自我介绍：

我熟悉大数据方面技术，如 Spark Hadoop 等，并有丰富的实战经验；对设计模式和 Java 虚拟机有较深的理解，有较强的需求分析能力。

平时有时间的话喜欢看技术和科技等方面的书籍和博客，读读感兴趣的技术的源代码。画思维导图和写博客完善自己的技能树。组织同行的朋友进行技术分享和交流。我现在研究机器学习和推荐系统方面的技术。如果有机会希望能一起交流学习。

数据结构和算法：

常用的数据结构和排序算法----数据结构与算法思维导图(ok)

手写数据结构和算法：比较重要，基础中的基础

1：冒泡排序

```
private static void bubbleSort(int [] array){
    int temp=0;
    for(int i=0;i<array.length-1;i++){          //第一个 for 循环控制排序要走多少趟，最多做 n-1 趟排序
        for(int j=0;j<array.length-1-i;j++){    //第 2 个 for 循环控制每趟比较多少次
            if(array[j+1]<array[j]){             //可以将排序的交换算法提出来
            temp=array[j];
            array[j]=array[j+1];
            array[j+1]=temp;
            }
        }
    }
}
```

2：二分查找

```
public static int binarySearch(int srcArray,int des){
int low=0;
int height=srcArray.length-1;
while(low<=height){
    int middle=(low+height)/2;
    if(des==srcArray[middle]){
    return middle;
    }else if(des<srcArray[middle]){
    height=middle-1;
    }else{
    low=middle+1;
    }
}
    return -1;
}
```

递归的方式实现：

```
public static int binarySearch(int[] dataset;int data,int beginIndex,int endIndex){
int midIndex=(beginIndex+endIndex)/2;
if(data<dataset[beginIndex] || data>dataset[endIndex] || beginIndex>endIndex){
```

```
return -1;
}
if(data<dataset[midIndex]){
return binarySearch(dataset,data,beginIndex,midelIndex-1);
}else if(data>dataset[midIndex]){
return binarySearch(dataset,data,midIndex+1,endIndex);
}else{
 return midIndex;
}
}
```

3：单链表反转

```
class ListNode {        int val;        ListNode next;        ListNode(int x) {                val = x;        } }
public static ListNode reverseList(ListNode head){
      ListNode prev=null;
      while(head != null){
      ListNode temp=head.next;
      head.next=prev;
      prev =head;
      head =temp;
      }
      return prev;
}
```

递归的方式实现：

```
public ListNode reverseList(ListNode head) {                    if(head==null||head.next ==null)
return head;              ListNode prev = reverseList(head.next);              head.next.next = head;
            head.next = null;              return prev;        }
```

插入排序：

初始时假设第一个记录自成一个有序序列，其余记录为无序序列。接着从第二个记录开始，按照记录的大小依次将当前处理的记录插入到其之前的有序序列中，直至最后一个记录插入到有序序列中为止

```
public static void insertSort(int[] a){
      int temp;
      for(int i=1;i<a.length;i++){
            for(int j=i;j>0;j--){
                  if(a[j-1]>a[j]){
                  temp=a[j-1]
                  a[j-1]=a[j]
                  a[j]=temp
                  }
            }
      }
}
```

选择排序：把最小或者最大的选择出来

对于给定的一组记录，经过第一轮比较后得到最小的记录，然后将该记录与第一个记录的位置进行交换；接着对不包括第一个记录以外的其他记录进行第二轮比较，得到最小的记录并与第二个记录进行位置交换；重复该过程，直到进行比较的记录只有一个时为止。

```
public static void selectSort(int[] a){
if (a == null || a.length <= 0) { //记住需要判断输入的数据
        return;
}
for(int i=0;i<a.length;i++){
      int min=i;
      for(int j=i+1;j<a.length;j++){
            if(a[j]<a[min]){
            min=j;
            }
      }
       if (i != min) {
           int tmp = a[min];
           a[min] = a[i];
           a[i] = tmp;
        }
}

}
```

快速排序：

基于分治的思想，是冒泡排序的改进型。首先在数组中选择一个基准点（该基准点的选取可能影响快速排序的效率，后面讲解选取的方法），然后分别从数组的两端扫描数组，设两个指示标志（lo 指向起始位置，hi 指向末尾），首先从后半部分开始，如果发现有元素比该基准点的值小，就交换 lo 和 hi 位置的值，然后从前半部分开始扫秒，发现有元素大于基准点的值，就交换 lo 和 hi 位置的值，如此往复循环，直到 lo>=hi,然后把基准点的值放到 hi 这个位置。一次排序就完成了。以后采用递归的方式分别对前半部分和后半部分排序，当前半部分和后半部分均有序时该数组就自然有序了。

```
public static int partition(int []array,int lo,int hi){          //固定的切分方式，将 lo 下标对应的 点 标 记 为 基 准 点                int  key=array[lo];                while(lo<hi){
while(array[hi]>=key&&hi>lo){//从后半部分向前扫描                hi--;                }
                array[lo]=array[hi];                while(array[lo]<=key&&hi>lo){从前半部分向后扫描                lo++;                }                array[hi]=array[lo];                }
            array[hi]=key;                return hi;        }                public static void sort(int[] array,int lo ,int hi){                if(lo>=hi){                return ;                }                int index=partition(array,lo,hi);                sort(array,lo,index-1);                sort(array,index+1,hi);
      }
```

设计模式：会问，但是问的不那么多

设计模式原则和 24 种设计模式----设计模式思维导图(ok)

注意手写单例模式：

懒汉模式:

```
public class Singleton{
```

```
private Singleton (){};
private static Singleton singleton=null;
public static Singleton getInstance(){
            if(singleton==null){
              singleton =new Singleton();
                }
return singleton;
      }
}
```

恶汉模式：

```
public class Singleton{
private Singleton (){};
private static Singleton singleton=new Singleton();
public static Singleton getinstance(){
    return singleton;
 }
}
```

线程安全的单例模式：对 getinstance 方法加锁

实现方式 1：

```
public class Singleton{
private Singleton(){}
private static Singleton singleton=null;
public static synchronized Singleton getInstance(){
    if(single==null){
    singleton=new Singleton();
 }
return singleton;
   }
}
```

实现方式 2：双重检查锁定：对 getinstance 方法加锁+if 判断

```
public class Singleton{
private Singleton(){}
private static Singleton singleton=null;
public static    Singleton getInstance(){
    if(single==null){
      synchronized(Singleton.class){
      if(singleton==null){
   singleton=new Singleton();
 }
}
return singleton;
   }
}
```

实现方式 3：静态内部类：

```
public class Singleton{
private static class getInstaceFac{
private static final Singleton     INSTANCE=new Singleton();
}
private Singleton (){}；
public static final Singleton getinstance(){
return getInstaceFac.INSTANCE;
}
}
```

JVM：面试，问的还行

炼数成金深入理解 JVM+深入理解 java 虚拟机-----JVM 思维导图(基本 ok，思维导图以后补充)

Java虚拟机常见总结问题

Java引用的四种状态

- 强引用StrongReference
  - 特点：垃圾收集器绝对不会回收，超出内存时OOM异常
- 软引用(SoftReference)
  - 特点：内容足够，不回收；内存不足，回收
  - 应用：内存敏感的高速缓存。例如从数据库中查询的信息再次被使用
- 弱引用(WeakReference)
  - 特点：只要GC发现，便收集
  - 应用：非敏感数据的缓存
    - WeakHashMap
- 虚引用(PhantomReference)
  - 特点：任何时候都会被GC收集

Java中的内存划分

- 程序计数器(线程私有)
  - 特点：线程创建时创建，执行本地方法时其值为undefined。
- 虚拟机栈(线程私有)
  - 特点：（栈内存）为虚拟机执行java方法服务；方法被调用时创建栈帧 -->局部变量表-> 局部变量、对象引用
    - 如果线程请求的栈深度超出了虚拟机所允许的深度，就会出现StackOverFlowError。-Xss规定了栈的最大空间
    - 虚拟机栈可以动态扩展，如果扩展到无法申请到足够的内存，会出现OOM
- 本地方法栈(线程私有)
  - 特点：java虚拟机栈是为虚拟机执行java方法服务的，而本地方法栈则为虚拟机执行使用到的Native方法服务。
    - Java虚拟机没有对本地方法栈的使用和数据结构做强制规定。Sun HotSpot把Java虚拟机栈和本地方法栈合二为一
    - 会抛出StackOverFlowError和OutOfMemoryError
- Java堆，即内存堆(线程共享)
  - 特点：被所有线程共享，在Java虚拟机启动时创建，几乎所有的对象实例都存放到堆中
    - GC的管理的主要区域
    - 物理不连续，逻辑上连续，并可以动态扩展，无法扩展时抛出OutOfMemoryError
- 方法区(线程共享)
  - 特点：用于存储已被虚拟机加载的类信息、常量、静态变量、即使编译器编译后的代码的数据
    - Sun HotSpot 虚拟机把方法区叫做永久待(Permanent Generation)
    - 特点：受到方法区的限制，触发OutOfMemoryError
- 运行时常量池(方法区的一部分)

Java对象在内存中的状态

- 可达的/可触及的
  - Java对象被创建后，如果被一个或多个变量引用，就是可达的
- 可恢复的
  - 特点：
    - Java对象不再被任何变量引用就进入可恢复状态
    - 在回收该对象之前，如果在finalize()方法中重新让变量引用该对象，则该对象在此变为可达状态，否则进入不可达状态
    - 在finalize方法中只能恢复一次
- 不可达的
  - 特点：
    - Java对象不被任何变量引用，且在调用对象的finalize()方法后依然没有变成可达状态
    - 当对象处于不可达状态时，系统才会真正回收该对象占有的资源
- 可达性判断
  - 单条引用路径可及性判断 在这条路径中，最弱的一个引用决定对象的可及性
  - 多条引用路径可及性判断 几条路径中，最强的一条的引用决定对象的可及性。

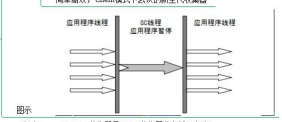
判断对象死亡的方法

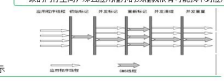
- 引用计数算法
  - 概念：
    - 给对象添加一个引用计数器，每当有一个地方引用它时，计数器加1，放引用失效时，计数器值减1，任何计数器为0的对象就是不可能再被使用的
    - 目前主流的Java虚拟机并没有选用引用计数算法来管理内存，因为它很难解决对象之间相互循环引用的问题
  - 优点：
    - 算法实现简单，判定效率高
  - 缺点：
    - 引用和去引用伴随加法和减法，影响性能
    - 致命缺陷，对循环引用的对象无法进行回收
- 根搜索算法
  - 概念：设立若干种根对象，当任何一个根对象(GC Root)到某一个对象均不可达时，则认为对象是可以被回收的
  - 根(GC Root)
    - 1、栈（栈帧中的本地变量表）中引用的对象。
    - 2、方法区中的静态成员。
    - 3、方法区中的常量引用的对象（全局变量）
    - 4、本地方法栈中JNI（一般说的Native方法）引用的对象。
  - 应用
    - 标记-清除
    - 标记整理
    - 标记复制

垃圾回收算法

- 标记-清除算法
  - 概念：
    - 标记阶段：先通过根节点，标记所有从根节点开始的对象，未被标记的为垃圾对象
    - 清除阶段：清除所有未被标记的对象
  - 缺点：
    - 标记和清除的过程效率不高
    - 标记清除后会产生大量不连续的碎片
- 复制算法
  - 概念：将原有的内存空间分为两块，每次只使用其中一块，在垃圾回收时，将正在使用的内存中的存活对象复制到未使用的内存块中，然后清除正在使用的内存块中的所有对象，
  - 优点：
    - 每次对整个半区进行回收，内存分配时不用考虑碎片等情况
    - 移动堆顶指针，实现简单，运行效率高
  - 缺点：
    - 空间浪费一半
- 标记-整理
  - 概念：
    - 标记阶段：先通过根节点，标记所有从根节点开始的可达对象，为被标记的为垃圾对象
    - 整理阶段：将所有的存活对象压缩到内存的一段，之后清理边界所有的空间
  - 优点：
    - 不会产生内存碎片
  - 缺点：
    - 在标记的基础上还需要进行对象的移动，成本相对较高，效率也不高
- 前三个算法的比较
  - （1）效率：复制算法 > 标记/整理算法 > 标记/清除算法（此处的效率只是简单的对比时间复杂度，实际情况不一定如此）。
  - （2）内存整齐度：复制算法=标记/整理算法>标记/清除算法。
  - （3）内存利用率：标记/整理算法=标记/清除算法>复制算法。
- 分代收集算法
  - 概念：
    - 根据对象的存活中期的不同，将内存分为几块。一般吧Java堆分为新生代和老生代：短命对象归为新生代，长命对象归为老生代
  - 特点：
    - 在新生代中，每次GC时就会有大批对象死去，只有少量存活，选用复制算法
    - 老年代中对象存活率高，没有额外空间对他进行分配担保，使用"标记-清理"/"标记整理算法"
    - 老年代中的对象，有一部分是因为在新生代回收时，老年代做担保。绝大多数对象是因为很多次GC后没有被回收而进入老年代的。

垃圾收集器

- Serial收集器
  - 概念：是一个单线程的收集器，不是只能使用一个CPU。在进行垃圾收集时，必须暂停其他所有的工作线程，直到收集结束。
  - 特点：
    - 新生代采用复制算法，Stop-The-World
    - 老年代采用标记-整理算法，Stop-The-World
    - 简单高效，Client模式下默认的新生代收集器
  - 图示： 

- ParNew收集器
  - 概念：ParNew收集器是Serial收集器的多线程版本
  - 特点：
    - 新生代采用复制算法，Stop-The-World
    - 老年代采用标记-整理算法，Stop-The-World
    - 它是运行在Server模式下首选新生代收集器
    - 除了Serial收集器之外，只有它能和CMS收集器配合工作
  - 图示： 
- ParNew Scavenge收集器
  - 概念：
    - 停顿时间和吞吐量不可能同时调优。我们一方面希望停顿时间少，另外一方面希望吞吐量高，其实这是矛盾的。因为：在GC时候，垃圾回收的工作总量是不变的，如果将停顿时间减少，那频率就会提高；既然频率提高了，说明就会频繁的进行GC，那吞吐量就会减少，性能就会降低。
  - 相关概念：
    - 吞吐量：CPU用于用户代码的时间/CPU总消耗时间的比值，即=运行用户代码的时间/(运行用户代码时间+垃圾收集时间)。比如，虚拟机总共运行了100分钟，其中垃圾收集花掉1分钟，那吞吐量就是99%。
- G1收集器
  - 概念：是当今收集器发展的最前沿成果之一，对垃圾回收进行了划分优先级的操作，这种有优先级的区域回收方式保证了它的高效率
  - 优点：
    - 最大的优点是结合了空间整合，不会产生大量的碎片，也降低了进行gc的频率
    - 让使用者明确指定指定停顿时间
- 以上所有的收集器当中，当执行GC时，都会 Stop-the-world. 但下面的CMS收集器不会这样
- CMS收集器：（Concurrent Mark Sweep：并发标记清除老年代收集器）
  - 概念：一种以获得最短回收停顿时间为目标的收集器，适用于互联网站或者B/S系统的服务器上
  - 运行过程
    - 初始标记(Stop-The-World)：根可以直接关联到的对象，速度快
    - 并发标记(和用户线程一起)：主要标记过程，标记全部对象
    - 重新标记(Stop-The-World)：由于并发标记时，用户线程依然运行，因此在正式清理前，再更修正
    - 并发清除(和用户线程一起)：基于标记结果，直接清理对象
  - 优点：并发收集，低停顿
  - 缺点：
    - （1）导致用户的执行速度降低。
    - （2）无法处理浮动垃圾。因为它采用的是标记-清除算法。有可能有些垃圾在标记之后，需要等到下一次GC才会被回收。如果CMS运行期间无法满足程序需要，那么就会临时启用Serial Old收集器来重新进行老年代的收集。
    - （3）由于采用的是标记-清除算法，那么就会产生大量的碎片。往往会出现老年代还有很大的空间剩余，但是无法找到足够大的连续空间来分配当前对象，不得不提前触发一次full GC
  - 疑问：
    - 问：既然标记-清除算法会造成内存空间的碎片化，CMS收集器为什么使用标记清除算法而不是使用标记整理算法：
    - 答：CMS收集器更加关注停顿，它在被GC的时候是和用户线程一起工作的（并发执行），如果使用标记整理算法的话，那么在清理的时候就会去移动可用对象的内存空间，那么应用程序的线程就很有可能找不到应用对象在哪里。
  - 图示： 

Java堆内存划分

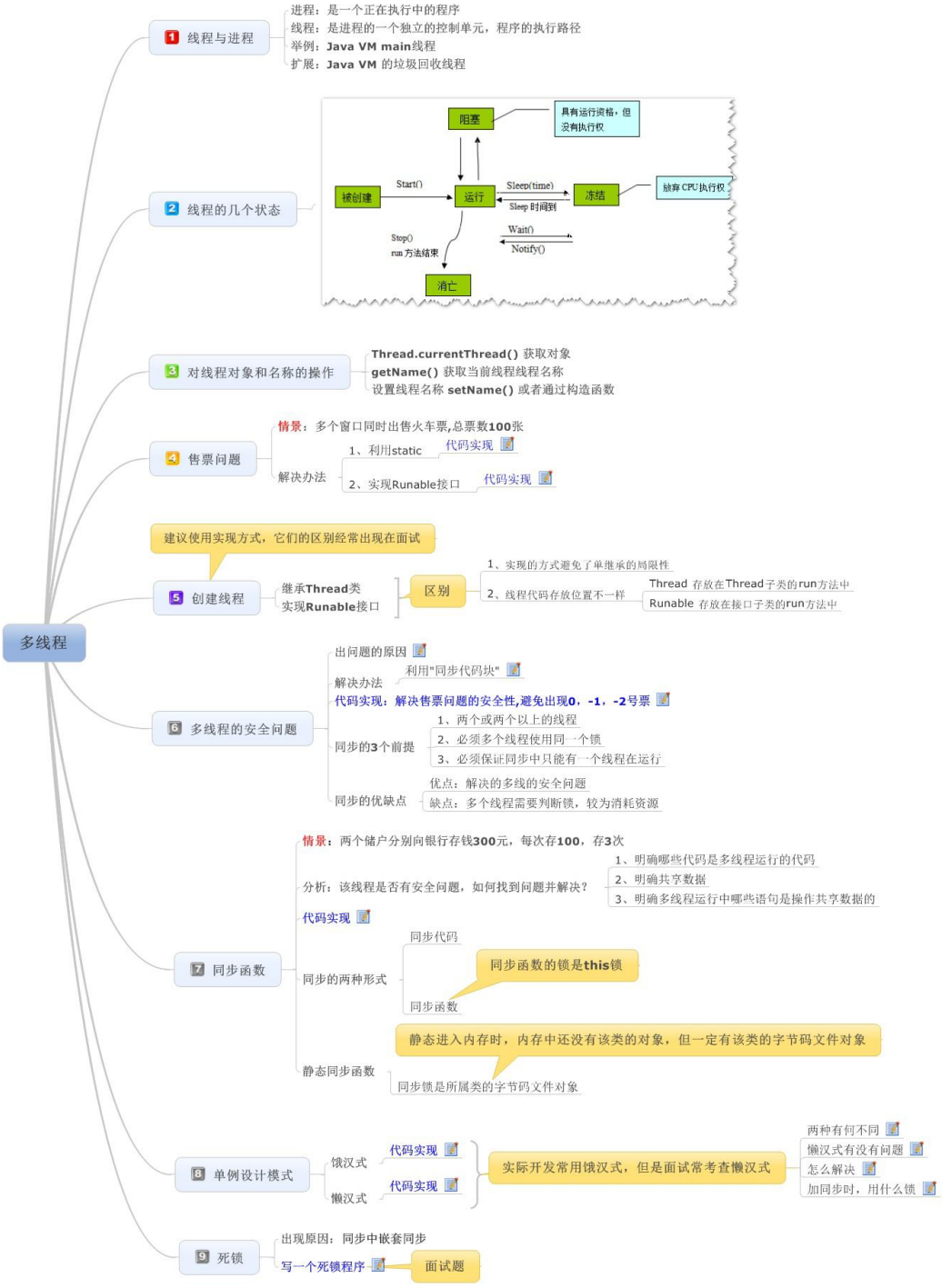
划分依据：根据对象的存活率(年龄)

- 新生代
  - 概念：在方法中去new一个对象，那这方法调用完毕后，对象就会被回收，这就是一个典型的新生代对象。
- 老生代
  - 概念：
    - 在新生代中经历了N次垃圾回收后仍然存活的对象就会被放到老年代中。而且大对象直接进入老年代
    - 当Survivor空间不够用时，需要依赖于老年代进行分配担保，所以大对象直接进入老年代
- 永久代
  - 概念：即方法区。

MiniorGC、FullGC

- Minor GC
  - 概念：新生代中的垃圾收集动作，采用的是复制算法
  - 特点：对于较大的对象，在Minor GC的时候可以直接进入老年代
- Full GC
  - 概念：Full GC是发生在老年代的垃圾收集动作，采用的是标记-清除/整理算法。
  - 特点：由于老年代的对象几乎都是在Survivor区熬过来的，不会那么容易死掉。因此Full GC发生的次数不会有Minor GC那么频繁，并且Time(Full GC)>Time(Minor GC)

java：多线程高并发编程思维导图(多线程非常重要)--面试常问



Java：集合类

scala 相关：scala 知识点思维导图总结(ok)

linux 相关：思维导图(常用的命令等)

大数据框架：

hadoop 体系：

hadoop 思维导图：HDFS；yarn ；MapReduce：

带相应的附件和案例：源文件在：有道云笔记：云计算-大数据-机器学习-人工智能/大数据

组件/Hadoop 下面。
hadoop 的重点是理解分布式的存储方式和分布式的计算方式及实现原理：
HDFS：
MR：
yarn：

Hive 思维导图：带相应的附件和案例：源文件在：有道云笔记：云计算-大数据-机器学习-人工智能/大数据组件/Hive/Hive 思维导图
http://note.youdao.com/noteshare?id=b1d43422fe74512a1d7755e74d66dec0&sub=FFC308FBFB5D4B3AB058844C4F248F49
Hbase 思维导图：
带相应的附件和案例：源文件在：有道云笔记：云计算-大数据-机器学习-人工智能/大数据组件/HBase/HBase 思维导图
spark 体系：
Spark Core 思维导图(ok)
SparkSQL 思维导图(ok)
SparkStreaming 思维导图(ok)
SparkMLLib 思维导图(ok)---深入研究逻辑回归
其他大数据组件：
Flume 思维导图：重点在 Agent 的设计和多 Flume 的架构设计，负载均衡和故障迁移
带相应的附件和案例：源文件在：有道云笔记：云计算-大数据-机器学习-人工智能/大数据组件/Flume/Flume 思维导图下面
Sqoop 思维导图：
Sqoop 思维导图：重点。在 Sqoop 的导入导出的 import 参数的设置和 export 的参数的设置
带相应的附件和案例：源文件在：有道云笔记：云计算-大数据-机器学习-人工智能/大数据组件/Sqoop/Sqoop 思维导图下面
Storm 思维导图：重点在 storm topology DAG 有向无环图的构建上面(spolt,bolt,topology)；Storm 的并发机制(进程 worker(slot),线程，rebalance)；Storm 的容错机制(集群，各进程组件(nimbus,supervisor，worker)，消息的完整性(acker 机制)，)；Storm 架构(Nimbus，supervisor，Zookeeper，worker，excutors:无状态，快速启停的设计)及和其他流式框架(SparkStreaming，Driud)的和 MR 的对比(架构，使用场景，实现等)；Storm 使用场景（同步(DRPCserver)，异步）；Storm DRPC 的实现(闭环的架构，实现方式：线性的 topologybuilder；普通的 topology)；Storm 的事务(实现 Storm 事务的 3 种思路：tuple(强有序事务 id)，batch(批量的强有序事务 id)，分阶段（能并行的强有序 spout 发送和强有序的事务提交）)
源文件在：有道云笔记：云计算-大数据-机器学习-人工智能/大数据组件/Storm/Storm 思维导图下面：
Redis 思维导图

Zookeeper 思维导图

Kafka 思维导图：重点
机器学习：
常用的机器学习算法总结---机器学习思维导图(ok)

推荐系统的总结：
扩展性思维题：（比较重要，问的也比较多）
自身的问题：
Hadoop 项目的总结：大数据分析平台
关于项目的面试问题思考：
http://note.youdao.com/noteshare?id=1031902275fa3ce51a0857a33de03773&sub=E715696374CF42D3A17E897F4AFFD869

推荐系统项目总结：项目讲解的步骤
（1）大数据项目的一个架构：
离线的批量计算+在线的实时计算
（2）推荐系统的必要性：
解决在信息过载（分类目录（雅虎和门户网站->搜索引擎（谷歌的搜索引擎）->亚马逊的推荐系统）+长尾理论（解决小部分用户的个性化需求））的且用户没有明确的目的搜索目的下根据用户的历史行为，兴趣，寻找用户感兴趣的物品的系统，搜索引擎是用户有明确的目的---一般采用的是搜索引擎+推荐系统的组合方式）
（3）推荐系统的应用：
电商的亚马逊（35%）+京东（18%）+netflix 网飞（推荐系统大赛）+豆瓣（豆瓣 FM），今日头条，都有很多的应用
（4）计算和获得指标的一些方法：
离线的实验（测试不同的算法，不参与真实的系统，难于获得一些指标：点击率，转化率等）
用户调查（保证双盲实验，成本较高，能够获得一些离线实验获得不了的重要的指标）
在线实验（在线 AB 测试：AB 测试系统，公平的获取不同的算法的性能指标，周期会长一些）
一个推荐算法的上线需要：离线实验指标优于现在的，用户调查用户满意度不低于现在的 AB 测试我们关心的指标优于现在的
（5）好的推荐系统的指标：用户+推荐系统+物品组成的生态，权衡各个方向的利益
好的推荐系统能够收集高质量的用户的反馈，不断的提高推荐系统质量
用户的满意度(最重要的指标：用户调查，在线实验获得)
准确率（衡量一个推荐算法预测用户行为的能力：最重要的推荐系统的离线的评测指标）
一般 TopN 的预测准确度的指标有：准确率和召回率
覆盖率（描述推荐系统发掘长尾的能力）:信息熵或者基尼系数，消除推荐系统的马太效应
多样性（覆盖用户的不同兴趣）：推荐列表中的物品的两两不相似度衡量
新颖性(用户没听说的但是又喜欢的)
信任度（提高用户和推荐系统的交互：做推荐的解释）
推荐系统的健壮性（防作弊的能力：数据清洗，使用代价用户比较高的用户行为）
实时性（实时的更新推荐列表满足用户的新的行为：通过推荐列表的变化速率来评测）
商业目的（计算广告：转化率什么的）
根据不同的评测维度来全面的理解推荐系统的性能
（5）推荐系统的三大组成部分
前端的展示页面：和用户交互的，产生显示的用户行为的反馈数据（喜欢，评分等）和隐式的用户的行为反馈数据（浏览数据，一般数据量比较大）

后台的日志系统：用户的行为日志
推荐算法系统：推荐引擎的算法模型
<1>:基于内容的推荐
<2>:基于协同过滤的推荐：
基于领域的方法：
（基于用户的协同过滤(相似用户喜欢的物品：余炫相似度和改进的余炫相似度(用户大的时候非常的耗时，改进的是对两个用户对冷门的物品都采取过行为，增加对兴趣列表用热门的商品的惩罚))；
缺点：用户的大需要计算很大的用户相似度矩阵，无法提供推荐解释
基于物品的协同过滤（和它之前喜欢的物品相似的物品：喜欢 A 的用户大部分都喜欢 B---维护一张物品相似度的表）：亚马逊提出的

基于隐语义模型(LFM)：通过隐含的特征联系用户额物品,具体的一些实现的细节我忘记了
基于图的随机游走算法：将用户的行为数据表示为图：二元组（U,i）,计算图中 2 点之间相关性的方法：基于图的随机游走算法，具体的细节我忘记了
<3>:
（6）推荐系统的冷启动的问题
用户的冷启动的问题：利用用户的注册的信息（人口统计信息，用户的兴趣的描述，提供一些物品供用户反馈：如何选择这些物品：使用决策树来解决）
物品的冷启动的问题：利用内容信息，给他们推荐喜欢过相似物品的用户喜欢的物品
系统的冷启动的问题：发挥专家作用，比如一个音乐基因的项目
（7）推荐系统的架构：
推荐系统 lambda 的架构：
线上：实时的根据用户的行为数据+模型==》计算推荐的 topN
线下 ：线下为推荐模型的训练将模型保存在 redis，solr，ES 中
1：离线存储到 hadoop 体系：hive HQL（数据的清洗，构建成训练集）---训练模型---存储模型---在线计算推荐结果列表
2：储存到关系性数据库：推荐系统在公司中一般作为一个服务使用，其他的系统也能够调用
采用的推荐算法是：
逻辑回归-------（线性(分界线--有无 w0，线性会遇到线性不可分的问题：升维：一般是将已知的维度两两相乘，将其映射到高维的空间，那么就能找到一个平面将其分开，计算复杂度变大了)有监督（有 y 值）的分类模型（阈值的分析：默认 0.5，调正阈值可以降低风险）：逻辑回归的函数，singmod 函数）
训练集和测试集
训练的方式
优化：

W 越小，模型的抗干扰能力越强。求得的 Z 对逻辑回归的 sigmoid 函数影响就越大，抗干扰的能力就越小，小小的变化就会影响到最后的分类结果。

加上正则提高你模型的推广能力-----设置 lambda 系数，一般情况从经验来看，会把 lambda 设置为 0.4

L1 正则和 L2 正则：L1 正则更倾向于取 1 和取 0，L2 正则更加的倾向于使得 w 整体偏小

SGD 和 LBFGS 优化，提高计算的速度

数值优化-----不会影响正确率，只会提升求解模型的速度

方差归一化，会考虑到一组数里面的所有数据，就是每个数去除以方差

带来的好处就是会使得各个 W 基本数量级一致

缺点未必会落到 0 到 1 之间

均值归一化，让找到最优的速度变快，让各个维度的数据有正有负，就会使得各个 W 在调整的时候有的变大有的变小

归一化的目的：归一化的目的是消除 x 之间的过大差异，从而消除 W 之间的过大差异，从而使得在训练（梯度下降法）的时候，使得 W 之间的变化是同步，均匀的，从而使得你求解机器学习模型的速度加快

推荐系统的一般的步骤：
需求的分析（架构的推荐方案）
数据的清洗（得到训练的数据）
算法的建模（得到模型的结果）
模型的使用（得到推荐的结果）
结果的评估（推荐结果的评估）

数据量？多少服务器？集群的规模？估算数据量，多少台服务器，多少磁盘？
日均 pv=46000，按照一条日志记录的等于 1-10k 左右，加上其他的一些数据，大概一天 500M 左右的数据，一天的数据量在 500M-1G，一个月的数据量在 15G-30G，一个月的数据在 1T 左右的数据。
当时是用测试集群测试跑的，大概是 10 台的样子。真正项目上线的时候的是其他的小组负责的，具体的我不清楚。
数据量：使用的是 4 个月左右的数据吧，大概是 4T 左右的数据，毕竟是一个区域性的网站，集群的规模我不知道，这个是运维人员管理的，我们都是在测试集群上面测试。测试集群的机器我也记不清具体的配置，只是跑应用程序的时候，使用多少核，多少内存。
项目相关的各种事情，以及我的职责，具体做的事情？项目的架构的问题？
项目的描述？推荐的必要性？我的职责？具体做的事情？大数据的项目的架构+推荐系统项目的架构+推荐系统的三大部分的详解？+推荐系统的核心：推荐引擎，推荐算法？推荐系统的流程？推荐系统的评估（分类器的评估问题）？+推荐系统的调优？
Spark 怎么生成正负例样本表?
spark 的数据分析的程序生成的 hive 表(打标签)--正负例样本表
解决数据量过大，导致计算时间过长的问题？
优化方法：最大似然估计；代码优化；数据处理优化；用户画像
逻辑回归的优化？
逻辑回归的本质及计算优化和数值优化：
http://note.youdao.com/noteshare?id=89242ec92cb6b30f27e132dd844b46a9&sub=6C86DBBA2CA24BE5AD195CABFEB9869B
需求分析：
日均的 pv：3w-5w

日均的 ip：5000-1w
算法模型：Hadoop
开发相关：
团队的人数：8 人
负责的模块-流量的统计，KPI 分析
每日的数据：20-30G 左右
项目开发的时间：4 个月
服务器的数量：10 台左右的测试集，具体的是运维人员管理，不清楚
技术架构：hadoop+flume+hive+hbase+mysql
负责流量的统计这个模块，提取 Kpi
实现的流程：
1：日志是由业务系统产生的，设置 web 服务器每天产生一个新的目录，目录下面会产生多个日志文件，每个日志文件设置的是 64M
2：设置系统的定时器 CRON：计划任务，是任务在约定的时间执行已经计划好的工作
在夜间 0 点以后，利用 Flume 向 HDFS 导入昨天的日志文件
3：完成导入后，设置系统的定时器，启动 mapreduce 程序，提取并计算统计的指标
4：完成计算后，设置系统定时器，利用 sqoop 从 HDFS 将统计指标数据导入到 mysql 数据库，方便以后的即时查询
KPI 的相关计算：
PV：页面的访问统计：pv 去重后求和
map 过程（key：$request,value:1）和 reduce 过程（key:$request,value:sum()）
IP:页面的独立 IP 的访问统计：uv 去重后求和
map(key ： $request,value:remote_addr （客户端的 IP 地址）)和 reduce 过程(key:$request,value:去重后求：sum(unique))
Time:用户每一个小时 pV 统计：key 有时间地址连接然后求和
Source：用户来源域名的统计:域名解析工具，然后按照域名为 key 求和
当时刚刚学大数据，也不是很懂，就是跟着打打酱油，很多都是别人帮忙一起解决的
短视频推荐：
PGC(专业生产内容)型)和非直接生产内容(*UGC*(用户生产内容)型)两种
主框架还是候选生成+排序模型的模式
《Youtube 短视频推荐系统变迁：从机器学习到深度学习》：
https://cloud.tencent.com/community/article/989677
新媒体环境下的短视频推荐模型及算法研究
http://media.people.com.cn/n1/2016/0316/c402797-28203806.html
协同过滤[2]，基于内容[3]，以及融合前两者的混合模型
Spark 比 MR 快的原因：
答案：
（1）：Spark 基于内存迭代，MR 基于磁盘迭代
MR 将中间的数据结果保存在文件，提高可靠性，牺牲性能
Spark 的数据在内存进行交换，可以将数据缓存在内存，减少数据的加载耗时
（2）DAG 计算模型的 DAGScheduler 相当于一个改进版的 MR，在大多数的情况下可以减少 shuffle 的次数。Stage 里面的 pipline 的计算模式可以流式的计算
（3）Spark 的资源的调度是粗粒度的，MR 的资源的调度是细粒度的。
Spark 提交任务的一些进程：

Client 提交的 Driver 的进程的名字：sparkSubmit
Cluster 提交 Driver 进程的名字：DriverWrapper