个线程进行读操作的效率。（可以通过readwritelock实现读写分离）

## 用过多线程编程吗，线程有哪些启动方式，线程有几种状态

创建并启动线程的6种方式：

1. 继承Thread类创建线程
2. 实现Runnable接口创建线程
3. 使用Callable和FutureTask创建线程
4. 使用线程池，例如用Executor框架
5. Spring实现多线程（底层是线程池）
6. 定时器Timer（底层封装了一个TimerThread对象）

线程有几种状态 : 线程共有6种状态

NEW、RUNNABLE、BLOCKED、WAITING、TIMED\_WAITING、TERMINATED

新建、运行、阻塞、等待、带超时的等待、终止

## Java多线程的开发与项目实际应用

1、后台任务，例如：定时向大量（100w以上）的用户发送邮件；

2、异步处理，例如：发微博、记录日志等；

3、分布式计算

## 对于JVM的的CPU资源占用过高的问题的排查

1.此时发现如果是Java的进程占用过高，并且一直下不来，则排查是什么线程导致占比过高。以图中进程举例，假如发现PID为31357的Java进程占CPU比一直很高，则记录下它的PID

2.查看Java进程里面的线程的占用情况

top -H -p PID端口号

说明：-H 指显示线程，-p 是指定进程

3.通过jstack命令获取占用资源异常的线程栈，可暂时保存到一个文件中查看 jstack 31357 > jstack.31357.log

以上能看到指定线程的堆栈信息。

如果想看到关于线程中的锁的附加信息，可以加一个-l参数

4.吐出的实际日志结果如下jstack -F “PID” > jstack.“PID”.txt

5.显然一直在跑的是19576这个线程，一直在执行EXCEL导出的相关方法，问题就出在这里，下面的任务就是排查这个地方的代码逻辑了。

## Synchronized 和 Static Synchronized 区别

Synchronized是对类的当前实例进行加锁，防止其他线程同时访问该类的该实例的所有synchronized块， 类的两个不同实例就没有这种约束了。那么static synchronized恰好就是要控制类的所有实例的访问了，static synchronized是限制线程同时访问jvm中该类的所有实例同时访问对应的代码快。实际上，在类中某方法或某代码块中有 synchronized，那么在生成一个该类实例后，改类也就有一个监视块，防止线程并发访问改实例synchronized保护快，而static synchronized则是所有该类的实例公用一个监视快了.

## **一个主线程，开 5 个线程同时执行，5 个线程都结束主线程才可以继续执行，怎么做？**

例如：CountDownLatch latch = new CountDownLatch(5) //声明计数器为5个

Thread t = new Thread() {

public void run() {

try {

//TODO 你的应用

} catch (Exception e) {

//TODO 异常处理

}

finally {

latch.countDown(); //这句是关键

}

}

};

t.start();

System.out.println("ok"); //5个线程都跑完后输出

然后让以上操作循环五次(就是说同时开5个线程)，那么这个"ok"就会在等到这5个线程都ok后才会被输出一次。

## **得到线程执行结果的方法有那些？**

1、根据反射原理实现

很多人都习惯在调用线程的时候，通过构造方法给线程传递参数，这里我们在构造方法里传两个参数，一个是类 callback，一个是方法名 method。这样我们在线程的run方法最后 执行callback.getMethod(method).invoke(null); 这样就可能执行你指定的某个类下的某个方法了！

2、Callable+ScheduledThreadPoolExecutor实现

这个是在Java1.5以后，添加了ScheduledThreadPoolExecutor和callable两个组件（接口），ScheduledThreadPoolExecutor大家可能比较熟悉，它的优点就不说了；它用来执行线程进行调度的时候，有一个方法.schedule(Callable<Object> callable, long delay, TimeUnit unit)，注意：这里的参数是Callable而不是Runnable，大家千万不要被它唬住了，其实它和Runnable差不多，都能实现有个线程，只不过Callable可以添加返回值！这正是我们想要的！

所以我们在用SclExecutor调度线程得到返回值 ScheduledFuture后执行get()方法，就可以得到返回值了

## 对网络编程有没有了解过?清楚阻塞IO和非阻塞IO的区别吗?听说过netty框架吗?

进程（线程）IO调用会不会阻塞进程自己。所以这里两个概念是相对调用进程本身状态来讲的。

在阻塞模式下，若从网络流中读取不到指定大小的数据量，阻塞IO就在那里阻塞着。比如，已知后面会有10个字节的数据发过来，但是我现在只收到8个字节，那么当前线程就在那傻傻地等到下一个字节的到来，对，就在那等着，啥事也不做，直到把这10个字节读取完，这才将阻塞放开通行。

在非阻塞模式下，若从网络流中读取不到指定大小的数据量，非阻塞IO就立即通行。比如，已知后面会有10个字节的数据发过来，但是我现在只收到8个字节，那么当前线程就读取这8个字节的数据，读完后就立即返回，等另外两个字节再来的时候再去读取。

netty框架：基于 JAVA NIO 类库的异步非阻塞通信框架。

（https://blog.csdn.net/mn\_kw/article/details/72780507）

## **tcp和udp的区别和应用场景**

1、TCP面向连接（如打电话要先拨号建立连接）;UDP是无连接的，即发送数据之前不需要建立连接

2、TCP提供可靠的服务。也就是说，通过TCP连接传送的数据，无差错，不丢失，不重复，且按序到达;UDP尽最大努力交付，即不保证可靠交付

3、TCP面向字节流，实际上是TCP把数据看成一连串无结构的字节流;UDP是面向报文的

UDP没有拥塞控制，因此网络出现拥塞不会使源主机的发送速率降低（对实时应用很有用，如IP电话，实时视频会议等）  
4、每一条TCP连接只能是点到点的;UDP支持一对一，一对多，多对一和多对多的交互通信

5、TCP首部开销20字节;UDP的首部开销小，只有8个字节

6、TCP的逻辑通信信道是全双工的可靠信道，UDP则是不可靠信道

应用场景：TCP 协议（如文件传输、重要状态的更新等）；反之，则使用 UDP 协议（如视频传输、实时通信等）。

## **有没有了解JVM的技术原理?平时工作有没有接触过**

**JVM工作原理：**

jvm最重要的一个优点就是跨平台，java源文件通过javac命令启动编译器，将java文件编译成class字节码文件，class字节码文件通过java命令启动jvm虚拟机加载class字节码文件。

**内存管理：**

内存分为两类:线程共享和线程私有内存

线程共享内存:堆、方法区、常量池

线程私有内存：java栈、本地方法栈、pc寄存器

执行流程：类加载机制：父类委托模式

垃圾回收：大部分的GC都是采用分代`收集算法的

工作中的使用：jvm调优

有用过就说有用过，没有就说没有，但是有了解过可以怎么调优。

最经典的就是GC调优。（GC调优https://blog.csdn.net/hellozhxy/article/details/80637199）

## **GC垃圾回收的几种算法**

1.标记-清除算法

该算法先标记，后清除，将所有需要回收的算法进行标记，然后清除；这种算法的缺点是：效率比较低；标记清除后会出现大量不连续的内存碎片，这些碎片太多可能会使存储大对象会触发GC回收，造成内存浪费以及时间的消耗。

2.复制算法

复制算法将可用的内存分成两份，每次使用其中一块，当这块回收之后把未回收的复制到另一块内存中，然后把使用的清除。这种算法运行简单，解决了标记-清除算法的碎片问题，但是这种算法代价过高，需要将可用内存缩小一半，对象存活率较高时，需要持续的复制工作，效率比较低。

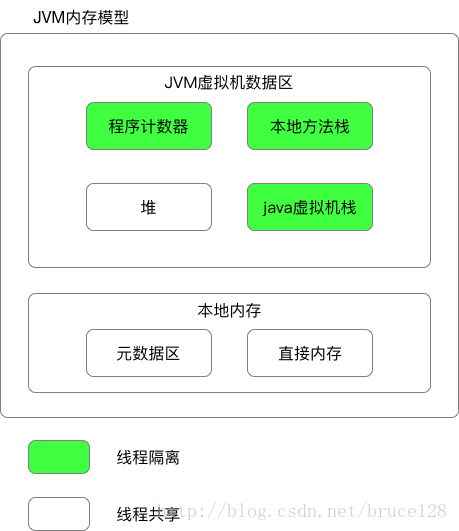
3.标记整理算法

标记整理算法是针对复制算法在对象存活率较高时持续复制导致效率较低的缺点进行改进的，该算法是在标记-清除算法基础上，不直接清理，而是使存活对象往一端游走，然后清除一端边界以外的内存，这样既可以避免不连续空间出现，还可以避免对象存活率较高时的持续复制。这种算法可以避免100%对象存活的极端状况，因此老年代不能直接使用该算法。

4.分代收集算法

分代收集算法就是目前虚拟机使用的回收算法，它解决了标记整理不适用于老年代的问题，将内存分为各个年代，在不同年代使用不同的算法，从而使用最合适的算法，新生代存活率低，可以使用复制算法。而老年代对象存活率搞，没有额外空间对它进行分配担保，所以只能使用标记清除或者标记整理算法。

## **jdk1.8的JVM的有哪几个区域**



1.8同1.7比，最大的差别就是：元数据区取代了永久代。元空间的本质和永久代类似，都是对JVM规范中方法区的实现。不过元空间与永久代之间最大的区别在于：元数据空间并不在虚拟机中，而是使用本地内存。

## **写出一段栈溢出和堆溢出的代码**

栈溢出：

public static void main（String[] args）{

main（args）；

}

堆溢出：

public static void main(String[] args) {

List<byte[]> list = new ArrayList<>();

int i=0;

while(true){

list.add(new byte[5\*1024\*1024]);

System.out.println("分配次数："+(++i));

}

}

## **Java内存泄漏与内存溢出的区别，什么情况下会造成内存溢出？**

内存泄漏memory leak :

是指程序在申请内存后，无法释放已申请的内存空间，一次内存泄漏似乎不会有大的影响，但内存泄漏堆积后的后果就是内存溢出。

内存溢出 out of memory :

指程序申请内存时，没有足够的内存供申请者使用，或者说，给了你一块存储int类型数据的存储空间，但是你却存储long类型的数据，那么结果就是内存不够用，此时就会报错OOM,即所谓的内存溢出。

两者的区别：

内存泄漏的堆积最终会导致内存溢出

内存溢出就是你要的内存空间超过了系统实际分配给你的空间，此时系统相当于没法满足你的需求，就会报内存溢出的错误。

内存泄漏是指你向系统申请分配内存进行使用(new)，可是使用完了以后却不归还(delete)，结果你申请到的那块内存你自己也不能再访问（也许你把它的地址给弄丢了），而系统也不能再次将它分配给需要的程序。就相当于你租了个带钥匙的柜子，你存完东西之后把柜子锁上之后，把钥匙丢了或者没有将钥匙还回去，那么结果就是这个柜子将无法供给任何人使用，也无法被垃圾回收器回收，因为找不到他的任何信息。

内存溢出：一个盘子用尽各种方法只能装4个果子，你装了5个，结果掉倒地上不能吃了。这就是溢出。比方说栈，栈满时再做进栈必定产生空间溢出，叫上溢，栈空时再做退栈也产生空间溢出，称为下溢。就是分配的内存不足以放下数据项序列,称为内存溢出。说白了就是我承受不了那么多，那我就报错。

内存溢出原因：

内存中加载的数据量过于庞大，如一次从数据库取出过多数据；

集合类中有对对象的引用，使用完后未清空，使得JVM不能回收；

代码中存在死循环或循环产生过多重复的对象实体；

使用的第三方软件中的BUG；

启动参数内存值设定的过小

## 常见的排序

我常用的就是 冒泡排序

最简单的一种排序算法。先从数组中找到最大值(或最小值)并放到数组最左端(或最右端)，然后在剩下的数字中找到次大值(或次小值)，以此类推，直到数组有序排列。

算法的时间复杂度为O(n^2)。

## 了解过二叉树吗 那些地方用到过它，能聊聊吗?

二叉树是每个结点最多有两个子树的树结构。通常子树被称作“左子树”和“右子树”。二叉树常被用于实现二叉查找树和二叉堆。一种属树形结构的数据结构

其中的索引其中就包含一种b-tree就是一种树形结构二叉树再排序、查找、大规模数据索引方面有很多很多应用。

## B-tree和B+tree的区别,谈谈平衡二叉树和红黑树

B-tree和B+tree的区别：

B+树是B-树的变体，也是一种多路搜索树，其定义基本与B-树同，区别：

1.非叶子结点的子树指针与关键字个数相同；

2.非叶子结点的子树指针P[i]，指向关键字值属于[K[i],K[i+1])的子树（B-树是开区间）；

3.为所有叶子结点增加一个链指针；在B+Tree的每个叶子节点增加一个指向相邻叶子节点的指针，就形成了带有顺序访问指针的B+Tree；目的：是为了提高区间访问的性能

4.所有关键字都在叶子结点出现

5.B+Tree中叶节点和内节点一般大小不同；虽然B-Tree中不同节点存放的key和指针可能数量不一致，但是每个节点的域和上限是一致的，所以在实现中B-Tree往往对每个节点申请同等大小的空间

平衡二叉树：它是一 棵空树或它的左右两个子树的高度差的绝对值不超过1，并且左右两个子树都是一棵平衡二叉树。构造与调整方法 平衡二叉树的常用算法有红黑树、AVL、[Treap](http://baike.baidu.com/view/956602.htm" \t "https://www.cnblogs.com/guweiwei/p/_blank)等。 最小二叉平衡树的节点的公式如下 F(n)=F(n-1)+F(n-2)+1 这个类似于一个递归的[数列](http://baike.baidu.com/view/39749.htm" \t "https://www.cnblogs.com/guweiwei/p/_blank)，可以参考Fibonacci数列，1是根节点，F(n-1)是左子树的节点数量，F(n-2)是右子树的节点数量。

红黑树在原有的排序二叉树增加了如下几个要求：

性质 1：每个节点要么是红色，要么是黑色。

性质 2：根节点永远是黑色的。

性质 3：所有的叶节点都是空节点（即 null），并且是黑色的。

性质 4：每个红色节点的两个子节点都是黑色。（从每个叶子到根的路径上不会有两个连续的红色节点）

性质 5：从任一节点到其子树中每个叶子节点的路径都包含相同数量的黑色节点。

## 一个非接口的类a和类b，类c怎么能够全部使用到a,b类的功能

这个需求有两个方式：

继承关系：使用多重继承，c继承a，a继承b，通过继承可以将功能继承过来

组合关系：c中包含a和b的对象作为c中的字段。

**注意：**一般我们都使用组合关系来实现，只有a，b，c三者之间有从属关系的时候我们才使用继承关系。

## 抽象类和接口的区别?抽象类能继承实体类吗?

|  |  |  |
| --- | --- | --- |
| 参数 | 抽象类 | 接口 |
| 默认的方法实现 | 它可以有默认的方法实现 | 接口完全是抽象的。它根本不存在方法的实现 |
| 实现 | 子类使用extends关键字来继承抽象类。如果子类不是抽象类的话，它需要提供抽象类中所有声明的方法的实现。 | 子类使用关键字implements来实现接口。它需要提供接口中所有声明的方法的实现 |
| 构造器 | 抽象类可以有构造器 | 接口不能有构造器 |
| 与正常Java类的区别 | 除了你不能实例化抽象类之外，它和普通Java类没有任何区别 | 接口是完全不同的类型 |
| 访问修饰符 | 抽象方法可以有public、protected和default这些修饰符 | 接口方法默认修饰符是public。你不可以使用其它修饰符。 |
| main方法 | 抽象方法可以有main方法并且我们可以运行它 | 接口没有main方法，因此我们不能运行它。 |
| 多继承 | 抽象方法可以继承一个类和实现多个接口 | 接口只可以继承一个或多个其它接口 |
| 速度 | 它比接口速度要快 | 接口是稍微有点慢的，因为它需要时间去寻找在类中实现的方法。 |
| 添加新方法 | 如果你往抽象类中添加新的方法，你可以给它提供默认的实现。因此你不需要改变你现在的代码。 | 如果你往接口中添加方法，那么你必须改变实现该接口的类。 |

抽象类能继承实体类, 但是抽象类不能直接被实例化对象

## **现再有一字符串\*abc\*\*gfe\*\*\*xyz\*，请写一个方法，将\*号移到首部，并不改变其他字母的顺序，即结果为 \*\*\*\*\*\*\*abcgfexyz**

**static** **void** strMove(**char**[] str) {

**int** i = str.length - 1; // 遍历的次数

**int** j = i; // j遍历字符串中的非\*字符

**while** (j > 0 && str[j] != '\*') // 跳过最后面的非'\*'字符

{

j--;

}

**while** (j >= 0 && i >= 0) {

**while** (str[i] != '\*' && i >= 0) {

i--;

}

**while** (str[j] == '\*' && j >= 0) {

j--;

}

// 交换j.i处字符

str[i--] = str[j]; // \*前面的字符后移放到\*的位置

str[j--] = '\*'; // 将\*前移

}

System.***out***.println(str);

}

## **用java代码怎样实现乐观锁的方案?**

表设计时，需要往表里加一个version字段。每次查询时，查出带有version的数据记录，更新数据时，判断数据库里对应id的记录的version是否和查出的version相同。若相同，则更新数据并把版本号+1；若不同，则说明，该数据发送并发，被别的线程使用了，进行递归操作，再次执行递归方法，知道成功更新数据为止。

# JavaWeb

## **MySQL数据库的引擎?**

数据库引擎是用于存储、处理和保护数据的核心服务。利用数据库引擎可控制访问权限并快速处理事务，从而满足企业内大多数需要处理大量数据的应用程序的要求。 使用数据库引擎创建用于联机事务处理或联机分析处理数据的关系数据库。这包括创建用于存储数据的表和用于查看、管理和保护数据安全的数据库对象（如索引、视图和存储过程）。

常用的数据库引擎：Innodb和MyISAM

Innodb引擎

　　Innodb引擎提供了对数据库ACID事务的支持，并且实现了SQL标准的四种隔离级别。该引擎还提供了行级锁和外键约束，它的设计目标是处理大容量数据库系统，它本身其实就是基于MySQL后台的完整数据库系统，MySQL运行时Innodb会在内存中建立缓冲池，用于缓冲数据和索引。当需要使用数据库事务时，该引擎当然是首选。由于锁的粒度更小，写操作不会锁定全表，所以在并发较高时，使用Innodb引擎会提升效率。但是使用行级锁也不是绝对的，如果在执行一个SQL语句时MySQL不能确定要扫描的范围，InnoDB表同样会锁全表。

MyIASM引擎

　　MyIASM是MySQL默认的引擎，但是它没有提供对数据库事务的支持，也不支持行级锁和外键，因此当INSERT(插入)或UPDATE(更新)数据时即写操作需要锁定整个表，效率便会低一些。不过和Innodb不同，MyIASM中存储了表的行数，于是SELECT COUNT(\*) FROM TABLE时只需要直接读取已经保存好的值而不需要进行全表扫描。如果表的读操作远远多于写操作且不需要数据库事务的支持，那么MyIASM也是很好的选择。

## **对于Tomcat容器有没有什么特殊的了解?用Tomcat的时候遇到过什么坑?**

tomcat容器是核心点connecter 连接器 跟container处理容器，connecter用于监听并接收http请求，并分配线程让container对请求进行处理。

## **tomcat可以承受多少的请求?tomcat做连接限制的实现原理?如果用java做限流怎么做(提示:多线程)**

tomcat默认的请求数为100， 可以通过acceptCount参数进行自定义。tomcat8以前一个请求对应一个socket对应一个线程，tomcat做连接限制其实就是对socket创建数量的限制。如果使用java做限流，一般可以在filter中或者拦截器中做限制。

## **JSP的原理**

WEB容器（Servlet引擎）接收到以.jsp为扩展名的URL的访问请求时，它将把该访问请求交给JSP引擎去处理。Tomcat中的JSP引擎就是一个Servlet程序，它负责解释和执行JSP页面。

每个JSP 页面在第一次被访问时，JSP引擎将它翻译成一个Servlet源程序，接着再把这个Servlet源程序编译成Servlet的class类文件，然后再由WEB容器（Servlet引擎）像调用普通Servlet程序一样的方式来装载和解释执行这个由JSP页面翻译成的Servlet程序。

## **注解的原理?**

注解在目前而言最主流的应用：代替配置文件

关于配置文件与注解开发的优缺点：

注解优点：开发效率高 成本低

注解缺点：耦合性大 并且不利于后期维护

使用注解最主要的部分在于对注解的处理，那么就会涉及到注解处理器。从原理上讲，注解处理器就是通过反射机制获取被检查方法上的注解信息，然后根据注解元素的值进行特定的处理。

## **联合索引查询遵守什么原则，如何定义联合索引的顺序**

最左前缀匹配原则 : 在MySQL建立联合索引时会遵循最左前缀匹配的原则，即最左优先，在检索数据时从联合索引的最左边开始匹配

建立组合索引要把经常等值查询的放前面

例如 表中有 name, time 两个字段，name经常被等值查询，time经常被范围查询

## **怎么查看 sql 执行效率，怎么查找慢查询的 sql 语句**

Explain命令在解决数据库性能上是第一推荐使用命令，大部分的性能问题可以通过此命令来简单的解决，Explain可以用来查看 SQL 语句的执行效 果，可以帮助选择更好的索引和优化查询语句，写出更好的优化语句。type：这列最重要，显示了连接使用了哪种类别,有无使用索引，是使用Explain命令分析性能瓶颈的关键项之一。

## **数据库左右链接和内链接的区别**

左连接：左边有的，右边没有的为null

右连接：左边没有的，右边有的为null

内连接：显示左边右边共有的

## **in和like查询可以使用索引么?**

in查询可以使用索引, 对于like查询, 只有不以%开头的时候，才会使用到索引,如果是其他情况就不会使用索引

## **事务的四大特性**

原子性（atomicity）：强调事务的不可分割；

一致性（consistency）：事务的执行前后数据的完整性保持一致；

隔离性（isolation）：一个事务的执行的过程中，不应该受到其他事务的干扰；

持久性（durability）：事务一旦结束，数据就持久到数据库。

## **spring的事务的传播机制**

propagation\_required 如果没有，就开启一个事务；如果有，就加入当前事务（方法B看到自己已经运行在 方法A的事务内部，就不再起新的事务，直接加入方法A）

propagation\_requires\_new 如果没有，就开启一个事务；如果有，就将当前事务挂起。（方法A所在的事务就会挂起，方法B会起一个新的事务，等待方法B的事务完成以后，方法A才继续执行）

propagation \_nested 如果没有，就开启一个事务；如果有，就在当前事务中嵌套其他事务

propagation \_supports 如果没有，就以非事务方式执行；如果有，就加入当前事务（方法B看到自己已经运行在 方法A的事务内部，就不再起新的事务，直接加入方法A）

propagation \_not\_supported 如果没有，就以非事务方式执行；如果有，就将当前事务挂起，（方法A所在的事务就会挂起，而方法B以非事务的状态运行完，再继续方法A的事务）

propagtion\_never 如果没有，就以非事务方式执行；如果有，就抛出异常。

propagtion\_mandatory 如果没有，就抛出异常；如果有，就使用当前事务

## **有一张表t\_login\_log，包含了登录用户user\_id、登录时间login\_time，每次用户登录后都会往表新增一条记录，请用sql找出登录天数最多的前10位用户user\_id。**

select user\_id,count(day\_count.c) days from (

select user\_id,count(d) c from (

select DISTINCT user\_id,DATE\_FORMAT(login\_time,'%Y-%m-%d') d from t\_login\_log

) day\_log GROUP BY user\_id,day\_log.d

) day\_count GROUP BY user\_id ORDER BY days desc limit 0,10

## **servlet、jsp区别？jsp的九大对象，七大动作，三大指令分别是？**

Servlet是服务端的程序，主要为了实现页面中数据的动态输出而存在。使用java代码完成所有数据的输出。

Jsp的本质也是Servlet，是为了解决Servlet中的HTML代码存在的，在jsp中可以编写HTML代码和java代码，但是我们通常不会选择在jsp中编写java代码。

Servlet是服务端的程序，主要为了实现页面中数据的动态输出而存在。使用java代码完成所有数据的输出。

Jsp的本质也是Servlet，是为了解决Servlet中的HTML代码存在的，在jsp中可以编写HTML代码和java代码，但是我们通常不会选择在jsp中编写java代码。

**Jsp的九大内置对象**

1、request对象

request 对象是 javax.servlet.httpServletRequest类型的对象。 该对象代表了客户端的请求信息，主要用于接受通过HTTP协议传送到服务器的数据。（包括头信息、系统信息、请求方式以及请求参数等）。request对象的作用域为一次请求。

2、response对象

response 代表的是对客户端的响应，主要是将JSP容器处理过的对象传回到客户端。response对象也具有作用域，它只在JSP页面内有效。

3、session对象

session 对象是由服务器自动创建的与用户请求相关的对象。服务器为每个用户都生成一个session对象，用于保存该用户的信息，跟踪用户的操作状态。session对象内部使用Map类来保存数据，因此保存数据的格式为 “Key/value”。 session对象的value可以使复杂的对象类型，而不仅仅局限于字符串类型。

4、application对象

 application 对象可将信息保存在服务器中，直到服务器关闭，否则application对象中保存的信息会在整个应用中都有效。与session对象相比，application对象生命周期更长，类似于系统的“全局变量”。

5、out 对象

out 对象用于在Web浏览器内输出信息，并且管理应用服务器上的输出缓冲区。在使用 out 对象输出数据时，可以对数据缓冲区进行操作，及时清除缓冲区中的残余数据，为其他的输出让出缓冲空间。待数据输出完毕后，要及时关闭输出流。

6、pageContext 对象

pageContext 对象的作用是取得任何范围的参数，通过它可以获取 JSP页面的out、request、reponse、session、application 等对象。pageContext对象的创建和初始化都是由容器来完成的，在JSP页面中可以直接使用 pageContext对象。

7、config 对象

config 对象的主要作用是取得服务器的配置信息。通过 pageConext对象的 getServletConfig() 方法可以获取一个config对象。当一个Servlet 初始化时，容器把某些信息通过 config对象传递给这个 Servlet。 开发者可以在web.xml 文件中为应用程序环境中的Servlet程序和JSP页面提供初始化参数。

8、page 对象

page 对象代表JSP本身，只有在JSP页面内才是合法的。 page隐含对象本质上包含当前 Servlet接口引用的变量，类似于Java编程中的 this 指针。

9、exception 对象

exception 对象的作用是显示异常信息，只有在包含 isErrorPage="true" 的页面中才可以被使用，在一般的JSP页面中使用该对象将无法编译JSP文件。excepation对象和Java的所有对象一样，都具有系统提供的继承结构。exception 对象几乎定义了所有异常情况。在Java程序中，可以使用try/catch关键字来处理异常情况； 如果在JSP页面中出现没有捕获到的异常，就会生成 exception 对象，并把 exception 对象传送到在page指令中设定的错误页面中，然后在错误页面中处理相应的 exception 对象。

**Jsp七大动作**

1.include 动态包含(分别编译):  
用jsp:include动作实现<jsp: include page="included.jsp" flush="true" />  
它总是会检查所含文件中的变化，适合用于包含动态页面，并且可以带参数。flush属性: 用true ，表示页面可刷新。默认为false;  
   
2.useBean动作(jsp页面使用javaBean的第二种方式):  
<jsp:useBean id="对象名" class="包名.类名" scope="作用范围(request/page/application/session)"/>  
作用域默认为page(本页面有效).  
3.getProperty动作(name为useBean动作中的id).  
从对象中取出属性值：<jsp:getProperty name="javaBean对象" property="javaBean对象属性名" />  
4.setProperty动作(name为useBean动作中的id):  
为对象设置属性值：<jsp:setProperty name="javaBean对象" property="javaBean对象属性名" value=http://www.hake.cc/kf/201109/"值"/>  
为对象设置属性值:<jsp:setProperty property="javaBean对象属性名" name="javaBean对象" param="username"/>  
(param="username" 相当于 value=http://www.hake.cc/kf/201109/<%=request.getParameter("username")%>)  
   
5.param动作(传递参数)):  
到达跳转页面可以通过 request.getParameter(“参数名”)方式取出参数值  
<jsp:include page="转向页面的url" >  
           <jsp:param   name="参数名1" value=http://www.hake.cc/kf/201109/"参数值1">

<jsp:param   name="参数名2" value=http://www.hake.cc/kf/201109/"参数值2">  
           ...........  
 </jsp:include>  
或:  
<jsp:forward page="转向页面的url" >  
           <jsp:param   name="参数名1" value=http://www.hake.cc/kf/201109/"参数值1">  
           <jsp:param   name="参数名2" value=http://www.hake.cc/kf/201109/"参数值2">  
           ...........  
</jsp:forward>  
   
6.forward动作:  
跳转页面:<jsp:forward page="login.jsp" />  
   
7.plugin动作:<jsp:plugin>:用于指定在客户端运行的插件

**Jsp的三大指令**

**Page:该指令是针对当前页面的指令**

<%@ page [Attribute="Value"] %>

 language:声明当前jsp页面使用的脚本语言的种类

通常是java,默认是java,无需设置

extends:指定jsp编译产生的类继承的父类,或者实现的接口

import:用于导入包,默认已经导入的包有

java.lang.\*,javax.servlet.\*,javax.servlet.jsp.\*,javax.servlet.http.\* session:设定这个jsp页面是否需要HttpSession

buffer:指定输出缓冲区的大小

autoflush:当缓冲区即将溢出时,是否自动强制输出缓冲区的内容

info:设置该JSp程序的信息,采用Servlet.getServletInfo()获取该值

errorPage:指定错误处理页

isErrorPage:设置本JSp是否为错误处理程序

ContentType:用于设定生成网页的文件格式和编码格式

**include:用于指定包含另一个页面**

<%@include file="path"%>

**taglib:用于定义和访问自定义标签**

## **说说tomcat类加载机制。**

tomcat作为一个服务器，在它上面可以部署多个应用。默认情况下是使用1中的应用加载器加载类的，但往往部署的应用都会有多个jar依赖，所以第一点为了解决依赖问题，必须保证每个应用的类库独立。 为什么要保证每个应用的类库相互独立呢？ 打个比方，我们都知道连接数据库需要数据库驱动，而数据库驱动的版本要与连接的数据库相对应，否则无法获取连接。 那假设部署在tomcat上的两个应用依赖的数据库驱动版本有很大差异，这样直接使用java的系统类加载器会导致这两个应用有一个无法启动。

           a)、要保证部署在tomcat上的每个应用依赖的类库相互独立，不受影响。

           b)、由于tomcat是采用java语言编写的，它自身也有类库依赖，为了安全考虑，tomcat使用的类库要与部署的应用的类库相互独立。

           c)、有些类库tomcat与部署的应用可以共享，比如说servlet-api，使用maven编写web程序时，servlet-api的范围是provided，表示打包时不打包这个依赖，因为我们都知道服务器已经有这个依赖了。

           d)、部署的应用之间的类库可以共享。这听起来好像与第一点相互矛盾，但其实这很合理，类被类加载器加载到虚拟机后，会生成代表该类的class对象存放在永久代区域，这时候如果有大量的应用使用spring来管理，如果spring类库不能共享，那每个应用的spring类库都会被加载一次，将会是很大的资源浪费。

           由于存在上述问题，tomcat实现了自己的类加载器，不仅仅是tomcat，所有的服务器基本都有或多或少上述所说的问题。

## **User表（id,name,address），现在要迁移到新表User2(id,address,state)，sql怎么写**

Insert into user2(id,address,state) select id,address,0 from user

## **线上系统在运行，有一张一千万数据的表，怎么做迁移。**

如果不用分区表，大致上就是一个 insert into 历史表 select from 表 where 条件，然后在删除掉表里历史的数据，你可以直接写个定时任务，也可以用数据库里的 定时任务

## **监听器的使用**

监听各大作用域。如监听servletContext,ServletRequest,HttpSession等域对象的创建，更新和销毁等事件。如：1、监听session的创建和销毁，可以统计网站的在线人数。2、监听容器的更新，系统启动后，创建admin账号等.

# 框架

## **Controller是不是单例?为什么这么设计?**

默认是单例的,单例对象接受请求访问的时候是线程安全的,而且不用重复创建,性能高;

## **Spring两大特性的了解?如何应用到业务场景中?**

答: Spring的核心特性就是IOC和AOP，IOC（Inversion of Control），即“控制反转”；AOP（Aspect-OrientedProgramming），即“面向切面编程”;

**IoC（控制反转）**是Spring的一个**容器**，他**不是一种技术**，而**是一种思想**，依旧是基于面向对象编程的。它能指导我们怎么样设计出松耦合、更优良的程序;

简单来说，如果现在有两个（甚至更多）类，A类和B类，A类要引用B类中的某个方法，传统编程是在A类中实例化一个B类，也就是通过new，然后调方法，而我们知道，代码高耦合最大的原因就是用了new。利用Spring框架就将实例化的过程交给了IoC容器，通过配置文件中的设置Bean或者B类中添加注解，A类可以不能new通过ApplicationContext的getBean方法得到实例，然后打点调方法就可以了，减少了两个类之间的耦合度

IoC三种注入方式：(1）接口注入（2）构造方法注入。时效性好，但是灵活性差。（3）赋值方式注入。时效性差，但是灵活性好，需要有set方法。

**AOP（面向接口编程）**，同样的，他也是一种思想，而不是技术。和OOP（面向对象编程）相比较，AOP是对OOP的补充。OOP是静态的抽象，而AOP是动态的抽象。关于AOP的概念，简单来说就是将一个工程中与源代码无关，但是很多地方都要用，抽出来也不影响源代码上下文的那一部分代码抽出来，然后要用的时候就织入进去，进行使用，至于是在指定的代码之前使用还是之后又或者异常使用等，可以动态的进行;

具体业务场景的使用:

IOC通过以下注入方式应用到具体业务中, (1）接口注入（2）构造方法注入。时效性好，但是灵活性差。（3）赋值方式注入,时效性差，但是灵活性好，需要有set方法。在应用开发中，当我们在设计组件时，往往需要引入和调用其他组件的服务时，这种依赖关系如果固化在组件设计中就会导致组件之间的耦合和维护难度的增大，这个时候如果使用 IoC 容器，把资源获取的方式反转，让 IoC 容器主动管理这些依赖关系，将依赖关系注入到组件中，那么这些依赖关系的适配和管理就会更加灵活。

AOP主要使用在如在应用场景: 记录日志; 监控方法运行时间 （监控性能） ; 权限控制  ; 缓存优化 （第一次调用查询数据库，将查询结果放入内存对象， 第二次调用， 直接从内存对象返回，不需要查询数据库 ）; 事务管理 （调用方法前开启事务， 调用方法后提交关闭事务 ）;

## **有没有用过aop?aop的特性有哪些? 有什么应用?用到的业务的场景是怎样的**

有,AOP（面向接口编程） 是基于动态代理实现的;和OOP（面向对象编程）相比较，AOP是对OOP的补充。OOP是静态的抽象，而AOP是动态的抽象。关于AOP的概念，简单来说就是将一个工程中与源代码无关，但是很多地方都要用，抽出来也不影响源代码上下文的那一部分代码抽出来，然后要用的时候就织入进去，进行使用，至于是在指定的代码之前使用还是之后又或者异常使用等，可以动态的进行;

应用:例如，在很多业务中都需要记录操作日志，结果我们不得不在业务流程中嵌入大量的日志记录代码。无论是对业务代码还是对日志记录代码来说，维护都是相当复杂的。由于系统中嵌入了这种大量的与业务无关的其他重复性代码，系统的复杂性、代码的重复性增加了。维护起来会更加复杂。

应用场景: 记录日志; 监控方法运行时间 （监控性能） ; 权限控制  ; 缓存优化 （第一次调用查询数据库，将查询结果放入内存对象， 第二次调用， 直接从内存对象返回，不需要查询数据库 ）; 事务管理 （调用方法前开启事务， 调用方法后提交关闭事务 ）

## **Spring动态代理的方式有几种, 他们之间有什么区别**

有两种方式:

JDK动态代理,基于接口(默认代理模式)，CGLIB动态代理（若要使用需要进行配置）;

区别:

jdk的代理对象与被代理对象实现了同一接口,底层通过clone()方法,将实现类的方法复制过来在按照invoke中的配置对方法进行增强。而cglib代理则是代理对象继承了被代理对象的方法,然后自身进行增强。这两种代理思想在外来的自动生成代码方向会经常被使用,实现完全体代码的复用性

两种方式的对比和选择如下:

若目标对象实现了若干接口，spring使用JDK的java.lang.reflect.Proxy类代理。

优点：因为有接口，所以使系统更加松耦合

缺点：为每一个目标类创建接口

若目标对象没有实现任何接口，spring使用CGLIB库生成目标对象的子类。

优点：因为代理类与目标类是继承关系，所以不需要有接口的存在。

缺点：因为没有使用接口，所以系统的耦合性没有使用JDK的动态代理好

## **请谈谈你对Mybatis的缓存的理解**

MyBatis的缓存分为一级缓存和二级缓存,一级缓存放在session里面,默认就有,二级缓存放在它的命名空间里,默认是打开的, 使用二级缓存属性类需要实现Serializable序列化接口(可用来保存对象的状态),可在它的映射文件中配置<cache/>

## **了解mybatis的半ORM的原理吗(怎么在一个mapper里面转化成sql去执行)?**

原理:

加载配置：配置来源于两个地方，一是配置文件，一是Java代码的注解，将SQL的配置信息加载成为一个个MappedStatement对象（包括了传入参数映射配置、执行的SQL语句、结果映射配置），存储在内存中。 SQL解析：当API接口层接收到调用请求时，会接收到传入SQL的ID和传入对象（可以是Map、JavaBean或者基本数据类型），Mybatis会根据SQL的ID找到对应的MappedStatement，然后根据传入参数对象对MappedStatement进行解析，解析后可以得到最终要执行的SQL语句和参。 SQL执行：将最终得到的SQL和参数拿到数据库进行执行，得到操作数据库的结果。 结果映射：将操作数据库的结果按照映射的配置进行转换，可以转换成HashMap、JavaBean或者基本数据类型，并将最终结果返回。

## **Spring MVC的原理**

原理分析如下:

01、用户发送出请求到前端控制器DispatcherServlet

02、DispatcherServlet收到请求调用HandlerMapping（处理器映射器）

03、HandlerMapping找到具体的处理器(可查找xml配置或注解配置)生成处理器对象及处理器拦截器(如果有)，再一起返回给DispatcherServlet

05、HandlerAdapter经过适配调用具体的处理器（Handler/Controller）

06、Controller执行完成返回ModelAndView对象

07、HandlerAdapter将Controller执行结果ModelAndView返回给DispatcherServlet

08、DispatcherServlet将ModelAndView传给ViewReslover（视图解析器）

09、ViewReslover解析后返回具体View（视图）

10、DispatcherServlet根据View进行渲染视图（即将模型数据填充至视图中）

11、DispatcherServlet响应用户

## **MVC架构模式**

MVC设计模式的任务是将包含业务数据的模块与显示模块的视图[解耦](https://www.baidu.com/s?wd=%E8%A7%A3%E8%80%A6&tn=24004469_oem_dg&rsv_dl=gh_pl_sl_csd" \t "_blank)。在模型和视图之间引入重定向层可以解决问题。此重定向层是控制器，控制器将接收请求，执行更新模型的操作，然后通知视图关于模型更改的消息

具体分析如下:

MVC: Model View Controller 模型-视图-控制器   
视图：用户看到并与之交互的界面。

模型：表示企业数据和业务规则。在MVC的3个部件中，模型拥有最多的处理任务。被模型返回的数据是独立的，与数据格式无关。这样的一个模式能够为多个视图提供数据，这样一来模型的代码只需要写一次便能够被多个视图重用，减少了代码的重复性。

控制器：接受用户的输入并调用相应的模型和视图来完成用户的需求，所以当用户点击web页面中的超链接和发送表单时，控制器本身不输出任何东西和做任何处理。它只是接收请求并根据请求来调用哪个模型去处理请求，然后在决定用哪个视图来显示返回的数据。

## **热部署怎么用?**

通过JRebel插件的方式来进行热部署.

1. 在IDEA中打开插件管理界面,正确安装JRebel插件,
2. 安装完插件后，需要去获取正版的激活码
3. 重启IDEA后，在IDEA的Settings中找到JRebel输入复制的激活码即可
4. 在左边工具栏中勾中JRebel方式后启动即可

## **spring 底层都用过哪些设计模式**

1. 简单工厂
2. 又叫做静态工厂方法（StaticFactory Method）模式, 简单工厂模式的实质是由一个工厂类根据传入的参数，动态决定应该创建哪一个产品类
3. 工厂方法（Factory Method）
4. 定义一个用于创建对象的接口，让子类决定实例化哪一个类, Spring中的FactoryBean就是典型的工厂方法模式
5. 3. 单例（Singleton）
6. 保证一个类仅有一个实例，并提供一个访问它的全局访问点。
7. 4. 适配器（Adapter）
8. 将一个类的接口转换成客户希望的另外一个接口。Adapter模式使得原本由于接口不兼容而不能一起工作的那些类可以一起工作。
9. 5.包装器（Decorator）
10. 动态地给一个对象添加一些额外的职责。就增加功能来说，Decorator模式相比生成子类更为灵活。
11. 6. 代理（Proxy）
12. 为其他对象提供一种代理以控制对这个对象的访问。
13. 从结构上来看和Decorator模式类似，但Proxy是控制，更像是一种对功能的限制，而Decorator是增加职责, Spring的Proxy模式在aop中有体现，比如JdkDynamicAopProxy和Cglib2AopProxy。
14. 7.观察者（Observer）
15. 定义对象间的一种一对多的依赖关系，当一个对象的状态发生改变时，所有依赖于它的对象都得到通知并被自动更新
16. 8. 策略（Strategy）
17. 定义一系列的算法，把它们一个个封装起来，并且使它们可相互替换。本模式使得算法可独立于使用它的客户而变化。
18. 9.模板方法（Template Method）
19. 定义一个操作中的算法的骨架，而将一些步骤延迟到子类中。Template Method使得子类可以不改变一个算法的结构即可重定义该算法的某些特定步骤。
20. 另外还有命令模式，职责链模式，抽象工厂模式。

## **mybatis 的#和$的区别**

都是取值符号 #相当于对数据 加上 双引号，$相当于直接显示数据 $有SQL注入的问题,但是如果是在order by或者group by 后面添加的一些字符串, 我们需要使用$而不是# 完成字符串的替换

## **springMVC 的整体流程**

1.spring mvc的所有请求都提交给DispatcherServlet,它会委托应用系统的其他模块负责对请求进行真正的处理工作。

2.DispatcherServlet查询一个或多个HandlerMapping,找到处理请求的Controller.

3.DispatcherServlet请求提交到目标Controller

4.Controller进行业务逻辑处理后，会返回一个ModelAndView

5.Dispathcher查询一个或多个ViewResolver视图解析器,找到ModelAndView对象指定的视图对象

6.视图对象负责渲染返回给客户端。

## **Mybatis 的作用**

MyBatis是一个支持普通SQL查询，存储过程和高级映射的优秀持久层框架。

MyBatis消除了几乎所有的JDBC代码和参数的手工设置以及对结果集的检索封装。MyBatis可以使用简单的XML或注解用于配置和原始映射，将接口和Java的POJO（Plain Old Java Objects，普通的Java对象）映射成数据库中的记录。

## **Ajax原理**

Ajax的原理简单来说通过XmlHttpRequest对象来向服务器发异步请求，从服务器获得数据，然后用javascript来操作DOM而更新页面。这其中最关键的一步就是从服务器获得请求数据。

缺点 ：ajax干掉了back按钮 安全问题 对搜索引擎支持比较弱。破坏了程序的异常机制

## **MyBatis中包扫描怎么变成具体的实现类?**

答:mybatis通过JDK的动态代理方式，在启动加载配置文件时，根据配置mapper的xml去生成具体的实现类.

细节如下:

1. 启动时加载解析mapper的xml

如果不是集成spring的，会去读取<mappers>节点，去加载mapper的xml配置.

如果是集成spring的，会去读spring的sqlSessionFactory的xml配置中的mapperLocations，然后去解析mapper的xml;

2. 绑定namespace(XMLMapperBuilder)

这里先去判断该namespace能不能找到对应的class,若可以则调用

configuration.addMapoer(boundType), configuration又委托给MapperRegistry:

3. 生成该mapper的代理工厂(MapperRegistry)

4. getMapper的时候生成mapper代理类

5. new出来MapperProxy

6.MapperProxy实现InvocationHandler接口进行拦截代理,

这里的代理拦截，主要是寻找到MapperMethod，通过它去执行SQL

7. MapperMethod委托给SqlSession去执行sql

## **请阐述Spring的事务隔离级别。**

Spring事务上提供以下的隔离级别:

ISOLATION\_DEFAULT: 使用后端数据库默认的隔离级别

ISOLATION\_READ\_UNCOMMITTED　: 允许读取未提交的数据变更，可能会导致脏读，幻读或不可重复读

ISOLATION\_READ\_COMMITTD : 允许读取为提交数据,可以阻止脏读，当时幻读或不可重复读仍可能发生

ISOLATION\_REPEATABLE\_READ: 对统一字段多次读取结果是一致的，除非数据是被本事务自己修改．可以阻止脏读，不可重复读，但幻读可能发生

ISOLATION\_SERIALIZABLE :　完全服从ACID

## **请阐述你对Spring的IOC和DI的理解**

比如A对象需要使用合作对象B, 那么A就对B产生了依赖，也就是A和B之间存在一种耦合关系，并且是紧密耦合在一起。

但是这种高度的耦合性在使用了Spring之后就不一样了，创建合作对象B的工作是由Spring来做的，Spring创建好B对象，然后存储到一个容器里面，当A对象需要使用B对象时，Spring就从存放对象的那个容器里面取出A要使用的那个B对象，然后交给A对象使用，至于Spring是如何创建那个对象，以及什么时候创建好对象的，A对象不需要关心这些细节问题，A得到Spring给我们的对象之后，两个人一起协作完成要完成的工作即可。

所以控制反转IoC(Inversion of Control)是说创建对象的控制权进行转移，以前创建对象的主动权和创建时机是由自己把控的，而现在这种权力转移到第三方，比如转移交给了IoC容器，它就是一个专门用来创建对象的工厂，你要什么对象，它就给你什么对象，有了 IoC容器，依赖关系就变了，原先的依赖关系就没了，它们都依赖IoC容器了，通过IoC容器来建立它们之间的关系。

DI(依赖注入)其实就是IOC的一种类型, IoC的一个重点是在系统运行中，动态的向某个对象提供它所需要的其他对象。这一点是通过DI（Dependency Injection，依赖注入）来实现的

## **使用过分页插件么?**

使用过MyBatis 分页插件 - PageHelper

该插件目前支持Mysql, Oracle数据库的物理分页:

使用步骤:

1在maven pom.xml中添加依赖

2在Mybatis-config.xml中sqlSessionFactory中添加插件配置

3.写sql、service

分页插件的原理：实现Mybatis提供的接口，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。

举例：select \* from student，拦截sql后重写为：select t.\* from (select \* from student) t limit 0，10

## **MyBatis的底层原理什么？**

1 Mybatis配置：sqlmapConfig.xml，此文件作为Mybatis的全局配置文件，配置了Mybatis的运行环境等信息。mapper.xml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。

2通过Mybatis环境等配置信息获得sqlsessionfactory即会话工厂

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

4 mybatis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。

5 Mapped Statement也是mybatis一个底层封装对象，它保障了mybatis配置信息及sql映射信息等。mapper.xml文件中一个sql对应一个Mapped Statement独享，sql的id就是mapped statement的id。

6 Mapped Statement对sql执行输入参数定义，包括Hashmap、基本类型、pojo、Executor通过Mapped Statement在执行sql前将输入的java对象映射到sql中，输入参数映射就是jdbc编程中对prepared Statement设置参数。

7 Mapped Statement对sql执行输出结果进行定义，包括Hashmap、基本类型、pojo、Executor通过Mapped Statement在执行sql后将输出结果映射到java对象中，输出结果映射过程相当于jdbc编程中对结果的解析处理过程

## **什么是spring**

它是一个容器框架，用来装javabean（java对象），中间层框架（万能胶）可以起一个连接作用，比如说把SpringMVC和Mybatis粘合在一起运用。简单来说，Spring是一个轻量级的控制反转(IoC)和面向切面(AOP)的容器框架。

## **什么是spring IOC**

由容器控制程序之间的（依赖）关系，而非传统实现中，由程序代码直接操控。这也就是所谓“控制反转”的概念所在：(依赖)控制权由应用代码中转到了外部容器，控制权的转移，是所谓反转。

 IoC还有一个另外的名字：“依赖注入 (DI=Dependency Injection)”  ，即由容器动态的将某种依赖关系注入到组件之中

## **spring IOC 如何控制反转**

所有的类都会在spring容器中登记，告诉spring你是个什么东西，你需要什么东西，然后spring会在系统运行到适当的时候，把你要的东西主动给你，同时也把你交给其他需要你的东西。所有的类的创建、销毁都由 spring来控制，也就是说控制对象生存周期的不再是引用它的对象，而是spring。对于某个具体的对象而言，以前是它控制其他对象，现在是所有对象都被spring控制，所以这叫控制反转。

# CRM+骡窝窝

## **你们前后端分离,那么你们提供接口需要写接口文档吗?接口有变动你们怎么做的管理?**

需要的，后端开发采用RESTful风格，接口文档自然使用一些开源的接口文档生成工具，之前公司使用的是swagger2来时生成接口文档，开发测试或者联调测试时直接通过设定好路径访问即可。至于接口变动，联调测试时发现接口设计不合理或者有bug会汇集一起提bug单，同时给对接的开发者及对应管理者抄送邮件。接口修复后，提git上提测试版本V2，同时回复回复邮件。

## **RESTful用了什么框架实现?**

我接触过的的RESTful框架实现框架有3个，分别为：jersey，play，SpringMVC/springBoot，前面2个是居于个人兴趣自学的，没有在工作中用过。用最多的还是springmvc/springBoot。

http://www.importnew.com/17138.html

## **说说你对RESTful的理解?**

RESTful架构是目前一种比较流行的互联网软件架构。它结构清晰、符合标准、易于理解、扩展方便，所以正得到越来越多网站的采用。RESTful是一种软件架构风格、设计风格，而不是标准，只是提供了一组设计原则和约束条件。它主要用于客户端和服务器交互类型的应用中。基于这个风格设计的应用可以更简洁，更有层次，更易于实现缓存等机制。其设计遵循REST原则，即Representational State Transfer的缩写，翻译过来是"表现层状态转化"。

其中涉及4个核心点：

资源(Resources):是网络上的一个实体，或者说是网络上的一个具体信息,也即URI

表现层(Representation):资源的具体呈现，也就是资源具体格式，一般在HTTP请求的头信息中用Accept和Content-Type字段指定，这两个字段才是对"表现层"的描述

状态转(State Transfer):客户端和服务器的一个互动过程。在这个过程中，势必涉及到数据和数据状态的变化。客户端用到的手段，只能是HTTP协议。具体来说，就是HTTP协议里面，

四个表示操作方式的动词：GET、POST、PUT、DELETE。它们分别对应四种基本操作：GET用来获取资源，POST用来新建资源（也可以用于更新资源），PUT用来更新资源，DELETE用来删除资源。

大白话讲：RESTful是一种软件架构风格、设计风格，而不是标准，只是提供了一组设计原则和约束条件。RESTful可以理解为是HTTP协议最佳实践，使用uri来标识要操作的资源对象，使用HTTP请求方法区分操作行为，使用请求头参数细化操作行为，最后通过响应状态码识别操作资源结果。对HTTP协议的极致利用可以说是BS架构应用的不二选择。

## **我看你们的项目用的是RESTful风格的api接口?你们是怎么处理日志的?**

对于接口的每次请求的原始报文都会以日志的形式保存到我们的日志存储系统, 通过logback框架按照天对日志存储到文件系统, 对于解析后的报表也会存储到对应的日志表中

前公司使用的自己开发的log日志系统，对一些核心的业务方法进行切面日志记录，将日志记录到日志表中。表中记录程序运行的类/方法/异常信息/线程号/执行时间的。系统异常时通过查询日志表即可快速定位异常发生点。

## **FreeMarker的原理?**

FreeMarker是一款比较好用的模板引擎，其原理可以概括为： 模板 + 数据 = 输出。操作

应的终端文件。这些文件可以是HTML网页、电子邮件、配置文件、源代码等。

FreeMarker它不是面向最终用户的，而是一个Java类库，是一款程序员可以嵌入他们所开发产品的组件。

## **Shiro的用法?有哪些注解?**

之前项目中用shiro做过登录认证跟权限控制，使用起来较为简单。

认证：对需要进行登录认证的资源路径配置authc 登录过滤器，使用loginUrl指定登录页面，get方式提交请求表示访问等页面，post方式提交请求进行登录认证。当请求验证资源时先对资源进行登录检查，已登录放行，未登录跳转回登录页面。

授权：授权使用注解 + 编程 + 页面标签方式实现。注解授权：在请求接口方法上贴上@ RequiresPermissions，在请求时进行权限检查。 编程授权：更多的是实现业务需求时，对操作角色进行权限限制。页面标签授权：跟编程授权一样，也是对角色进行权限限制，具体体现是页面上是否显示操作入口，比如说按钮之类的。

注解：

@RequiresAuthentication 使用该注解标注的类，实例，方法在访问或调用时，当前Subject必须在当前session中已经过认证。

@RequiresGuest 使用该注解标注的类，实例，方法在访问或调用时，当前Subject可以是“gust”身份，不需要经过认证或者在原先的session中存在记录。

@RequiresPermissions 当前Subject需要拥有某些特定的权限时，才能执行被该注解标注的方法。如果当前Subject不具有这样的权限，则方法不会被执行。

@RequiresRoles 当前Subject必须拥有所有指定的角色时，才能访问被该注解标注的方法。如果当天Subject不同时拥有所有指定角色，则方法不会执行还会抛出AuthorizationException异常。

@RequiresUser 当前Subject必须是应用的用户，才能访问或调用被该注解标注的类，实例，方法。

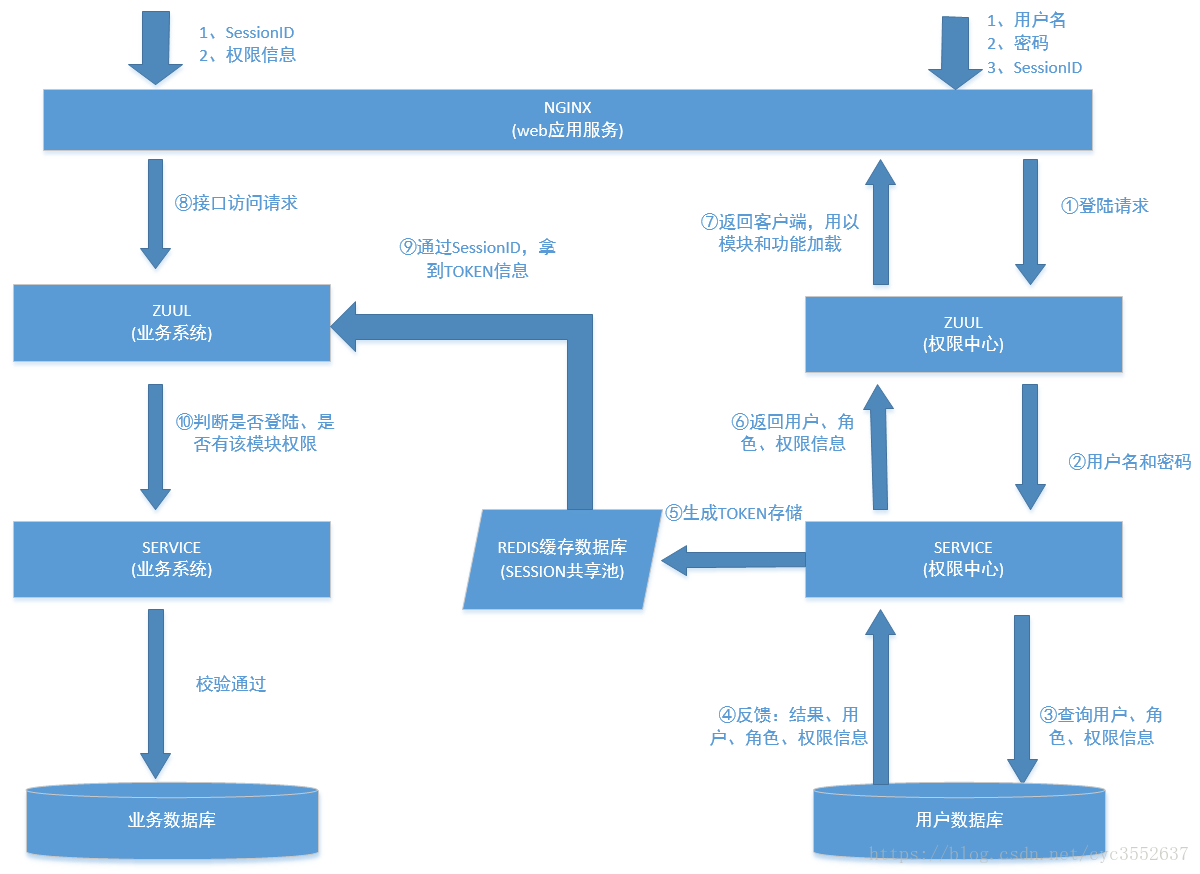
## **分布式架构权限管理**

分布式架构权限管理的两种情况，一种是针对统一授权访问的，一种是跨平台接口访问的。

虽然分布式架构会做业务的切割，将整体的业务切割为独立的子业务或者子平台，但是同一平台下往往会有统一的授权和单点登录，客户端而言平台是整体的，这种是统一授权访问的权限管理。但是也会遇到多平台协作的情况，这种情况不用考虑其他平台的架构，只需要为其提供数据接口跟其对接就可以，这种情况就要考虑跨平台接口访问的权限管理。

一、统一授权访问

前端采用web服务器，nginx或者haproxy之类的都可以，利用nginx做第一层反向代理，用zuul做第二层反向代理，两层反向代理对于常见的网络渗透和爬虫基本可以轻松应对。在这个前提下，我们处理接口授权和访问安全等问题，而且需要从客户端和服务端都进行安全控制。



1、用户通过用户名、密码发起登陆请求，这里可以配合验证码、短信验证、微信验证等提高安全级别，登陆请求访问到权限中心的ZUUL。

2、由ZUUL反向代理到权限中心的SERVICE。

3、查询用户、角色信息，进行查询匹配。

4、有查询结果以后反馈，反馈用户状态，用户、角色、权限信息等。

5、生产TOKEN，TOKEN的算法需要自己编写，建议加入时间戳等信息进行加密。存放在SESSION或者直接放入REDIS中，一般建议放入SESSION中，因为分布式架构要做SESSION共享，必须有一个SESSION共享池，SESSION共享池一般会用REDIS来做，而SESSION可以利用SESSIONID，确认唯一用户，比较方便。

6、返回给客户端用户、角色、权限信息。

7、浏览器客户端通过得到的用户、角色、权限信息进行前端功能和菜单的渲染，隐藏非授权的功能，并且可以基于这些信息做前端的校验，但是这种校验由于都在客户端，很容易被人篡改，只能做一些基本防护，但是这是必要的。

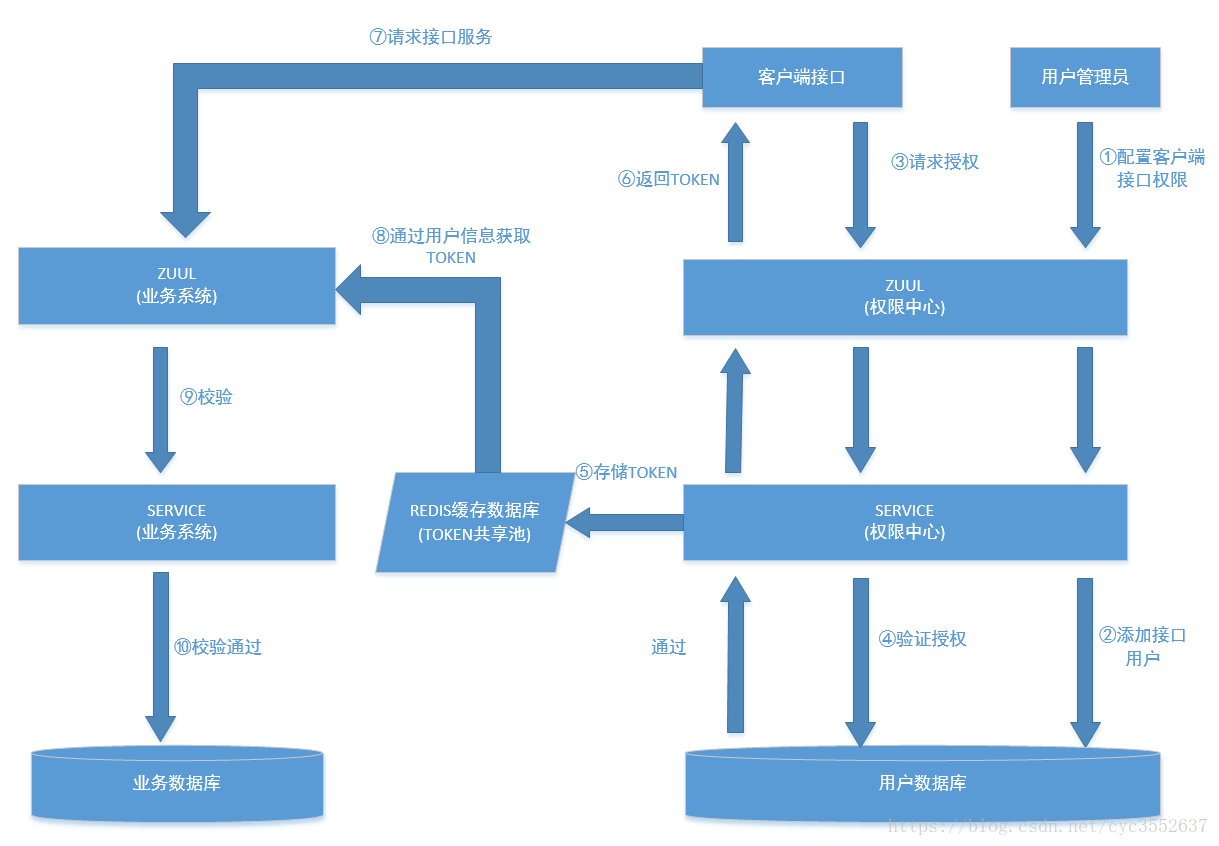
8、浏览器客户端发送业务请求到服务端，请求会附带客户端菜单或者功能的权限信息，服务端通过ZUUL的拦截器拦截业务请求，访问SESSION共享池。

9、然后通过SESSIONID得到TOKEN信息，通过我们的TOKEN算法进行解密，得到用户、角色、权限信息。

10、进行匹配校验，查看本次请求是否有接口访问权限，校验通过就可以访问接口，校验不通过就不能访问。

二、跨平台接口访问

这里的接口是指一下跨平台的接口服务，类似webservice这样，没有单点登录，没有统一的授权，往往是其他平台跟我们平台进行远程交互的，这些往往两个平台不是统一的公司或部门，所以接口不是长期使用的，有一定的时间限制。



1、用户管理员配置外部客户端用户及其权限信息。

2、添加用户信息、权限信息到用户数据库之中。

3、外部客户端接口如果没有TOKEN，需要先发送授权请求到权限中心。

4、权限中心通过验证是否配置外部客户端用户，并且取得该用户的权限信息，校验通过以后通过加密算法生产TOKEN，这里的TOKEN根据实际情况，如果安全要求高最好加入时间戳，让TOKEN过期作废，让对方重新请求。

5、将TOKEN存入REDIS中，形成TOKEN共享池，这里由于没有浏览器的SESSIONID，所以存放在SESSION中意义不大。

6、返回TOKEN给客户端。

7、客户端拿到TOKEN以后发送业务请求给业务接口地址。

8、业务接口通过ZUUL拦截请求，并且将TOKEN跟REDIS的TOKEN进行比对。

9、比对成功以后，对TOKEN进行解密，然后看是否有接口授权。

10、如果有接口授权则通过，访问接口，如果没有则不能访问。

## **你们的项目是用哪种图片服务器?**

第一种说法：公司自己有自己的文件服务器。

第二种说法：租用阿里的oss 七牛 新浪图床

## **http常见的响应状态码和其意义**

200 OK //客户端请求成功

302 重定向错误。eg:

400 Bad Request //客户端请求有语法错误，不能被服务器所理解.eg:controller接收时间格式参数时，没对时间格式进行一个转换操作。

401 Unauthorized //请求未经授权，这个状态代码必须和WWW-Authenticate报头域一起使用

403 Forbidden //服务器收到请求，但是拒绝提供服务

404 Not Found //请求资源不存在，eg：输入了错误的URL

500 Internal Server Error //服务器发生不可预期的错误

503 Server Unavailable //服务器当前不能处理客户端的请求，一段时间后可能恢复正常

## **http的请求方式有哪些**

1、opions 返回服务器针对特定资源所支持的HTML请求方法 或web服务器发送\*测试服务器功能（允许客户端查看服务器性能）

2、Get 向特定资源发出请求（请求指定页面信息，并返回实体主体）

3、Post 向指定资源提交数据进行处理请求（提交表单、上传文件），又可能导致新的资源的建立或原有资源的修改

4、Put 向指定资源位置上上传其最新内容（从客户端向服务器传送的数据取代指定文档的内容）

5、Head 与服务器索与get请求一致的相应，响应体不会返回，获取包含在小消息头中的原信息（与get请求类似，返回的响应中没有具体内容，用于获取报头）

6、Delete 请求服务器删除request-URL所标示的资源\*（请求服务器删除页面）

7、Trace 回显服务器收到的请求，用于测试和诊断

8、Connect HTTP/1.1协议中能够将连接改为管道方式的代理服务器

http服务器至少能实现get、head、post方法，其他都是可选的

# P2P+商城项目

## **商城项目的人员构成**

这个根据自己简历写的按项目大小来说

一般小型项目组人员构成:

UI 1() / andriod (1) / IOS(1) /前端(1-2) / 后台(2-6) / 测试(1-2)

其中UI , android , ios 前端,测试 都可能是多个项目组共享的,自己看情况回答

## **你们微服务有几台服务器, 每台服务器主要有哪些功能模块**

服务器数量大家可以根据自己项目大小和自己所说的并发数量计算一下,一个服务器可能会部署三四个服务,这样才能更好利用服务器资源

**需要部署的服务:**

数据库 主(可以分多个数据库) + 从(可以有多个从库) 3(1主两从)

Redis(规模大的可以是集群3主+3从) 6

activeMQ 规模大的可以是集群3主+3从 6

zookeeper(可能是集群最少3个) 3

各种服务端(商品,订单,支付,用户,内容管理 等...) 8

手机接口项目 2

后台管理项目 1

Nginx (如果Nginx 做了高可用则需要最少两台服务器) 2

**按照上面的部署来计算的话:**

服务器总数= 3+6+6+3+8+2+1+2/4(假设一台服务器混合部署个服务) = 8 台服务器

但是对于小型app来说可能根本不会去做Redis集群和MQ集群,各种服务也不会部署太多,这样服务器可能只需要四五台就能支撑了

大家回答的时候可以根据自己简历所写的项目大小做适当增减, 也不需要回答的非常精确,项目部署一般由技术经理来做,维护一般也是公司骨干或者技术经理,作为一个普通程序员只知道大概信息是正常的,说的太熟悉了反而会引来更多相关的问题

## **你们项目前后端分离用什么工具进行交接, 如果出现数据错误怎么解决**

我们一般使用swagger2来自动生成接口文档,因为公司不大,大家关系都比较熟悉,所以有疑问的地方前端一般就直接找对应的后端沟通了

## **除了你自己做的模块,你还对你的项目的其他模块了解吗?详细说说业务架构(有哪些模块,模块之间的关系)?业务上的关系你能描述清楚吗?**

## **你的商城项目用户量多大？你觉得用分布式架构合理吗？**

我们项目据项目经理说是按照千万级别用户量来考虑的,目前用户量不大,但是还是要考虑未来的拓展问题. 分布式架构虽然开发上复杂度上有一些增加,但是未来做大了拓展方便,我觉的也是很值得的

## **你这个商城用了多少个数据库?**

数据库我们暂时分了三个商品,用户的订单相关的,主要是方便拓展,但是又不想过多涉及分布式事务,就把订单,积分,优惠券相关的都放在一个数据了

(分布式事务不要过多提及,如果问到就说是经理写的,我们只需要调用相应的方法就行了)

## **你们怎么分库分表的?**

答案一：可以使用开源的数据库中间件mycat，需要按照对应的业务规则来分片。

这个问题建议先看看相关资料，然后再答，如果没有把握就讲是项目经理弄的。

（一般做了分库之后，查询是很少使用join的，因为是会涉及到跨库的join,性能较低）

答案2：我们的分库分表是项目经理搞的,我也大概了解过,主要是按照业务来进行划分,但是考虑到我们业务并没有那么大就没有划分太多数据库,分表方面是把商品,商品详情,商品图片这些大表拆分成了小表优化性能,并没有使用MyCat之类的中间件

(MayCat之类的中间件我们最好回避,不要说是自己负责的)

## **订单你处理过的并发量有多大?**

我们项目初期阶段的的设计目标是1000(或者两千)的并发,目前用户量还没有上来,所以并发量并没有太高, 测试那边说目前并发1000情况下支撑1000QPS问题不大,用户量大了我们随时扩容就行

## **第三方支付怎么做的**

支付这块我们接入了支付宝和微信两中支付方式,我负责了一部分支付宝的支付流程.微信的大概看了一遍,也都差不多的流程吧!

支付宝这边是调用支付宝SDK的支付接口(假设我们项目是只有APP,没有PC端), 如果用户手机安装的有支付宝就会拉起支付宝程序进行支付,如果用户手机没有没有安装支付宝,就会跳转到支付宝网页端要求用户输入用户名/密码进行支付, 支付之后支付宝会异步调用我们指定的接口通知支付结果, 我们根据支付结果是成功还是失败来执行相应的业务逻辑.

业务逻辑主要是更改订单状态,更改支付单状态,增加相应积分,发放相应优惠券之类的业务, 业务逻辑稍微复杂一些. 主要是涉及到金钱的业务必须非常严谨,需要考虑到各种情况,并且详细的记录日志,保证就算出了问题也能够根据相应的记录或者日志恢复数据

PS: 还有一个就是涉及到分布式事务, 因为订单支付成功之后会去商品库修改订单状态和支付单状态,同时在支付数据库插入相关表,这就涉及到分布式事务问题,这种问题尽量回避,就说我们就是调用经理写好的接口就行了. 如果对分布式事务了解过的同学可以简单说下对分布式事务的理解,但是不宜过深入

## **是否参与过项目的升级部署?怎么做?**

我们项目部署的时候都要求全体在岗的,升级部署主要是技术经理在搞,我也学习也一些,我们部署还是比较原始的,没有使用jenkins之类的持续集成软件,都是手工打包部署的,开发环境我自己部署过,生产环境倒是没有去部署过!

## **Liunx的操作怎么关闭防火墙？**

如果是centos6版本的话, 可以通过 service iptables stop关闭防火墙服务,通过 chkconfig iptables off 禁止防火墙开机启动

如果是centos7版本的话, 通过 systemctl stop firewalld.service关闭防火墙服务,通过systemctl disable firewalld禁止防火墙开机启动

## **Linux查看日志的常用命令**

查看日志常用的命令有less, more, tail 一般情况使用less查看比较多, 而且可以进行翻页查找,还可以通过对应的查找命令定位到指定的位置

如果需要实时查询日志信息的话, 我会使用tail -100f这样的命令直接查看文件的最后100行, 而且会实时更新

## **Linux的less命令和tail命令有什么区别**

less命令,局部打开文件, 可以通过翻页集合查找命令进行定位查找

tail命令, 经常用于查询文件尾部以及有实时更新的日志信息, tail可以实时追踪文件的内容改变

## **Linux的常用命令有哪些**

cd 切换目录

ls 显示当前目录下的文件和目录

less

tail

top

ps -ef |grep java 查看所有的java进程

netstat -ntpl 查看当前正在运行的程序的端口号等信息

## **谈谈你对Redis的理解**

答案1：开放性题目。主要讲讲Redis在互联网项目中起到了哪些作用。

比如提高性能，从架构的层面来讲讲Redis的作用，比如一般项目的瓶颈主要在数据库这块。

所以我们优化的目标主要是减少数据库的访问，对应经常读的数据，使用缓存可以极大的减少访问数据库的次数。缓存有很多种，比如EHCache。但是因为我们的应用设计到集群和分布式，我们需要这些机器都共享一份缓存，所以使用Redis作为缓存服务器。

Redis除了做缓存之外，还可以对海量数据进行处理。

随便提下Redis的高可用(哨兵模式，集群模式)，可靠性(数据持久化AOP,RDB)

最重要是讲讲你项目中什么地方用到了Redis.

答案2：

在我们项目中Redis最主要的用途就是作为分布式缓存,但是其实同时它也可以完成简单的发布/订阅, 注册中心等功能, 只是我们项目中主要就是作为分布式缓存来使用. 相对于其他NoSQL数据库来讲它的数据类型更丰富,性能更高,很适合用作缓存, 也能够利用其多种数据类型来支持更多的使用场景

其缺点就是相对于关系型数据看来说,对于数据分组查询,多表关联查询这种需求就显得无能为力了. 还有一个问题就是作为内存数据库存在着数据丢失的问题,虽然Redis 3.x版本提供了两种持久化方式, Redis 4.0 提供了3种持久化方式, 但是依然无法像关系型数据库那样保证数据的完整性

所以我觉得如果用Redis的话就应该充分考虑到Redis的特性,然后按照Redis的特性来思考和设计我们的业务和表

## **为什么使用Redis做缓存**

性能高,分布式的缓存.

很多优点，上面的回答也可以讲.

## **Redis除了做缓存, 还有其他应用么**

数据的处理啊：

1. 定时帖子总数统计
2. 秒杀库存原子性递减
3. 点赞
4. 使用bitmap完成上亿用户登录标记

把上课给的案例转变成你自己项目中的案例

## **Redis的常见的数据结构有哪些**

List,HASH,STRING,ZSET,SET

## **为什么Redis的读取性能会比数据库的快?(不仅仅因为内存)**

数据结构简单，内存操作。这题你也可以反问一下面试官.

还有对于数据库是有事务控制的, 但是对于Redis来说不需要去处理事务

## **用Redis解决session的问题?是单机部署还是集群部署?**

1. 可以不使用session,直接使用cookie+token的方式
2. 如果项目集成了Shiro,是可以直接集成redis完成session共享的问题的.

项目比较小就说是使用单机部署.(可能会问可靠性问题，回答AOF每秒持久化)

但是对方问你高可用问题时，你可以把上课讲的集群和他说说。你说你们公司项目小，访问量不大，所以没考虑集群。但是你知道对于这种情况应该怎么处理

## **Redis多机部署的架构下,怎么做到数据的共享?(怎么做到key读到准确的value),要考虑value的存储形式,主机的数据量太大了?需不需要做Redis的分机处理**

Redis是分片处理了，总共有1w多个槽位，这些槽位对应着你配置的redis集群主机。

在存储的时候会根据算法算出这个key存储到哪个槽位，然后存储到对应的物理机。读的时候也是通过相同的算法算出这个key存储到哪个槽位，然后对应的物理机取出对应数据。

这个操作都是redis完成的，不需要我们处理.

Redis集群是支持水平拓展的，在不关闭redis实例的情况下动态完集群中添加新的主机.

## **Redis分布式锁,解决的是分布式的线程问题,Redis分布式锁的实现思路,怎样做到线程安全?**

分布式锁无非就是使用指定的标的物作为标识, 以此判断执行某个操作的权限是否已经被占用,如果被占用该线程就继续等待,没有被占用就可以对标的物加锁然后执行操作

从原理上来讲实现分布式锁的方式非常多,比如使用指定的文件,使用zookeeper中的指定节点等都可以

使用Redis作为分布式锁的原理其实就是:

1. 获取锁的服务在Redis中创建指定的key
2. 如果其他服务判断该key存在并且没有过期就认为指定的操作已经被别的服务加锁,继续等待.
3. 如果判断指定的key不存在或者已经过期, 则认为没有人占用锁,或者锁已经超时导致失效. 就可以新建/重置key 标识自己占用了该锁, 之后占用锁的服务就可以指定对应的操作

## **Redis有哪些操作是原子性的?**

Redis所有的操作都是原子性的。常见的使用比如数值的自增等操作

## **Redis的使用场景**

令牌机制登录，热点数据存储，如商品分类，用户的信息，帖子点赞，点赞数，浏览数

## **Redis为什么是单线程的而不是多线程的?从cpu的角度来看,单线程和多线程有什么性能的区别?为什么单线程也能实现这么高的性能?**

单线程指的是网络请求模块使用了一个线程（所以不需考虑并发安全性），即一个线程处理所有网络请求，其他模块仍用了多个线程。

 绝大部分请求是纯粹的内存操作（非常快速）

 采用单线程,避免了不必要的上下文切换和竞争条件

 非阻塞IO - IO多路复用

## **有没有做Redis的集群?有哪几种方式,Redis的集群最少要有多少个节点?如果其中一个节点挂掉会怎样?如果只有两个主节点Redis能不能跑起来?**

有主从模式，和集群模式.

需要6个节点，3主三从.

对应的从节点会变成主节点继续工作.

答案2：

PS: Redis三种集群方案:https://blog.csdn.net/sanpo/article/details/52839044

Redis常见的集群方案有三种:

1. 主从
2. 哨兵机制(sentinel )
3. 集群(redis cluster)

三种集群方式原理各不相同, 我们使用的是3.x之后官方推荐的redis cluster方式集群,三主三从.

其中一个节点挂掉其他节点能不能提供服务是可以通过cluster-require-full-coverage yes 进行配置的, yes 表示有节点挂掉其他可达节点也将停止提供服务, no 则相反

更深入一些的话,会涉及到怎么解决节点挂掉后解决可用性的问题,大家可以参考这篇文章: https://yq.aliyun.com/ziliao/411825

## **Redis中的持久化方式**

RDB方式( 默认) .snapsho ting(快照)默认方式.将内存中以快照的方式写入到二进制文件中.默认为dump.rdb.可以配置设置自动做快照持久化方式.我们可以配置Redis在n秒内如果超过m个key就修改自动做快照.

AOF方式 就是通过记录操作命令的方式进行持久化的 重启Redis的话 执行对应的日志文件即可

更有效的使用到第二种方法 因为可能保存的比较的好

## **请简单描述一下Redis的发布订阅**

Redis的发布订阅功能用于消息的传输，Redis的发布订阅包括三个部分，发布者，订阅者和Channel。 发布者和订阅者都是Redis客户端，Channel则为Redis服务器端，发布者将消息发送到某个的频道，订阅了这个频道的订阅者就能接收到这条消息。Redis的这种发布订阅机制与基于主题的发布订阅类似，Channel相当于主题。

## **Redis缓存穿透**

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。

解决

我们的解决方案:如果一个查询返回的数据为空（不管是数 据不存在，还是系统故障），我们仍然把这个空结果进行缓存，但它的过期时间会很短，最长不超过五分钟。

## **Redis可以作为Mybatis的二级缓存么?需要怎么配置**

可以，首先需要创建一个缓存类，该缓存类实现mybatis的Cache接口，并实现该接口上的抽象方法，所有的这些抽象方法包含着设置对象，获取对象，删除对象，清理所有缓存，获取缓存大小，实现方法体中使用redis去操作数据。然后在mapper的xml文件中使用<Cache>标签使用该自定义的缓存对象。

## **阐述你对于Redis穿透和雪崩的一个理解**

1. 雪崩：

雪崩是指缓存在同一时间内大量键过期（失效），接着来的一大波请求瞬间都落在了数据库中导致连接异常。

1. 穿透：

穿透是指短时间内大量的请求缓存中不存在的数据，导致这些请求全都落在了数据库中，使数据库瘫痪或者宕机。

解决：缓存和穿透都可以使用加锁，或者使用MQ解决，避免短时间大量请求同时进入数据库。

## **除了Redis缓存还用过其他的缓存吗?**

EHCache，自己了解哪些就说哪些

## **list 和Redis里面的list有什么区别?**

Redis中的list功能更加强大。

对于Redis中的list的数据结构和Java中的List结构不一样, Redis的数据结构具有链表, 栈等特性, 操作更加方便和高效

## **使用Dubbo的版本是哪个?有没有了解过Dubbo比较新的特性以及消息?**

我们用的是2.5.7 大概看了一下,新版本主要解决了一些BUG,比如优雅停机,新增Netty4模块等, 但是没来得及过多关注

## **在Dubbo中如果zookeeper挂了以后还能用吗?**

服务能调用，因为客户端有缓存服务列表，但是新的服务无法往zookeeper注册

## **Dubbo消费端怎么调用生产端的?**

在业务层代码中调用远程接口的方法时，实际上是调用的本地创建的代理对象; 在该代理对象中，调用了InvokerInvocationHander.invoke方法。远程调用以Invcation、Result为中心，在这个方法中根据调用的远程方法和传入的参数构建RpcInvcation对象，该对象中的成员变量包括远程调用的方法名、方法参数类型、参数值等值，该RpcInvcation对象作为后续invoke方法的请求参数。

## **Dubbo基本的原理**

Dubbo的基本原理其实就是RPC协议的基本原理，而RPC协议的原理是数据的序列化和反序列化，并且把序列化后的数据通过TCP进行传输

## **Dubbo都支持哪些协议**

Dubbo支持Dubbo、rmi、hessian、http、webservice、thrift、Redis等多种协议，但是Dubbo官网是推荐我们使用Dubbo协议的。

缺省协议，使用基于mina1.1.7+hessian3.2.1的tbremoting交互。

传输协议：TCP

传输方式：NIO异步传输

序列化：Hessian 二进制序列化

适用范围：传入传出参数数据包较小（建议小于100K），消费者比提供者个数多，单一消费者无法压满提供者，尽量不要用Dubbo协议传输大文件或超大字符串。

适用场景：常规远程服务方法调用 Dubbo默认采用Dubbo协议，Dubbo协议采用单一长连接和NIO异步通讯，适合于小数据量大并发的服务调用，以及服务消费者机器数远大于服务提供者机器数的情况

他不适合传送大数据量的服务，比如传文件，传视频等，除非请求量很低。

## **http和RPC协议的区别**

RPC（即Remote Procedure Call，远程过程调用）和HTTP（HyperText Transfer Protocol，超文本传输协议）他们最本质的区别，就是RPC主要工作在TCP协议之上，而HTTP服务主要是工作在HTTP协议之上，我们都知道HTTP协议是在传输层协议TCP之上的，所以效率来看的话，RPC要更胜一筹，但是从开发难度来看，HTTP协议的接口要简单很多。

## **Dubbo的服务治理功能体现在哪里?**

体现在了我们可以统一的管理服务，监控服务运行的状态，给服务升级和降级，设置服务启用和禁用，为服务做负载均衡。

## **Dubbo底层采用Socket进行通信**

Socket 是一种应用接口, TCP/IP 是网络传输协议，虽然接口相同, 但是不同的协议会有不同的服务性质。创建Socket 连接时，可以指定使用的传输层协议，Socket 可以支持不同的传输层协议（TCP 或UDP ），当使用TCP 协议进行连接时，该Socket 连接就是一个TCP 连接。Soket 跟TCP/IP 并没有必然的联系。Socket 编程接口在设计的时候，就希望也能适应其他的网络协议。所以，socket 的出现只是可以更方便的使用TCP/IP 协议栈而已。

## **请阐述一下对Dubbo的理解**

* 1. 默认推荐使用netty框架
  2. 默认是阻塞的，可以异步调用，没有返回值的时候可以这么做
  3. 推荐使用zookeeper注册中心
  4. 默认使用Hessian序列化，还有Dubbo，FastJSON。Java自带序列化

Dubbo是高集群高可用，挂的几率很小，就算是挂了重新启动就好了

## **Dubbo的事务的管理**

1结合MQ消息中间件实现的可靠消息最终一致性

点对点，对列模型：

应用一（消息生产者）—> jms对列 --> 接收消息

应用二（消息生成者）--> jms对列 --> 接收消息

Jms点对点对列模式特点：

1、消息生成者生成消息发送到queue中，然后消息消息者从queue中取出并且消费消息

2、消息被消费以后，queue不会再存储，所有消息消费者不可能消费到已经被消费的消息

3、Queue支持多个消费者，但是对一个消息而言，只会有一个消费者可以消费

## **如果Dubbo中的服务的其中一个节点的服务挂掉了,Dubbo怎么知道?如果集群轮询到的某一台服务器挂掉,会怎样?**

 zookeeper通过心跳机制可以检测挂掉的机器并将挂掉机器的ip和服务对应关系从列表中删除。

客户端默认是1秒超时就报异常。

## **对象序列化的含义和意义**

序列化机制允许将实现序列化的Java对象转换成字节序列 ,这些字节序列可以被保存在磁盘上,或者通过网络传输,

java对象序列化可以保留一个对象的数据，而且还可以递归保存对象引用的每个对象的数据。可以将整个对象层次写入字节流中，可以保存在文件中或在网络连接上传递。利用对象序列化可以进行对象的“深复制”，即复制对象本身及引用的对象本身。序列化一个对象可能得到整个对象序列。

比如要进行网络传输的时候 或者在进行分布式架构搭建的时候这个时候就需要进行序列化和反序列化

## **谈谈你对zookeeper的理解**

我们zookeeper主要是给分布式应用提供服务注册中心的，Dubbo服务端往zookeeper注册

服务，然后Dubbo客户端从zookeeper调用服务

zookeeper应用场景：分布式通知、协调、负载均衡、配置中心、分布式锁、分布式对列

分布式锁：

查看目标Node是否已经创建，已经创建，name等待锁

如果未创建，创建一个瞬时node，表示已经占有锁

如果创建失败，那么证明锁已经被其他线程占有了，那么同样等待锁

当释放锁，或者当前session超时的时候，节点被删除，唤醒之前等待锁

zookeeper节点类型：持久节点、持久顺序节点、临时节点、临时顺序节点

部署模式：单机模式，伪集群模式、集群模式

集群规则为2N+1台

集群需要一半以上的机器可用，所以3台挂掉一台还能工作

3.5版本开始支持动态扩容

常见命令：is get set create delete等

zookeeper存储数据：

Zookeeper的数据结构是标准的树结构，在zookeeper中每个节点叫：znode，每个节点有唯一的路径，并且可以给该节点设置描述数据。节点名称和节点描述数据来存储数据的

zookeeper挂了还能正常调用服务吗？

在先去有过调用该服务切提供者还存活的情况下，可以正常调用，应为Dubbo在获取注册中心地址时会将返回的提供者地址缓存到本地，提供者地址没有改变的情况下是可以根据缓存数据调用服务的而非每次都要向注册中心取。

如果先前没有调用过该服务，那么可以做注册中心集群来避免注册中心挂掉的问题，毕竟一个挂了其他备用的注册中心可以使用

zookeeper：消息共享，实现一个分布式共享锁就可以了

## **谈谈你对分布式锁的理解**

在很多场景中，我们为了保证数据的最终一致性，需要很多的技术方案来支持，比如分布式事务、分布式锁等。有的时候，我们需要保证一个方法在同一时间内只能被同一个线程执行。在单机环境中，Java中其实提供了很多并发处理相关的API，但是这些API在分布式场景中就无能为力了。也就是说单纯的Java Api并不能提供分布式锁的能力。所以针对分布式锁的实现目前有多种方案。

基于缓存（Redis）实现分布式锁

基于缓存来实现在性能方面会表现的更好一点。而且很多缓存是可以集群部署的，可以解决单点问题。 使用的是Redis中的原子性操作的Redis为单进程单线程模式，采用队列模式将并发访问变成串行访问，且多客户端对Redis的连接并不存在竞争关系。Redis的SETNX命令可以方便的实现分布式锁。setNX（SET if Not eXists）

如果一个持有锁的客户端失败或崩溃了不能释放锁，该怎么解决？我们可以通过锁的键对应的时间戳来判断这种情况是否发生了，如果当前的时间已经大于lock.foo的值，说明该锁已失效，可以被重新使用

## **分布式事务的处理方式?**

使用MQ来实现数据最终一致性，TCC,LCN开源的框架。

## **你觉得MQ是在架构的哪一个层次用? 作为一个中间件起到了什么作用?**

服务和服务之间的解耦。

解耦，削峰，提高可靠性

## **为什么要使用ActiveMQ？**

ActiveMQ的应用场景有：1存在并发访问的业务2耗时比较久的业务3需要解耦出来的业务，我们的项目出现了ActiveMQ可以解决的场景，所以有用ActiveMQ

## **ActiveMQ和JSM有什么关系？**

JMS是Java针对消息系统的一套接口标准，ActiveMQ实现了这套表标准技术，他们是接口和实现的关系

## **ActiveMQ的消息传送模式有哪些？**

消息传送模式有持久化和非持久化，持久化会把消息持久化到磁盘的，如果消息没有被消费，服务器挂了，也不会丢失。非持久化只是存在于内存中，所以服务器挂了消息会丢失

## **ActiveMQ有哪几种消息签收方式？**

ActiveMQ的消息签收方式总的来说4中，如果使用的session是事务的，那么消息的签收是提交事务就签收了的，如果不带事务的session那么消息的签收方式有3种，分别是自动签收，手动签收，不必须签收

## **ActiveMQ的原理：**

生产者将消息发送给AvtiveMQ服务端，服务端会根据该消息对应的目标模型，（点对点对列，发布/订阅）将消息发送给可以接受的消费者，期间默认回见数据进行持久化，并等待消费者签收消息后才将消息删除避免消息丢失

## **哪些地方用到了ActiveMQ**

订单系统里面用了ActiveMQ，因为订单的处理流程比较长，比较耗时，避免并发量高时DB压力处理不过来，我们会将待创建的订单放到对列中，然后给用户类似正在创建订单的消息，当后台创建完成时再直接提示创建成功，这样在高并发情况下消息入队，再由订单服务完成订单创建操作，则可以极大地提升性能，当订单支付完成之后，还可以直接发布订单创建成功的消息，通知财务记账、物流生成发布订单等多个操作并行处理

## **使用消息队列是为了解决什么问题**

解耦，削峰，提高服务可靠性

## **对于ActiveMQ丢单的情况的处理?**

 可以建立本地消息表，我往mq发送消息前先往本地消息表中插入一条数据，然后定时器轮训这张表，往mq里面写数据。

消费者消费之后需要回调修改你的本地消息表，删除本地消息。

如果出现刚好的丢单，我们轮训本地消息表，比如5分钟还没有删除的本地的消息，我们再重复发到mq中。(下一次可能就变成15分钟检查，但只重发三次，还不能消费就标记需要手工处理。)

## **你觉得微服务和分布式的区别?**

微服务就是分布式的一种类型，但是跟分布式应用又有细微的区别，微服务可以是在业务拆分上粒度更细，并且自己是一小套完整的独立运行的程序，跟其他系统不相互依赖，而分布式不一样是完全独立的，有可能是跟其他系统存在依赖关系。

## **谈谈对于接口限制防刷的实现?**

1、验证码

2、限制ip，客户端请求的时候 ， 把ip记录下来，每次访问这个ip访问次数+1，如果查过制定次数，把这个ip拉黑

3、签名机制。像调用资金托管接口一样。需要调用接口需要获取签名参数，接口返回值也要对其进行回签。这样，如果用户直接调用接口的话是不行的，因为没获取到签名的参数。

## **秒杀系统的亮点在哪里?开发中遇到了什么难点?怎么解决超卖的问题?**

## **超卖的功能用到了乐观锁(回答是否应该加入乐观锁?)?乐观锁你是怎么设计的?怎么保证商品是按顺序去抢购的?**

## **后台服务最后保障秒杀不超卖的方式?如果用乐观锁怎么做?乐观锁能不能不用version而是用库存来判断?有没有考虑过最后的减完的数据不是在Redis中判断而是在数据库中判断?**

我们在减库存的时候 update 库存表 where 库存>0. 我们在做秒杀前会把库存预读到redis中，利用redis的原子自减来判断库存。

## **商品秒杀功能的实现流程**

## **秒杀用了队列?不是一个同步的系统?怎么保证用户的体验?**

在保证性能的情况下再保证用户的体验.

这个问题真的很广泛。即使是阿里巴巴在双十一也不能保证所有用户的体验都很好，现在双十一做了快10年了，今年还是无法选择收货地址的问题。

而且用户体验这个阿里巴巴也是做了很多的改变的。

## **秒杀系统怎么做限流? nginx做限流会是怎样的情况?只有小部分的请求过来了,其他请求过不来,公平性保证有问题,怎么解决?**

我们系统中的限流就通过Nginx来完成的，当然除了Nginx还有其他的限流的措施,比如 Guava中开源出来一个令牌桶算法的工具类RateLimiter,可以轻松实现限流的工作。

我们通过Nginx的限流根据ip限制请求数和请求频率，方式过大的流量导致服务器奔溃。

我们尽可能保证公平性，但不能保证绝对的公平。我们会限制单个用户只能抢购一个商品。

应该优先考虑服务器性能的问题，然后再考虑公平性的问题。

这个时候就可以反问一下面试官，对于性能和公平性的问题怎么取舍或者处理?

## **nginx做了负载均衡,除了轮询还有哪些选择?**

轮询 默认方式

Weight 权重方式

Ip\_hash 根据ip分配方式

Least\_conn 最少连接方式

Fair(第三方) 响应时间 方式

Url\_hash(第三发) 根据url分配方式

## **MySQL的优化**

1. 尽量减少不必要的要求

2. 冗余数据的处理；

3. 大表拆小表，有大数据的列单独拆成小表

4. 把常用属性分离成小表

5. 添加对应的索引

## **MySQL数据库的常见的索引有几种实现方式?**

1.b-tree使用平衡树实现索引，是MySQL中使用最多的索引类型；在innodb中，存在两种索引类型，第一种是主键索引（primary key），在索引内容中直接保存数据的地址；第二种是其他索引，在索引内容中保存的是指向主键索引的引用；所以在使用innodb的时候，要尽量的使用主键索引，速度非常快；

2,hash:把索引的值做hash运算，并存放到hash表中，使用较少，一般是memory引擎使用；优点:因为使用hash表存储，按照常理，hash的性能比B-TREE效率高很多。

1，hash索引只能适用于精确的值比较，=，in，或者<>；无法使用范围查询;

2，无法使用索引排序；

3，组合hash索引无法使用部分索引；

4，如果大量索引hash值相同，性能较低；

## **MySQL的执行流程**

1. 客户端发送一条查询给服务器；

2. 服务器通过权限检查之后,先会检查查询缓存，如果命中了缓存，则立即返回存储在缓存中的结果。否则进入下一阶段；

3. 服务器端进行SQL解析、预处理，再由优化器根据该SQL所涉及到的数据表的统计信息进行计算，生成对应的执行计划；

4. MySQL根据优化器生成的执行计划，调用存储引擎的API来执行查询；

5. 将结果返回给客户端。

SQL执行的最大瓶颈在于磁盘的IO，即数据的读取；不同SQL的写法，会造成不同的执行计划的执行，而不同的执行计划在IO的上面临完全不一样的数量级，从而造成性能的差距；

所以,我们说,优化SQL,其实就是让查询优化器根据程序猿的计划选择匹配的执行计划,来减少查询中产生的IO;

## **MySQL数据库的常见优化**

1、索引，建立索引是数据库优化各种方案之中成本最低，见效最快的解决方案.

索引一般加在查询条件的关键字上，如果有多个查询条件关键字，还可以添加组合索引，写sql的时候需要注意，索引字段和sql字段需要保持一致，否则索引会无效

2. 分库分表分区 分库，可以按照业务分库，分流数据库并发压力，使数据库表更加有条理性，最起码更加好找吧 分表，刚才说了，索引适合应对百万级别的数据量，千万级别数据量使用的好，勉强也能凑合，但如果是上亿级别的数据量

3. 数据库引擎 myisam innodb

4. 预处理 一般来说，实时数据(当天的数据)还是比较有限的，真正数据量比较大的是历史数据，基于大表历史数据的查询，如果再涉及一些大表关联，这种sql是非常难以优化的

5. MySQL like查询

6. 读写分离

## **MySQL的行锁和表锁**

innodb支持行锁和表锁，mysiam不支持行锁。

行锁会出现死锁情况。行锁包括：共享锁和排他锁。

表锁则不会出现死锁的情况。

共享锁又称：读锁。当一个事务对某几行上读锁时，允许其他事务对这几行进行读操作，但不允许其进行写操作，也不允许其他事务给这几行上排它锁，但允许上读锁。

排它锁又称：写锁。当一个事务对某几个上写锁时，不允许其他事务写，但允许读。更不允许其他事务给这几行上任何锁。包括写锁

上共享锁的写法：lock in share mode

例如： select math from zje where math>60 lock in share mode；

上排它锁的写法：for update

例如：select math from zje where math >60 for update；

锁冲突：例如说事务A将某几行上锁后，事务B又对其上锁，锁不能共存否则会出现锁冲突。（但是共享锁可以共存，共享锁和排它锁不能共存，排它锁和排他锁也不可以）

死锁：例如说两个事务，事务A锁住了1~5行，同时事务B锁住了6~10行，此时事务A请求锁住6~10行，就会阻塞直到事务B施放6~10行的锁，而随后事务B又请求锁住1~5行，事务B也阻塞直到事务A释放1~5行的锁。死锁发生时，会产生Deadlock错误。

## **MySQL的主从配置**

1、添加一个新的MySQL实例

(1)复制MySQL安装目录

(2)修改新MySQL的相关配置（端口，安装目录，数据目录）

(3)为新的MySQL创建数据目录

(4)为新的MySQL创建window系统服务

(5)检查创建的系统服务，并且启动测试

2、配置主从

3、启动主数据库执行 show master status命令

4、配置从服务器

5、启动从服务器

6、恢复主数据库的备份

7、让从数据库指定Master库

8、启动MySQL

## **如何实现读写分离**

1、一个service方法必须定位到唯一的数据库上

2、引入路由DataSource

(1)在应用中，需要自己去确定这个要访问的真实目标DataSource

(2)让路由DS知道哪些真实DataSource和他门对应的名字

(3)让路由DS根据传入的名字去返回真实的DS

3、完成读写分离

(1) 1,认识Spring中的AbstractRountingDataSource;

(2) 在Spring中提供了AbstractRountingDataSource来完成路由DS的功能;

(3) 1,targetDataSources:用于配置真实的datasource,这个属性是一个Map,Map的key就是DS的名字,Map的value是真实的DS对象;

(4) 2,defaultTargetDataSource:如果路由DS没有找到你当前请求的DS,直接使用默认的DS;

(5) 3,abstract Object determineCurrentLookupKey():这个方法是需要我们自己实现的,这个方法需要返回一个值,这个值就是DS的名字;这个方法就是让应用(我们)提供给Spring怎么找DS名字的逻辑;

(6) 2,步骤:

(7) 1,创建一个继承AbstractRountingDataSource的对象,并在Spring中配置;

(8) 2,修改原来的datasource的配置:

## **Springboot拦截器的使用**

1、定义一个拦截类，该类实现HandlerInterceptor接口，实现preHandler方法。

2、定义一个配置类，该类继承WebMvcConfigurerAdapter，覆写addInterceptors方法，其作用是将拦截器类注册都容器里面去，从而起到作用，并且指定拦截的请求的路径。

## **Springboot和Spring之间的区别**

额，其实这两个东西都不是同一类型的东西，来做区别也不太好讲。

Spring其实是一个很大的框架，包含了很多的模块，SpringBoot,Spring Framework,Spring DATA,Spring batch,Spring Security.

而SpringBoot只是Spring中的一个模块，是Spring团队在Spring4.0版本上二次开发并开源公布出来的。简而言之，SpringBoot就是一个轻量级，简化配置和开发流程的web整合框架

## **Spring mvc 和 Springboot的区别**

Spring boot 实现了自动配置的，降低了项目搭建的复杂度可以直接打成jar包，里面默认Tomcat启动的jar包

Spring MVC是基于 Servlet 的一个 MVC 框架 主要解决 WEB 开发的问题，因为 Spring 的配置非常复杂，各种XML、 JavaConfig、hin处理起来比较繁琐。于是为了简化开发者的使用，从而创造性地推出了Spring boot，约定优于配置，简化了spring的配置流程。

## **简单描述Springboot的自动装配原理**

在SpringBoot的@SpringBootApplication注解中有一个注解@EnableAutoConfiguration

，这个注解的功能就是实现自动装配的。他的作用主要是扫描导入每个jar中的spring.factory中的内容，然后把里面的内容读取到内存中，其实就是读取一堆的配置类。然后引入之后里面有些@Codictional的注解，根据特定的条件帮我们创建bean并注入，省去我们自己配置bean的过程，简化了开发步骤.

## **Springboot你们用的是哪个版本? 你们是怎么处理全局的异常的?**

关于SpringBoot的版本号，提供几个时间点给同学们，你们根据自己的简历来说对应的版本.

SprintBoot1.3.1的版本大概在2015年12月发布

SpringBoot1.4.1的版本大概在2016年09月发布

SpringBoot1.5.1的版本大概在2017年2月发布

SpringBoot2.0的版本大概在2018年3月发布

[对于全局异常，我们项目中使用@ControllerAdvice+@ExceptionHandler来捕获全局的异常，我们根据抛出来的不同的自定义异常做不同的处理.有些异常为了方便查询检索，存储到数据库中，有些异常会通过日志文件的方式存储。](mailto:对于全局异常，我们项目中使用@ControllerAdvice+@ExceptionHandler来捕获全局的异常，我们根据抛出来的不同的自定义异常做不同的处理.有些异常为了方便查询检索，存储到数据库中，有些异常会通过日志文件的方式存储。)

(如果这么回答的话，就要考虑你们的日志是按什么方式存储，怎么切分的。比如使用logback按天来进行切割)

自己准备好说的内容里面涉及到其他的技术点，要了解一下，防止别人发散发问.

## **Springboot的常用的注解**

@SpringBootApplication,@Bean,@PropertySource.@Profile 在写项目中用了哪些注解就写哪些.

这种问题只要把你做项目用过的注解说出来就可以了.可能面试官还会问你其他的SpringBoot的注解，如果真的不知道没关系，直接说这个没用过，面试完之后简单看一下就可以了.

## **Springboot 默认的支持对 Json 的解析**

Springboot 默认使用的json解析框架是jackson,使用fastjson需要配置可以使用阿里的fastjson的方法进行解析

## **进入docker容器的几种方法**

attach和exec子命令，也可以nsenter工具进入容器

## **Nginx的作用?nginx的负载均衡怎么做?除了nginx还有哪些可以做负载均衡?**

Nginx的主要应用场景:负载均衡，反向代理，代理缓存，限流等。

Nginx的负载均衡需要在配置文件中添加upstream节点，然后再这个节点下配置对于的主机。

里面主要涉及到的负载均衡策略有:轮训（默认）,ip\_hash,权重等.

比如HAProxy,LVS等，面试的时候说简单了解即可，不要给套进去了.

## **谈谈你对ElasticSearch的理解?**

Elasticsearch是一个基于Lucene搜索引擎为核心构建的开源，分布式，RESTful搜索服务器。设计用于云计算中，能够达到实时搜索，稳定，可靠，快速，安装使用方便，轻松扩展服务节点。 Elasticsearch是用Java开发的，但它却不是只支持Java语言，因为它支持RESTful方式调用，那理论上它是支持所有开发语言的，除此之外，如果你不想使用RESTful方式调用Elasticsearch服务器，那Elasticsearch还提供了各种语言的api供我们使用。

## **ElasticSearch的算法优化**

这个问题问题的有点问题，使用elasticsearch不需要我们直接去优化算法，所有的优化都是通过配置完成。

1分片的数量不少越多越好，因为查询结果是所有分片数据组装起来的，分片越多，消耗的IO越大，最好的是分片数等于节点数乘以3，比如有3个节点，那么索引的分片就创建9个。

2默认情况下，一个分片对应着一个副本，副本可以起到负载均衡的效果，因为分片和副本都可以处理搜索的请求，所以适当的增加副本的数量，可以大大的提高系统的吞吐量，不过副本也会增加系统的压力，还会增加磁盘消耗，所以一般来说，设置2-3个副本就可以了。

3设置线程池大小

4尽量减少不需要的字段，这样可以节省一个文档所消耗的空间，在相同数据文档量的前提下，字段越少，搜索的速度越快

5锁定内存，不让JVM写入swapping，避免降低ES的性能，对应的配置为bootstrap.mlockall: true

6定时处理缓存，GET http://127.0.0.1:9200/\*/\_cache/clear

定时合并优化资源，GET http://127.0.0.1:9200/\*/\_optimize

## **ElasticSearch使用的是query还是filter, 两者之间有什么区别**

query查询过程：

1，比较查询条件；

2，然后计算分值，最后返回文档结果。

这种查询方式适合于全文检索类的查询。

filter查询过程

1，判断是否满足查询条件，如果不满足，会缓存查询过程（记录该文档不满足结果）；

2，满足的话，就直接缓存结果。

这种查询方式适合于精确值匹配方式的查询。

综上所述，filter快在两个方面：

1 对结果进行缓存；

2 避免计算文档相关性分值。

当我们不关心搜索结果的评分或者不使用全文检索的时候，为了提高性能，注意使用filter。

## **什么是fastDFS**

2.1 fastDSF介绍

FastDFS是用c语言编写的一款开源的分布式文件系统，它是由淘宝资深架构师余庆编写并开源。FastDFS专为互联网量身定制，充分考虑了冗余备份、负载均衡、线性扩容等机制，并注重高可用、高性能等指标，使用FastDFS很容易搭建一套高性能的文件服务器集群提供文件上传、下载等服务。

​ 为什么要使用fastDFS呢？

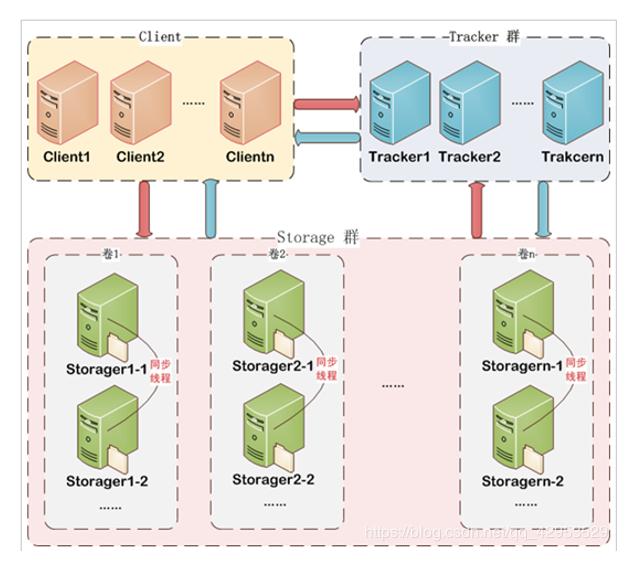
​ 上边介绍的NFS、GFS都是通用的分布式文件系统，通用的分布式文件系统的优点的是开发体验好，但是系统复杂性高、性能一般，而专用的分布式文件系统虽然开发体验性差，但是系统复杂性低并且性能高。fastDFS非常适合存储图片等那些小文件，fastDFS不对文件进行分块，所以它就没有分块合并的开销，fastDFS网络通信采用socket，通信速度很快。

2.2 fastDSF工作原理

2.2.1 fastDSF架构

FastDFS架构包括 Tracker server和Storageserver。客户端请求Tracker server进行文件上传、下载，通过Tracker server调度最终由Storage server完成文件上传和下载。

如下图：



1）Tracker

​ Tracker Server作用是负载均衡和调度，通过Tracker server在文件上传时可以根据一些策略找到Storage server提供文件上传服务。可以将tracker称为追踪服务器或调度服务器。

​ FastDFS集群中的Tracker server可以有多台，Tracker server之间是相互平等关系同时提供服务，Tracker server不存在单点故障。客户端请求Tracker server采用轮询方式，如果请求的tracker无法提供服务则换另一个tracker。

2）Storage

​ Storage Server作用是文件存储，客户端上传的文件最终存储在Storage服务器上，Storage server没有实现自己的文件系统而是使用操作系统的文件系统来管理文件。可以将storage称为存储服务器。

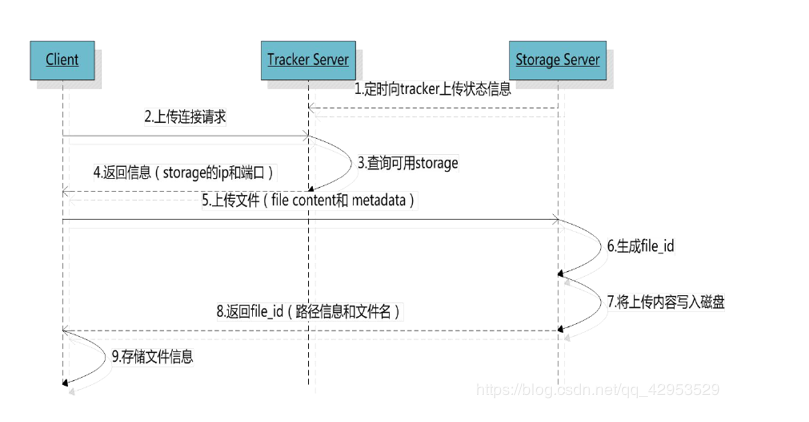
​ Storage集群采用了分组存储方式。storage集群由一个或多个组构成，集群存储总容量为集群中所有组的存储容量之和。一个组由一台或多台存储服务器组成，组内的Storage server之间是平等关系，不同组的Storage server之间不会相互通信，同组内的Storage server之间会相互连接进行文件同步，从而保证同组内每个storage上的文件完全一致的。一个组的存储容量为该组内的存储服务器容量最小的那个，由此可见组内存储服务器的软硬件配置最好是一致的。

​ 采用分组存储方式的好处是灵活、可控性较强。比如上传文件时，可以由客户端直接指定上传到的组也可以由tracker进行调度选择。一个分组的存储服务器访问压力较大时，可以在该组增加存储服务器来扩充服务能力（纵向扩容）。当系统容量不足时，可以增加组来扩充存储容量（横向扩容）。

3）Storage状态收集

​ Storage server会连接集群中所有的Tracker server，定时向他们报告自己的状态，包括磁盘剩余空间、文件同步状况、文件上传下载次数等统计信息。

2.2.2文件上传流程



客户端上传文件后存储服务器将文件ID返回给客户端，此文件ID用于以后访问该文件的索引信息。文件索引信息包括：组名，虚拟磁盘路径，数据两级目录，文件名。



组名：文件上传后所在的storage组名称，在文件上传成功后有storage服务器返回，需要客户端自行保存。

虚拟磁盘路径：storage配置的虚拟路径，与磁盘选项store\_path 对应。如果配置了store\_path0则是M00，如果配置了store\_path1则是M01，以此类推。

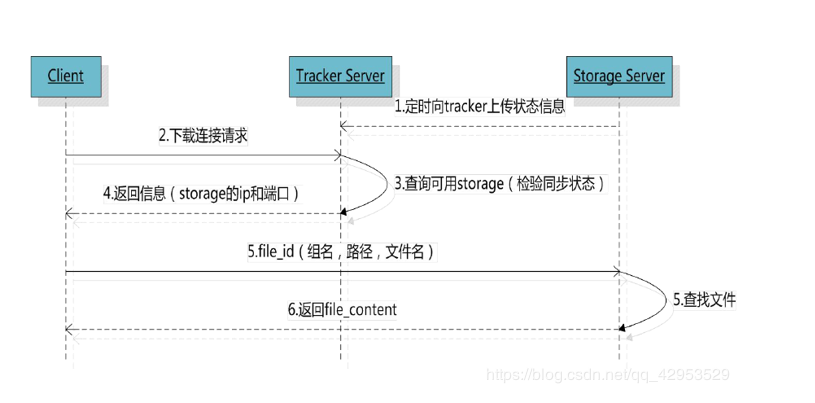
数据两级目录：storage服务器在每个虚拟磁盘路径下创建的两级目录，用于存储数据文件。

文件名：与文件上传时不同。是由存储服务器根据特定信息生成，文件名包含：源存储服务器IP地址、文件创建时间戳、文件大小、随机数和文件拓展名等信息。

2.2.3 文件下载流程

tracker根据请求的文件路径即文件ID 来快速定义文件。

比如请求下边的文件：



1.通过组名tracker能够很快的定位到客户端需要访问的存储服务器组是group1，并选择合适的存储服务器提供客户端访问。

2.存储服务器根据“文件存储虚拟磁盘路径”和“数据文件两级目录”可以很快定位到文件所在目录，并根据文件名找到客户端需要访问的文件。