about 云

大数据面试宝典

2015 年 02 月 27 日

序言：

文章内容来自 about 云大数据面试题板块经典内容，包括搜集的网络面试题，这里对它们进行了整理。其中包括技术题，面试经验分享，及一些资源。面试题及答案会随时更新。也欢迎一起交流学习。

阿里四轮面试总结

第一轮面试电话（5 月 6 号）：

1.自我介绍，包括做过项目。

2.有看过哪些 JDK 源码，了解哪些常用库。

3.集合框架 HashMap 的扩容机制，ConcurrnetHashMap 的原理

4.jvm 内存模型与 gc 内存回收机制

5.classloader 结构，是否可以自己定义一个 java.lang.String 类，为什么？ 双亲代理机制。

6.了解哪些设计模式，6 个设计原则分别是什么？每种设计原则体现的设计模式是哪个？

7.关于设计模式看了哪些书？书名是什么？

8.uml 模型图画过哪些？ 类图中类之间的关系有哪些，区别分别是什么？

9.画 uml 中类图时候用过一种虚线么？做什么用的？

10.做过应用相关性能测试的，举个例子，实际项目中怎么使用的。

用过并发框架相关的哪些内容

11.了解哪些 osgi 的框架？

12 有没有做过 jvm 内存调优，如何做的，举例子，用过哪些工具？

//一些不记得了

第二轮面试视频

1.自我介绍

2.看过哪些源码

3.java 的 io 库的类结构图所用到的设计模式如何体现

4.画出自己设计过的设计模式如何体现，画出结构图，并进行讲解。

5.画出自己做的架构的项目架构图 如何扩展等

6.数据库设计中主键 id 设计的原则

7.jvm 内存调优用过哪些工具，jstate 做什么用的？如何 dump 出当前线程状态？

8.并发框架是否有了解

9.classloader 的双亲代理机制

10.应用服务器的 jvm 调优实际经验，如何做的，在哪里用到的

11.在哪里获取最新资讯，逛什么论坛。最新的 Swift 语言有什么看法

12.设计原则与设计模式对应

13.servlet/filter 作用原理配置

14.ibatis in 操作 以及一个属性的作用

15.spring aop 用了什么设计原则，自动注入配置是做什么用的

16.jboss 的类加载器

17.session 共享机制

18.做过最成功的一件事情是什么？

19.最大的争执是什么？

20.为什么想要离职去阿里

//问题很多，一些不记得了

第三轮面试电话：

1.现在公司负责什么？

2.项目主要目的是做什么的？

3.公司管理方式、项目问题反馈机制是什么？

4.Java 的序列化做什么用的 序列化 id 会出现哪些问题？

5.OSGi 用过哪些？类加载器结构如何，如何在一个 bundle 中加载另外一个 bundle 中的一个类？

6.nio 是否了解 阻塞之后通知机制是怎样的？

7.uml 设计类图如何画，类之间关系以及区别

8.spring 如何不许要配置文件加载 bean 定义，可能是问自动注解或者是 properties 文件定义 bean

9.ibatis 等框架是不是都是实际在使用的,技术细节

10.为什么想离职去阿里

//一些不记得了

第四轮总监面电话面试：

1.自我介绍

2.公司做什么，业务， 负责内容，汇报机制等

3.企业级应用安全相关

4.http 协议，返回码，301 与 302 区别

5.多线程并发用过哪些

6.应用服务器相关，谈最熟悉的

7.为什么离职

//这个太多不记得了，很多不太会。

优酷 hadoop，mapred 面试题及答案

mapred 找共同朋友，数据格式如下

```
1.  A B C D E F
2.  B A C D E
3.  C A B E
4.  D A B E
5.  E A B C D
6.  F A
```

第一字母表示本人，其他是他的朋友，找出有共同朋友的人，和共同朋友是谁

答案如下

```
1.  import java.io.IOException;
2.  import java.util.Set;
3.  import java.util.StringTokenizer;
4.  import java.util.TreeSet;
5.
6.  import org.apache.hadoop.conf.Configuration;
7.  import org.apache.hadoop.fs.Path;
8.  import org.apache.hadoop.io.Text;
9.  import org.apache.hadoop.mapreduce.Job;
10. import org.apache.hadoop.mapreduce.Mapper;
11. import org.apache.hadoop.mapreduce.Reducer;
12. import org.apache.hadoop.mapreduce.Mapper.Context;
13. import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
14. import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
15. import org.apache.hadoop.util.GenericOptionsParser;
16.
17. public class FindFriend {
```

```
18.
19.           public static class ChangeMapper extends Mapper<Object, Text, Text,
    Text>{
20.                   @Override
21.                   public void map(Object key, Text value, Context context) throws
    IOException, InterruptedException {
22.                             StringTokenizer itr = new StringTokenizer(value.toString());
23.                                 Text owner = new Text();
24.                                 Set<String> set = new TreeSet<String>();
25.                             owner.set(itr.nextToken());
26.                             while (itr.hasMoreTokens()) {
27.                                     set.add(itr.nextToken());
28.                             }
29.                             String[] friends = new String[set.size()];
30.                             friends = set.toArray(friends);
31.
32.                             for(int i=0;i<friends.length;i++){
33.                                     for(int j=i+1;j<friends.length;j++){
34.                                             String outputkey = friends[i]+friends[j];
35.                                             context.write(new Text(outputkey),owner);
36.                                     }
37.                             }
38.                   }
39.           }
40.
41.           public static class FindReducer extends Reducer<Text,Text,Text,Text>
    {
42.                         public void reduce(Text key, Iterable<Text> values,
43.                                         Context context) throws IOException,
    InterruptedException {
44.                                  String  commonfriends ="";
```

```
45.                             for (Text val : values) {
46.                                 if(commonfriends == ""){
47.                                         commonfriends = val.toString();
48.                                 }else{
49.                                         commonfriends =
    commonfriends+":"+val.toString();
50.                                 }
51.                              }
52.                             context.write(key, new
    Text(commonfriends));
53.                         }
54.           }
55.
56.
57.         public static void main(String[] args) throws IOException,
58.         InterruptedException, ClassNotFoundException {
59.
60.             Configuration conf = new Configuration();
61.             String[] otherArgs = new GenericOptionsParser(conf, args).getRemainingArgs();
62.             if (otherArgs.length < 2) {
63.               System.err.println("args error");
64.               System.exit(2);
65.             }
66.             Job job = new Job(conf, "word count");
67.             job.setJarByClass(FindFriend.class);
68.             job.setMapperClass(ChangeMapper.class);
69.             job.setCombinerClass(FindReducer.class);
70.             job.setReducerClass(FindReducer.class);
71.             job.setOutputKeyClass(Text.class);
72.             job.setOutputValueClass(Text.class);
73.             for (int i = 0; i < otherArgs.length - 1; ++i) {
```

```
74.               FileInputFormat.addInputPath(job, new Path(otherArgs[i]));
75.             }
76.             FileOutputFormat.setOutputPath(job,
77.               new Path(otherArgs[otherArgs.length - 1]));
78.             System.exit(job.waitForCompletion(true) ? 0 : 1);
79.
80.         }
81.
82. }
```

```
1.  AB      E:C:D
2.  AC      E:B
3.  AD      B:E
4.  AE      C:B:D
5.  BC      A:E
6.  BD      A:E
7.  BE      C:D:A
8.  BF      A
9.  CD      E:A:B
10. CE      A:B
11. CF      A
12. DE      B:A
13. DF      A
14. EF      A
15.
```

hadoop 面试题，测试一下你的实力

1 使用 Hive 或者自定义 MR 实现如下逻辑

product_no	lac_id	moment	start_time	user_id	county_id	staytime	city_id
13429100031	22554	8	2013-03-11 08:55:19.151754088	571	571	282	571
13429100082	22540	8	2013-03-11 08:58:20.152622488	571	571	270	571
13429100082	22691	8	2013-03-11 08:56:37.149593624	571	571	103	571
13429100087	22705	8	2013-03-11 08:56:51.139539816	571	571	220	571
13429100087	22540	8	2013-03-11 08:55:45.150276800	571	571	66	571
13429100082	22540	8	2013-03-11 08:55:38.140225200	571	571	133	571
13429100140	26642	9	2013-03-11 09:02:19.151754088	571	571	18	571
13429100082	22691	8	2013-03-11 08:57:32.151754088	571	571	287	571
13429100189	22558	8	2013-03-11 08:56:24.139539816	571	571	48	571
13429100349	22503	8	2013-03-11 08:54:30.152622440	571	571	211	571

字段解释：

product_no：用户手机号；

lac_id：用户所在基站；

start_time：用户在此基站的开始时间；

staytime：用户在此基站的逗留时间。

需求描述：

根据 lac_id 和 start_time 知道用户当时的位置，根据 staytime 知道用户各个基站的逗留时长。根据轨迹合并连续基站的 staytime。

最终得到每一个用户按时间排序在每一个基站驻留时长

期望输出举例：

13429100082	22540	8	2013-03-11 08:58:20.152622488	571	571	270	571
13429100082	22691	8	2013-03-11 08:56:37.149593624	571	571	390	571
13429100082	22540	8	2013-03-11 08:55:38.140225200	571	571	133	571
13429100087	22705	8	2013-03-11 08:56:51.139539816	571	571	220	571
13429100087	22540	8	2013-03-11 08:55:45.150276800	571	571	66	571

2 Linux 脚本能力考察

2.1 请随意使用各种类型的脚本语言实现：批量将指定目录下的所有文件中的$HADOOP_HOME$替换成/home/ocetl/app/hadoop

2.2 假设有 10 台主机，H1 到 H10，在开启 SSH 互信的情况下，编写一个或多个脚本实现在所有的远程主机上执行脚本的功能

例如：runRemoteCmd.sh "ls -l"

期望结果：

H1:

XXXXXXXX

XXXXXXXX

XXXXXXXX

H2:

XXXXXXXX

XXXXXXXX

XXXXXXXX

H3:

...

3 Hadoop 基础知识与问题分析的能力

3.1 描述一下 hadoop 中，有哪些地方使用了缓存机制，作用分别是什么

3.2 请描述 https://issues.apache.org/jira/browse/HDFS-2379 说的是什么问题，最终解决的思路是什么？

4 MapReduce 开发能力

请参照 wordcount 实现一个自己的 map reduce，需求为：

a 输入文件格式：

xxx,xxx,xxx,xxx,xxx,xxx,xxx

b 输出文件格式：

xxx,20

xxx,30

xxx.40

c 功能：根据命令行参数统计输入文件中指定关键字出现的次数，并展示出来

例如：hadoop jar xxxxx.jar keywordcount xxx,xxx,xxx,xxx(四个关键字）

5 MapReduce 优化

请根据第五题中的程序，提出如何优化 MR 程序运行速度的思路

6 Linux 操作系统知识考察

请列举曾经修改过的/etc 下的配置文件，并说明修改要解决的问题？

7 Java 开发能力

7.1 写代码实现 1G 大小的文本文件，行分隔符为\x01\x02,统计一下该文件中的总行数，要求注意边界情况的处理

7.2 请描述一下在开发中如何对上面的程序进行性能分析，对性能进行优化的过程

回帖中答案可参考

可以参考另外一套面试题

另一套面试题答案

面试过程中经常被问道的问题记录

1、如何实现 hadoop 的安全机制。

2、在使用 hadoop 或者是 spark 中遇到过哪些问题，是如何处理解决的。

3、hadoop 的调度策略的实现，你们使用的是那种策略，为什么。

4、hadoop 和 spark 使用场景。

5、hdfs 和 hbase 各自使用场景。

6、hbase 的 rowkey 设计，影响 hbase 的性能有哪些。

7、storm 中如何实现统计 uv 的不重复。

8、redis 分布式实现原理。如何实现读写分离，在这个过程当中使用了哪些算法，有什么好处。

9、spark 如何保证宕机迅速恢复。

10、hadoop 中两个大表实现 join 的操作，简单描述。

下面答案并不是标准答案，仅供参考，由 about 云会员 **xpy888** 提供，非常感谢

1、如何实现 hadoop 的安全机制。

1.1 共享 hadoop 集群:

a: 管理人员把开发人员分成了若干个队列，每个队列有一定的资源，每个用户及用户组只能使用某个队列中指定资源。

b: HDFS 上有各种数据，公用的，私有的，加密的。不用的用户可以访问不同的数据。

1.2 HDFS 安全机制

client 获取 namenode 的初始访问认证(使用 kerberos)后，会获取一个 delegation token，这个 token 可以作为接下来访问 HDFS 或提交作业的认证。同样，读取 block 也是一样的。

1.3 mapreduce 安全机制

所有关于作业的提交或者作业运行状态的追踪均是采用带有 Kerberos 认证的 RPC 实现的。授权用户提交作业时，JobTracker 会为之生成一个 delegation token，该 token 将被作为 job 的一部分存储到 HDFS

上并通过 RPC 分发给各个 TaskTracker，一旦 job 运行结束，该 token 失效。

1.4 DistributedCache 是安全的。

DistribuedCache 分别两种，一种是 shared，可以被所有作业共享，而 private 的只能被该用户的作业共享。

1.5 RPC 安全机制

在 Hadoop RP 中添加了权限认证授权机制。当用户调用 RPC 时，用户的 login name 会通过 RPC 头部传递给 RPC，之后 RPC 使用 Simple Authentication and Security Layer（SASL）确定一个权限协议（支持 Kerberos 和 DIGEST-MD5 两种），完成 RPC 授权。

2、在使用 hadoop 或者是 spark 中遇到过哪些问题，是如何处理解决的。

2.1 数据倾斜。

出现这种情况：多数是由于代码的质量写的不够健壮。查看日志：发现问题。

2.2 spark-出现 OOM

小数据量的情况可以 cache，数据量大的情况必须考虑内存使用。

3、hadoop 的调度策略的实现，你们使用的是那种策略，为什么。

3.1 默认情况下 hadoop 使用的 FIFO, 先进先出的调度策略。按照作业的优先级来处理。

3.2 计算能力调度器(Capacity Scheduler) 支持多个队列，每个队列可配置一定的资源量，每个队列采用 FIFO, 为了防止同一个用户的作业独占资源，那么调度器会对同一个用户提交的作业所占资源进行限定，首先按以下策略选择一个合适队列：计算每个队列中正在运行的任务数与其应该分得的计算资源之间的比值，选择一个该比值最小的队列；然后按以下策略选择该队列中一个作业：按照作业优先级和提交时间顺序选择，同时考虑用户资源量限制和内存限制。

3.3 公平调度器(Fair Scheduler) 支持多队列多用户，每个队列中的资源量可以配置，同一队列中的作业公平共享队列中所有资源。

3.4 异构集群的调度器 LATE

3.5 实时作业的调度器 Deadline Scheduler 和 Constraint-based Scheduler

4、hadoop 和 spark 使用场景。

Hadoop/MapReduce 和 Spark 最适合的都是做离线型的数据分析，但 Hadoop 特别适合是单次分析的数据量“很大”的情景，而 Spark 则适用于数据量不是很大的情景。

4.1 一般情况下，对于中小互联网和企业级的大数据应用而言，单次分析的数量都不会“很大”，因此可以优先考虑使用 Spark。

4.2 业务通常认为 Spark 更适用于机器学习之类的“迭代式”应用,80GB 的压缩数据（解压后超过 200GB），10 个节点的集群规模，跑类似“sum+group-by”的应用，MapReduce 花了 5 分钟，而 spark 只需要 2 分钟。

5、hdfs 和 hbase 各自使用场景。

整理总结：

首先一点需要明白：Hbase 是基于 HDFS 来存储的。

HDFS：

1、一次性写入，多次读取。

2、保证数据的一致性。

3、主要是可以部署在许多廉价机器中，通过多副本提高可靠性，提供了容错和恢复机制。

Hbase：

1、瞬间写入量很大，数据库不好支撑或需要很高成本支撑的场景。

2、数据需要长久保存，且量会持久增长到比较大的场景

3、hbase 不适用与有 join，多级索引，表关系复杂的数据模型

4、大数据量 （100s TB 级数据） 且有快速随机访问的需求。

如：淘宝的交易历史记录。数据量巨大无容置疑，面向普通用户的请求必然要即时响应。

5、容量的优雅扩展

大数据的驱使，动态扩展系统容量的必须的。例如：webPage DB。

6、业务场景简单，不需要关系数据库中很多特性（例如交叉列、交叉表，事务，连接等等）

7、优化方面：合理设计 rowkey。因为 hbase 的查

6、hbase 的 rowkey 设计，影响 hbase 的性能有哪些。

由于使用 hbase 不多，hbase 的 rowkey 的设计就不多说了，哪位大神使用过，做一下补充。

1 hbase.hregion.max.filesize 应该设置多少合适。

默认是 256，HStoreFile 的最大值。如果任何一个 Column Family（或者说 HStore）的 HStoreFiles 的大小超过这个值，那么，其所属的 HRegion 就会 Split 成两个。

众所周知 hbase 中数据一开始会写入 memstore，当 memstore 满 64MB 以后，会 flush 到 disk 上而成为 storefile。当 storefile 数量超过 3 时，会启动 compaction 过程将它们合并为一个 storefile。这个过程中会删除一些 timestamp 过期的数据，比如 update 的数据。而当合并后的 storefile 大小大于 hfile 默认最大值时，会触发 split 动作，将它切分成两个 region。

2、autoflush=false 的影响

无论是官方还是很多 blog 都提倡为了提高 hbase 的写入速度而在应用代码中设置 autoflush=false，然后 lz 认为在在线应用中应该谨慎进行该设置　　原因如下：

2.1、autoflush=false 的原理是当客户端提交 delete 或 put 请求时，将该请求在客户端缓存，直到数据超过 2M(hbase.client.write.buffer 决定)或用户执行了 hbase.flushcommits()时才向 regionserver 提交请求。因此即使 htable.put()执行返回成功，也并非说明请求真的成功了。假如还没有达到该缓存而 client 崩溃，该部分数据将由于未发送到 regionserver 而丢失。这对于零容忍的在线服务是不可接受的。

2.2、autoflush=true 虽然会让写入速度下降 2-3 倍，但是对于很多在线应用来说这都是必须打开的，也正是 hbase 为什么让它默认值为 true 的原因。当该值为 true 时，每次请求都会发往 regionserver,而 regionserver 接收到请求后第一件事就是写 hlog，因此对 io 的要求是非常高的，为了提高 hbase 的写入速度，应该尽可能高地提高 io 吞吐量，比如增加磁盘、使用 raid 卡、减少 replication 因子数等 。

从性能的角度谈 table 中 family 和 qualifier 的设置

3、对于传统关系型数据库中的一张 table，在业务转换到 hbase 上建模时，从性能的角度应该如何设置 family 和 qualifier 呢？

最极端的，①每一列都设置成一个 family，②一个表仅有一个 family，所有列都是其中的一个 qualifier，那么有什么区别呢？

从读的方面考虑：

family 越多，那么获取每一个 cell 数据的优势越明显，因为 io 和网络都减少了。

如果只有一个 family，那么每一次读都会读取当前 rowkey 的所有数据，网络和 io 上会有一些损失。

当然如果要获取的是固定的几列数据，那么把这几列写到一个 family 中比分别设置 family 要更好，因为只需一次请求就能拿回所有数据。

从写的角度考虑：

首先，内存方面来说，对于一个 Region，会为每一个表的每一个 Family 分配一个 Store，而每一个 Store，

都会分配一个 MemStore，所以更多的 family 会消耗更多的内存。

其次，从 flush 和 compaction 方面说，目前版本的 hbase，在 flush 和 compaction 都是以 region 为单位的，也就是说当一个 family 达到 flush 条件时，该 region 的所有 family 所属的 memstore 都会 flush 一次，即使 memstore 中只有很少的数据也会触发 flush 而生成小文件。这样就增加了 compaction 发生的机率，而 compaction 也是以 region 为单位的，这样就很容易发生 compaction 风暴从而降低系统的整体吞吐量。

第三，从 split 方面考虑，由于 hfile 是以 family 为单位的，因此对于多个 family 来说，数据被分散到了更多的 hfile 中，减小了 split 发生的机率。这是把双刃剑。更少的 split 会导致该 region 的体积比较大，由于 balance 是以 region 的数目而不是大小为单位来进行的，因此可能会导致 balance 失效。而从好的方面来说，更少的 split 会让系统提供更加稳定的在线服务。而坏处我们可以通过在请求的低谷时间进行人工的 split 和 balance 来避免掉。

因此对于写比较多的系统，如果是离线应该，我们尽量只用一个 family 好了，但如果是在线应用，那还是应该根据应用的情况合理地分配 family

7、storm 中如何实现统计 uv 的不重复。

storm 主要是通过 Transactional topology，确保每次 tuple 只被处理一次。给每个 tuple 按顺序加一个 id，在处理过程中，将成功处理的 tuple id 和计算保存在数据库当中，但是这种机制使得系统一次只能处理一个 tuple，无法实现分布式计算。

我们要保证一个 batch 只被处理一次，机制和上一节类似。只不过数据库中存储的是 batch id。batch 的中间计算结果先存在局部变量中，当一个 batch 中的所有 tuple 都被处理完之后，判断 batch id，如果跟数据库中的 id 不同，则将中间计算结果更新到数据库中。

Storm 提供的 Transactional Topology 将 batch 计算分为 process 和 commit 两个阶段。Process 阶段可以同时处理多个 batch，不用保证顺序性；commit 阶段保证 batch 的强顺序性，并且一次只能处理一个 batch，第 1 个 batch 成功提交之前，第 2 个 batch 不能被提交。这样就保证多线程情况下值能处理一个 batch。

8、redis 分布式实现原理。如何实现读写分离，在这个过程当中使用了哪些算法，有什么好处。

memcache 只能说是简单的 kv 内存数据结构,而 redis 支持的数据类型比较丰富。Redis 在 3.0 以后实现集群机制。目前 Redis 实现集群的方法主要是采用一致性哈稀分片（Shard），将不同的 key 分配到不同的 redis server 上，达到横向扩展的目的。

使用了一致性哈稀进行分片，那么不同的 key 分布到不同的 Redis-Server 上，当我们需要扩容时，需要增加机器到分片列表中，这时候会使得同样的 key 算出来落到跟原来不同的机器上，这样如果要取某一个值，会出现取不到的情况，对于这种情况，Redis 的提出了一种名为 Pre-Sharding 的方式：

使用了 redis 的集群模式：存在以下几个问题

A：扩容问题：

Pre-Sharding 方法是将每一个台物理机上，运行多个不同断口的 Redis 实例，假如有三个物理机，每个物理机运行三个 Redis 实际，那么我们的分片列表中实际有 9 个 Redis 实例，当我们需要扩容时，增加一台物理机，步骤如下：

1、在新的物理机上运行 Redis-Server；

2、该 Redis-Server 从属于(slaveof)分片列表中的某一 Redis-Server（假设叫 RedisA）；

3、等主从复制(Replication)完成后，将客户端分片列表中 RedisA 的 IP 和端口改为新物理机上 Redis-Server 的 IP 和端口；

4、停止 RedisA.

B: 单点故障问题

将一个 Redis-Server 转移到了另外一台上。Prd-Sharding 实际上是一种在线扩容的办法，但还是很依赖 Redis 本身的复制功能的，如果主库快照数据文件过大，这个复制的过程也会很久，同时会给主库带来压力。

9、spark 如何保证宕机迅速恢复。

了解了 RDD，那么这个问题就迎刃而解了
RDD 的好处
RDD 只能从持久存储或通过 Transformations 操作产生，相比于分布式共享内存（DSM）可以更高效实现容错，对于丢失部分数据分区只需根据它的 lineage 就可重新计算出来，而不需要做特定的 Checkpoint。
RDD 的不变性，可以实现类 Hadoop MapReduce 的推测式执行。
RDD 的数据分区特性，可以通过数据的本地性来提高性能，这与 Hadoop MapReduce 是一样的。
RDD 都是可序列化的，在内存不足时可自动降级为磁盘存储，把 RDD 存储于磁盘上，这时性能会有大的下降但不会差于现在的 MapReduce。
RDD 的存储与分区
用户可以选择不同的存储级别存储 RDD 以便重用。
当前 RDD 默认是存储于内存，但当内存不足时，RDD 会 spill 到 disk。
RDD 在需要进行分区把数据分布于集群中时会根据每条记录 Key 进行分区（如 Hash 分区），以此保证两个数据集在 Join 时能高效。
RDD 的内部表示
在 RDD 的内部实现中每个 RDD 都可以使用 5 个方面的特性来表示：
分区列表（数据块列表）
计算每个分片的函数（根据父 RDD 计算出此 RDD）
对父 RDD 的依赖列表
对 key-value RDD 的 Partitioner【可选】
每个数据分片的预定义地址列表(如 HDFS 上的数据块的地址)【可选】
10、hadoop 中两个大表实现 join 的操作，简单描述。

这个简单了。

10.1 map 读取两个表，分别标示一下，然后 context 输出。reduce 做笛卡尔积。

10.2 map 之前 setup 时先 DistributeCache，然后 map 阶段扫描大表。

机器学习、大数据面试问题及答题思路

自己的专业方向是机器学习、数据挖掘，就业意向是互联网行业与本专业相关的工作岗位。各个企业对这类岗位的命名可能有所不同，比如数据挖掘/自然语言处理/机器学习算法工程师，或简称算法工程师，还有的称为搜索/推荐算法工程师，甚至有的并入后台工程师的范畴，视岗位具体要求而定。

机器学习、大数据相关岗位的职责

自己参与面试的提供算法岗位的公司有 BAT、小米、360、飞维美地、宜信、猿题库 等，根据业务的不同，岗位职责大概分为：

1、平台搭建类

- 数据计算平台搭建，基础算法实现，当然，要求支持大样本量、高维度数据，所以可能还需要底层开发、并行计算、分布式计算等方面的知识；

2、算法研究类

- 文本挖掘，如领域知识图谱构建、垃圾短信过滤等；
- 推荐，广告推荐、APP 推荐、题目推荐、新闻推荐等；
- 排序，搜索结果排序、广告排序等；
- 广告投放效果分析；
- 互联网信用评价；
- 图像识别、理解。

3、数据挖掘类

- 商业智能，如统计报表；
- 用户体验分析，预测流失用户。

以上是根据本人求职季有限的接触所做的总结。有的应用方向比较成熟，业界有足够的技术积累，比如搜索、推荐，也有的方向还有很多开放性问题等待探索，比如互联网金融、互联网教育。在面试的过程中，一方面要尽力向企业展现自己的能力，另一方面也是在增进对行业发展现状与未来趋势的理解，特别是可以从一些刚起步的企业和团队那里，了解到一些有价值的一手问题。

以下首先介绍面试中遇到的一些真实问题，然后谈一谈答题和面试准备上的建议。

面试问题

1、你在研究/项目/实习经历中主要用过哪些机器学习/数据挖掘的算法？

2、你熟悉的机器学习/数据挖掘算法主要有哪些？

3、你用过哪些机器学习/数据挖掘工具或框架？

4、基础知识

- 无监督和有监督算法的区别？
- SVM 的推导，特性？多分类怎么处理？
- LR 的推导，特性？
- 决策树的特性？
- SVM、LR、决策树的对比？
- GBDT 和 决策森林 的区别？
- 如何判断函数凸或非凸？
- 解释对偶的概念。
- 如何进行特征选择？
- 为什么会产生过拟合，有哪些方法可以预防或克服过拟合？
- 介绍卷积神经网络，和 DBN 有什么区别？
- 采用 EM 算法求解的模型有哪些，为什么不用牛顿法或梯度下降法？
- 用 EM 算法推导解释 Kmeans。
- 用过哪些聚类算法，解释密度聚类算法。
- 聚类算法中的距离度量有哪些？
- 如何进行实体识别？
- 解释贝叶斯公式和朴素贝叶斯分类。
- 写一个 Hadoop 版本的 wordcount。
- ……

5、开放问题

- 给你公司内部群组的聊天记录，怎样区分出主管和员工？
- 如何评估网站内容的真实性（针对代刷、作弊类）？
- 深度学习在推荐系统上可能有怎样的发挥？
- 路段平均车速反映了路况，在道路上布控采集车辆速度，如何对路况做出合理估计？采集数据中的异常值如何处理？
- 如何根据语料计算两个词词义的相似度？
- 在百度贴吧里发布 APP 广告，问推荐策略？
- 如何判断自己实现的 LR、Kmeans 算法是否正确？
- 100 亿数字，怎么统计前 100 大的？
- ……

答题思路

1、用过什么算法？

- 最好是在项目/实习的大数据场景里用过，比如推荐里用过 CF、LR，分类里用过 SVM、GBDT；
- 一般用法是什么，是不是自己实现的，有什么比较知名的实现，使用过程中踩过哪些坑；
- 优缺点分析。

2、熟悉的算法有哪些？

- 基础算法要多说，其它算法要挑熟悉程度高的说，不光列举算法，也适当说说应用场合；
- 面试官和你的研究方向可能不匹配，不过在基础算法上你们还是有很多共同语言的，你说得太高大上可能效果并不好，一方面面试官还是要问基础的，另一方面一旦面试官突发奇想让你给他讲解高大上的内容，而你只是泛泛的了解，那就傻叉了。

3、用过哪些框架/算法包？

- 主流的分布式框架如 Hadoop，Spark，Graphlab，Parameter Server 等择一或多使用了解；
- 通用算法包，如 mahout，scikit，weka 等；
- 专用算法包，如 opencv，theano，torch7，ICTCLAS 等。

4、基础知识

- 个人感觉高频话题是 SVM、LR、决策树（决策森林）和聚类算法，要重点准备；
- 算法要从以下几个方面来掌握

产生背景，适用场合（数据规模，特征维度，是否有 Online 算法，离散/连续特征处理等角度）；

原理推导（最大间隔，软间隔，对偶）；

求解方法（随机梯度下降、拟牛顿法等优化算法）；

优缺点，相关改进；

和其他基本方法的对比；

- 不能停留在能看懂的程度，还要

对知识进行结构化整理，比如撰写自己的 cheet sheet，我觉得面试是在有限时间内向面试官输出自己知识的过程，如果仅仅是在面试现场才开始调动知识、组织表达，总还是不如系统的梳理准备；

从面试官的角度多问自己一些问题，通过查找资料总结出全面的解答，比如如何预防或克服过拟合。

5、开放问题

- 由于问题具有综合性和开放性，所以不仅仅考察对算法的了解，还需要足够的实战经验作基础；

- 先不要考虑完善性或可实现性，调动你的一切知识储备和经验储备去设计，有多少说多少，想到什么说什么，方案都是在你和面试官讨论的过程里逐步完善的，不过面试官有两种风格：引导你思考考虑不周之处 or 指责你没有考虑到某些情况，遇到后者的话还请注意灵活调整答题策略;
- 和同学朋友开展讨论，可以从上一节列出的问题开始。

准备建议

1、基础算法复习两条线

- 材料阅读 包括经典教材（比如 PRML，模式分类）、网上系列博客（比如 研究者 July），系统梳理基础算法知识；
- 面试反馈 面试过程中会让你发现自己的薄弱环节和知识盲区，把这些问题记录下来，在下一次面试前搞懂搞透。

2、除算法知识，还应适当掌握一些系统架构方面的知识，可以从网上分享的阿里、京东、新浪微博等的架构介绍 PPT 入手，也可以从 Hadoop、Spark 等的设计实现切入。

3、如果真的是以就业为导向就要在平时注意实战经验的积累，在科研项目、实习、比赛（Kaggle，Netflix，天猫大数据竞赛等）中摸清算法特性、熟悉相关工具与模块的使用。

总结

如今，好多机器学习、数据挖掘的知识都逐渐成为常识，要想在竞争中脱颖而出，就必须做到

- 保持学习热情，关心热点；
- 深入学习，会用，也要理解；
- 在实战中历练总结；
- 积极参加学术界、业界的讲座分享，向牛人学习，与他人讨论。

最后，希望自己的求职季经验总结能给大家带来有益的启发。

Hadoop 面试题,看看书找答案，看看你能答对多少（2）答案公布

Hadoop 面试题,看看书找答案，看看你能答对多少(2)

以下答案经过查阅资料与 about 云群（39327136）友，hadoop 爱好者朋友，讨论后，二次修改答案。

1. 下面哪个程序负责 HDFS 数据存储。
a)NameNode b)Jobtracker c)Datanode d)secondaryNameNode e)tasktracker

答案 C datanode

2. HDfS 中的 block 默认保存几份？
a)3 份 b)2 份 c)1 份 d)不确定

答案 A 默认 3 分

3. 下列哪个程序通常与 NameNode 在一个节点启动？
a)SecondaryNameNode b)DataNode c)TaskTracker d)Jobtracker

答案 D
分析：
hadoop 的集群是基于 master/slave 模式，namenode 和 jobtracker 属于 master，datanode 和 tasktracker 属于 slave，master 只有一个，而 slave 有多个

SecondaryNameNode 内存需求和 NameNode 在一个数量级上，所以通常 secondary NameNode（运行在单独的物理机器上）和 NameNode 运行在不同的机器上。
JobTracker 和 TaskTracker
JobTracker 对应于 NameNode
TaskTracker 对应于 DataNode
DataNode 和 NameNode 是针对数据存放来而言的
JobTracker 和 TaskTracker 是对于 MapReduce 执行而言的

mapreduce 中几个主要概念，mapreduce 整体上可以分为这么几条执行线索：
jobclient，JobTracker 与 TaskTracker。
1、JobClient 会在用户端通过 JobClient 类将应用已经配置参数打包成 jar 文件存储到 hdfs，

并把路径提交到 Jobtracker,然后由 JobTracker 创建每一个 Task（即 MapTask 和 ReduceTask）
并将它们分发到各个 TaskTracker 服务中去执行
2、JobTracker 是一个 master 服务，软件启动之后 JobTracker 接收 Job，负责调度 Job 的每一个子任务 task 运行于 TaskTracker 上，
并监控它们，如果发现有失败的 task 就重新运行它。一般情况应该把 JobTracker 部署在单独的机器上。
3、TaskTracker 是运行在多个节点上的 slaver 服务。TaskTracker 主动与 JobTracker 通信，接收作业，并负责直接执行每一个任务。
TaskTracker 都需要运行在 HDFS 的 DataNode 上

4. Hadoop 作者
a)Martin Fowler b)Kent Beck c)Doug cutting

答案 C Doug cutting

5. HDFS 默认 Block Size
a)32MB b)64MB c)128MB

答案：B
(因为版本更换较快，这里答案只供参考)

6. 下列哪项通常是集群的最主要瓶颈
a)CPU b)网络 c)磁盘 IO d)内存

答案：C 磁盘
首先集群的目的是为了节省成本，用廉价的 pc 机，取代小型机及大型机。小型机和大型机有什么特点？

1.cpu 处理能力强
2.内存够大
所以集群的瓶颈不可能是 a 和 d
3.网络是一种稀缺资源，但是并不是瓶颈。

4.由于大数据面临海量数据，读写数据都需要 io，然后还要冗余数据，hadoop 一般备 3 份数据，所以 IO 就会打折扣。
同样可以参考下面内容（磁盘 IO：磁盘输出输出）
对于磁盘 IO：当我们面临集群作战的时候，我们所希望的是即读即得。可是面对大数据，读取数据需要经过 IO，这里可以把 IO 理解为水的管道。管道越大越强，我们对于 T 级的数据读取就越快。所以 IO 的好坏，直接影响了集群对于数据的处理。
集群瓶颈：磁盘 IO 必读
集群瓶颈为什么磁盘 io

7. 关于 SecondaryNameNode 哪项是正确的？
a)它是 NameNode 的热备　　b)它对内存没有要求
c)它的目的是帮助 NameNode 合并编辑日志，减少 NameNode 启动时间
d)SecondaryNameNode 应与 NameNode 部署到一个节点

答案 C。
D 答案可以参考第三题

多选题：
8. 下列哪项可以作为集群的管理？
a)Puppet b)Pdsh c)Cloudera Manager d)Zookeeper

答案 1：ABD
具体可查看
什么是 Zookeeper，Zookeeper 的作用是什么，在 Hadoop 及 hbase 中具体作用是什么
二次整理
修改后答案：ABC
分析：
A：puppetpuppet 是一种 Linux、Unix、windows 平台的集中配置管理系统
B：pdsh 可以实现在在多台机器上执行相同的命令
详细参考：集群管理小工具介绍-pdsh
C：可以参考 Cloudera Manager 四大功能【翻译】
首先这里给管理下一个定义：部署、配置、调试、监控，属于管理
因为 zookeeper 不满足上面要求，所以不纳入管理范围。

9. 配置机架感知的下面哪项正确
a)如果一个机架出问题，不会影响数据读写
b)写入数据的时候会写到不同机架的 DataNode 中
c)MapReduce 会根据机架获取离自己比较近的网络数据

答案 ABC
具体可以参考
hadoop 机架感知--加强集群稳固性，该如何配置 hadoop 机架感知

10. Client 端上传文件的时候下列哪项正确
a)数据经过 NameNode 传递给 DataNode
b)Client 端将文件切分为 Block，依次上传

c)Client 只上传数据到一台 DataNode，然后由 NameNode 负责 Block 复制工作
答案 B
分析：
Client 向 NameNode 发起文件写入的请求。

NameNode 根据文件大小和文件块配置情况，返回给 Client 它所管理部分 DataNode 的信息。

Client 将文件划分为多个 Block，根据 DataNode 的地址信息，按顺序写入到每一个 DataNode 块中。
具体查看
HDFS 体系结构简介及优缺点

11. 下列哪个是 Hadoop 运行的模式
a)单机版 b)伪分布式 c)分布式

答案 ABC

12. Cloudera 提供哪几种安装 CDH 的方法
a)Cloudera manager b)Tarball c)Yum d)Rpm
答案：ABCD 具体可以参考
Hadoop CDH 四种安装方式总结及实例指导

判断题：
13. Ganglia 不仅可以进行监控，也可以进行告警。（ 正确）
分析：
此题的目的是考 Ganglia 的了解。严格意义上来讲是正确。

ganglia 作为一款最常用的 Linux 环境中的监控软件，它擅长的的是从节点中按照用户的需求以较低的代价采集数据。但是 ganglia 在预警以及发生事件后通知用户上并不擅长。最新的 ganglia 已经有了部分这方面的功能。但是更擅长做警告的还有 Nagios。Nagios，就是一款精于预警、通知的软件。通过将 Ganglia 和 Nagios 组合起来，把 Ganglia 采集的数据作为 Nagios 的数据源，然后利用 Nagios 来发送预警通知，可以完美的实现一整套监控管理的系统。
具体可以查看
完美集群监控组合 ganglia 和 nagios

14. Block Size 是不可以修改的。（错误 ）
它是可以被修改的

Hadoop 的基础配置文件是 hadoop-default.xml，默认建立一个 Job 的时候会建立 Job 的 Config，Config

首先读入hadoop-default.xml的配置，然后再读入hadoop-site.xml的配置（这个文件初始的时候配置为空），hadoop-site.xml中主要配置需要覆盖的hadoop-default.xml的系统级配置。具体配置可以参考下

```
1.  <property>
2.    <name>dfs.block.size</name>//block的大小，单位字节，后面会提到用处，必须是512的倍数，因为采用crc作文件完整性校验，默认配置512是checksum的最小单元。
3.    <value>5120000</value>
4.    <description>The default block size for new files.</description>
5.  </property>
```

复制代码

15. Nagios 不可以监控 Hadoop 集群，因为它不提供 Hadoop 支持。（错误 ）

分析：
Nagios是集群监控工具，而且是云计算三大利器之一

16. 如果 NameNode 意外终止，SecondaryNameNode 会接替它使集群继续工作。（错误 ）

分析：
SecondaryNameNode是帮助恢复，而不是替代，如何恢复，可以查看
hadoop 根据SecondaryNameNode恢复Namenode

17. Cloudera CDH 是需要付费使用的。（错误 ）

分析：
第一套付费产品是 Cloudera Enterpris，Cloudera Enterprise 在美国加州举行的 Hadoop 大会 (Hadoop Summit) 上公开，以若干私有管理、监控、运作工具加强 Hadoop 的功能。收费采取合约订购方式，价格随用的 Hadoop 叢集大小变动。

18. Hadoop 是 Java 开发的，所以 MapReduce 只支持 Java 语言编写。（错误 ）

分析：
rhadoop是用R语言开发的，MapReduce是一个框架，可以理解是一种思想，可以使用其他语言开发。
具体可以查看

Hadoop 简介(1):什么是 Map/Reduce

19. Hadoop 支持数据的随机读写。（错 ）

分析：
lucene 是支持随机读写的，而 hdfs 只支持随机读。但是 HBase 可以来补救。
HBase 提供随机读写，来解决 Hadoop 不能处理的问题。HBase 自底层设计开始即聚焦于各种可伸缩性问题：表可以很"高"，有数十亿个数据行；也可以很"宽"，有数百万个列；水平分区并在上千个普通商用机节点上自动复制。表的模式是物理存储的直接反映，使系统有可能提高高效的数据结构的序列化、存储和检索。

20. NameNode 负责管理 metadata，client 端每次读写请求，它都会从磁盘中读取或则会写入 metadata 信息并反馈 client 端。（错误）

修改后分析：
分析：
NameNode 不需要从磁盘读取 metadata，所有数据都在内存中，硬盘上的只是序列化的结果，只有每次 namenode 启动的时候才会读取。
1）文件写入
Client 向 NameNode 发起文件写入的请求。
NameNode 根据文件大小和文件块配置情况，返回给 Client 它所管理部分 DataNode 的信息。
Client 将文件划分为多个 Block，根据 DataNode 的地址信息，按顺序写入到每一个 DataNode 块中。

2）文件读取
Client 向 NameNode 发起文件读取的请求。
NameNode 返回文件存储的 DataNode 的信息。
Client 读取文件信息。

具体查看
hadoop 中 NameNode、DataNode 和 Client 三者之间协作关系

21. NameNode 本地磁盘保存了 Block 的位置信息。（ 个人认为正确，欢迎提出其它意见）

分析：
DataNode 是文件存储的基本单元，它将 Block 存储在本地文件系统中，保存了 Block 的 Meta-data，同时周期性地将所有存在的 Block 信息发送给 NameNode。

具体同样查看
hadoop 中 NameNode、DataNode 和 Client 三者之间协作关系

22. DataNode 通过长连接与 NameNode 保持通信。（ ）

这个有分歧：具体正在找这方面的有利资料。下面提供资料可参考。
首先明确一下概念：

（1）.长连接
Client 方与 Server 方先建立通讯连接，连接建立后不断开，
然后再进行报文发送和接收。这种方式下由于通讯连接一直
存在，此种方式常用于点对点通讯。

（2）.短连接
Client 方与 Server 每进行一次报文收发交易时才进行通讯连
接，交易完毕后立即断开连接。此种方式常用于一点对多点
通讯，比如多个 Client 连接一个 Server.

23. Hadoop 自身具有严格的权限管理和安全措施保障集群正常运行。（错误 ）

hadoop 只能阻止好人犯错，但是不能阻止坏人干坏事
具体可查看
hadoop 安全性需不断加强

24. Slave 节点要存储数据，所以它的磁盘越大越好。（ 错误）

分析：
一旦 Slave 节点宕机，数据恢复是一个难题

25. hadoop dfsadmin –report 命令用于检测 HDFS 损坏块。（错误 ）

分析：
hadoop dfsadmin -report

用这个命令可以快速定位出哪些节点 down 掉了，HDFS 的容量以及使用了多少，以及每个节点的硬盘使用
情况。

当然 NameNode 有个 http 页面也可以查询，但是这个命令的输出更适合我们的脚本监控 dfs 的使用状况

```
1.  Configured Capacity: 77209395855360 (70.22 TB)
2.  Present Capacity: 76079914600683 (69.19 TB)
3.  DFS Remaining: 60534707015680 (55.06 TB)
4.  DFS Used: 15545207585003 (14.14 TB)
5.  DFS Used%: 20.43%
6.
7.  -------------------------------------------------
8.  Datanodes available: 107 (109 total, 2 dead)
9.
10. Name: 172.16.218.232:50010
11. Rack: /lg/dminterface0
12. Decommission Status : Normal
13. Configured Capacity: 1259272216576 (1.15 TB)
14. DFS Used: 185585852416 (172.84 GB)
15. Non DFS Used: 39060951040 (36.38 GB)
16. DFS Remaining: 1034625413120(963.57 GB)
17. DFS Used%: 14.74%
18. DFS Remaining%: 82.16%
19. Last contact: Wed Nov 18 10:19:44 CST 2009
20.
21. Name: 172.16.216.126:50010
22. Rack: /lg/dminterface2
23. Decommission Status : Normal
24. Configured Capacity: 661261402112 (615.85 GB)
25. DFS Used: 123147280384 (114.69 GB)
26. Non DFS Used: 8803852288 (8.2 GB)
27. DFS Remaining: 529310269440(492.96 GB)
28. DFS Used%: 18.62%
29. DFS Remaining%: 80.05%
```

```
30. Last contact: Wed Nov 18 10:19:46 CST 2009
```

复制代码

26. Hadoop 默认调度器策略为 FIFO（正确 ）

具体参考

Hadoop 集群三种作业调度算法介绍

27. 集群内每个节点都应该配 RAID，这样避免单磁盘损坏，影响整个节点运行。（错误 ）

分析：

首先明白什么是 RAID，可以参考百科磁盘阵列。

这句话错误的地方在于太绝对，具体情况具体分析。题目不是重点，知识才是最重要的。

因为 hadoop 本身就具有冗余能力，所以如果不是很严格不需要都配备 RAID。具体参考第二题。

28. 因为 HDFS 有多个副本，所以 NameNode 是不存在单点问题的。（错误 ）

分析：

NameNode 存在单点问题。了解详细信息，可以参考

Hadoop 中 Namenode 单点故障的解决方案及详细介绍 AvatarNode

29. 每个 map 槽就是一个线程。（错误 ）

分析：首先我们知道什么是 map 槽,map 槽->map slot

map slot 只是一个逻辑值 (org.apache.hadoop.mapred.TaskTracker.TaskLauncher.numFreeSlots)，而不是对应着一个线程或者进程

具体见：

hadoop 中槽-slot 是线程还是进程讨论

30. Mapreduce 的 input split 就是一个 block。（错误 ）

InputFormat 的数据划分、Split 调度、数据读取三个问题的浅析

32. Hadoop 环境变量中的 HADOOP_HEAPSIZE 用于设置所有 Hadoop 守护线程的内存。它默认是 200 GB。（ 错误）

hadoop 为各个守护进程（namenode,secondarynamenode,jobtracker,datanode,tasktracker）统一分配的内存在 hadoop-env.sh 中设置，参数为 HADOOP_HEAPSIZE，默认为 1000M。

具体参考 hadoop 集群内存设置

33. DataNode 首次加入 cluster 的时候，如果 log 中报告不兼容文件版本，那需要 NameNode

执行“Hadoop namenode -format”操作格式化磁盘。（错误 ）

分析：

首先明白介绍，什么 ClusterID

ClusterID

添加了一个新的标识符 ClusterID 用于标识集群中所有的节点。当格式化一个 Namenode，需要提供这个标识符或者自动生成。这个 ID 可以被用来格式化加入集群的其他 Namenode。

二次整理

有的同学问题的重点不是上面分析内容：内容如下：

这个报错是说明 DataNode 所装的 Hadoop 版本和其它节点不一致，应该检查 DataNode 的 Hadoop 版本

详细内容可参考

hadoop 集群添加 namenode 的步骤及常识

以上答案通过多个资料验证，对于资料不充分的内容，都标有”个人观点“，给出本测试题抱着谨慎的态度，希望大家多批评指正。

上面只是选择与判断，可以看另外一套实战面试[实战面试题]

转载请注明：出自 about 云 http://www.aboutyun.com/thread-6787-1-1.html

百度 2015 校园招聘面试题（成功拿到 offer）

引言

盼望着，盼望着.....今年终于轮到我找工作了，还深深记得去年跟在师兄后面各种打酱油的经历，当时觉得找工作好难啊，怎么面一个败一个，以后还能找到工作不？

不过当时的失败也是理所当然的，那时候没有做任何准备（连进程间有几种通信方式这样老掉牙的题我都不知道），没有任何找工作的经验，甚至一个简单的自我介绍都吞吞吐吐的。

经过一年时间的磨练，特别是近几个月的强度知识吸收，感觉个人在能力和知识储备方面有了质的提高，这大大提高了我的自信心，也让我在这个秋季的求职生涯最终以较满意收场。

截止目前为止，找工作总算告一段落。初次找工作，只投了前面的几家公司（有百度、阿里、美团、搜狗、华为等），今年比较幸运，面的几家公司都成功拿到 offer，也算是初战告捷。这些公司的招聘结束后我就没再参加后面的一些大型互联网公司了，比如奇虎 360、网易、爱奇艺、金山等等，不像我的一些同学，手里 offer 多的都拿不动了，也有满意的了，可还是满怀干劲的参加后面的招聘，真心不知道他们的精力和耐力从何而来，可能是他们想成为传说中的 offer 帝和面霸吧。我个人无喜于那些称谓，既然有了较满意的 offer 就不想再耗费太多的精力去争根本不会考虑的 offer，也可能是自己的惰性所为.....

OK，废话到此为止，下面简单回顾下自己在百度面试过程中的一些题目，给还在找工作或将来要找工作的同学一个浅薄的参考。

百度面试题

一面（1 hour）：

1. 面试官从简历里抽了一个较感兴趣的项目，让把项目简单介绍了下，针对项目问了几个技术问题
2. 介绍 Java 中垃圾回收机制，程序员平时需要关注这个吗？为什么？请举例说明。
3. 数据库隔离级别介绍、举例说明。
4. override 和 overload 的区别。
5. 求二叉树的最大距离（即相距最远的两个叶子节点），写代码。
6. 两个栈实现一个队列，写代码。
7. 你觉得你的优势是什么？有什么技术薄弱点吗？
8. 目前手上有 offer 吗？

二面（40 minutes）：

1. 详细介绍研究生期间的小论文项目。
2. 求二叉树的宽度，先简介思路再写代码。
3. Hashmap、Hashtable 和 cocurrentHashMap 的区别，要讲出它们各自的实现原理才行，比如 Hashmap 的扩容机制、cocurrentHashMap 的桶分割原理、多线程安全性。
4. 进程调度算法，有哪些算法比较难实现？
5. linux 下如何修改进程优先级？（nice 命令的使用）。
6. linux 下性能监控命令 uptime 介绍，平均负载的具体含义是什么？建议看 server load 概念。
7. linux 下如何调试程序？说到 gdb，具体如何调试？如何查看 core 文件中的堆栈信息等（bt 指令）。

三面（1 hour and twenty minutes）：

1. 介绍我研究生期间的论文，讲的很详细，每个点具体采用的技术、实现方法等，花了较长时间。
2. 打印二叉树两个叶子节点间的路径，写代码（汗，百度这么喜欢问二叉树）。
3. 字符串中第一个只出现一次的字符，如何优化算法使得遍历次数更少？
4. socket 编程相关，如果服务器这边调用 write 写了 100 个字节的数据，客户端想要获得这个数据，是直接用 read 系统调用，参数也是 100 吗？
5. 百度新闻缓存预算问题：一般为了追求时间性能，都需要缓存一些新闻数据，你怎么计算所需预算？然后申请需要的主机……

6. 多线程的适用场景是什么？为啥要用多线程？
7. 问是否会 go 语言，.....
8. 为啥对技术感兴趣，一些相关问题讨论。
9. 聊北京、谈 offer。

最后面试官说像计算机体系结构、操作系统这样的书一定要看国外的，国内的有时候会误导人。

总结

三面都是技术面，总体下来没有特别难的题目，从我的面试情况来看，百度这次非常看重面试者对二叉树的掌握情况，还有所做的项目详细介绍。后面我会继续分享自己在面试过程中的一些个人经验和技巧。

文章出处：

[百度 2015 校园招聘面试题（成功拿到 offer）]

去公司面试，记录下的最新 hadoop 面试题

1.简要描述如何安装配置一个 apache 开源版 hadoop，描述即可，列出步骤更好

2.请列出正常工作的 hadoop 集群中 hadoop 都需要启动哪些进程，他们的作用分别是什么？

3.启动 hadoop 报如下错误，该如何解决？
error org.apache.hadoop.hdfs.server.namenode.NameNode

org.apache.hadoop.hdfs.server.common.inconsistentFSStateExceptio

n Directory /tmp/hadoop-root/dfs/name is in an inconsistent

state storage direction does not exist or is not accessible?

4.请写出以下执行命令
1）杀死一个 job?
2)删除 hdfs 上的/tmp/aaa 目录
3 加入一个新的存储节点和删除一个计算节点需要刷新集群状态命令？

5.请列出你所知道的 hadoop 调度器，并简要说明其工作方法？
6.请列出在你以前工作中所使用过的开发 mapreduce 的语言？
7.当前日志采样格式为

```
1.  a,b,c,d
2.  b,b,f,e
3.  a,a,c,f
```

复制代码

请用你最熟悉的语言编写一个 mapreduce，并计算第四列每个元素出现的个数

8.你认为用 Java，Streaming,pipe 方式开发 mapreduce,各有哪些优缺点？
9.hive 有哪些方式保存元数据，各有哪些特点？
10.请简述 hadoop 怎么样实现二级排序？
11.简述 hadoop 实现 join 的几种方法？
12.请用 Java 实现非递归二分查找？
13.请简述 mapreduce 中，combiner，partition 作用？
14.某个目录下有两个文件 a.txt 和 b.txt,文件格式为（ip，username），

列如：

a.txt
127.0.0.1 zhangsan
127.0.0.1 wangxiaoer
127.0.0.2 lisi
127.0.0.3 wangwu

b.txt
127.0.0.4 lixiaolu
127.0.0.1 lisi

每个文件至少 100 万行，请使用 Linux 命令完成如下工作：

1）每个文件各自的 ip 数
2）出现在 b.txt 而没有出现在 a.txt 的 ip
3）每个 user 出现的次数以及每个 user 对应的 ip 数

程序员找不到开发工作的 6 大原因

导读

在这个信息畅通的时代，我们有时候面试，不止是你当时面试的情况，同样面试时的表现虽然暂居了大部分，但是一些额外的信息，可能会决定你面试的成功与失败。

面试更多的是一种修炼与思考，一个人的框架、思想决定了你面试的内容，不同人，面试官会问不同的内容：

1.你认为面试的时候，你该如何做?

2.你认为面试中，决定性因素是什么？

3.为什么不同的人，面试官会问不同的问题？

如果思考透上面的问题，相信你已经对面试有所了解，成功的概率也会增大。

你申请了自己梦寐以求的开发工作，得到了面试机会，感觉一切都很顺利，但是最后所有的努力换来的竟然只是一封拒绝信！？

原因可能是以下几点：

1.我们 Google 了你的名字

"这家伙不错"，但是当我用你的名字和电子邮件地址作了个快速搜索之后，我整个人都不好了。

嗯，你有一个可爱的小博客，但是貌似已经四年没有更新了，当然这不是我关注的重点。

可是貌似你的电子邮件地址别名 GolDieHoRE 正在网上销售《World of Warcraft gold》。好吧，这个我也不多说了，可是你的网页遍布色情广告，甚至在上面大声叫嚣"那些混球能强迫我坐在办公桌前，但是不能强迫我去为他们工作，哼哈哈哈！"这就是个大问题了。

2.你毫无礼貌地将自己的简历扔在我的办公桌上，自命不凡地嚷嚷"快膜拜我吧，小子！"

在你走进我的办公室之前，我就已经看过你长达 10 页的简历了，你有多少斤两，我心知肚明。

甚至，我还知道你刚刚看完《Engineering The Alpha》（因为你在 Facebook 有提到过它，还提及正要让一些傻瓜看看你的实力，我想知道这所谓的傻瓜指的是我吗？）。

但是，你也不应该毫无礼貌地将自己的简历扔在我的办公桌上，自命不凡地嚷嚷"快膜拜我吧，小子！"，我只想说，小子，傲什么傲，从哪儿来回哪儿去吧！

3.号称有着 15 年的 Angular.js 经验

我知道你很牛，哪怕你还只有 25 岁，但是已经有 20 年的编程经验。我也可以相信你的学习进度很快，但让我感到不舒服的是，你话里话外都在暗示你在特定技术上有着超长的工作经验。

要知道，Angular.js 是最近 3 年才出现在我们视野中的，所以，可以判定要么是你在撒谎，要么你已经神奇地发现了一种方法来规避线性时间的压迫限制。

我的意思是，可能你买到了时光机，又或者是穿越的，才能规避线性时间的压迫限制。但是不管是前面哪种不可能的可能，我只觉得你不是我们想要的那种人。

4.过于激进

亲，这是在面试，不是在和你讨论宪法条文，也不是议论政治立场，所以不要这么激动好不好，弄得好像有深仇大恨要打起来一样。

而且，我觉得如果要我面对你不停把玩的武器问一些可能你难以回答的问题，我想我小小心肝还没有那么强大。我只想哭着说，完全不能好好对话啊！

可能那些脾气比较好的人带着武器来面试还可以接受，但是我很担心要是面对一些苛刻的客户，你会不会怒而奋起，挥舞你漂亮的左轮手枪？恕我冒犯，您这尊大佛小庙请不起。

5.忒实诚了点

首先我得表明我的立场——诚实是美德。但是坦白告诉你，我们并不希望你在面试中犹如竹篓倒豆子一样把你所有的想法都说出来。

你一走进来就指着我的裤子说"漂亮的套装"，虽然可以夸你"你的观察力还算敏锐"，但是这完全不是我们需要探讨的方向，好不好？

我不喜欢的是，我都没开口说话，你就自己热情洋溢地高声畅谈起为什么你需要找一份新的开发工作。但是我想提一提，你可不可以不要将心里所有的想法都说出来啊——要知道你这样"诋毁"你的现任老板以及如此描述你们之间不堪的关系，让我着实很担心你在我们这儿是否也会如此。

然后，在你离开之际，甚至还火上浇油地跟我说"面试过我的那么多人中，你不是最聪明的，要是我在的话，你就只有炒鱿鱼的份！"你说我会请你不？我脑门又不是被驴给踢了。

6.拿不出什么有价值的东西

再重申一次，我不是在指责谁，如有雷同，纯属巧合。

我不得不说，你把开发工作想得太过于简单了。我们又不是慈善企业，你有价值我们才会愿意聘请你，知道不。

我能体会你迫切地想要一份工作"就像一只疯狗即使在你睡觉的时候也不断地朝你嘶吼"的心情。

但是，即使我们再想帮你，也没法说服自己让你一个只会用 Excel 做电子表格的家伙担任技术架构师和首席软件开发人员的职位。

甚至，当我，纯粹是处于礼貌地询问，关于你所掌握的 Excel 技能时，你竟然不屑一顾地表示不想多说，然后一个劲地重复"听着，要是你们不雇佣我的话，我就没钱吃饭了。而且我都说了几百遍了，我很会写代码。"骚年，请你不要这么狂妄自大，行不！

上面的是最新的笑话？

No，No，No，当然不是了。这是在用一种夸张手法表达一些非常实用的建议。

下面是由此我们可以得到的经验教训：

1.不要随便将自己的想法写到网上。因为东西一旦到了网上，那么每个人都可以搜索并看到它们。据我所知，很多所谓的"专家"就是因为这样才"阴沟里翻船"的。

2.自信是好的，但到了自负的境地就糟了。有这么一句话，自负通常意味着极度缺乏信心，因为自信的人根本没必要公开显摆自己的技能和力量，是金子到哪里都会发光。

3.不要撒谎！一个谎言需要无数个谎言来掩盖，并且很容易露马脚。假设你说你知道某技术，但是问你你又回答不上来有关该技术的任何问题，那么你就会被贴上无知和骗子的标志，而不仅仅是无知。无知不可怕，可怕的是以后没人会相信你。

4.面试和工作场合不是表达你个人观点想法的地方。即使你热血沸腾无惧无畏，但是如果你想给人留下好的印象，最好管好自己，不要表现得像个愤青。

5.不要说出自己的真实想法。诚实是美德，但是你得学会先将自己要说的话过滤一遍，不要什么话都脱口而出——不要说出太多的否定意见，也不要透露太多的个人信息。不说谎话，可不是要你主动说一些可能对你面试有碍或者显得你在藐视权威的话。还有，当心马屁拍到马腿上。

6.展示自己的价值。我不止一次地告诉开发人员，不是工作适应薪水，而是薪水适应工作。你有多少价值，就能拿到多少薪水，但是却永远不会高过你所能创造的财富。很多求职者似乎都忽略了这一点，只想着要找好工作，而不去想自己有几斤几两。

http://www.codeceo.com/article/6-resons-you-lose-jobs.html

大数据面试题

一般采用分治法!，大文件映射成小文件

1. 给定 a、b 两个文件，各存放 50 亿个 url，每个 url 各占 64 字节，内存限制是 4G，让你找出 a、b 文件共同的 url？

方案 1：将大文件分成能够被内存加载的小文件。

可以估计每个文件安的大小为 50G×64=320G，远远大于内存限制的 4G。所以不可能将其完全加载到内存中处理。考虑采取分而治之的方法。

s 遍历文件 a，对每个 url 求取 ，然后根据所取得的值将 url 分别存储到 1000 个小文件（记为 ）中。这样每个小文件的大约为 300M。

s 遍历文件 b，采取和 a 相同的方式将 url 分别存储到 1000 各小文件（记为 ）。这样处理后，所有可能相同的 url 都在对应的小文件（ ）中，不对应的小文件不可能有相同的 url。然后我们只要求出 1000 对小文件中相同的 url 即可。

s 求每对小文件中相同的 url 时，可以把其中一个小文件的 url 存储到 hash_set 中。然后遍历另一个小文件的每个 url，看其是否在刚才构建的 hash_set 中，如果是，那么就是共同的 url，存到文件里面就可以了。

方案 2：内存映射成 BIT 最小存储单元。

如果允许有一定的错误率，可以使用 Bloom filter，4G 内存大概可以表示 340 亿 bit。将其中一个文件中的 url 使用 Bloom filter 映射为这 340 亿 bit，然后挨个读取另外一个文件的 url，检查是否与 Bloom filter，如果是，那么该 url 应该是共同的 url（注意会有一定的错误率）。

2. 有 10 个文件，每个文件 1G，每个文件的每一行存放的都是用户的 query，每个文件的 query 都可能重复。要求你按照 query 的频度排序。

方案 1：

s 顺序读取 10 个文件，按照 hash(query)%10 的结果将 query 写入到另外 10 个文件（记为 ）中。这样新生成的文件每个的大小大约也 1G（假设 hash 函数是随机的）。

s 找一台内存在 2G 左右的机器，依次对 用 hash_map(query, query_count)来统计每个 query 出现的次数。利用快速/堆/归并排序按照出现次数进行排序。将排序好的 query 和对应的 query_cout 输出到文件中。这样得到了 10 个排好序的文件（记为 ）。

s 对 这 10 个文件进行归并排序（内排序与外排序相结合）。

方案 2：

一般 query 的总量是有限的，只是重复的次数比较多而已，可能对于所有的 query，一次性就可以加入到内存了。这样，我们就可以采用 trie 树/hash_map 等直接来统计每个 query 出现的次数，然后按出现次数做快速/堆/归并排序就可以了。

方案 3：

与方案 1 类似，但在做完 hash，分成多个文件后，可以交给多个文件来处理，采用分布式的架构来处理（比如 MapReduce），最后再进行合并。

//一般在大文件中找出出现频率高的，先把大文件映射成小文件，模 1000，在小文件中找到高频的。

3. 有一个 1G 大小的一个文件，里面每一行是一个词，词的大小不超过 16 字节，内存限制大小是 1M。返回频数最高的 100 个词。

方案 1：顺序读文件中，对于每个词 x，取 ，然后按照该值存到 5000 个小文件（记为 ）中。这样每个文件大概是 200k 左右。如果其中的有的文件超过了 1M 大小，还可以按照类似的方法继续往下分，知道分解得到的小文件的大小都不超过 1M。 对每个小文件，统计每个文件中出现的词以及相应的频率（可以采用 trie 树/hash_map 等），并取出出现频率最大的 100 个词（可以用含 100 个结 点的最小堆），并把 100 词及相应的频率存入文件，这样又得到了 5000 个文件。下一步就是把这 5000 个文件进行归并（类似与归并排序）的过程了。

4. 海量日志数据，提取出某日访问百度次数最多的那个 IP。

方案 1：首先是这一天，并且是访问百度的日志中的 IP 取出来，逐个写入到一个大文件中。注意到 IP

是 32 位的，最多有 个 IP。同样可以采用映射的方法，比如模 1000，把整个大文件映射为 1000 个小文件，再找出每个小文中出现频率最大的 IP（可以采用 hash_map 进行频率统计，然后再找出频率最大的几个）及相应的频率。然后再在这 1000 个最大的 IP 中，找出那个频率最大的 IP，即为所求。

5. 在 2.5 亿个整数中找出不重复的整数，内存不足以容纳这 2.5 亿个整数。

方案 1：采用 2-Bitmap（每个数分配 2bit，00 表示不存在，01 表示出现一次，10 表示多次，11 无意义）进 行，共需内存 内存，还可以接受。然后扫描这 2.5 亿个整数，查看 Bitmap 中相对应位，如果是 00 变 01，01 变 10，10 保持不变。所描完事后，查看 bitmap，把对应位是 01 的整数输出即可。

方案 2：也可采用上题类似的方法，进行划分小文件的方法。然后在小文件中找出不重复的整数，并排序。然后再进行归并，注意去除重复的元素。

6. 海量数据分布在 100 台电脑中，想个办法高校统计出这批数据的 TOP10。

方案 1：

s 在每台电脑上求出 TOP10，可以采用包含 10 个元素的堆完成（TOP10 小，用最大堆，TOP10 大，

用最小堆）。比如求 TOP10 大，我们首先取前 10 个元素调整成最小堆，如果发现，然后扫描后面的数据，并与堆顶元素比较，如果比堆顶元素大，那么用该元素替换堆顶，然后再调整为最小堆。最后堆中的元 素就是 TOP10 大。

s 求出每台电脑上的 TOP10 后，然后把这 100 台电脑上的 TOP10 组合起来，共 1000 个数据，再利用上面类似的方法求出 TOP10 就可以了。

7. 怎么在海量数据中找出重复次数最多的一个？

方案 1：先做 hash，然后求模映射为小文件，求出每个小文件中重复次数最多的一个，并记录重复次数。然后找出上一步求出的数据中重复次数最多的一个就是所求（具体参考前面的题）。

8. 上千万或上亿数据（有重复），统计其中出现次数最多的钱 N 个数据。

方案 1：上千万或上亿的数据，现在的机器的内存应该能存下。所以考虑采用 hash_map/搜索二叉树/红黑树等来进行统计次数。然后就是取出前 N 个出现次数最多的数据了，可以用第 6 题提到的堆机制完成。

9. 1000 万字符串，其中有些是重复的，需要把重复的全部去掉，保留没有重复的字符串。请怎么设计

和实现？

方案 1：这题用 trie 树比较合适，hash_map 也应该能行。

10. 一个文本文件，大约有一万行，每行一个词，要求统计出其中最频繁出现的前 10 个词，请给出思想，给出时间复杂度分析。

方案 1：这题是考虑时间效率。用 trie 树统计每个词出现的次数，时间复杂度是 O(n*le)（le 表示单词的平准长 度）。然后是找出出现最频繁的前 10 个词，可以用堆来实现，前面的题中已经讲到了，时间复杂度是 O(n*lg10)。所以总的时间复杂度，是 O(n*le)与 O(n*lg10)中较大的哪一个。

11. 一个文本文件，找出前 10 个经常出现的词，但这次文件比较长，说是上亿行或十亿行，总之无法一次读入内存，问最优解。

方案 1：首先根据用 hash 并求模，将文件分解为多个小文件，对于单个文件利用上题的方法求出每个文件件中 10 个最常出现的词。然后再进行归并处理，找出最终的 10 个最常出现的词。

12. 100w 个数中找出最大的 100 个数。

方案 1：在前面的题中，我们已经提到了，用一个含 100 个元素的最小堆完成。复杂度为 O(100w*lg100)。

方案 2：采用快速排序的思想，每次分割之后只考虑比轴大的一部分，知道比轴大的一部分在比 100 多的时候，采用传统排序算法排序，取前 100 个。复杂度为 O(100w*100)。

方案 3：采用局部淘汰法。选取前 100 个元素，并排序，记为序列 L。然后一次扫描剩余的元素 x，与排好序的 100 个元素 中最小的元素比，如果比这个最小的要大，那么把这个最小的元素删除，并把 x 利用插入排序的思想，插入到序列 L 中。依次循环，知道扫描了所有的元素。复杂度 为 O(100w*100)。

13. 寻找热门查询：

搜索引擎会通过日志文件把用户每次检索使用的所有检索串都记录下来，每个查询串的长度为 1-255 字节。假设目前有一千 万个记录，这些查询串的重复读比较高，虽然总数是 1 千万，但是如果去除重复和，不超过 3 百万个。一个查询串的重复度越高，说明查询它的用户越多，也就越热 门。请你统计最热门的 10 个查询串，要求使用的内存不能超过 1G。

(1) 请描述你解决这个问题的思路；

(2) 请给出主要的处理流程，算法，以及算法的复杂度。

方案 1：采用 trie 树，关键字域存该查询串出现的次数，没有出现为 0。最后用 10 个元素的最小推来对出现频率进行排序。

14. 一共有 N 个机器，每个机器上有 N 个数。每个机器最多存 O(N)个数并对它们操作。如何找到 个数中的中数？

方案 1：先大体估计一下这些数的范围，比如这里假设这些数都是 32 位无符号整数（共有 个）。我们把 0 到 的整数划分为 N 个范围段，每个段包含 个整数。比如，第一个段位 0 到 ，第二段为 到 ，...，第 N 个段为 到 。然后，扫描每个机器上的 N 个数，把属于第一个区段的数放到第一个机器上，属于第二个区段的数放到第二个机器上，...，属于第 N 个区段的数放到第 N 个机器 上。注意这个过程每个机器上存储的数应该是 O(N)的。下面我们依次统计每个机器上数的个数，一次累加，直到找到第 k 个机器，在该机器上累加的数大于或等 于 ，而在第 k-1 个机器上的累加数小于 ，并把这个数记为 x。那么我们要找的中位数在第 k 个机器中，排在第 位。然后我们对第 k 个机器的数排序，并找出第 个数，即为所求的中位数。复杂度是 的。

方案 2：先对每台机器上的数进行排序。排好序后，我们采用归并排序的思想，将这 N 个机器上的数归并起来得到最终的排序。找到第 个便是所求。复杂度是 的。

15. 最大间隙问题

给定 n 个实数 ，求着 n 个实数在实轴上向量 2 个数之间的最大差值，要求线性的时间算法。

方案 1：最先想到的方法就是先对这 n 个数据进行排序，然后一遍扫描即可确定相邻的最大间隙。但该方法不能满足线性时间的要求。故采取如下方法：

s 找到 n 个数据中最大和最小数据 max 和 min。

s 用 n-2 个点等分区间[min, max]，即将[min, max]等分为 n-1 个区间（前闭后开区间），将这些区间看作桶，编号为 ，且桶 的上界和桶 i+1 的下届相同，即每个桶的大小相同。每个桶的大小为： 。实际上，这些桶的边界构成了一个等差数列（首项为 min，公差为 ），且认为将 min 放入第一个桶，将 max 放入第 n-1 个桶。

s 将 n 个数放入 n-1 个桶中：将每个元素 分配到某个桶（编号为 index），其中 ，并求出分到每个桶

的最大最小数据。

s 最大间隙：除最大最小数据 max 和 min 以外的 n-2 个数据放入 n-1 个桶中，由抽屉原理可知至少有一个桶是空的，又因为每个桶的大小相同，所以最大间隙 不会在同一桶中出现，一定是某个桶的上界和气候某个桶的下界之间隙，且该量筒之间的桶（即便好在该连个便好之间的桶）一定是空桶。也就是说，最大间隙在桶 i 的上界和桶 j 的下界之间产生 。一遍扫描即可完成。

16. 将多个集合合并成没有交集的集合：给定一个字符串的集合，格式如： 。要求将其中交集不为空的集合合并，要求合并完成的集合之间无交集，例如上例应输出 。

(1) 请描述你解决这个问题的思路；

(2) 给出主要的处理流程，算法，以及算法的复杂度；

(3) 请描述可能的改进。

方案 1：采用并查集。首先所有的字符串都在单独的并查集中。然后依扫描每个集合，顺序合并将两个相邻元素合并。例如，对 于 ，首先查看 aaa 和 bbb 是否在同一个并查集中，如果不在，那么把它们所

在的并查集合并，然后再看 bbb 和 ccc 是否在同一个并查集中，如果不在，那么也 把它们所在的并查集合并。接下来再扫描其他的集合，当所有的集合都扫描完了，并查集代表的集合便是所求。复杂度应该是O(NlgN)的。改进的话，首先可 以记录每个节点的根结点，改进查询。合并的时候，可以把大的和小的进行合，这样也减少复杂度。

17. 最大子序列与最大子矩阵问题 数组的最大子序列问题：给定一个数组，其中元素有正，也有负，找出其中一个连续子序列，使和最大。

方案 1：这个问题可以动态规划的思想解决。设 表示以第 i 个元素 结尾的最大子序列，那么显然 。基于这一点可以很快用代码实现。

最大子矩阵问题：给定一个矩阵（二维数组），其中数据有大有小，请找一个子矩阵，使得子矩阵的和最大，并输出这个和。

方案 1：可以采用与最大子序列类似的思想来解决。如果我们确定了选择第 i 列和第 j 列之间的元素，那么在这个范围内，其实就是一个最大子序列问题。如何确定第 i 列和第 j 列可以词用暴搜的方法进行。

hbase 40 道测试题

1. HBase 来源于哪篇博文？ C
A The Google File System
B MapReduce
C BigTable
D Chubby

2. 下面对 HBase 的描述哪些是正确的？ B、C、D
A 不是开源的
B 是面向列的
C 是分布式的
D 是一种 NoSQL 数据库

3. HBase 依靠（）存储底层数据 A
A HDFS
B Hadoop
C Memory
D MapReduce

4. HBase 依赖（）提供消息通信机制 A
A Zookeeper
B Chubby
C RPC
D Socket

5. HBase 依赖（）提供强大的计算能力 D
A Zookeeper
B Chubby
C RPC
D MapReduce

6. MapReduce 与 HBase 的关系，哪些描述是正确的？ B、C
A 两者不可或缺，MapReduce 是 HBase 可以正常运行的保证
B 两者不是强关联关系，没有 MapReduce，HBase 可以正常运行
C MapReduce 可以直接访问 HBase
D 它们之间没有任何关系

7. 下面哪些选项正确描述了 HBase 的特性？ A、B、C、D
A 高可靠性
B 高性能

C 面向列
D 可伸缩

8. 下面与 Zookeeper 类似的框架是？D
A Protobuf
B Java
C Kafka
D Chubby

9. 下面与 HDFS 类似的框架是？C
A NTFS
B FAT32
C GFS
D EXT3

10. 下面哪些概念是 HBase 框架中使用的？A、C
A HDFS
B GridFS
C Zookeeper
D EXT3

第二部分：HBase 核心知识点
11. LSM 含义是？A
A 日志结构合并树
B 二叉树
C 平衡二叉树
D 长平衡二叉树

12. 下面对 LSM 结构描述正确的是？ A、C
A 顺序存储
B 直接写硬盘
C 需要将数据 Flush 到磁盘
D 是一种搜索平衡树

13. LSM 更能保证哪种操作的性能？B
A 读
B 写
C 随机读
D 合并

14. LSM 的读操作和写操作是独立的？A
A 是。

B 否。
C LSM 并不区分读和写
D LSM 中读写是同一种操作

15. LSM 结构的数据首先存储在（）。 B
A 硬盘上
B 内存中
C 磁盘阵列中
D 闪存中

16 HFile 数据格式中的 Data 字段用于（）。A
A 存储实际的 KeyValue 数据
B 存储数据的起点
C 指定字段的长度
D 存储数据块的起点

17 HFile 数据格式中的 MetaIndex 字段用于（）。D
A Meta 块的长度
B Meta 块的结束点
C Meta 块数据内容
D Meta 块的起始点

18 HFile 数据格式中的 Magic 字段用于（）。A
A 存储随机数，防止数据损坏
B 存储数据的起点
C 存储数据块的起点
D 指定字段的长度

19 HFile 数据格式中的 KeyValue 数据格式，下列选项描述正确的是（）。A、D
A 是 byte[]数组
B 没有固定的结构
C 数据的大小是定长的
D 有固定的结构

20 HFile 数据格式中的 KeyValue 数据格式中 Value 部分是（）。C
A 拥有复杂结构的字符串
B 字符串
C 二进制数据
D 压缩数据
第三部分：HBase 高级应用介绍
31 HBase 中的批量加载底层使用（）实现。A
A MapReduce
B Hive

C Coprocessor
D Bloom Filter

32. HBase 性能优化包含下面的哪些选项？A、B、C、D
A 读优化
B 写优化
C 配置优化
D JVM 优化

33. Rowkey 设计的原则，下列哪些选项的描述是正确的？A、B、C
A 尽量保证越短越好
B 可以使用汉字
C 可以使用字符串
D 本身是无序的

34. HBase 构建二级索引的实现方式有哪些？ A、B
A MapReduce
B Coprocessor
C Bloom Filter
D Filter

35. 关于 HBase 二级索引的描述，哪些是正确的？A、B
A 核心是倒排表
B 二级索引概念是对应 Rowkey 这个“一级”索引
C 二级索引使用平衡二叉树
D 二级索引使用 LSM 结构

36. 下列关于 Bloom Filter 的描述正确的是？A、C
A 是一个很长的二进制向量和一系列随机映射函数
B 没有误算率
C 有一定的误算率
D 可以在 Bloom Filter 中删除元素
第四部分：HBase 安装、部署、启动
37. HBase 官方版本可以安装在什么操作系统上？A、B、C
A CentOS
B Ubuntu
C RedHat
D Windows

38. HBase 虚拟分布式模式需要（）个节点？A
A 1
B 2
C 3

D 最少 3 个

39. HBase 分布式模式最好需要（）个节点？C
A 1
B 2
C 3
D 最少

40. 下列哪些选项是安装 HBase 前所必须安装的？A、B
A 操作系统
B JDK
C Shell Script
D Java Code

41. 解压.tar.gz 结尾的 HBase 压缩包使用的 Linux 命令是？A
A tar -zxvf
B tar -zx
C tar -s
D tar -nf

阿里 2015 校招面试回忆（成功拿到 offer）

1. 引言

继上次《百度 2015 校园招聘面试题回忆（成功拿到 offer）》文章过后，大家都希望除了题目之外，最好能给出自己当时的回答情况，看看有没有什么回答技巧，这样更有参考价值。

嗯，建议的很对，因此这次对于阿里的面试回忆，我下面以对话的形式尽可能复现我当初的面试场景。

声明：下面只复述我觉得有参考价值的面试题，实际面试题比这多些（有些不记得了），需要找工作的请认真看完（对 Java 方向的同学更有帮助），不需要的大牛们请一笑置之。

2. 阿里面试回忆

在说具体的面试场景之前，一个小插曲很有必要说一下：

由于我面的是 Java 开发，但当时负责人员给我安排了一个 C++面试官（在面试官桌子上放着一个"C++方向"的牌子），然后我就跟负责人说"我是面 Java 方向的，不是 C++方向的"，结果那个面试官笑呵呵的说"没关系，在我这面一样的，如果你不懂 C++，我可以不问你 C++语言方面的细节问题"，当时我心里想：虽然我 C++学的和 Java 差不多，就算面 C++也没有问题。但是既然面试官这么说了，肯定不问 C++了，Java 估计也不会问（C++方向的工程师不一定懂 Java 呢），应该是问数据结构、算法、网络、操作系统方面的问题了。虽然都有所准备，面起来也可以，但是鉴于两个方面的原因，我还是没答应在这面。

（1）据说阿里 90%都是招 Java 工程师，如果在这面后面肯定会以为我是面 C++的，胜算就少了；

（2）我这几个月基本上都在专研 Java 方向的知识，也研究了不少 JDK 源码和 Java 相关的项目，感觉自己的 Java 方向知识有了一个质的提高，因此希望面试官能够考查和检验我这方面的知识。

因此我当时跟那个面试官说"不好意思，我还是希望能面 Java 方向的知识"，那个面试官依旧笑呵呵"如果你对 Java 知识的确非常自信，那么可以给你换个 Java 方向的面试官"，然后我只能说"是的，我有研究过很多 JDK 源码，也做了不少 Java 项目"。最后，我出去找负责人给我安排 Java 方向的面试官，负责人说"那你要等会儿才行，估计要半个小时"。"哦，没关系，那我等会吧"，心里想：等不怕，方向搞错了才要命。

说上面这段插曲的目的是想告诫大家，如果你有比较明确的方向，比如"我以后一定要做 C++/Java 方向的开发"，那么一定要强调出来。如果像阿里这样每个面试官都有个方向牌那很容易搞定，如果没有则可以在面试一开始的自我介绍或找其它机会说出来让面试官知道，可能有人认为语言不重要，关键是算法、数据结构、操作系统云云……是的，很多面试官都跟我说过：在以后的工作过程中，语言不重要，重要的是你以后做的东西是不是你感兴趣的，但是我认为语言在面试过程中是有非常大的关系的，如果你是 C++方向，那么面试官可能会问你一些虚函数机制、Linux 下的内存分配策略、内存管理、常用系统调用等这方面的知识；如果你主攻 Java，可能会问集合类区别、可研究过 jdk 源码、数据库等方面的知识。

因此面试的第一步就是让面试官明确知道你主攻的语言方向（可能也有较好的面试官会在提问之前首先问你懂 Java 还是 C++），如果你两个方向都非常懂并且没有比较偏爱的方向，那就无所谓了。

一面：技术面（大概 40 minutes）

半个小时没到，终于给我安排了个 Java 方向的面试官，这次交流非常愉快。下面的"面"代表面试官。

第一阶段：自我介绍

面： 请简单自我介绍下。
我： 我是 XX 大学计算机专业的一名 XX，我研究生期间的方向是 XX，.....（方向简单描述）。12 年暑期在 XX 公司实习了 4 个月，做的是 XX（在一所不知名的本地小公司实习过）.....（其它重要项目的简述）。另外，自己是名开源积极分子，有自己的 Github，而且今年暑期参加了 CSDN 举办的开源夏令营活动，做的是.....

阶段总结：上面只是我自己当时的一个简单自我介绍，经验不足讲的比较乱，而且忘了讲自己十分热衷技术，平时喜欢写技术博客等事情（有些情况下可以加分呢）。建议大家都定制好自己的版本，讲出自己的亮点。

第二阶段：介绍研究生期间的论文项目，针对介绍提出几个相关的技术问题

面： 好，我看下简历。（然后对着我的简历看了十几秒，指着我的第一个项目）这是你刚说的研究生的论文项目吧（我嗯），那你把这个项目详细说下。
我： 我从项目的选题（为啥要做这个方向）、项目采用的技术、新颖的地方、最终达到的效果（由于我做的是某个算法的性能提高，那么我就会讲速度提高了多少？空间压缩了多少？）这几个方面详细介绍了自己的项目。

下面就是针对我的叙述具体问了几个技术相关点，这没什么好说的，因为每个人做的项目不一样，问的东西也不一样。

阶段总结：其实要求讲的这个项目是用 C++写的而不是 Java，不过没关系，对于项目而言，语言就不是很重要了，关键是项目的架构、所采用的技术、能达到什么样的效果。面试官选的项目一般要么是简历中项目经验的第一个、要么是有他感兴趣的、要么项目做的时间比较长的，因此建议在写简历时，把你认为最有把握的项目放在第一位（而不是传说中的要按时间倒序来写项目经验），没太大把握的不要写（被抽问到就惨了）。只要你真真正正的吃透了被抽到的项目，那么这个项目提问阶段是完全 easy 的。

第三阶段：Java 方向的知识，包括 JVM 原理、垃圾回收机制等

面： 你 Java 学的怎么样？
我： 还可以，有研究过部分 JDK 源码，比如常用的集合类如 HashMap/Hashtable、ArrayList/LinkedList、Vector 等，还有 Java5 之后的并发包 JUC 如 concurrentHashMap、Executor 框架、CopyOnWrite 容器等。自己很欣赏 Java 巧妙的垃圾回收机制，看过周志明的《深入理解 Java 虚拟机》，因此对 JVM 相关的知识有所掌握……

面： 嗯，学的挺深的，那你把 JVM 的结构和类加载原理说下。
我： 马上拿起桌上的笔和纸，把虚拟机运行时包含的几个数据区和执行引擎画了下，包括方法区、虚拟机栈、本地方法栈、堆和程序计数器，然后介绍每个区域有什么作用，最后讲 ClassLoader 的类加载机制，还顺便说了下双亲委派机制。

面： （面试官点头表示满意）你刚刚说 Java 的 GC 机制很巧妙，那么它的巧妙之处在哪里？
我： 我从两个方面说下自己的理解：一是 Java 的内存分配原理与 C/C++不同，C/C++每次采用 malloc 或 new 申请内存时都要进行 brk 和 mmap 等系统调用，而系统调用发生在内核空间，每次都要中断进行切换，这需要一定的开销，而 Java 虚拟机是先一次性分配一块较大的空间，然后每次 new 时都在该空间上进行分配和释放，减少了系统调用的次数，节省了一定的开销，这有点类似于内存池的概念；二是有了这块空间过后，如何进行分配和回收就跟 GC 机制有关了，然后我详细介绍了 GC 原理、画图表示年轻代（Eden 区和 Survival 区）、年老代、比例分配及为啥要这样分代回收（我认为巧妙就在于这里），有了 GC 基本结构后，我又详述了下 GC 是具体如何进行内存分配和垃圾回收的。

面： （面试官一直点头表示对我回答的赞同）嗯，看来你对这块的确掌握了，对了，你说你参加的 CSDN 开源夏令营项目是阿里的是吧（我点头），这个夏令营是什么情况？
我： 我简单介绍了 CSDN 举办此次夏令营的目的，顺便说道此次夏令营活动当初有 2000 多人报名参加，最终只筛选出 60 多名，自己凭着开题报告和对开源的热爱赢得了导师的青睐得以入选。

面： 你导师是谁？
我： 淘宝的 XX。
面： 哦，他啊，我认识呢，他是……（后面就简单闲聊了几句，该阶段结束，面试官让等会儿准备二面）

阶段总结：上面的对话有人看了过后可能会说：好简单啊，问的题目都是你会的，当然能过啦。是的，其实这是有技巧在里面的，就是要想办法“先下手为强”，啥意思？即让自己成为主动摊牌者而不是被动回答者，找机会跟面试官说自己熟练掌握了哪些方面的知识、自己喜欢专研什么等等，就像上面我所做的，一开始摊牌说明自己掌握的知识处在哪些地方，引导面试官去问你想让他问的知识点，这样达到双赢的目的（你爽了，面试官也轻松了，因为他不用老是猜你可能知道哪些东西然后试探性的问你这个会吗那个了解过吗）。

二面：技术面 + HR 面（大概 1 hour）

一面很轻松的就过了，但是二面就相对而言有些吃力，问的完全是项目相关，而且不是我最熟的研究生期间的论文项目，而是另外两个项目，由于复习不到位，某些地方回答的不完善。

第一阶段：自我介绍，同上第二阶段：介绍面试官感兴趣的两个项目,一个与推荐系统相关，另一个与 Java web 相关

面： 介绍下你简历上的这个电影个性化推荐引擎，使用的是哪种推荐算法？
我： 改进的基于用户的协同过滤推荐算法。

面： 那好，那你从项目的基本架构、所使用的算法原理、如何改进的、数据如何处理这几个方面介绍下你的项目吧。
我： 我首先画了下项目的架构图，据此图详细讲了下 UserCF 的原理及如何使用用户的社交数据和六维理论改进传统的 UserCF，并写出了改进后的算法公式。然后又说这个项目的数据多大，代码中采用什么数据结构进行处理的。

面： （介绍原理中提到了利用用户相似性来作为推荐的一个参考，面试官追问）那用户的相似性你怎么算的？
我： （汗，这个有个计算公式，我不太记得了，最后根据自己的理解讲了下余弦相似性的计算方式，公式没写全，面试官问公式里的根号怎么算的，我说直接用 Java 的库函数）

面： 你这数据哪来的？有几类数据？数据的存储格式是什么？
我： （该项目时间有点久了，前几天只复习了项目的整体架构和算法原理，忘了看具体的数据了，这里只能凭自己的记忆讲了下数据的存储格式，回来后发现自己讲的虽然没错但不够具体）

附：该电影个性化推荐引擎我早已经放到了自己的 Github 上面，是自己在老师的指导下做的，纯算法，还比较简单有待于改进。

介绍完了这个项目，马上面试官又看中了另一个 Java web 相关的项目，马上追问。

面： 嗯，你这个 XX 系统是用 ssh2 框架做的，那你对这个框架熟吗？

我： 嗯，当时在公司实习时对 ssh 的掌握程度只是会使用级别，那时候没时间去研究框架背后的原理。后来有闲暇时间后，我就深入研究了下这几个框架的原理，还看了部分 spring 的源码，学到了不少知识。

面： 嗯，那你把这三个框架都介绍下。
我： 我开始按自己的理解按 Hibernate、Struts、Spring 的顺序开始讲，Hibernate 讲到它的使用原理及与 iBATIS 的对比，顺便说了下现在似乎大家更倾向于使用 iBATIS、myBATIS 这样更加灵活的轻量级框架。struts 讲了下它的作用就是“将请求与视图分开”，然后讲述从输入 url 到使用 struts 处理的控制流程（struts 从 tomcat 那接管、action 处理），然后也说 struts 现在似乎也不那么倾向于使用因为它有漏洞。最后重点讲了下重头戏 Spring，详细讲述了它解耦的功能、AOP 原理及自己有利用动态代理简单模拟实现过一个简单的 AOP 功能、IOC(DI)等。最后说，从 web 应用层面上看，Hibernate 属于持久层，struts 属于表示层，而 Spring 却贯穿所有于所有层（表示层、业务层、持久层），Spring 也有自己的 MVC 模块、web 模块及 JDBC 和 DAO 模块，只是很少使用，也就是只用一个 Spring 也是完全可以的。

面： （点头表示肯定）你刚说到 struts 有漏洞，那么 Hibernate 是安全的吗？有没有可能发生 xss 攻击和 sql 注入攻击？
我： （汗，这个问题真心没想过，对 Hibernate 的掌握没有 Spring 那么深，只能硬着头皮按自己的理解回答）这个问题没想过，不过我觉得框架没有绝对的安全，Hibernate 是用来操作数据库的，hql 语句里也有 select、where 判断，应该有可能发生 sql 注入攻击，xss 攻击就不太清楚了。（这个回答太糟了）

面试官没说啥，一直在电脑上写着什么东西。这时候旁边的 HR 终于发话了。

HR： 你本科是哪的？为什么选择考研？
我： 开始说出我的“发家史”，从一所不知名的小二本考到了中科大，……

HR： 那你技术上是怎么学习的？
我： 又从本科说起，本科技术很差，到了研究生期间才真正开始技术上的修炼，……balabala

HR： 你的职业规划是什么？
我： ……（每个人的想法不一样）

HR： 你最大的优势是什么？
我： （自己吹吧，也要根据实际情况看）
.....

阶段总结：再次说明项目的重要性，第一个项目有些记忆模糊，答的有瑕疵，这里要引以为戒（一定要对项目知根知底），第二个项目感觉答的还可以，不过 Hibernate 安全问题没答出来，我觉得只要你其它问题答的很好，有个别问题答不出来是不会影响最终的 offer 的。HR 面也很重要，你得说通了，需要提前考虑好常见问题的回答。

第三阶段：到你提问了

自由发挥阶段，可以问问公司内部的培养计划、晋升机制、是否经常有大牛分享技术让我们学习等等.....

---

3. 总结

（1）整个面试过程中没让写代码，没问 Linux 下的一些知识，也没问操作系统、计算机网络相关，我觉得可能是 Java 面试更倾向于从项目中问相关的技术问题，如果你没项目或项目不多，那么就可能问这些计算机基础知识了。

（2）由于之前内推电面的失败，让我丧失了一些小自信，因此在这次阿里的整体面试过程中还是有些紧张，大家请引以为戒，务必在面试中保持淡定的心态，就当是和朋友在一起交流技术问题。

（3）最后，希望我上面对话形式的面经能够给正在找工作或以后找工作的同学们带来一些借鉴意义，希望你们能够从中看出某些问题的答题技巧和所做的准备工作。

面试 hadoop 可能被问到的问题，附部分参考答案

尽信书不如无书，尽信答案不如无答案，下面只供参考：

1、hadoop 运行的原理?

hadoop 主要由三方面组成:

1、HDFS

2、MapReduce

3、Hbase

Hadoop 框架中最核心的设计就是：MapReduce 和 HDFS。MapReduce 的思想是由 Google 的一篇论文所提及而被广为流传的， 简单的一句话解释 MapReduce 就是"任务的分解与结果的汇总"。HDFS 是 Hadoop 分布式文件系统（Hadoop Distributed File System）的缩写 ，为分布式计算存储提供了底层支持。

MapReduce 从它名字上来看就大致可以看出个缘由，两个动词 Map 和 Reduce，"Map（展开）"就是将一个任务分解成为多个任 务，"Reduce"就是将分解后多任务处理的结果汇总起来，得出最后的分析结果。这不是什么新思想，其实在前面提到的多线程，多任务的设计就可以找到这 种思想的影子。不论是现实社会，还是在程序设计中，一项工作往往可以被拆分成为多个任务，任务之间的关系可以分为两种：一种是不相关的任务，可以并行执 行；另一种是任务之间有相互的依赖，先后顺序不能够颠倒，这类任务是无法并行处理的。回到大学时期，教授上课时让大家去分析关键路径，无非就是找最省时的 任务分解执行方式。在分布式系统中，机器集群就可以看作硬件资源池，将并行的任务拆分，然后交由每一个空闲机器资源去处理，能够极大地提高计算效率，同时 这种资源无关性，对于计算集群的扩展无疑提供了最好的设计保证。（其实我一直认为 Hadoop 的卡通图标不应该是一个小象，应该是蚂蚁，分布式计算就好比 蚂蚁吃大象，廉价的机器群可以匹敌任何高性能的计算机，纵向扩展的曲线始终敌不过横向扩展的斜线）。任务分解处理以后，那就需要将处理以后的结果再汇总起 来，这就是 Reduce 要做的工作。

2、mapreduce 的原理?

Hadoop 中的 MapReduce 是一个使用简易的软件框架，基于它写出来的应用程序能够运行在由上千个商用机器组成的大型集群上，并以一种可靠容错的式并 行处理上 T 级别的数据集。

一个 MapReduce 作业（job）通常会把输入的数据集切分为若干独立的数据块，由 map 任务（task）以完全并行的方式处理它们。框架会对 map 的输出先进行排序，然后把结果输入给 reduce 任务。通常作业的输入和输出都会被存储在文件系统中。整个框架负责任务的调度和监控，以及重新执行已经失败的任务。

通常，MapReduce 框架和分布式文件系统是运行在一组相同的节点上的，也就是说，计算节点和存储节点通常在一起。这种配置允许框架在那些已经存好数据的节点上高效地调度任务，这可以使整个集群的网络带宽被非常高效地利用。

MapReduce 框架由一个单独的 master JobTracker 和每个集群节点一个 slave TaskTracker 共同组成。master 负责调度构成一个作业的所有任务，这些任务分布在不同的 slave 上，master 监控它们的执行，重新执行已经失败的任务。而 slave 仅负责执行由 master 指派的任务

3、HDFS 存储的机制?

HDFS 的三个实体

数据块

每个磁盘都有默认的数据块大小,这是磁盘进行读写的基本单位.构建于单个磁盘之上的文件系统通过磁盘块来管理该文件系统中的块.该文件系统中的块一般为磁盘块的整数倍.磁盘块一般为 512 字节.HDFS 也有块的概念,默认为 64MB(一个 map 处理的数据大小).HDFS 上的文件也被划分为块大小的多个分块,与其他文

件系统不同的是,HDFS 中小于一个块大小的文件不会占据整个块的空间.
HDFS 用块存储带来的第一个明显的好处一个文件的大小可以大于网络中任意一个磁盘的容量,数据块可以利用磁盘中任意一个磁盘进行存储.第二个简化了系统的设计,将控制单元设置为块,可简化存储管理,计算单个磁盘能存储多少块就相对容易.同时也消除了对元数据的顾虑,如权限信息,可以由其他系统单独管理.
DataNode 节点
DataNode 是 HDFS 文件系统的工作节点,它们根据需要存储并检索数据块,受 NameNode 节点调度.并且定期向 NameNode 发送它们所存储的块的列表
NameNode 节点
NameNode 管理 HDFS 文件系统的命名空间,它维护着文件系统树及整棵树的所有的文件及目录.这些文件以两个文件形式永久保存在本地磁盘上(命名空间镜像文件和编辑日志文件).NameNode 记录着每个文件中各个块所在的数据节点信息但并不永久保存这些块的位置信息,因为这些信息在系统启动时由数据节点重建.
没有 NameNode,文件系统将无法使用.如提供 NameNode 服务的机器损坏,文件系统上的所有文件丢失,我们就不能根据 DataNode 的块来重建文件.因此,对 NameNode 的容错非常重要.第一种机制,备份那些组成文件系统元数据持久状态的文件.通过配置使 NameNode 在多个文件系统上保存元数据的持久状态或将数据写入本地磁盘的同时,写入一个远程挂载的网络文件系统.当然这些操作都是原子操作.第二种机制是运行一个辅助的 NameNode,它会保存合并后的命名空间镜像的副本,并在 Name/Node 发生故障时启用.但是辅助 NameNode 保存.态总是滞后于主力节点,所以在主节点全部失效后难免丢失数据.在这种情况下,一般把存储在远程挂载的网络文件系统的数据复制到辅助 NameNode 并作为新的主 NameNode 运行

4、举一个简单的例子说明 mapreduce 是怎么来运行的 ？

5、面试的人给你出一些问题,让你用 mapreduce 来实现？
比如:现在有 10 个文件夹,每个文件夹都有 1000000 个 url.现在让你找出 top1000000url。

6、hadoop 中 Combiner 的作用?
1、combiner 最基本是实现本地 key 的聚合，对 map 输出的 key 排序，value 进行迭代。如下所示：
map: (K1, V1) → list(K2, V2)
combine: (K2, list(V2)) → list(K2, V2)
reduce: (K2, list(V2)) → list(K3, V3)
2、combiner 还具有类似本地的 reduce 功能.
例如 hadoop 自带的 wordcount 的例子和找出 value 的最大值的程序，combiner 和 reduce 完全一致。如下所示：
map: (K1, V1) → list(K2, V2)
combine: (K2, list(V2)) → list(K3, V3)
reduce: (K3, list(V3)) → list(K4, V4)
3、如果不用 combiner，那么，所有的结果都是 reduce 完成，效率会相对低下。使用 combiner，先完成的 map 会在本地聚合，提升速度。
4、对于 hadoop 自带的 wordcount 的例子,value 就是一个叠加的数字,所以 map 一结束就可以进行 reduce 的 value 叠加，而不必要等到所有的 map 结束再去进行 reduce 的 value 叠加。
combiner 使用的合适，可以在满足业务的情况下提升 job 的速度，如果不合适，则将导致输出的结果不正确。

程序员生存定律--表达背后的力量

去除性格和习惯中的致命缺陷

性格决定人缘，而人缘影响沟通成效，最终影响一个人的表达力。想成为一个道德完美的人是非常困难的，但只要稍微注意，去除一些谁都厌烦的性格缺陷还是可能的。

1. 人情练达

在《红楼梦》第八十二回里有一小段对话很有意思：

袭人道："你还提香菱呢，这才苦呢，撞着这位太岁奶奶，难为她怎么过！"把手伸着两个指头道："说起来，比他还利害，连外头的脸面都不顾了。"黛玉接着道："他也够受了，尤二姑娘怎么死了。"袭人道："可不是。想来都是一个人，不过名分里头差些，何苦这样毒？外面名声也不好听。"黛玉从不闻袭人背地里说人，今听此话有因，便说道："这也难说。但凡家庭之事，不是东风压了西风，就是西风压了东风。"

最末一句鲜明的体现了林黛玉和薛宝钗的性格差异。如果是薛宝钗估计会讲，姐妹们需要互相扶持。从《红楼梦》的故事里也可以看到这两种人格会导致的不同结局。

这对我们有一定的启示意义，我们可以抽象出一个极其绝对的场景：

两个人作为一个团队而存在的时候，如果张三无求于李四，或者张三具有绝对的控制权，那么张三不需要和李四做沟通，只要保持沉默或者命令也就足够。否则的话，两者就需要协作，进行更多的交流，今天是你帮助我一点，明天是我帮助你一点，这样彼此工作上都可以有比较好的进境。

以程序员的工作状况来看，期望东风压倒西风式的绝对控制基本上是不可能的。把自己封闭在某个独立的领域里（比如算法），达到绝对高度，做孤狼型的人倒是可能，但终究罕见。而与人协作，从他人那里获得更多的支持并取得成绩这一事情则需要人情练达。

走纯粹技术路线的程序员之间不需要很多这方面的考量，但程序员也是人，一点情商也不要也是不可能的。

想象一个很常见的场景：

张三和李四同时加入公司，张三的水平高一点，因此李四在遇到程序问题时，总是会问到张三，而张三也总是很热心的给予帮助。有一天，张三有事，下午要请假半天，但有一个功能还没有对应掉。这时张三找到李四请他帮忙。但李四头也不回地说：没空，你自己解决吧。

这种情境下，一般来讲张三会愤怒，在可做可不做的时候会拒绝向李四提供帮助。李四不是不能拒绝，但他应该认识到自己欠人人情，拒绝的时候需要诚恳的表示歉意，解释一下自己的困难。

李四如果持续自己的做事风格，可以想见他会越来越被孤立，也许他的技术能力不断提高，但对他的评价则会下浮。除非有一天他达到了一种别人只能仰望的地步，事情也许会有变化。如果李四想往管理方向发展，那么影响就更为致命，这种行事方式几乎堵死了自己取得成绩的可能性。

对于程序员而言，在这个上面需要注意的点并不多，也不需要把自己搞的很累，但有几条传统的智慧还是要的：

- 欠人的要记清楚，别人欠自己的可以含糊。不要认为任何对自己的帮助都是理所应当的。
- 不要为无谓的事情争吵，乃至口出恶言。人与人的关系坏起来容易，修复起来难。
- 要言而有信，确实无法信守承诺时，要主动道歉。
- 不要通过贬低别人来证明自己，也不要因为言辞不当让人以为是在贬低别人。
- 不要恶意欺骗他人。想想当你被恶意欺骗了，你会什么感觉，就知道恶意欺骗别人能造成多大的伤害。

傲气与狂妄的差别

程序员是需要有点傲气的，一点傲气都没有的程序员往往就会失去对技术的追求并失去对自己的信任。这在技术上是很致命的。

但狂妄则是走向灭亡的前兆，要引起警觉。傲气的人会坚持自己的看法，在没有事实和逻辑支撑时绝不轻易认输；但狂妄的人则会在坚持自己的同时贬低甚至羞辱他人。傲气的人大致知道自己骄傲的边界，能够在工作中找出自己的位置；狂妄的人则眼里只有自己，认为公司的规则、所有的同事都得围着自己转。

一旦一个人由傲气转向狂妄，那必然会人嫌狗不爱，这样一来这个人能创造的价值往往会降低，但他的索取却会因为狂妄而不断增加，这就为未来可能的悲剧打下了伏笔。

2. 有条件的顺应环境

中国古代的钱币外形是圆的，但中间则是一个方空，这可以是一种很有含义的隐喻。全无个性的人往往是平庸的，但在那里都张扬个性的人往往是痛苦的。因为公司必然有自己的规则和文化，而这种规则和文化并不会因为某个人而突然发生变化。

我们可以强调职业精神，说拿了钱必须干活，个性完全不关键，但只有这个是不够的。这里面必须把握一种限度，在这种限度下，不只要拿了钱干活，还要努力适应选定的公司。而一旦超过这种限度，那则意味着需要尽快离开，而不是继续的抱怨。这时有两个关键点需要被认识到：一是天下间没有完美的公司；一是要知道那类事情需要顺应。

- 天下没有完美的公司

2012 年 CSDN 转过一篇 Facebook 员工对公司的抱怨，其中的几条非常特别：

l Zuck 的过于关注。既然都成为上市公司了，作为公司的 CEO，你主要接头的应该是：投资者、分析师、博学者等。但你仍然和我们这些工程师谈产品的规划和战略！这是彻彻底底的侵吞时间。你忘了你主要的责任是提高公司的股价而不是原材料的加工。

l 太多的决策由工程师给出。有些决定甚至是一个工程师单独下的，更甚至在午饭中就做出了决定。让缺少公司运作经验的工程师去做这些决定是不是太草率了？！

l 对于内部员工的过度信任。

我们有理由相信，完全相反的抱怨也绝对存在：

l CEO 完全不关注技术。

l 工程师没有决策权。

l 员工完全不被信任。

这充分说明，只要你想抱怨，那就总会有可抱怨的东西。这点起源于人思维的善变以及欲望的无边界特质，实属正常。其实事情并没有那么麻烦。喜欢和不喜欢就像天平的两端，临界点就一个：走还是留。

想走的可以尽情抱怨，自不必说，想留的就要适应某些自己并不喜欢的东西。而选择留下来却使劲抱怨则是最不明智的选择。

这似乎很消极，但以人生而论，无法改变的，无法抛弃的，就要考虑如何去适应。想象一下，不管你如何生气，地球也不会围着你转。

选择了留下来，却去抱怨完全不可能改变的东西，进而总是认为自己受到了不公正的待遇，总是满腹怨气，这不可能不影响到工作，也不可能不影响到别人眼中的你，所以说这也是一种表达。

这里其实有个陷阱：越是认为自己怀才不遇的，那就越真的会怀才不遇。当然也可能其实才华也只是自己认为的。这点在容在有些知名学校的毕业生身上，体现出来。假设说对应某一个学校有一个大致的就业水平，这似乎会对这个学校的学生产生一种心理暗示，他们就应该在某个水平以上的公司里。一旦进入了低于这个水平的公司，心理先天就会有优越感，可能会想：这个人怎么能来领导我？这么多这么差水平的人每天干的都是什么事？逐渐下来就很容易眼高手低，评价也会走低，反倒是越来越沉底。

认不清这点会很麻烦，但凡是多人聚集在一起地方几乎必然是名利场，而名利场中几乎一定有不堪的地方，公司也不例外。总是期望公司百分百与自己的期望相符会导致所有的公司都可以抱怨，进一步导致工作状

态变坏，并对自己造成损伤。当然，接受某些自己不如意的东西也是有底线的，这是下一节的话题。

- 知道那类事情必须顺应

受到委屈的时候，首先要判定的是环境是否公正，不要因为升职的不是自己而郁闷，更可怕的是升职的人不具备对应的能力---后者说明整体环境有问题，这是更应该引起警觉的事情。

坦诚的讲，大部分人并不具备改变周围环境的能力，而更像行业或者公司历史中的一片尘埃。当一个企业的基因确定，其中所蕴含的力量是无比宏大的，当这个企业并没有突破基本公正的底线时，最优的选择只能是在大多地方进行顺应，而非是消极对抗。

最不应该顺应的东西主要有两个：一个是公司中处处显失公平；一是个人在公司里面完全看不到发挥的机会和未来。这两点对个人未来是致命的，弄不清楚还不只是适应不适应的问题，而是糊涂不糊涂的问题。

其他的东西则大多是要适应的。不要看很多大人物今天站在台上无限风光，但在取得成绩的路上，几乎每个人都调整过自己来适应周围的环境。

据说杨元庆先生曾经在事业挫折时流泪过，而柳传志先生曾经对杨元庆先生讲：当你真像鸵鸟那么大时，小鸡才会心服。只有赢得这种"心服"，才具备了在同代人中做核心的条件。

这里隐含的一层意思是，两个公鸡可能一个尾巴长，一个冠子亮，但这时候人们往往无法区分究竟那个更好。选尾巴长的，冠子亮的可能会抱怨；选冠子亮的，尾巴长的可能会抱怨。但这种愤怒是格局不够的一种体现，与其抱怨，不如考虑怎么让自己成为鸵鸟。但恰如前面所说，环境要相对公正。

具体来讲，人不能老等着上司变的开明，变得更英明，这些事很多时候，你改变不了。

你想干个什么事，你得自己做准备，把脏活累活都干了，当然大家看不见这些的，能看见的只有成绩。不能老指望自己动动嘴巴，事情搞定，功劳到手。你得去了解，公司里可能不太好的流程，利害关系人可能有些奇怪的想法，这些都得去理解和摆平。因为换个公司它更可能还是这样子。

你可以讲这太烦了，那也 OK，关键是要能接受平凡的结果。做点事情其实远比想的麻烦，即使是在开明的公司里面，唯有抱怨最容易，但抱怨什么也换不来。

适应环境里有一个极端的情形，也很危险：

很多人可能会认为反正我就赚这么多钱，混混日子也没什么，这也算是彻底适应环境了。但这时候，可能没认识到只要这个状态持续五年，诚然你可能赚到几十万，但失去的却是人生最为黄金时期的五年，一生中所有剩下的时间都可能需要为此而背负债务。在相对公平的环境里主动就是人家跟着你跑，被动就是你跟着别人跑。从长期视角来看，主动去做，错了也是对的；被动做事，对了也是错的。所以被动混日子是危险的，做事的时候要尽可能主动。

这点之所以需要针对程序员群体专门一提是因为程序的世界里是非比较分明，但公司里不是的，再怎么优秀的公司里，也需要一些模糊区域。如果用看待程序的眼光来看待公司，那就 Bug 太多了，并且很多时候很多 Bug 你还不能修，还得假设它是对的，并顺应它，简直是岂有此理，但这也确实是一种现实的规则，无论喜欢不喜欢，都要学会给予它一定的尊重。如果你真的很长情，很有理想，那不妨耐心等待，直到有足够力量把你不喜欢的击个粉碎。当然这不意味着，有意见不能提，而是说提了意见没被采纳，大多时候实属正常。

遭遇可怕的上司怎么办？

很久以前看过慕名看过杰克韦尔奇写的《赢》，可能是自己记忆力不太好，书里说过什么大多是很快忘记了，但其中记录的一件小事却记得特别清楚。

杰克韦尔奇在书里说，2004 年在中国的时候，听众中一个年轻女性流着泪问到，“在只有老板才有发言权”的情况下，又有那个商业人士能够实践坦诚精神和推行区别考评制度呢？我们这些在基层工作的人们有非常多的想法。但很多人甚至想都不敢想能把它们讲出来，除非自己成为老板。”

我之所以记得这个片段，倒不是因为问题本身，而是因为一个人会在公众场合哭着发言---这必然是因为心里累积太多的压力。这也让我私下猜测，想必是我们的商业环境里有很多特别之处。

这种特别之处往往会让我们以更大的频度遭遇一个麻烦的问题：真遇到一个可怕的上司，程序员该怎么办？

在细说这个问题前，首先还是要再强调一下选择权。选择权是博弈的基础，而上一章里提到的自身价值则是选择权的基础。这点虽然在后续章节里不会总强调，但他明显比其他因素有更高的权重。

如果真的遭遇了可怕的上司，并感觉遭遇了不公正的待遇。首先倒不是去和他吵一架，而后辞职，而是要先反省下，看看问题是不是出在自己身上，或者说自己究竟有多大责任。

培根说：聪明者反省自身，愚蠢者欺惑大众，还是很智慧的。

这里可能的原因就太多了。

可能是价值观的冲突，你的上司并非只是针对你而是有自己的是非标准和行事原则，找你麻烦只是因为你的价值观和他的不一样。这种时候如果工作本身没问题，可能需要考虑适应，因为你换个工作可能还有问题。

可能是你年少轻狂做人失误，在很多场合对上司过于藐视。要是这种，要看看能不能修补。毕竟如果当前工作很适合自己，并不适合因为意气之争而换工作。

也可能真是上司纯属个人瞧你不顺眼（因为内斗等）或者他自己过于古怪，这种大致没办法，要考虑尽快换个地方。

3. 去除致命的坏习惯

谈习惯的书很多，但基本上是在告诉你，什么样的习惯更好。但在考虑改善表达力时，却要做逆向思维，

在这里认清什么样的习惯更差是更加关键问题。很少有人会期望程序员八面玲珑，因此很多程序员的习惯都是可接受的，那么不可容忍的到底是什么？

我们来看一个每天都会发生的例子：

A 是一名程序员，每当他宣称自己的工作完成时，你总是能在他的代码或者文档中发现缺陷。比如：代码中不遵守大家约定好的编码规范，使用文件时可能会使用绝对路径并导致基本测试无法通过，文档中记录错操作系统的名字等。

想象一下，长久下来 A 身上会发生么？很简单，他会逐渐失去周围人的信任，也许 A 的能力并不差，能解决比较复杂的问题，但是做程序的时候，有这样的队友也还是很可怕的。

这类问题并不涉及高深的知识，基本上是因为习惯不好而导致的。这类习惯里充满了负能量，会让周围的人倾向于看低你。会导致下面两种结果的习惯等价于职场上的核弹，如果你有，没准那天会被他们炸的粉身碎骨。

一是忽视细节，这会导致别人认为你不具备做事能力。一是负不起责任，这会导致别人认为你不用心做事。“能力不足”和“态度不好”这两顶帽子只要带上一个，个人前景立刻会变的非常暗淡。

- 关于忽视细节。

有的人工作习惯比较好，做的时候稳扎稳打，自己做完会做双重检查，表现出来的结果就是工作的一次成型能力强。与之相反，有的人则做事的时候分心，做完之后不做自我检查，表现出来的结果就是小错误很多，在文档上可能就表现为拼写错误，版本号不对，字体混乱等等。总之，让人感觉就是个半成品。能够一次成型其实是一种很关键，也很被看重的能力，而要想保证这个，只能在小习惯上下功夫。

- 关于推卸责任

写程序的时候给自己的问题找借口特别容易，因为纯粹的像 int add(int i, int j){ return i+j;}这类代码特别少，总是要用一点别人的东西，因此总是可以在别人的身上找到借口。可以抱怨开源的文档少，可以抱怨微软代码不公开，诸如此类。但其实这一点意义也没有，只会让人认为对工作负不起责任。

上述这两类不良习惯中蕴含着巨大的负能量，是每个人要用心规避的。如果说一个人的天分、才华、知识、能力都像水一样，那么上述这两个坏习惯就像漏勺，不知不觉中就拉低了你可以达到的高度。

关于我自己的各种信息，在左边栏可找到，想了解下写这书的人是不是骗子和大忽悠的可以瞄。
最后希望感兴趣的支持 V 众投，感觉上这应该是国内最靠谱的生活购物等的问答社区了吧，都是朋友给朋友做的答案，同时实行一人一号，一人一票制度，想找什么答案关注公众号：vzhongtou（左侧有二维码）就行了。

看到这本书，很不错，给大家推荐下。

java 面试之大数据

第一部分、十道海量数据处理面试题
1、海量日志数据，提取出某日访问百度次数最多的那个 IP。
首先是这一天，并且是访问百度的日志中的 IP 取出来，逐个写入到一个大文件中。注意到 IP 是 32 位的，最多有个 2^32 个 IP。同样可以采用映射的方法， 比如模 1000，把整个大文件映射为 1000 个小文件，再找出每个小文中出现频率最大的 IP（可以采用 hash_map 进行频率统计，然后再找出频率最大 的几个）及相应的频率。然后再在这 1000 个最大的 IP 中，找出那个频率最大的 IP，即为所求。
或者如下阐述（雪域之鹰）：
算法思想：分而治之+Hash
1.IP 地址最多有 2^32=4G 种取值情况，所以不能完全加载到内存中处理；
2.可以考虑采用“分而治之”的思想，按照 IP 地址的 Hash(IP)%1024 值，把海量 IP 日志分别存储到 1024 个小文件中。这样，每个小文件最多包含 4MB 个 IP 地址；
3.对于每一个小文件，可以构建一个 IP 为 key，出现次数为 value 的 Hash map，同时记录当前出现次数最多的那个 IP 地址；
4.可以得到 1024 个小文件中的出现次数最多的 IP，再依据常规的排序算法得到总体上出现次数最多的 IP；

2、搜索引擎会通过日志文件把用户每次检索使用的所有检索串都记录下来，每个查询串的长度为 1-255 字节。

假设目前有一千万个记录（这些查询串的重复度比较高，虽然总数是 1 千万，但如果除去重复后，不超过 3 百万个。一个查询串的重复度越高，说明查询它的用户越多，也就是越热门。），请你统计最热门的 10 个查询串，要求使用的内存不能超过 1G。

典型的 Top K 算法，还是在这篇文章里头有所阐述，详情请参见：十一、从头到尾彻底解析 Hash 表算法。

文中，给出的最终算法是：

第一步、先对这批海量数据预处理，在 O（N）的时间内用 Hash 表完成**统计**（之前写成了排序，特此订正。July、2011.04.27）；

第二步、借助堆这个数据结构，找出 Top K，时间复杂度为 N‘logK。

即，借助堆结构，我们可以在 log 量级的时间内查找和调整/移动。因此，维护一个 K(该题目中是 10)大小的小根堆，然后遍历 300 万的 Query，分别 和根元素进行对比所以，我们最终的时间复杂度是：O（N） + N’*O（logK），（N 为 1000 万，N’为 300 万）。ok，更多，详情，请参考原文。

或者：采用 trie 树，关键字域存该查询串出现的次数，没有出现为 0。最后用 10 个元素的最小推来对出现频率进行排序。

3、有一个 1G 大小的一个文件，里面每一行是一个词，词的大小不超过 16 字节，内存限制大小是 1M。返回频数最高的 100 个词。

方案：顺序读文件中，对于每个词 x，取 hash(x)%5000，然后按照该值存到 5000 个小文件（记为 x0,x1,…x4999）中。这样每个文件大概是 200k 左右。

如果其中的有的文件超过了 1M 大小，还可以按照类似的方法继续往下分，直到分解得到的小文件的大小都不超过 1M。

对每个小文件，统计每个文件中出现的词以及相应的频率（可以采用 trie 树/hash_map 等），并取出出现频率最大的 100 个词（可以用含 100 个结 点的最小堆），并把 100 个词及相应的频率存入文件，这样又得到了 5000 个文件。下一步就是把这 5000 个文件进行归并（类似与归并排序）的过程了。

4、有 10 个文件，每个文件 1G，每个文件的每一行存放的都是用户的 query，每个文件的 query 都可能重复。要求你按照 query 的频度排序。

还是典型的 TOP K 算法，解决方案如下：

方案 1：

顺序读取 10 个文件，按照 hash(query)%10 的结果将 query 写入到另外 10 个文件（记为）中。这样新生成的文件每个的大小大约也 1G（假设 hash 函数是随机的）。

找一台内存在 2G 左右的机器，依次对用 hash_map(query, query_count)来统计每个 query 出现的次数。利用快速/堆/归并排序按照出现次数进行排序。将排序好的 query 和对应的 query_cout 输出到文件中。这样得到了 10 个排好序的文件（记为）。

对这 10 个文件进行归并排序（内排序与外排序相结合）。

方案 2：

一般 query 的总量是有限的，只是重复的次数比较多而已，可能对于所有的 query，一次性就可以加入到内存了。这样，我们就可以采用 trie 树/hash_map 等直接来统计每个 query 出现的次数，然后按出现次数做快速/堆/归并排序就可以了。

方案 3：

与方案 1 类似，但在做完 hash，分成多个文件后，可以交给多个文件来处理，采用分布式的架构来处理（比如 MapReduce），最后再进行合并。

5、 给定 a、b 两个文件，各存放 50 亿个 url，每个 url 各占 64 字节，内存限制是 4G，让你找出 a、b 文件共同的 url？

方案 1：可以估计每个文件安的大小为 5G×64=320G，远远大于内存限制的 4G。所以不可能将其完全加载到内存中处理。考虑采取分而治之的方法。

遍历文件 a，对每个 url 求取 hash(url)%1000，然后根据所取得的值将 url 分别存储到 1000 个小文件（记为 a0,a1,…,a999）中。这样每个小文件的大约为 300M。

遍历文件 b，采取和 a 相同的方式将 url 分别存储到 1000 小文件（记为 b0,b1,…,b999）。这样处理后，所有可能相同的 url 都在对应的小 文件（a0vsb0,a1vsb1,…,a999vsb999）中，不对应的小文件不可能有相同的 url。然后我们只要求出 1000 对小文件中相同的 url 即可。

求每对小文件中相同的 url 时，可以把其中一个小文件的 url 存储到 hash_set 中。然后遍历另一个小文件的每个 url，看其是否在刚才构建的 hash_set 中，如果是，那么就是共同的 url，存到文件里面就可以了。

方案 2：如果允许有一定的错误率，可以使用 Bloom filter，4G 内存大概可以表示 340 亿 bit。将其中一个文件中的 url 使用 Bloom filter 映射为这 340 亿 bit，然后挨个读取另外一个文件的 url，检查是否与 Bloom filter，如果是，那么该 url 应该是共同的 url（注意会有一定的错误率）。

Bloom filter 日后会在本 BLOG 内详细阐述。

6、在 2.5 亿个整数中找出不重复的整数，注，内存不足以容纳这 2.5 亿个整数。

方案 1：采用 2-Bitmap（每个数分配 2bit，00 表示不存在，01 表示出现一次，10 表示多次，11 无意义）进行，共需内存 2^32 * 2 bit=1 GB 内存，还可以接受。然后扫描这 2.5 亿个整数，查看 Bitmap 中相对应位，如果是 00 变 01，01 变 10，10 保持不变。所描完事后，查看 bitmap，把对应位是 01 的整数输出即可。

方案 2：也可采用与第 1 题类似的方法，进行划分小文件的方法。然后在小文件中找出不重复的整数，并排序。然后再进行归并，注意去除重复的元素。

7、腾讯面试题：给 40 亿个不重复的 unsigned int 的整数，没排过序的，然后再给一个数，如何快速判断这个数是否在那 40 亿个数当中？

与上第 6 题类似，我的第一反应时快速排序+二分查找。以下是其它更好的方法：

方案 1：oo，申请 512M 的内存，一个 bit 位代表一个 unsigned int 值。读入 40 亿个数，设置相应的 bit 位，读入要查询的数，查看相应 bit 位是否为 1，为 1 表示存在，为 0 表示不存在。

dizengrong：

方案 2：这个问题在《编程珠玑》里有很好的描述，大家可以参考下面的思路，探讨一下：

又因为 2^32 为 40 亿多，所以给定一个数可能在，也可能不在其中；

这里我们把 40 亿个数中的每一个用 32 位的二进制来表示

假设这 40 亿个数开始放在一个文件中。

然后将这 40 亿个数分成两类:

1.最高位为 0

2.最高位为 1

并将这两类分别写入到两个文件中，其中一个文件中数的个数<=20 亿，而另一个>=20 亿（这相当于折半了）；

与要查找的数的最高位比较并接着进入相应的文件再查找

再然后把这个文件为又分成两类:

1.次最高位为 0

2.次最高位为 1

并将这两类分别写入到两个文件中，其中一个文件中数的个数<=10 亿，而另一个>=10 亿（这相当于折半了）；

与要查找的数的次最高位比较并接着进入相应的文件再查找。

.....

以此类推，就可以找到了,而且时间复杂度为 O(logn)，方案 2 完。

附：这里，再简单介绍下，位图方法：

使用位图法判断整形数组是否存在重复

判断集合中存在重复是常见编程任务之一，当集合中数据量比较大时我们通常希望少进行几次扫描，这时双重循环法就不可取了。

位图法比较适合于这种情况，它的做法是按照集合中最大元素 max 创建一个长度为 max+1 的新数组，然后再次扫描原数组，遇到几就给新数组的第几位置上 1，如遇到 5 就给新数组的第六个元素置 1，这样下次再遇到 5 想置位时发现新数组的第六个元素已经是 1 了，这说明这次的数据肯定和以前的数据存在着重复。这 种给新数组初始化时置零其后置一的做法类似于位图的处理方法故称位图法。它的运算次数最坏的情况为 2N。如果已知数组的最大值即能事先给新数组定长的话效 率还能提高一倍。

欢迎，有更好的思路，或方法，共同交流。

8、怎么在海量数据中找出重复次数最多的一个？

方案 1：先做 hash，然后求模映射为小文件，求出每个小文件中重复次数最多的一个，并记录重复次数。然后找出上一步求出的数据中重复次数最多的一个就是所求（具体参考前面的题）。

9、上千万或上亿数据（有重复），统计其中出现次数最多的钱 N 个数据。

方案 1：上千万或上亿的数据，现在的机器的内存应该能存下。所以考虑采用 hash_map/搜索二叉树/红黑树等来进行统计次数。然后就是取出前 N 个出现次数最多的数据了，可以用第 2 题提到的堆机制完成。

10、一个文本文件，大约有一万行，每行一个词，要求统计出其中最频繁出现的前 10 个词，请给出思想，给出时间复杂度分析。

方案 1：这题是考虑时间效率。用 trie 树统计每个词出现的次数，时间复杂度是 O(n*le)（le 表示单词的平准长度）。然后是找出出现最频繁的前 10 个词，可以用堆来实现，前面的题中已经讲到了，时间复杂度是 O(n*lg10)。所以总的时间复杂度，是 O(n*le)与 O(n*lg10)中较大的哪一 个。

附、100w 个数中找出最大的 100 个数。

方案 1：在前面的题中，我们已经提到了，用一个含 100 个元素的最小堆完成。复杂度为 O(100w*lg100)。

方案 2：采用快速排序的思想，每次分割之后只考虑比轴大的一部分，知道比轴大的一部分在比 100 多的时候，采用传统排序算法排序，取前 100 个。复杂度为 O(100w*100)。

方案 3：采用局部淘汰法。选取前 100 个元素，并排序，记为序列 L。然后一次扫描剩余的元素 x，与排好序的 100 个元素中最小的元素比，如果比这个最小的 要大，那么把这个最小的元素删除，并把 x 利用插入排序的思想，插入到序列 L 中。依次循环，知道扫描了所有的元素。复杂度为 O(100w*100)。

文章出处：

阿里 2015 校招面试回忆（成功拿到 offer）

hadoop、大数据笔试、面试都会问那些问题

1、hdfs 原理，以及各个模块的职责

2、mr 的工作原理

3、map 方法是如何调用 reduce 方法的

4、shell 如何判断文件是否存在，如果不存在该如何处理？

5、fsimage 和 edit 的区别？

6、hadoop1 和 hadoop2 的区别？

笔试：

1、hdfs 中的 block 默认保存几份？

2、哪个程序通常与 nn 在一个节点启动？并做分析

3、列举几个配置文件优化？

4、写出你对 zookeeper 的理解

5、datanode 首次加入 cluster 的时候，如果 log 报告不兼容文件版本，那需要 namenode 执行格式化操作，

这样处理的原因

是？

6、谈谈数据倾斜，如何发生的，并给出优化方案

7、介绍一下 hbase 过滤器

8、mapreduce 基本执行过程

9、谈谈 hadoop1 和 hadoop2 的区别

10、hbase 集群安装注意事项

11、记录包含值域 F 和值域 G，要分别统计相同 G 值的记录中不同的 F 值的数目，简单编写过程。

信息技术有限公司

1、你们的集群规模？

2、你们的数据是用什么导入到数据库的？导入到什么数据库？

3、你们业务数据量多大？有多少行数据？(面试了三家，都问这个问题)

4、你们处理数据是直接读数据库的数据还是读文本数据？

5、你们写 hive 的 hql 语句，大概有多少条？

6、你们提交的 job 任务大概有多少个？这些 job 执行完大概用多少时间？(面试了三家，都问这个问题)

7、hive 跟 hbase 的区别是？

8、你在项目中主要的工作任务是？

9、你在项目中遇到了哪些难题，是怎么解决的？

10、你自己写过 udf 函数么？写了哪些？

11、你的项目提交到 job 的时候数据量有多大？(面试了三家，都问这个问题)

12、reduce 后输出的数据量有多大？

hadoop 面试题

一、

1、hadoop 集群搭建过程，写出步骤。

2、hadoop 集群运行过程中启动那些线程，各自的作用是什么？

3、/tmp/hadoop-root/dfs/name　　the path is not exists or is not accessable.

NameNode main 中报错，该怎么解决。（大意这样 一个什么异常）

4、工作中编写 mapreduce 用到的语言，编写一个 mapreduce 程序。

5、hadoop 命令

1）杀死一个 job 任务　（杀死 50030 端口的进程即可）

2）删除/tmp/aaa 文件目录

3）hadoop 集群添加或删除节点时，刷新集群状态的命令

6、日志的固定格式：

a,b,c,d

a,a,f,e

b,b,d,f

使用一种语言编写 mapreduce 任务，统计每一列最后字母的个数。

7、hadoop 的调度器有哪些，工作原理。

8、mapreduce 的 join 方法有哪些？

9、Hive 元数据保存的方法有哪些，各有什么特点？

10、java 实现非递归二分法算法。

11、mapreduce 中 Combiner 和 Partition 的作用。

12、用 linux 实现下列要求：

```
ip            username
a.txt
210.121.123.12 zhangsan
34.23.56.78    lisi
11.56.56.72    wanger
.....

b.txt
58.23.53.132   liuqi
34.23.56.78    liba
.....
```

a.txt,b.txt 中至少 100 万行。

1）a.txt,b.txt 中各自的 ip 个数，ip 的总个数。

2）a.txt 中存在的 ip 而 b.txt 中不存在的 ip。

3）每个 username 出现的总个数，每个 username 对应的 ip 个数。

13、大意是 hadoop 中 java、streaming、pipe 处理数据各有特点。

14、如何实现 mapreduce 的二次排序。

二、

15、面试官上来就问 hadoop 的调度机制，

16、机架感知，

17、MR 数据倾斜原因和解决方案，

18、集群 HA

三、

19、如果让你设计，你觉得一个分布式文件系统应该如何设计，考虑哪方面内容；

每天百亿数据入 hbase，如何保证数据的存储正确和在规定的时间里全部录入完毕，

不残留数据。

20、对于 hive，你写过哪些 UDF 函数，作用是什么

21、hdfs 的数据压缩算法

22、mapreduce 的调度模式

23、hive 底层与数据库交互原理

24、hbase 过滤器实现原则

25、对于 mahout，如何进行推荐、分类、聚类的代码二次开发分别实现那些借口

四、

26、请问下，直接将时间戳作为行健，在写入单个 region 时候会发生热点问题，为什么呢？

hadoop、storm、hbase 面试题、工作日常问答

工作日常问答

1.一个网络商城 1 天大概产生多少 G 的日志？

2.大概有多少条日志记录（在不清洗的情况下）？

3.日访问量大概有多少个？

4.注册数大概多少？

5.我们的日志是不是除了 apache 的访问日志是不是还有其他的日志？

6.假设我们有其他的日志是不是可以对这个日志有其他的业务分析？这些业务分析都有什么（*********）

1.问：你们的服务器有多少台？

2.问：你们服务器的内存多大？

3.问：你们的服务器怎么分布的？（这里说地理位置分布，最好也从机架方面也谈谈）

4.问：你平常在公司都干些什么？

hbase 的问题

1.hbase 怎么预分区？

2.hbase 怎么给 web 前台提供接口来访问？

3.htable API 有没有线程安全问题，在程序中是单例还是多例？

4.我们的 hbase 大概在公司业务中（主要是网上商城）大概都几个表，几个表簇，大概都存什么样的数据？

5.hbase 的并发问题？

Storm 的问题：

1.metaq 消息队列 zookeeper 集群 storm 集群（包括 zeromq,jzmq,和 storm 本身）就可以完成对商城推荐系统功能吗？还有没有其他的中间件？

2.storm 怎么完成对单词的计数？（个人看完 storm 一直都认为他是流处理，好像没有积攒数据的能力，都是处理完之后直接分发给下一个组件）

3.storm 其他的一些面试经常问的问题？

程序员如何快速准备面试中的算法

阅读导读：

1.备战面试中的算法，可以进行哪些步骤？

2.如果要面机器学习一类的岗位，可以看看哪些书籍？

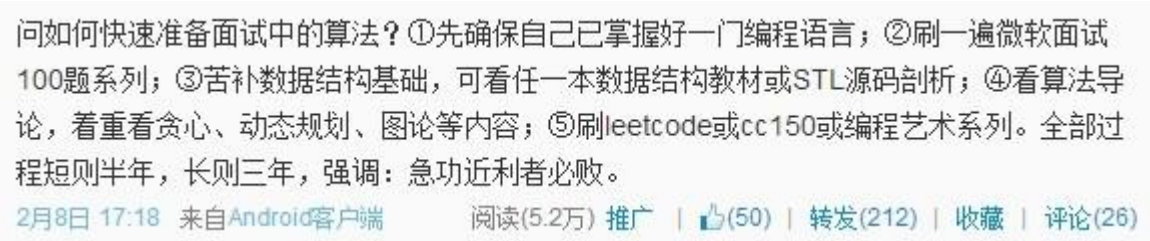
3.去国外找工作的话，可以查看哪些国外的编程面试网站？

4.校招和社招分别需要注意哪些事项？

****

前言

我决定写篇短文，即为此文。之所以要写这篇文章，缘于微博上常有朋友询问，要毕业找工作了，如何备战算法。尽管在微博上简单梳理过，如下图所示：



但因字数限制，许多问题无法一次性说清楚，故特撰此文着重阐述下 程序员如何快速准备面试中的算法，继而推荐一些相关的书籍或资料。顺便也供节后跳槽、3 月春季招聘小高潮、及 6 月毕业找工作的朋友参考。

备战面试中算法的五个步骤　对于立志进一线互联网公司，同时不满足于一辈子干纯业务应用开发，希望在后端做点事情的同学来说，备战面试中的算法，分为五个步骤，如下：

1、掌握一门编程语言

首先你得确保你已掌握好一门编程语言：

- C 的话，推荐 Dennis M. Ritchie & Brian W. Kernighan 合著的《C 程序设计语言》、《C 和指针》，和《征服 C 指针》；
- C++ 则推荐《C++ Primer》，《深度探索 C++对象模型》，《Effective C++》 。　掌握一门语言并不容易，不是翻完一两本书即可了事，语言的细枝末节需要在平日不断的编程练习中加以熟练。

2、过一遍微软面试 100 题系列

我从 2010 年起开始整理微软面试 100 题系列，见过的题目不可谓不多，但不管题目怎般变化，依然是那些常见的题型和考察点，当然，不考察任何知识点，纯粹考察编程能力的题目也屡见不鲜。故不管千变

万化，始终不离两点：

- 看你基本知识点的掌握情况
- 编程基本功

而当你看了一遍微软面试 100 题之后（不要求做完，且这个系列的有些答案存在不少问题，建议以编程艺术 github 版 为准），你自会意识到：数据结构和算法在笔试面试中的重要性。

3、苦补数据结构基础

如果学数据结构，可以看我们在大学里学的任一本数据结构教材都行，包括链表、数组、字符串、矩阵、树、图等等，如果你觉得实在不够上档次，那么可以再看看《STL 源码剖析》。

4、看算法导论

《算法导论》上的前大部分的章节都在阐述一些经典常用的数据结构和典型算法(如二分查找 ,快速排序、Hash 表)，以及一些高级数据结构（诸如红黑树、B 树），如果你已经学完了一本数据结构教材，那么建议你着重看贪心、动态规划、图论等内容，这 3 个议题每一个议题都大有题目可出。同时，熟悉常用算法的时间复杂度。

如果算法导论看不懂，你可以参看本博客。

5、刷 leetcode 或 cc150 或编程艺术系列

如主要在国外找工作，推荐两个编程面试网站：一个是国外一网站 leetcode，它上面有个 OJ 对于找工作的同学来说非常值得一刷 https://oj.leetcode.com/；另外一个是 http://www.careercup.com/，而后这个网站的创始人写了本书，叫《careercup cracking coding interview》，最终这本英文书被图灵教育翻译出版为《程序员面试金典》。

若如果是国内找工作，则郑重推荐我编写的《程序员编程艺术》，有编程艺术博客版，以及在博客版本基础上精简优化的编程艺术 github 版。除此之外，还可看看《编程之美》，与《剑指 offer》。 而不论是准备国内还是国外的海量数据处理面试题，此文必看 :教你如何迅速秒杀掉 :99%的海量数据处理面试题。

此外，多看看优秀的开源代码，如 nginx 或 redis，多做几个项目加以实践之，尽早实习（在一线互联网公司实习 3 个月可能胜过你自个黑灯瞎火摸爬滚打一年)。

当然，如果你是准备社招，且已经具备了上文所说的语言 & 数据结构 & 算法基础，可以直接跳到本第五步骤，开始刷 leetcode 或 cc150 或编程艺术系列。

后记

学习最忌心浮气躁，急功近利，即便练习了算法，也不一定代表能万无一失通过笔试面试关，因为总体说来，在一般的笔试面试中，70%基础+ 30%coding 能力(含算法)，故如果做到了上文中的 5 个步骤，还远远不够，最后，我推荐一份非算法的书单，以此为大家查漏补缺(不必全部看完，欢迎大家补充)：

- 《深入理解计算机系统》
- W.Richard Stevens 著的《TCP/IP 详解三卷》,《UNIX 网络编程二卷》,《UNIX 环境高级编程：第 2 版》
- 你如果要面机器学习一类的岗位，建议看看相关的算法（如支持向量机通俗导论（理解 SVM 的三层境界））,及老老实实补补数学基础，包括微积分、线性代数、概率论与数理统计（除了教材，推荐一本《数理统计学简史》）、矩阵论（推荐《矩阵分析与应用》）等..[/ol]　综上：上述全部过程短则半年，长则三年。　最后要强调的是：急功近利者必败，越想快速越要循序渐进，踏实前进，若实在觉得算法 & 编程太难，转产品、运营、测试、运维、前端、设计都是不错的选择，因为虽然编程有趣，但不一定人人适合编程。

Hero 在线编程判题、出题系统的演进与优化

问题导读

本文更要的传播的一种思想，及一个工具的发展的历程

关键点在于，

程序如何判断用户的代码是否正确?

前言

以前出门在外玩的时候，经常跑去网吧，去网吧也不干啥事，看看博客，改改博客，但若想修改博客上的一段代码，却发觉网吧没有装编译器这个东西，可一想到安装它需要不少时间，所以每次想在网吧写代码都作罢。

当时，便想，如果某一天打开浏览器，便能在网页上直接敲代码，那该有多好，随时随地，不受编译器限制。好事多磨，今年 3 月终于来 CSDN 来做这样一个在线编程网站 Hero 了：http://hero.csdn.net/，以项目负责人的身份总体负责它的产品和运营、包括出题。

为何要写此文？本文不谈 Hero 如何实现，也不谈今年 3 月至今，它的 PV 涨了多少倍，不谈每一道题的具体解法、思路、代码是怎样的（日后可能会写），更不谈它的界面是如何一步步优化的，只谈谈它的判题系统、出题系统是如何一步步演进和优化的，即它背后是怎样的一种判题机制（用来判断每天几千个用户提交的程序正确与否），以及如何做到让每一个用户都可以来 Hero 上出题的。

顺便对很多朋友询问“Hero 后台到底是怎样判题的，为何我的程序提交出错？”的一个集中回答，把判题机制开放出来，对每一个 Hero 的用户做到公平公正。最后年终将至，也算是对自己近一年工作的部分回

顾与总结。

OK，本文有何问题，欢迎随时指正，对 Hero 有任何改进或建议，欢迎随时向我反馈，thanks。

第三十八章、Hero 在线编程判题、出题系统的演进与优化

一、最初的人工肉眼判题　Hero 从头至尾的实现没有借用过任何开源工具，所以它的每一步探索都显得进展缓慢、推动艰难。在今年 3 月份之前，在 Hero 上玩的人不多，所以我刚来公司时，是完全人工肉眼去看每一个用户的程序思路是否正确，不确定的便得自己复制用户的代码粘贴到编译器里进行编译，看结果是否正常。也就是说如果我出一道题：求 N 个字符的全排列。系统后台只做一件事情，就是把用户的代码简单保存起来。

但打开许多用户的答案后，才发觉他并没有实现全排列，他只是写了一个“hello world”：

```
1.  public class HelloWorld
2.  {
3.          public static void main(String args[])
4.          {
5.                  System.out.println("Hello World");
6.          }
7.  }
```

复制代码

即用户的程序是否正确，我得人工判断。这样的人工判题持续了整整一个月，后来发觉来 Hero 上玩的人越来越多，每天从之前只看几份代码到需要看几百份代码，我便立马觉得不对劲了。

是的，必须得让机器实现自动判题。

二、写测试代码让机器自动判题

2.1、简单粗暴的一系列 if else 判断

怎么让机器实现自动判题呢？其实原理也挺简单，可以在出题时写一段测试代码，然后用这段包含了很多组测试数据的测试代码去验证用户的程序是否正确。

比如现在有一道题是这样子的：

“最长有效括号的长度：给定只包含括号字符'('和 ')"的字符串，请找出最长的有效括号内子括号的长度。举几个例子如下：

例如对于"(()"，最长的有效的括号中的子字符串是"()" ，有效双括号数 1 个，故它的长度为 2。
再比如对于字符串") () ())"，其中最长的有效的括号中的子字符串是"() ()"，有效双括号数 2 个，故它的长度为 4。
再比如对于"(() ())"，它的长度为 6.

换言之，便是有效双括号"()"数的两倍。

给定函数原型

```
1.
2.  int longestValidParentheses(string s),
3.  #include
4.  #include
5.  using namespace std;
6.  class Solution
7.  {
8.  public:
9.      int longestValidParentheses(string s) {
```

```
10.          // Start typing your C/C++ solution below
11.          // DO NOT write int main() function
12.         //....
13.          return length;
14.     }
15. };
16. //start 提示：自动阅卷起始唯一标识，请勿删除或增加。
17. int main()
18. {
19.     //write your code
20.     return 0;
21. }
22. //end //提示：自动阅卷结束唯一标识，请勿删除或增加。
23. 请完成此函数，实现题目所要求的功能。”
```

复制代码

如此，我可能会写这样一段测试代码来验证每一个用户的程序是否正确，如下：

```
1.  //start 提示：自动阅卷起始唯一标识，请勿删除或增加。
2.  int main()
3.  {
4.      char* la ="";
5.      char* b =    "(";
6.      char* c =    ")"    ;
7.      char* d =    ")(";
8.
9.      char* e =    "()";
10.     char* f =     "(()";
11.     char* g =    "())";
```

```
12.
13.     char* h    = "()()";
14.     char* i    = "(())())";
15.     char* j =    "(()()";
16.     char* k =    "()(()";
17.     char* l =    "(()()";
18.     char* m =    "(()())";
19.     char* n =    "((()))())";
20.     char* o =    ")()())()()(";
21.     char* p =    ")(((((())()()))()(()))(";
22.
23.     Solution a;
24.     if (a.longestValidParentheses(la) == 0 && a.longestValidParentheses(b) == 0 &&
    a.longestValidParentheses(c) == 0
25.         && a.longestValidParentheses(d) == 0 && a.longestValidParentheses(e) == 2 &&
    a.longestValidParentheses(f) == 2
26.         && a.longestValidParentheses(g) == 2 && a.longestValidParentheses(h) == 4 &&
    a.longestValidParentheses(i) == 6
27.         && a.longestValidParentheses(j) == 4 && a.longestValidParentheses(k) == 2 &&
    a.longestValidParentheses(l) == 4
28.         && a.longestValidParentheses(m) == 6 && a.longestValidParentheses(n) == 8 &&
    a.longestValidParentheses(o) == 4
29.         && a.longestValidParentheses(p) == 22
30.         )
31.     {
32.         cout<<"Y!"<<endl;
33.     }
34.     else
35.     {
36.         cout<<"N!"<<endl;
37.     }
```

```
38.
39.     return 0;
40. }
41. //end //提示：自动阅卷结束唯一标识，请勿删除或增加。
```

```
1.  int main()
2.  {
3.  //write your code
4.  return 0;
5.  }
6.  //end //提示：自动阅卷结束唯一标识，请勿删除或增加。
```

这样，只要在出题时写好测试代码，机器便能实现自动判题了。但很快，我们发现上述这样的测试代码有两个可以优化的地方：

一个 for 循环代替一系列 if else；不用让机器跑完所有测试数据，而是只要有一组测试数据没有通过，程序立即退出，即只要机器找到用户第一组没有通过的测试数据即可。

2.2、for 循环代替 if else

如上节所说，如果测试数据比较少量，还好说，但数据量一大，那么就得写很长很长一段的 if else，那对 coding 的人来说显得非常业余。于是，针对下面这样一道题来看：

"合法字符串：用 n 个不同的字符（编号 1 - n），组成一个字符串，有如下 2 点要求：

1、对于编号为 i 的字符，如果 2 * i > n，则该字符可以作为最后一个字符，但如果该字符不是作为最后一个字符的话，则该字符后面可以接任意字符；

2、2、对于编号为 i 的字符，如果 2 * i <= n，则该字符不可以作为最后一个字符，且该字符后面所紧接着的下一个字符的编号一定要 >= 2 * i。

问有多少长度为 M 且符合条件的字符串。

例如：N = 2，M = 3。则 abb, bab, bbb 是符合条件的字符串，剩下的均为不符合条件的字符串。
输入：输入：n,m (2<=n,m<=1000000000)；
输出：满足条件的字符串的个数，由于数据很大，输出该数 Mod 10^9 + 7 的结果。
函数头部

```
1.  int validstring(int n,int m) {}”
```

我们便可以写出如下的测试代码：

```
1.  //start 提示：自动阅卷起始唯一标识，请勿删除或增加。
2.  int main() {
3.      const int n[] = {2,2,66,123,31542/*因为题目还在 Hero 线上，故只公开一部分测试数据*/};
4.      const int m[] = {2,1000000000,634,10000,55555535 /*只公开一部分测试数据*/};
5.      const int answer[] = {2,999999994,171104439,8789556,605498333 /*只公开一部分测试数据
    */};
6.      int i;
7.      for (i = 0; i < 4/*实际上 i 不止 4 组*/; ++i) {
8.          if (validstring(n[i],m[i]) != answer[i]) {
9.              break;
10.         }
11.     }
12.     if (i >= 4) {
13.         puts("Y!");
14.     }
15.     else {
16.         printf("N!");
17.     }
18.     return 0;
19. }
20. //end //提示：自动阅卷结束唯一标识，请勿删除或增加。
```

如此，上面这样的一段测试代码便能让机器自动判断用户提交的每一个程序是否正确。但就像你现在去做也会体会到，系统光告诉我的程序是对是错，估计还远远不够，即如果用户的程序错了，那系统得告诉他怎么错了呀？是因为超时，还是程序本身的逻辑错了。

于是，系统很快便反馈了用户出错的第一组数据，怎么实现的呢？很简单，只要把上面那段判断出用户的程序是错的那部分加上出错的那一组数据即可：

```
1.  //start 提示：自动阅卷起始唯一标识，请勿删除或增加。
2.  int main() {
3.          const int n[] = {2,2,66,123,31542/*因为题目还在 Hero 线上，故只公开一部分测试数据*/};
4.          const int m[] = {2,1000000000,634,10000,55555535 /*只公开一部分测试数据*/};
5.          const int answer[] = {2,999999994,171104439,8789556,605498333 /*只公开一部分测试
    数据*/};
6.          int i;
7.          for (i = 0; i = 4) {
8.                  puts("Y!");
9.          }
10.         else {
11.                 printf("N!\n%d %d\n",n,m);
12.         }
13.         return 0;
14. }
15. //end //提示：自动阅卷结束唯一标识，请勿删除或增加。
```

很快，我又发现，对于出题本身来说，构造它的完整而全面的测试数据已属不易，现在还要针对每一道

题目去写一份测试代码，而且是每一种语言 C、C++、Java、C#都写一遍，做一次就意识到这不对劲了。可这样一段要求为每一道题每一种编程语言的写痛苦的测试代码的过程持续了整整半年，直到今年 10 月份。

那是否可以简化写这个测试代码的工作，让系统本身变得更加智能呢？因为既然关键是测试数据的构造，那么在有了测试数据的前提下，是否只要填测试数据了，而不必再写测试代码呢？请看下面本文第 3 节部分。

三、出题系统本身的持续改进与优化

3.1、漫长的写测试代码的过程

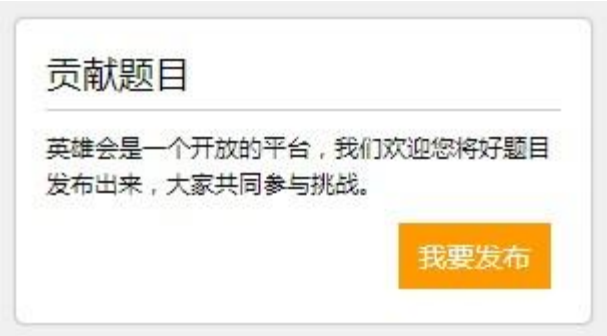
如上文所述，直到今年 10 月份，hero 后台的判题机制一直都是，针对每一道题每一种语言单独写一份带 main 函数的测试代码，用这段测试代码替换掉用户程序里的 main 函数。如下图所示：右边的 start end 代码 替换掉用户程序里的 start end 代码：

当然，这个写测试代码的过程中，得到了好友曹鹏的鼎力相助，若不是他的支持，我也坚持不了 6 个

月，thanks。但即便有他的帮助，这个漫长的写测试代码的过程还是令人非常煎熬。

此外，推动自己一定把出题的过程简化，不想写测试代码的重要原因还有一个：即正因为自己要对每一道题每一种编程语言都写一份测试代码，导致这种出题效率异常底下，这对于整个 Hero 系统是十分不利的。

因此，团队决定，把出题的接口开放，让所有人都可以来 Hero 上出题，此功能为：社会化出题。在 Hero 首页的右侧边栏，如下：



这个时候，问题就来了，让自己写测试代码也就算了，虽然不轻松，但至少在曹鹏的帮助下还能应对，但怎么可以让用户也去为每一道题每一种语言写测试代码呢？然这一切只是自己的主观判断，并没有太多的实际证据支撑我的判断，于是团队决定，暂时先让社会化出题上线后再说。

3.2、出题时只填测试数据，不写测试代码

3.2.1、出题时一个框一个框的填测试数据

我最担心的糟糕结果还是出现了。10 月初社会化出题上线，直到 10 月底，尽管有一些热心的朋友在没有任何奖励的情况下来 Hero 上出题了，但几乎没有任何人愿意写测试代码。

尽管我们走了不少弯路，导致整个进展的过程非常缓慢，但至少还是在一直前进。

得益于整个团队，在 11 月份的时候，出题终于不用再手写测试代码了，只需要一组一组一个框一个框的去填测试数据了。

3.2.2、出题时批量填测试数据

如果一道题目只有不到 10 组测试数据，那么出题时一个框一个框的去填测试数据是没什么问题的。但问题是，不存在某一道题的测试数据少于 10 组的情况，多的话几百组，甚至上千组，因此，我们很快发现必须支持批量填测试数据，于是到了今年 12 月底，出题系统改造成了如下图所示：

3.3.3、后续的改进、优化

当然，直到现在，出题系统还有很多需要改进、优化的地方，如需支持数组，支持多行输入对应多行输出，直到最终的完全 OJ 模式，这些请待本文后续更新。团队还有很多事情要做，但我们一直在努力，在整个团队的推动下，Hero 也一直在前进，从未退步。

后记

感谢大家关注本编程艺术系列，欢迎大家继续关注或贡献编程艺术 github：https://github.com/julycoding/The-Art-Of-Programming-by-July，也希望各位继续支持 Hero。有何问题，欢迎随时向我反馈，本文完。s

面试题：分别使用 Hadoop MapReduce、hive 统计手机流量

问题导读

1.hive 实现统计的查询语句是什么？

2.生产环境中为什么建议使用外部表？

3.hadoop mapreduce 创建类 DataWritable 的作用是什么？

4.为什么创建类 DataWritable？

5.如何实现统计手机流量？

6.对比 hive 与 mapreduce 统计手机流量的区别？

1.使用 Hive 进行手机流量统计

很多公司在使用 hive 对数据进行处理。

hive 是 hadoop 家族成员，是一种解析 like sql 语句的框架。它封装了常用 MapReduce 任务，让你像执行 sql 一样操作存储在 HDFS 的表。

hive 的表分为两种，内表和外表。

Hive 创建内部表时，会将数据移动到数据仓库指向的路径；若创建外部表，仅记录数据所在的路径，不对数据的位置做任何改变。

在删除表的时候，内部表的元数据和数据会被一起删除， 而外部表只删除元数据，不删除数据。这样外部表相对来说更加安全些，数据组织也更加灵活，方便共享源数据。

Hive 的内外表，还有一个 Partition 的分区的知识点，用于避免全表扫描，快速检索。后期的文章会提到。

接下来开始正式开始《Hive 统计手机流量》

原始数据：

```
1363157985066    13726230503    00-FD-07-A4-72-B8:CMCC  120.196.100.82  i02.c.aliimg.com
       24       27       2481     24681    200
1363157995052    13826544101    5C-0E-8B-C7-F1-E0:CMCC  120.197.40.4                    4        0
     264      0        200
1363157991076    13926435656    20-10-7A-28-CC-0A:CMCC  120.196.100.99                  2        4
      132      1512     200
1363154400022    13926251106    5C-0E-8B-8B-B1-50:CMCC  120.197.40.4                    4        0
     240      0        200
1363157993044    18211575961    94-71-AC-CD-E6-18:CMCC-EASY     120.196.100.99  iface.qiyi.com
 瑙.?缃..        15    2       1527     2106     200
1363157995074    84138413       5C-0E-8B-8C-E8-20:7DaysInn      120.197.40.4    122.72.52.12
       20       16       4116     1432     200
1363157993055    13560439658    C4-17-FE-BA-DE-D9:CMCC  120.196.100.99                  18       1
5       1116     954      200
1363157995033    15920133257    5C-0E-8B-C7-BA-20:CMCC  120.197.40.4    sug.so.360.cn   淇℃.
瀹..        20       20    156      2936     200
```

操作步骤：

```
1.  #配置好 Hive 之后，使用 hive 命令启动 hive 框架。hive 启动属于懒加载模式，会比较慢
2.  hive;
3.  #使用 show databases 命令查看当前数据库信息
4.  hive> show databases;
5.  OK
6.  default
7.  hive
8.  Time taken: 3.389 seconds
9.  #使用 use hive 命令，使用指定的数据库 hive 数据库是我之前创建的
10. use hive;
11. #创建表，这里是创建内表。内表加载 hdfs 上的数据，会将被加载文件中的内容剪切走。
12. #外表没有这个问题，所以在实际的生产环境中，建议使用外表。
13. create table ll(reportTime string,msisdn string,apmac string,acmac string,host
    string,siteType string,upPackNum bigint,downPackNum bigint,upPayLoad bigint,downPayLoad
    bigint,httpStatus string)row format delimited fields terminated by '\t';
14. #加载数据，这里是从 hdfs 加载数据，也可用 linux 下加载数据 需要 local 关键字
15. load data inpath'/HTTP_20130313143750.dat' into table ll;
```

```
16. #数据加载完毕之后，hdfs 的
17. #执行 hive 的 like sql 语句,对数据进行统计
18. select msisdn,sum(uppacknum),sum(downpacknum),sum(uppayload),sum(downpayload) from ll
    group by msisdn;
```

执行结果如下：

```
1.  hive> select msisdn,sum(uppacknum),sum(downpacknum),sum(uppayload),sum(downpayload)
    from ll group by msisdn;
2.  Total MapReduce jobs = 1
3.  Launching Job 1 out of 1
4.  Number of reduce tasks not specified. Estimated from input data size: 1
5.  In order to change the average load for a reducer (in bytes):
6.    set hive.exec.reducers.bytes.per.reducer=<number>
7.  In order to limit the maximum number of reducers:
8.    set hive.exec.reducers.max=<number>
9.  In order to set a constant number of reducers:
10.   set mapred.reduce.tasks=<number>
11. Starting Job = job_201307160252_0006, Tracking URL =
    http://hadoop0:50030/jobdetails.jsp?jobid=job_201307160252_0006
12. Kill Command = /usr/local/hadoop/libexec/../bin/hadoop
    job  -Dmapred.job.tracker=hadoop0:9001 -kill job_201307160252_0006
13. Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1
14. 2013-07-17 19:51:42,599 Stage-1 map = 0%,  reduce = 0%
15. 2013-07-17 19:52:40,474 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 48.5 sec
16. 2013-07-17 19:52:41,690 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 48.5 sec
17. 2013-07-17 19:52:42,693 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 48.5 sec
18. 2013-07-17 19:52:43,698 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 48.5 sec
19. 2013-07-17 19:52:44,702 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 48.5 sec
20. 2013-07-17 19:52:45,707 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 48.5 sec
21. 2013-07-17 19:52:46,712 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 48.5 sec
22. 2013-07-17 19:52:47,715 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 48.5 sec
23. 2013-07-17 19:52:48,721 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 48.5 sec
```

```
2013-07-17 19:52:49,758 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 48.5 sec
2013-07-17 19:52:50,763 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 48.5 sec
2013-07-17 19:52:51,772 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 50.0 sec
2013-07-17 19:52:52,775 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 50.0 sec
2013-07-17 19:52:53,779 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 50.0 sec
MapReduce Total cumulative CPU time: 50 seconds 0 msec
Ended Job = job_201307160252_0006
MapReduce Jobs Launched:
Job 0: Map: 1  Reduce: 1   Cumulative CPU: 50.0 sec   HDFS Read: 2787075 HDFS Write: 16518
    SUCCESS
Total MapReduce CPU Time Spent: 50 seconds 0 msec
OK
13402169727        171        108        11286        130230
13415807477        2067        1683        169668        1994181
13416127574        1501        1094        161963        802756
13416171820        113        99        10630        32120
13417106524        160        128        18688        13088
13418002498        240        256        22136        86896
13418090588        456        351        98934        67470
13418117364        264        152        29436        49966
13418173218        37680        48348        2261286        73159722
13418666750        22432        26482        1395648        39735552
13420637670        20        20        1480        1480
......
Time taken: 75.24 seconds
```

2.Hadoop MapReduce 手机流量统计

自定义一个 writable

```
1.  package cn.maoxiangyi.hadoop.wordcount;
2.
3.  import java.io.DataInput;
4.  import java.io.DataOutput;
5.  import java.io.IOException;
6.
7.  import org.apache.hadoop.io.Writable;
8.
9.  public class DataWritable implements Writable {
10.
11.         private int upPackNum;
12.         private int downPackNum;
13.         private int upPayLoad;
14.         private int downPayLoad;
15.
16.         public DataWritable() {
17.                 super();
18.         }
19.
20.         public DataWritable(int upPackNum, int downPackNum, int upPayLoad,
21.                         int downPayLoad) {
22.                 super();
23.                 this.upPackNum = upPackNum;
24.                 this.downPackNum = downPackNum;
25.                 this.upPayLoad = upPayLoad;
26.                 this.downPayLoad = downPayLoad;
27.         }
28.
29.         @Override
```

```
30.         public void write(DataOutput out) throws IOException {
31.                 out.writeInt(upPackNum);
32.                 out.writeInt(downPackNum);
33.                 out.writeInt(upPayLoad);
34.                 out.writeInt(downPayLoad);
35.         }
36.
37.         @Override
38.         public void readFields(DataInput in) throws IOException {
39.                 upPackNum = in.readInt();
40.                 downPackNum = in.readInt();
41.                 upPayLoad = in.readInt();
42.                 downPayLoad =in.readInt();
43.         }
44.
45.         public int getUpPackNum() {
46.                 return upPackNum;
47.         }
48.
49.         public void setUpPackNum(int upPackNum) {
50.                 this.upPackNum = upPackNum;
51.         }
52.
53.         public int getDownPackNum() {
54.                 return downPackNum;
55.         }
56.
57.         public void setDownPackNum(int downPackNum) {
58.                 this.downPackNum = downPackNum;
59.         }
60.
```

```
61.         public int getUpPayLoad() {
62.                 return upPayLoad;
63.         }
64.
65.         public void setUpPayLoad(int upPayLoad) {
66.                 this.upPayLoad = upPayLoad;
67.         }
68.
69.         public int getDownPayLoad() {
70.                 return downPayLoad;
71.         }
72.
73.         public void setDownPayLoad(int downPayLoad) {
74.                 this.downPayLoad = downPayLoad;
75.         }
76.
77.         @Override
78.         public String toString() {
79.                 return "        " + upPackNum + "        "
80.                                 + downPackNum + "        " + upPayLoad + "        "
81.                                 + downPayLoad;
82.         }
83.
84.
85. }
```

MapReduc 函数

```
1.  package cn.maoxiangyi.hadoop.wordcount;
2.
3.  import java.io.IOException;
4.
5.  import org.apache.hadoop.conf.Configuration;
```

```
6.  import org.apache.hadoop.fs.Path;
7.  import org.apache.hadoop.io.IntWritable;
8.  import org.apache.hadoop.io.LongWritable;
9.  import org.apache.hadoop.io.Text;
10. import org.apache.hadoop.mapreduce.Job;
11. import org.apache.hadoop.mapreduce.Mapper;
12. import org.apache.hadoop.mapreduce.Reducer;
13. import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
14. import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
15.
16. public class DataTotalMapReduce {
17.
18.         public static void main(String[] args) throws Exception {
19.                 Configuration configuration = new Configuration();
20.                 Job job = new Job(configuration);
21.                 job.setJarByClass(DataTotalMapReduce.class);
22.                 job.setMapperClass(DataTotalMapper.class);
23.                 job.setReducerClass(DataTotalReducer.class);
24.                 job.setOutputKeyClass(Text.class);
25.                 job.setOutputValueClass(DataWritable.class);
26.                 job.setCombinerClass(DataTotalReducer.class);
27.                 Path inputDir = new Path("hdfs://hadoop0:9000/HTTP_20130313143750.dat");
28.                 FileInputFormat.addInputPath(job, inputDir);
29.                 Path outputDir = new Path("hdfs://hadoop0:9000/dataTotal");
30.                 FileOutputFormat.setOutputPath(job, outputDir);
31.                 job.waitForCompletion(true);
32.         }
33.
34. }
35.
36. /**
```

```
37. *
38. *
39. *
40. * 1363157985066 13726230503 00-FD-07-A4-72-B8:CMCC 120.196.100.82
41. * i02.c.aliimg.com 24 27 2481 24681 200 1363157995052 13826544101
42. * 5C-0E-8B-C7-F1-E0:CMCC 120.197.40.4 4 0 264 0 200 1363157991076 13926435656
43. * 20-10-7A-28-CC-0A:CMCC 120.196.100.99 2 4 132 1512 200
44. *
45. *
46. */
47.
48. class DataTotalMapper extends Mapper<LongWritable, Text, Text, DataWritable> {
49.
50.         @Override
51.         protected void map(LongWritable key, Text value, Context context)
52.                         throws IOException, InterruptedException {
53.                 String lineStr = value.toString();
54.                 String[] strArr = lineStr.split("\t");
55.                 String phpone = strArr[1];
56.                 String upPackNum = strArr[6];
57.                 String downPackNum = strArr[7];
58.                 String upPayLoad = strArr[8];
59.                 String downPayLoad = strArr[9];
60.                 context.write(
61.                                 new Text(phpone),
62.                                 new DataWritable(Integer.parseInt(upPackNum), Integer
63.                                                 .parseInt(downPackNum),
    Integer.parseInt(upPayLoad),
64.                                                 Integer.parseInt(downPayLoad)));
65.         }
66.
```

```
67. }
68.
69. class DataTotalReducer extends Reducer<Text, DataWritable, Text, DataWritable> {
70.
71.         @Override
72.         protected void reduce(Text k2, Iterable<DataWritable> v2, Context context)
73.                         throws IOException, InterruptedException {
74.                 int upPackNumSum = 0;
75.                 int downPackNumSum = 0;
76.                 int upPayLoadSum = 0;
77.                 int downPayLoadSum = 0;
78.                 for (DataWritable dataWritable : v2) {
79.                         upPackNumSum += dataWritable.getUpPackNum();
80.                         downPackNumSum += dataWritable.getDownPackNum();
81.                         upPayLoadSum += dataWritable.getUpPayLoad();
82.                         downPayLoadSum += dataWritable.getDownPayLoad();
83.                 }
84.                 context.write(k2, new DataWritable(upPackNumSum, downPackNumSum,
    upPayLoadSum, downPayLoadSum));
85.         }
86.
87. }
```

结果节选

```
1.  13402169727            171     108     11286   130230
2.  13415807477            2067    1683    169668  1994181
3.  13416127574            1501    1094    161963  802756
4.  13416171820            113     99      10630   32120
5.  13417106524            160     128     18688   13088
6.  13418002498            240     256     22136   86896
7.  13418090588            456     351     98934   67470
```

```
8.  13418117364            264     152     29436   49966
9.  13418173218            37680   48348   2261286 73159722
10. 13418666750            22432   26482   1395648 39735552
11. 13420637670            20      20      1480    1480
12. 13422149173            40      32      4000    3704
13. 13422311151            465     535     33050   661790
14. 13424077835            84      72      15612   9948
15. 13424084200            765     690     60930   765675
16. 13428887537            43892   44830   2925330 65047620
17. 13430219372            454     352     33792   192876
18. 13430234524            27852   39056   1767220 52076614
19. 13430237899            1293    1165    166346  808613
20. 13430258776            4681    4783    350511  6609423
21. 13430266620            10544   9377    11600817        5728002
22. 13432023893            40      0       2400    0
```

十道大数据的题与十个海量数据处理的方法

问题导读

1、海量数据处理的常用方法有哪些？

2、大数据的问题主要有哪些？

3、双层桶划分的思想是什么？

第一部分、十道海量数据处理面试题

1、海量日志数据，提取出某日访问百度次数最多的那个 IP。

此题，在我之前的一篇文章算法里头有所提到，当时给出的方案是：IP 的数目还是有限的，最多 2^32 个，所以可以考虑使用 hash 将 ip 直接存入内存，然后进行统计。

再详细介绍下此方案：首先是这一天，并且是访问百度的日志中的 IP 取出来，逐个写入到一个大文件中。注意到 IP 是 32 位的，最多有个 2^32 个 IP。同样可以采用映射的方法，比如模 1000，把整个大文件映射为 1000 个小文件，再找出每个小文中出现频率最大的 IP（可以采用 hash_map 进行频率统计，然后再找出频率最大的几个）及相应的频率。然后再在这 1000 个最大的 IP 中，找出那个频率最大的 IP，即为所求。

2、搜索引擎会通过日志文件把用户每次检索使用的所有检索串都记录下来，每个查询串的长度为 1-255 字节。

假设目前有一千万个记录（这些查询串的重复度比较高，虽然总数是 1 千万，但如果除去重复后，不超过 3 百万个。一个查询串的重复度越高，说明查询它的用户越多，也就是越热门。），请你统计最热门的 10 个查询串，要求使用的内存不能超过 1G。

典型的 Top K 算法，还是在这篇文章里头有所阐述。 文中，给出的最终算法是：第一步、先对这批海量数据预处理，在 O（N）的时间内用 Hash 表完成排序；然后，第二步、借助堆这个数据结构，找出 Top K，时间复杂度为 N‘logK。 即，借助堆结构，我们可以在 log 量级的时间内查找和调整/移动。因此，维护一个 K(该题目中是 10)大小的小根堆，然后遍历 300 万的 Query，分别和根元素进行对比所以，我们最终的时间复杂度是：O（N） + N'*O（logK），（N 为 1000 万，N’为 300 万）。ok，更多，详情，请参考原文。

或者：采用 trie 树，关键字域存该查询串出现的次数，没有出现为 0。最后用 10 个元素的最小推来对出现频率进行排序。

3、有一个 1G 大小的一个文件，里面每一行是一个词，词的大小不超过 16 字节，内存限制大小是 1M。返回频数最高的 100 个词。

方案：顺序读文件中，对于每个词 x，取 hash(x)%5000，然后按照该值存到 5000 个小文件（记为

x0,x1,...x4999）中。这样每个文件大概是 200k 左右。

如果其中的有的文件超过了 1M 大小，还可以按照类似的方法继续往下分，直到分解得到的小文件的大小都不超过 1M。 对每个小文件，统计每个文件中出现的词以及相应的频率（可以采用 trie 树/hash_map 等），并取出出现频率最大的 100 个词（可以用含 100 个结点的最小堆），并把 100 个词及相应的频率存入文件，这样又得到了 5000 个文件。下一步就是把这 5000 个文件进行归并（类似与归并排序）的过程了。

4、有 10 个文件，每个文件 1G，每个文件的每一行存放的都是用户的 query，每个文件的 query 都可能重复。要求你按照 query 的频度排序。

还是典型的 TOP K 算法，解决方案如下： 方案 1： 顺序读取 10 个文件，按照 hash(query)%10 的结果将 query 写入到另外 10 个文件（记为）中。这样新生成的文件每个的大小大约也 1G（假设 hash 函数是随机的）。 找一台内存在 2G 左右的机器，依次对用 hash_map(query, query_count)来统计每个 query 出现的次数。利用快速/堆/归并排序按照出现次数进行排序。将排序好的 query 和对应的 query_cout 输出到文件中。这样得到了 10 个排好序的文件（记为）。

对这 10 个文件进行归并排序（内排序与外排序相结合）。

方案 2： 一般 query 的总量是有限的，只是重复的次数比较多而已，可能对于所有的 query，一次性就可以加入到内存了。这样，我们就可以采用 trie 树/hash_map 等直接来统计每个 query 出现的次数，然后按出现次数做快速/堆/归并排序就可以了。

方案 3： 与方案 1 类似，但在做完 hash，分成多个文件后，可以交给多个文件来处理，采用分布式的架构来处理（比如 MapReduce），最后再进行合并。

5、 给定 a、b 两个文件，各存放 50 亿个 url，每个 url 各占 64 字节，内存限制是 4G，让你找出 a、b 文件共同的 url？

方案 1：可以估计每个文件安的大小为 5G×64=320G，远远大于内存限制的 4G。所以不可能将其完全加载到内存中处理。考虑采取分而治之的方法。

遍历文件 a，对每个 url 求取 hash(url)%1000，然后根据所取得的值将 url 分别存储到 1000 个小文件（记为 a0,a1,...,a999）中。这样每个小文件的大约为 300M。

遍历文件 b，采取和 a 相同的方式将 url 分别存储到 1000 小文件（记为 b0,b1,...,b999）。这样处理后，所有可能相同的 url 都在对应的小文件（a0vsb0,a1vsb1,...,a999vsb999）中，不对应的小文件不可能有相

同的 url。然后我们只要求出 1000 对小文件中相同的 url 即可。

求每对小文件中相同的 url 时，可以把其中一个小文件的 url 存储到 hash_set 中。然后遍历另一个小文件的每个 url，看其是否在刚才构建的 hash_set 中，如果是，那么就是共同的 url，存到文件里面就可以了。

方案 2：如果允许有一定的错误率，可以使用 Bloom filter，4G 内存大概可以表示 340 亿 bit。将其中一个文件中的 url 使用 Bloom filter 映射为这 340 亿 bit，然后挨个读取另外一个文件的 url，检查是否与 Bloom filter，如果是，那么该 url 应该是共同的 url（注意会有一定的错误率）。

Bloom filter 日后会在本 BLOG 内详细阐述。

6、在 2.5 亿个整数中找出不重复的整数，注，内存不足以容纳这 2.5 亿个整数。

方案 1：采用 2-Bitmap（每个数分配 2bit，00 表示不存在，01 表示出现一次，10 表示多次，11 无意义）进行，共需内存内存，还可以接受。然后扫描这 2.5 亿个整数，查看 Bitmap 中相对应位，如果是 00 变 01，01 变 10，10 保持不变。所描完事后，查看 bitmap，把对应位是 01 的整数输出即可。

方案 2：也可采用与第 1 题类似的方法，进行划分小文件的方法。然后在小文件中找出不重复的整数，并排序。然后再进行归并，注意去除重复的元素。

7、腾讯面试题：给 40 亿个不重复的 unsigned int 的整数，没排过序的，然后再给一个数，如何快速判断这个数是否在那 40 亿个数当中？

与上第 6 题类似，我的第一反应时快速排序+二分查找。以下是其它更好的方法： 方案 1：oo，申请 512M 的内存，一个 bit 位代表一个 unsigned int 值。读入 40 亿个数，设置相应的 bit 位，读入要查询的数，查看相应 bit 位是否为 1，为 1 表示存在，为 0 表示不存在。

dizengrong： 方案 2：这个问题在《编程珠玑》里有很好的描述，大家可以参考下面的思路，探讨一下：又因为 2^32 为 40 亿多，所以给定一个数可能在，也可能不在其中；这里我们把 40 亿个数中的每一个用 32 位的二进制来表示假设这 40 亿个数开始放在一个文件中。

然后将这 40 亿个数分成两类: 1.最高位为 0 2.最高位为 1 并将这两类分别写入到两个文件中，其中一个文件中数的个数<=20 亿，而另一个>=20 亿（这相当于折半了）；与要查找的数的最高位比较并接着进入相应的文件再查找

再然后把这个文件为又分成两类: 1.次最高位为 0 2.次最高位为 1

并将这两类分别写入到两个文件中，其中一个文件中数的个数<=10 亿，而另一个>=10 亿（这相当于折半了）； 与要查找的数的次最高位比较并接着进入相应的文件再查找。 ..... 以此类推，就可以找到了，而且时间复杂度为 O(logn)，方案 2 完。

附：这里，再简单介绍下，位图方法： 使用位图法判断整形数组是否存在重复 判断集合中存在重复是常见编程任务之一，当集合中数据量比较大时我们通常希望少进行几次扫描，这时双重循环法就不可取了。

位图法比较适合于这种情况，它的做法是按照集合中最大元素 max 创建一个长度为 max+1 的新数组，然后再次扫描原数组，遇到几就给新数组的第几位置上 1，如遇到 5 就给新数组的第六个元素置 1，这样下次再遇到 5 想置位时发现新数组的第六个元素已经是 1 了，这说明这次的数据肯定和以前的数据存在着重复。这种给新数组初始化时置零其后置一的做法类似于位图的处理方法故称位图法。它的运算次数最坏的情况为 2N。如果已知数组的最大值即能事先给新数组定长的话效率还能提高一倍。

8、怎么在海量数据中找出重复次数最多的一个？

方案 1：先做 hash，然后求模映射为小文件，求出每个小文件中重复次数最多的一个，并记录重复次数。然后找出上一步求出的数据中重复次数最多的一个就是所求（具体参考前面的题）。

9、上千万或上亿数据（有重复），统计其中出现次数最多的钱 N 个数据。

方案 1：上千万或上亿的数据，现在的机器的内存应该能存下。所以考虑采用 hash_map/搜索二叉树/红黑树等来进行统计次数。然后就是取出前 N 个出现次数最多的数据了，可以用第 2 题提到的堆机制完成。

10、一个文本文件，大约有一万行，每行一个词，要求统计出其中最频繁出现的前 10 个词，请给出思想，给出时间复杂度分析。

方案 1：这题是考虑时间效率。用 trie 树统计每个词出现的次数，时间复杂度是 O(n*le)（le 表示单词的平准长度）。然后是找出出现最频繁的前 10 个词，可以用堆来实现，前面的题中已经讲到了，时间复杂度是 O(n*lg10)。所以总的时间复杂度，是 O(n*le)与 O(n*lg10)中较大的哪一个。

附、100w 个数中找出最大的 100 个数。

方案 1：在前面的题中，我们已经提到了，用一个含 100 个元素的最小堆完成。复杂度为 O(100w*lg100)。

方案 2：采用快速排序的思想，每次分割之后只考虑比轴大的一部分，知道比轴大的一部分在比 100 多的时候，采用传统排序算法排序，取前 100 个。复杂度为 O(100w*100)。

方案 3：采用局部淘汰法。选取前 100 个元素，并排序，记为序列 L。然后一次扫描剩余的元素 x，与排好序的 100 个元素中最小的元素比，如果比这个最小的要大，那么把这个最小的元素删除，并把 x 利用插入排序的思想，插入到序列 L 中。依次循环，知道扫描了所有的元素。复杂度为 O(100w*100)。

第二部分、十个海量数据处理方法大总结

ok，看了上面这么多的面试题，是否有点头晕。是的，需要一个总结。接下来，本文将简单总结下一些处理海量数据问题的常见方法。

下面的方法全部来自 http://hi.baidu.com/yanxionglu/blog/博客，对海量数据的处理方法进行了一个一般性的总结，当然这些方法可能并不能完全覆盖所有的问题，但是这样的一些方法也基本可以处理绝大多数遇到的问题。下面的一些问题基本直接来源于公司的面试笔试题目，方法不一定最优，如果你有更好的处理方法，欢迎讨论。

一、Bloom filter

适用范围：可以用来实现数据字典，进行数据的判重，或者集合求交集

基本原理及要点：

对于原理来说很简单，位数组+k 个独立 hash 函数。将 hash 函数对应的值的位数组置 1，查找时如果发现所有 hash 函数对应位都是 1 说明存在，很明显这个过程并不保证查找的结果是 100%正确的。同时也不支持删除一个已经插入的关键字，因为该关键字对应的位会牵动到其他的关键字。所以一个简单的改进就是 counting Bloom filter，用一个 counter 数组代替位数组，就可以支持删除了。

还有一个比较重要的问题，如何根据输入元素个数 n，确定位数组 m 的大小及 hash 函数个数。当 hash 函数个数 k=(ln2)*(m/n)时错误率最小。在错误率不大于 E 的情况下，m 至少要等于 n*lg(1/E)才能表示任意 n 个元素的集合。但 m 还应该更大些，因为还要保证 bit 数组里至少一半为 0，则 m 应该>=nlg(1/E)*lge 大概就是 nlg(1/E)1.44 倍(lg 表示以 2 为底的对数)。

举个例子我们假设错误率为 0.01，则此时 m 应大概是 n 的 13 倍。这样 k 大概是 8 个。

注意这里 m 与 n 的单位不同，m 是 bit 为单位，而 n 则是以元素个数为单位(准确的说是不同元素的个数)。通常单个元素的长度都是有很多 bit 的。所以使用 bloom filter 内存上通常都是节省的。

扩展：

Bloom filter 将集合中的元素映射到位数组中，用 k（k 为哈希函数个数）个映射位是否全 1 表示元素在不在这个集合中。Counting bloom filter（CBF）将位数组中的每一位扩展为一个 counter，从而支持了元素的删除操作。Spectral Bloom Filter（SBF）将其与集合元素的出现次数关联。SBF 采用 counter 中的最小值来近似表示元素的出现频率。

问题实例：

给你 A,B 两个文件，各存放 50 亿条 URL，每条 URL 占用 64 字节，内存限制是 4G，让你找出 A,B 文件共同的 URL。如果是三个乃至 n 个文件呢？

根据这个问题我们来计算下内存的占用，4G=2^32 大概是 40 亿*8 大概是 340 亿，n=50 亿，如果按出错率 0.01 算需要的大概是 650 亿个 bit。现在可用的是 340 亿，相差并不多，这样可能会使出错率上升些。另外如果这些 urlip 是一一对应的，就可以转换成 ip，则大大简单了。

二、Hashing

适用范围：快速查找，删除的基本数据结构，通常需要总数据量可以放入内存

基本原理及要点：

hash 函数选择，针对字符串，整数，排列，具体相应的 hash 方法。

碰撞处理，一种是 open hashing，也称为拉链法；另一种就是 closed hashing，也称开地址法，opened addressing。

扩展：

d-left hashing 中的 d 是多个的意思，我们先简化这个问题，看一看 2-left hashing。2-left hashing 指的是将一个哈希表分成长度相等的两半，分别叫做 T1 和 T2，给 T1 和 T2 分别配备一个哈希函数，h1 和 h2。在存储一个新的 key 时，同时用两个哈希函数进行计算，得出两个地址 h1[key]和 h2[key]。这时需要检查 T1 中的 h1[key]位置和 T2 中的 h2[key]位置，哪一个位置已经存储的（有碰撞的）key 比较多，然后将新 key 存储在负载少的位置。如果两边一样多，比如两个位置都为空或者都存储了一个 key，就把新 key 存储在左边的 T1 子表中，2-left 也由此而来。在查找一个 key 时，必须进行两次 hash，同时查找两个位置。

问题实例：

1).海量日志数据，提取出某日访问百度次数最多的那个 IP。

IP 的数目还是有限的，最多 2^32 个，所以可以考虑使用 hash 将 ip 直接存入内存，然后进行统计。

三、bit-map

适用范围：可进行数据的快速查找，判重，删除，一般来说数据范围是 int 的 10 倍以下

基本原理及要点：使用 bit 数组来表示某些元素是否存在，比如 8 位电话号码

扩展：

bloom filter 可以看做是对 bit-map 的扩展

问题实例：

1)已知某个文件内包含一些电话号码，每个号码为 8 位数字，统计不同号码的个数。

8 位最多 99 999 999，大概需要 99m 个 bit，大概 10 几 m 字节的内存即可。

2)2.5 亿个整数中找出不重复的整数的个数，内存空间不足以容纳这 2.5 亿个整数。

将 bit-map 扩展一下，用 2bit 表示一个数即可，0 表示未出现，1 表示出现一次，2 表示出现 2 次及以上。或者我们不用 2bit 来进行表示，我们用两个 bit-map 即可模拟实现这个 2bit-map。

四、堆

适用范围：海量数据前 n 大，并且 n 比较小，堆可以放入内存

基本原理及要点：最大堆求前 n 小，最小堆求前 n 大。方法，比如求前 n 小，我们比较当前元素与最大堆里的最大元素，如果它小于最大元素，则应该替换那个最大元素。这样最后得到的 n 个元素就是最小的 n 个。适合大数据量，求前 n 小，n 的大小比较小的情况，这样可以扫描一遍即可得到所有的前 n 元素，效率很高。

扩展：双堆，一个最大堆与一个最小堆结合，可以用来维护中位数。

问题实例：

1)100w 个数中找最大的前 100 个数。

用一个 100 个元素大小的最小堆即可。

五、双层桶划分----其实本质上就是【分而治之】的思想，重在分的技巧上！

适用范围：第 k 大，中位数，不重复或重复的数字

基本原理及要点：因为元素范围很大，不能利用直接寻址表，所以通过多次划分，逐步确定范围，然后最后在一个可以接受的范围内进行。可以通过多次缩小，双层只是一个例子。

扩展：

问题实例：

1).2.5 亿个整数中找出不重复的整数的个数，内存空间不足以容纳这 2.5 亿个整数。

有点像鸽巢原理，整数个数为 2^32,也就是，我们可以将这 2^32 个数，划分为 2^8 个区域(比如用单个文件代表一个区域)，然后将数据分离到不同的区域，然后不同的区域在利用 bitmap 就可以直接解决了。也就是说只要有足够的磁盘空间，就可以很方便的解决。

2).5 亿个 int 找它们的中位数。

这个例子比上面那个更明显。首先我们将 int 划分为 2^16 个区域，然后读取数据统计落到各个区域里的数的个数，之后我们根据统计结果就可以判断中位数落到那个区域，同时知道这个区域中的第几大数刚好是中位数。然后第二次扫描我们只统计落在这个区域中的那些数就可以了。

实际上，如果不是 int 是 int64，我们可以经过 3 次这样的划分即可降低到可以接受的程度。即可以先将 int64 分成 2^24 个区域，然后确定区域的第几大数，在将该区域分成 2^20 个子区域，然后确定是子区域的第几大数，然后子区域里的数的个数只有 2^20，就可以直接利用 direct addr table 进行统计了。

六、数据库索引

适用范围：大数据量的增删改查

基本原理及要点：利用数据的设计实现方法，对海量数据的增删改查进行处理。

七、倒排索引(Inverted index)

适用范围：搜索引擎，关键字查询

基本原理及要点：为何叫倒排索引？一种索引方法，被用来存储在全文搜索下某个单词在一个文档或者一组文档中的存储位置的映射。

以英文为例，下面是要被索引的文本： T0 = "it is what it is" T1 = "what is it" T2 = "it is a banana"

我们就能得到下面的反向文件索引：

"a": {2} "banana": {2} "is": {0, 1, 2} "it": {0, 1, 2} "what": {0, 1}

检索的条件"what","is"和"it"将对应集合的交集。

正向索引开发出来用来存储每个文档的单词的列表。正向索引的查询往往满足每个文档有序频繁的全文查询和每个单词在校验文档中的验证这样的查询。在正向索引中，文档占据了中心的位置，每个文档指向了一个它所包含的索引项的序列。也就是说文档指向了它包含的那些单词，而反向索引则是单词指向了包含它的文档，很容易看到这个反向的关系。

扩展：

问题实例：文档检索系统，查询那些文件包含了某单词，比如常见的学术论文的关键字搜索。

八、外排序

适用范围：大数据的排序，去重

基本原理及要点：外排序的归并方法，置换选择败者树原理，最优归并树

扩展：

问题实例：

1).有一个 1G 大小的一个文件，里面每一行是一个词，词的大小不超过 16 个字节，内存限制大小是 1M。返回频数最高的 100 个词。

这个数据具有很明显的特点，词的大小为 16 个字节，但是内存只有 1m 做 hash 有些不够，所以可以用来排序。内存可以当输入缓冲区使用。

九、trie 树

适用范围：数据量大，重复多，但是数据种类小可以放入内存

基本原理及要点：实现方式，节点孩子的表示方式

扩展：压缩实现。

问题实例：

1).有 10 个文件，每个文件 1G，每个文件的每一行都存放的是用户的 query，每个文件的 query 都可能重复。要你按照 query 的频度排序。

2).1000 万字符串，其中有些是相同的(重复),需要把重复的全部去掉，保留没有重复的字符串。请问怎么设计和实现？

3).寻找热门查询：查询串的重复度比较高，虽然总数是 1 千万，但如果除去重复后，不超过 3 百万个，每个不超过 255 字节。

十、分布式处理 mapreduce

适用范围：数据量大，但是数据种类小可以放入内存

基本原理及要点：将数据交给不同的机器去处理，数据划分，结果归约。

扩展：

问题实例：

1).The canonical example application of MapReduce is a process to count the appearances ofeach different word in a set of documents:

2).海量数据分布在 100 台电脑中，想个办法高效统计出这批数据的 TOP10。

3).一共有 N 个机器，每个机器上有 N 个数。每个机器最多存 O(N)个数并对它们操作。如何找到 N^2 个数的中数(median)？

经典问题分析

上千万 or 亿数据（有重复），统计其中出现次数最多的前 N 个数据,分两种情况：可一次读入内存，不可一次读入。

可用思路：trie 树+堆，数据库索引，划分子集分别统计，hash，分布式计算，近似统计，外排序

所谓的是否能一次读入内存，实际上应该指去除重复后的数据量。如果去重后数据可以放入内存，我们可以为数据建立字典，比如通过 map，hashmap，trie，然后直接进行统计即可。当然在更新每条数据的出现次数的时候,我们可以利用一个堆来维护出现次数最多的前 N 个数据,当然这样导致维护次数增加，不如完全统计后在求前 N 大效率高。

如果数据无法放入内存。一方面我们可以考虑上面的字典方法能否被改进以适应这种情形，可以做的改变就是将字典存放到硬盘上，而不是内存，这可以参考数据库的存储方法。

当然还有更好的方法，就是可以采用分布式计算，基本上就是 map-reduce 过程，首先可以根据数据值或者把数据 hash(md5)后的值，将数据按照范围划分到不同的机子，最好可以让数据划分后可以一次读入内存，这样不同的机子负责处理各种的数值范围，实际上就是 map。得到结果后，各个机子只需拿出各自的出现次数最多的前 N 个数据，然后汇总，选出所有的数据中出现次数最多的前 N 个数据，这实际上就

是 reduce 过程。

实际上可能想直接将数据均分到不同的机子上进行处理，这样是无法得到正确的解的。因为一个数据可能被均分到不同的机子上，而另一个则可能完全聚集到一个机子上，同时还可能存在具有相同数目的数据。比如我们要找出现次数最多的前 100 个，我们将 1000 万的数据分布到 10 台机器上，找到每台出现次数最多的前 100 个，归并之后这样不能保证找到真正的第 100 个，因为比如出现次数最多的第 100 个可能有 1 万个，但是它被分到了 10 台机子，这样在每台上只有 1 千个，假设这些机子排名在 1000 个之前的那些都是单独分布在一台机子上的，比如有 1001 个，这样本来具有 1 万个的这个就会被淘汰，即使我们让每台机子选出出现次数最多的 1000 个再归并，仍然会出错，因为可能存在大量个数为 1001 个的发生聚集。因此不能将数据随便均分到不同机子上，而是要根据 hash 后的值将它们映射到不同的机子上处理，让不同的机器处理一个数值范围。

而外排序的方法会消耗大量的 IO，效率不会很高。而上面的分布式方法，也可以用于单机版本，也就是将总的数据根据值的范围，划分成多个不同的子文件，然后逐个处理。处理完毕之后再对这些单词的及其出现频率进行一个归并。实际上就可以利用一个外排序的归并过程。

另外还可以考虑近似计算，也就是我们可以通过结合自然语言属性，只将那些真正实际中出现最多的那些词作为一个字典，使得这个规模可以放入内存。

hive 面试题目：表大概有 2T 左右，对表数据转换

有一张很大的表：TRLOG

该表大概有 2T 左右

```
TRLOG：

CREATE TABLE TRLOG

(PLATFORM string,

USER_ID int,

CLICK_TIME string,

CLICK_URL string)

row format delimited

fields terminated by '\t';
```

数据：

PLATFORM	USER_ID	CLICK_TIME	CLICK_URL
WEB	12332321	2013-03-21 13:48:31.324	/home/
WEB	12332321	2013-03-21 13:48:32.954	/selectcat/er/
WEB	12332321	2013-03-21 13:48:46.365	/er/viewad/12.html
WEB	12332321	2013-03-21 13:48:53.651	/er/viewad/13.html
WEB	12332321	2013-03-21 13:49:13.435	/er/viewad/24.html
WEB	12332321	2013-03-21 13:49:35.876	/selectcat/che/
WEB	12332321	2013-03-21 13:49:56.398	/che/viewad/93.html
WEB	12332321	2013-03-21 13:50:03.143	/che/viewad/10.html
WEB	12332321	2013-03-21 13:50:34.265	/home/
WAP	32483923	2013-03-21 23:58:41.123	/m/home/
WAP	32483923	2013-03-21 23:59:16.123	/m/selectcat/fang/
WAP	32483923	2013-03-21 23:59:45.123	/m/fang/33.html
WAP	32483923	2013-03-22 00:00:23.984	/m/fang/54.html
WAP	32483923	2013-03-22 00:00:54.043	/m/selectcat/er/
WAP	32483923	2013-03-22 00:01:16.576	/m/er/49.html
……	……	……	……

需要把上述数据处理为如下结构的表 ALLOG：

```
CREATE TABLE ALLOG

(PLATFORM string,

USER_ID int,

SEQ int,

FROM_URL string,

TO_URL string)

row format delimited

fields terminated by '\t';
```

整理后的数据结构：

```
PLATFORM        USER_ID        SEQ        FROM_URL        TO_URL

WEB        12332321        1        NULL        /home/

WEB        12332321        2        /home/        /selectcat/er/
```

WEB	12332321	3	/selectcat/er/	/er/viewad/12.html
WEB	12332321	4	/er/viewad/12.html	/er/viewad/13.html
WEB	12332321	5	/er/viewad/13.html	/er/viewad/24.html
WEB	12332321	6	/er/viewad/24.html	/selectcat/che/
WEB	12332321	7	/selectcat/che/	/che/viewad/93.html
WEB	12332321	8	/che/viewad/93.html	/che/viewad/10.html
WEB	12332321	9	/che/viewad/10.html	/home/
WAP	32483923	1	NULL	/m/home/
WAP	32483923	2	/m/home/	/m/selectcat/fang/
WAP	32483923	3	/m/selectcat/fang/	/m/fang/33.html
WAP	32483923	4	/m/fang/33.html	/m/fang/54.html
WAP	32483923	5	/m/fang/54.html	/m/selectcat/er/
WAP	32483923	6	/m/selectcat/er/	/m/er/49.html
.....	.....	.....	.....	

PLATFORM 和 USER_ID 还是代表平台和用户 ID；SEQ 字段代表用户按时间排序后的访问顺序，FROM_URL 和 TO_URL 分别代表用户从哪一页跳转到哪一页。对于某个平台上某个用户的第一条访问记录，其

FROM_URL 是 NULL（空值）。

面试官说需要用两种办法做出来：

1、实现一个能加速上述处理过程的 Hive Generic UDF，并给出使用此 UDF 实现 ETL 过程的 Hive SQL

2、实现基于纯 Hive SQL 的 ETL 过程，从 TRLOG 表生成 ALLOG 表；（结果是一套 SQL）

给你个 JAVA 写的 RowNumber 方法

```
1.
2.  public class RowNumber extends org.apache.hadoop.hive.ql.exec.UDF {
3.
4.          private static int MAX_VALUE = 50;
5.          private static String comparedColumn[] = new String[MAX_VALUE];
6.          private static int rowNum = 1;
7.
8.          public int evaluate(Object... args) {
9.                  String columnValue[] = new String[args.length];
10.                 for (int i = 0; i < args.length; i++)
11.                         columnValue[i] = args[i].toString();
12.                 if (rowNum == 1)
13.                 {
14.
15.                         for (int i = 0; i < columnValue.length; i++)
16.                                 comparedColumn[i] = columnValue[i];
17.                 }
```

```
18.
19.                 for (int i = 0; i < columnValue.length; i++)
20.                 {
21.
22.                         if (!comparedColumn[i].equals(columnValue[i]))
23.                         {
24.                                 for (int j = 0; j < columnValue.length; j++)
25.                                 {
26.                                         comparedColumn[j] = columnValue[j];
27.                                 }
28.                                 rowNum = 1;
29.                                 return rowNum++;
30.                         }
31.                 }
32.                 return rowNum++;
33.         }
34. }
35.
```

把这个 JAVA 打包，编译成 JAR 包，比如 RowNumber.jar。这个你总会吧~~~
然后放到 HIVE 的机器上
在 HIVE SHELL 里执行下面两条语句：

```
1.  add jar /root/RowNumber.jar;
2.  #把 RowNumber.jar 加载到 HIVE 的 CLASSPATH 中
3.  create temporary function row_number as 'RowNumber';
4.  #在 HIVE 里创建一个新函数，叫 row_number ，引用的 CLASS 就是 JAVA 代码里的 RowNumber
5.
```

提示成功后，执行下面这条 HIVE SQL

```
1.
```

```
2.  #INSERT OVERWRITE TABLE ALLOG 如果要写入 ALLOG 表，可以把注释去掉
3.  SELECT t1.platform,t1.user_id,row_number(t1.user_id)seq,t2.click_url
    FROM_URL,t1.click_url TO_URL FROM
4.  (select *,row_number(user_id)seq from trlog)t1
5.  LEFT OUTER JOIN
6.  (select *,row_number(user_id)seq from trlog)t2
7.  on t1.user_id = t2.user_id and t1.seq = t2.seq + 1;
8.
```

第一题中的 RN 貌似是 HIVE 转译 SQL 的 BUG，你可以把外层的 ROW_NUMBER 去掉，用 T1 的 SEQ，就能发现问题了。

第二题：

```
1.
2.  INSERT OVERWRITE TABLE ALLOG
3.  SELECT t1.platform,t1.user_id,t1.seq,t2.click_url FROM_URL,t1.click_url TO_URL FROM
4.  (SELECT platform,user_id,click_time,click_url,count(1) seq FROM (SELECT a.*,b.click_time
    click_time1,b.click_url click_url2  FROM trlog a left outer join trlog b on a.user_id =
    b.user_id)t WHERE click_time>=click_time1 GROUP BY
    platform,user_id,click_time,click_url)t1
5.  LEFT OUTER JOIN
6.  (SELECT platform,user_id,click_time,click_url,count(1) seq FROM (SELECT a.*,b.click_time
    click_time1,b.click_url click_url2  FROM trlog a left outer join trlog b on a.user_id =
    b.user_id)t WHERE click_time>=click_time1 GROUP BY
    platform,user_id,click_time,click_url )t2
7.  on t1.user_id = t2.user_id and t1.seq = t2.seq + 1;
8.
```

腾讯面试题：有一千万条短信，有重复，以文本文件的形式保存，一行一条，有重复

有一千万条短信，有重复，以文本文件的形式保存，一行一条，有重复。

请用 5 分钟时间，找出重复出现最多的前 10 条。

分析：

常规方法是先排序，在遍历一次，找出重复最多的前 10 条。但是排序的算法复杂度最低为 nlgn。

可以设计一个 hash_table, hash_map<string, int>，依次读取一千万条短信，加载到 hash_table 表中，并且统计重复的次数，与此同时维护一张最多 10 条的短信表。

这样遍历一次就能找出最多的前 10 条，算法复杂度为 O(n)。

实现如下：

```
1.  #include<iostream>
```

```
2.  #include<map>
3.  #include<iterator>
4.  #include<stdio.h>
5.  using namespace std;
6.
7.  #define HASH __gnu_cxx
8.  #include<ext/hash_map>
9.  #define uint32_t unsigned int
10. #define uint64_t unsigned long int
11. struct StrHash
12. {
13. uint64_t operator()(const std::string& str) const
14. {
15. uint32_t b = 378551;
16. uint32_t a = 63689;
17. uint64_t hash = 0;
18.
19. for(size_t i = 0; i < str.size(); i++)
20. {
21. hash = hash * a + str[i];
22. a = a * b;
23. }
24.
25. return hash;
26. }
27. uint64_t operator()(const std::string& str, uint32_t field) const
28. {
29. uint32_t b = 378551;
30. uint32_t a = 63689;
31. uint64_t hash = 0;
32. for(size_t i = 0; i < str.size(); i++)
```

```
33. {
34. hash = hash * a + str[i];
35. a = a * b;
36. }
37. hash = (hash<<8)+field;
38. return hash;
39. }
40. };
41. struct NameNum{
42. string name;
43. int num;
44. NameNum():num(0),name(""){}
45. };
46. int main()
47. {
48. HASH::hash_map< string, int, StrHash > names;
49. HASH::hash_map< string, int, StrHash >::iterator it;
50. NameNum namenum[10];
51. string l = "";
52. while(getline(cin, l))
53. {
54. it = names.find(l);
55. if(it != names.end())
56. {
57. names[l] ++;
58. }
59. else
60. {
61. names[l] = 1;
62. names[l] = 1;
63. }
```

```
64. }
65. int i = 0;
66. int max = 1;
67. int min = 1;
68. int minpos = 0;
69. for(it = names.begin(); it != names.end(); ++ it)
70. {
71. if(i < 10)
72. {
73. namenum[i].name = it->first;
74. namenum[i].num = it->second;
75. if(it->second > max)
76. max = it->second;
77. else if(it->second < min)
78. {
79. min = it->second;
80. minpos = i;
81. }
82. }
83. else
84. {
85. if(it->second > min)
86. {
87. namenum[minpos].name = it->first;
88. namenum[minpos].num = it->second;
89. int k = 1;
90. min = namenum[0].num;
91. minpos = 0;
92. while(k < 10)
93. {
94. if(namenum[k].num < min)
```

```
95. {
96. min = namenum[k].num;
97. minpos = k;
98. }
99. k ++;
100. }
101. }
102. }
103. i++;
104.
105. }
106. i = 0;
107. cout << "maxlength (string,num): " << endl;
108. while( i < 10)
109. {
110. cout << "(" << namenum[i].name.c_str() << "," << namenum[i].num << ")" << endl;
111. i++;
112. }
113. return 0;
114. }
115.
```

使用 **g++编译如下：**

g++ main.cpp -o main

短信文本文件为：msg.txt

运行：./main < msg.txt

输出结果为：

maxlength (string,num):

(点点母婴坊,4)

(农机配件维修,5)

(红胜超市,6)

(龙溪大酒店,8)

(张记饺子馆,3)

(友谊旅店,3)

(明珠通讯,3)

(金源旅馆,3)

(洞庭山天然泉水,2)

(清源超市,3)

Hadoop（2.0）面试题

（1） 解释“hadoop”和“hadoop 生态系统”两个概念
（2） 说明 Hadoop 2.0 的基本构成
（3） 相比于 HDFS1.0, HDFS 2.0 最主要的改进在哪几方面？
（4） 试使用“步骤 1，步骤 2，步骤 3…..”说明 YARN 中运行应用程序的基本流程
（5） “MapReduce 2.0”与“YARN”是否等同，尝试解释说明
（6） MapReduce 2.0 中，MRAppMaster 主要作用是什么，MRAppMaster 如何实现任务容错的？

58 同城电话号码识别核心算法

算法描述

基于要识别的图像生成 01 二维数组 pixarr

将所有的模板读入内存

将所有的特征模板读入内存

将 pixarr 扫描一遍，去掉孤立点。(孤立点就是指其附近都是 0 的 1 点)

找到首次出现 1 的那一行，记为 top，以后就在(top--->top+18)行的范围内识别

```
row=top  col=0
while(true)
  while(col 列没出现 1&&col<width)
     col++
  if(col==width)
    break
```

以(top,col)为起点，沿着右和下的方向匹配每一个模板，找到吻合率最大的那个模板 template0

那么这块区域所代表的字符就应该是模板 template0 对应的字符

if(该字符为'-'并且匹配率不等于 1)

表明匹配有误，往后倒退一列，基于特征模板进行匹配，为保险起见，最多只扫描三列

将当前得到的字符修改为基于特征模板匹配得到的字符

记录下该字符

清理当前区域，为下一步迭代做准备

else

记录下该字符

width0=模板 template0 的宽度

以(top,col)为起点，沿着右和下的方向对比模板 template0，将 template0 对应位置处的 1 置为 0

在(top,col)--->(top+18,col+width0-1)rectangle 范围内以深度优先搜寻的方式递归遍历，每次找到一块连通区域，考察该区域

如果该区域的最右边不超过 rectangle 右边框，把该区域所有 1 置为 0

ShibiesimpleGraphical.java

```
1.  package zhanlianyzm;
2.
3.  import java.awt.image.BufferedImage;
4.  import java.io.File;
5.  import java.io.IOException;
6.
7.  import javax.imageio.ImageIO;
8.
9.  public class ShibiesimpleGraphical {
10.         static int[][] pixarr = null;
11.         static boolean[][] vistedarr = null;
12.         static int[] dx = new int[8], dy = new int[8];// 方向数组
13.         static {
14.                 dx[0] = 0;
15.                 dy[0] = -1;
16.                 dx[1] = -1;
17.                 dy[1] = -1;
18.                 dx[2] = -1;
19.                 dy[2] = 0;
20.                 dx[3] = -1;
21.                 dy[3] = 1;
22.                 dx[4] = 0;
23.                 dy[4] = 1;
24.                 dx[5] = 1;
25.                 dy[5] = 1;
26.                 dx[6] = 1;
27.                 dy[6] = 0;
28.                 dx[7] = 1;
29.                 dy[7] = -1;
30.         }
31.
```

```
32.         static void resetvistedarr() {
33.                 for (int row = 0; row < vistedarr.length; row++)
34.                         for (int col = 0; col < vistedarr[0].length; col++)
35.                                 vistedarr[row][col] = false;
36.         }
37.
38.         static void makematrix(String imgfile) {
39.                 int plex = -1;
40.                 try {
41.                         BufferedImage img = ImageIO.read(new File(imgfile));
42.                         int width = img.getWidth(null);
43.                         int height = img.getHeight(null);
44.                         pixarr = new int[height][width];
45.                         System.out.println("width=" + width + " height=" + height);
46.                         for (int y = 0; y < height; y++) {
47.                                 for (int x = 0; x < width; x++) {
48.                                         plex = img.getRGB(x, y);
49.                                         if (plex == -1)
50.                                                 plex = 0;
51.                                         else
52.                                                 plex = 1;
53.                                         pixarr[y][x] = plex;
54.                                 }
55.                         }
56.
57.                 } catch (IOException e) {
58.                         // TODO Auto-generated catch block
59.                         e.printStackTrace();
60.                 }
61.                 vistedarr = new boolean[pixarr.length][pixarr[0].length];
62.         }
```

```
63.
64.         static int getfirst1row(int[][] a) {// 返回首次出现 1 的行下标,返回-1 表示 a 中没有 1
65.                 for (int i = 0; i < a.length; i++)
66.                         for (int j = 0; j < a[0].length; j++) {
67.                                 if (a[i][j] == 1)
68.                                         return i;
69.                         }
70.                 return -1;
71.         }
72.
73.         static void displaymatrix(int[][] a) {
74.                 for (int row = 0; row < a.length; row++) {
75.                         for (int col = 0; col < a[0].length; col++)
76.                                 System.out.print(a[row][col] + " ");
77.                         System.out.println();
78.                 }
79.         }
80.
81.         public static void shibie(String imgfile) {
82.                 makematrix(imgfile);// 基于图片生成 01 二维数组
83.                 displaymatrix(pixarr);
84.                 Template[] templatearr = MakeTemplate.maketemplate();// 将所有的模板读入
    内存,保存在 templatearr 中
85.                 int[][] matrix = templatearr[9].getMatrix();
86.                 // displaymatrix(matrix);
87.                 Templatefeature[] templatefeaturearr = MakeTemplatefeature
88.                                 .maketemplatefeature();
89.                 filteroutlier(pixarr); // 考察 pixarr 整体范围，过滤孤立点
90.
91.                 int top = getfirst1row(pixarr); // 找到首次出现 1 的那一行
92.                 if (top == -1)
```

```
93.                         return;
94.                 int row = top, col = 0;
95.                 StringBuilder builder = new StringBuilder();
96.                 int debug = 0;
97.                 while (true) {
98.                         if (debug == 6)
99.                                 System.out.println();
100.                        debug++;
101.            boolean featurematch=false;
102.                        while (col < pixarr[0].length && !have1(col))
103.                                col++;
104.                        if (col == pixarr[0].length)
105.                                break;
106.                        int max = 0, ps = 0;
107.                        for (int i = 0; i < templatearr.length; i++) {// 以(top,col)为起
    点,循环匹配每一个模板
108.                                int Consistent_rate = match(top, col, templatearr[i]);
109.                                if (Consistent_rate > max) {
110.                                        max = Consistent_rate;
111.                                        ps = i;
112.                                        // System.out.println("max:" + max);
113.                                }
114.                        }
115.                        System.out.println("ch=" + templatearr[ps].getCh());
116.                        char ch = templatearr[ps].getCh();
117.                        if (ch == '-' && max != 100) {
118.                                String result = null;
119.                                for (int i = 0; i < templatefeaturearr.length; i++) {// 以
    (top,col-1)为起点,循环匹配每一个特征模板模板
120.                                        result = featurematch(top, col - 1,
    templatefeaturearr[i]);// "12,53"
```

```
121.                                     if (result != null) {
122.                                             ch = templatefeaturearr[i].getCh();
123.                                             int firstfeaturepoint_row =
    templatefeaturearr[i]
124.                                                             .getFeatureps()[0][0];//
    0
125.                                             int firstfeaturepoint_col =
    templatefeaturearr[i]
126.                                                             .getFeatureps()[0][1];//
    5
127.                                             int orow =
    templatefeaturearr[i].getOrow(); // 5
128.
                                                 // firstfeaturepoint_row 在 ch 模板中的相对位
    置
129.                                               int ocol =
    templatefeaturearr[i].getOcol(); // 5
130.
                                                 // firstfeaturepoint_col 在 ch 模板中的相对位
    置
131.                                               String[] strarr = result.split("\\,");
132.                                               int feature_startrow =
    Integer.parseInt(strarr[0]);
133.                                               int feature_startcol =
    Integer.parseInt(strarr[1]);
134.                                               int template_startrow = feature_startrow
135.                                                             + firstfeaturepoint_row -
    orow;
136.                                               int template_startcol = feature_startcol
137.                                                             + firstfeaturepoint_col -
    ocol;
```

```
138.                                                System.out.println(template_startrow +
    ","+ template_startcol);
139.                                                Template template = null;
140.                                                for (int im = 0; im < templatearr.length;
    im++) { // 寻找 ch 模板
141.                                                        template = templatearr[im];
142.                                                        if (template.getCh() == ch)
143.                                                                break;
144.                                                }
145.                                                int width0=template.getWidth();
146.                                                cleared(template_startrow,
    template_startcol, template.getMatrix(), ch); // 以(top,col)为起点，对比模板 template0，
    将 template0 对应位置处的 1 置为 0
147.                                                filternoise(pixarr, template_startrow,
    template_startcol, template_startrow + 18, col + width0 - 1); // 扫描局部范围，过滤噪音
148.                                                resetvistedarr();
149.                                                System.out.println("过滤噪音后
    ========================================================================");
150.                                                displaymatrix(pixarr);
151.                                                featurematch=true;
152.                                                break;
153.                                        }//end if (result != null)
154.
155.                                }//end for(int i = 0
156.                                if (result == null)
157.                                        break;
158.                        }
159.                        builder.append(ch);// 当前阶段吻合率最大的模板是 templatearr[ps]，
    记下它所对应的字符
160.            if(featurematch)
161.                    continue;
```

```
162.                        int width0 = templatearr[ps].getWidth();
163.
164.                        if (ch == '1')
165.                                System.out.println();
166.                        displaymatrix(pixarr);
167.                        cleared(top, col, templatearr[ps].getMatrix(), ch); // 以(top,col)
    为起点，对比模板 template0，将 template0 对应位置处的 1 置为 0
168.                        System.out.println("==========================================
    ====================================");
169.                        System.out.println();
170.                        displaymatrix(pixarr);
171.                        // filteroutlier(pixarr, top, col, top + 18, col + width0 - 1); //
172.                        // 扫描局部范围，过滤孤立点
173.                        filternoise(pixarr, top, col, top + 18, col + width0 - 1); // 扫
    描局部范围，过滤噪音
174.                        resetvistedarr();
175.                        System.out.println("过滤噪音后
    ========================================================================");
176.                        displaymatrix(pixarr);
177.                }
178.                System.out.println(builder.toString());
179.        }
180.
181.        private static String featurematch(int top, int col,
182.                        Templatefeature templatefeature) {// 按特征模板匹配，如果成功，则
    返回特征模板矩阵当时所处位置
183.                int startcol = col;
184.                int[][] pixmatrix = templatefeature.getMatrix();
185.                int[][] featureps = templatefeature.getFeatureps();
186.                int h = pixmatrix.length;
187.
```

```
188.                for (int x = col; x < col + 2; x++) {// 为保险起见，最多只扫描两列
189.                        for (int y = top; y <= top + 18 - h; y++) {
190.                                if (scanmatch(y, x, pixmatrix, featureps))
191.                                        return y + "," + x;
192.                        }
193.                }
194.                return null;
195.        }
196.
197.        private static boolean scanmatch(int y, int x, int[][] pixmatrix,
198.                        int[][] featureps) {// 从(y,x)起对比 pixarr 和 pixmatrix,重点考察
    featureps 所表示的点是否匹配
199.                for (int p = 0; p < featureps.length; p++) {
200.                        int row = featureps[p][0];// 行
201.                        int col = featureps[p][1];// 列
202.                        if (pixmatrix[row][col] != pixarr[row + y][col + x])
203.                                return false;
204.                }
205.                return true;
206.        }
207.
208.        static int localtop = 0, localleft = 0, localbottom = 0, localright = 0;
209.
210.        private static void filternoise(int[][] pixarr, int top, int left,int bottom, int
    right) {// 扫描局部范围，过滤噪音
211.                for (int row = top; row <= bottom; row++)
212.                        for (int col = left; col <= right; col++) {
213.                                if (pixarr[row][col] == 1) {
214.                                        if (vistedarr[row][col])
215.                                                continue;
216.                                        localtop = row;
```

```
217.                                    localleft = col;
218.                                    localbottom = row;
219.                                    localright = col;
220.                                    int localtop0 = localtop, localleft0 = localleft;
221.                                    int localbottom0 = localbottom, localright0 =
    localright;
222.                                    dfsTraversal(row, col, top, left, bottom);// 以
    (row,col)点为起点，在该局部范围内，找到一块连通区域
223.                                    /*
224.                                     *
    if(localtop==localtop0&&localleft0==localleft&&localbottom0==localbottom&&localright0=
    =localright)
225.                                     * continue;
226.                                     */
227.
228.                                    if (localright <= right) {// 如果该区域的最右边不
    超过 rectangle 右边框,把该区域所有 1 置为 0
229.                                            for (int y = localtop; y <= localbottom;
    y++)
230.                                                    for (int x = localleft; x <=
    localright; x++)
231.                                                            pixarr[y][x] = 0;
232.                                    }
233.                            }
234.                    }
235.
236.        }
237.
238.        private static void dfsTraversal(int row, int col, int top, int left,
239.                        int bottom) {// 以(row,col)点为起点，深度优先遍历找到一块连通区域
```

```
240.                for (int i = 0; i < dx.length; i++) {// 分别往 5 个方向测探,上边的方向之前
    已经扫描处理过了，可以不必再考虑
241.                        int row_1 = row + dy[i];
242.                        int col_1 = col + dx[i];
243.                        // if(row_1 <top || row > bottom || col_1 < left)
244.                        // continue;
245.                        if (vistedarr[row_1][col_1])
246.                                continue;
247.                        vistedarr[row_1][col_1] = true;
248.                        if (pixarr[row_1][col_1] == 1) {
249.                                if (row_1 > localbottom)
250.                                        localbottom = row_1;
251.                                if (col_1 < localleft)
252.                                        localleft = col_1;
253.                                if (col_1 > localright)
254.                                        localright = col_1;
255.                                dfsTraversal(row_1, col_1, top, left, bottom);
256.                        }
257.
258.                }
259.
260.        }
261.
262.        private static void filteroutlier(int[][] pixarr) {
263.                filteroutlier(pixarr, 0, 0, pixarr.length - 1, pixarr[0].length - 1);
264.
265.        }
266.
267.        private static void filteroutlier(int[][] pixarr, int top, int left,
268.                        int bottom, int right) {// 扫描 pixarr 局部，去掉孤立点
269.                for (int row = top; row <= bottom; row++)
```

```
270.                        for (int col = left; col <= right; col++) {
271.                                changepoint(pixarr, top, left, bottom, right, row, col);
272.
273.                        }
274.
275.        }
276.
277.        private static void changepoint(int[][] pixarr, int top, int left,
278.                        int bottom, int right, int row, int col) {
279.                if (pixarr[row][col] == 0)
280.                        return;
281.                boolean have1 = false;
282.                for (int i = 0; i < dx.length; i++) {// 测探八个方向存在 1 否?
283.                        int row_1 = row + dy[i];
284.                        int col_1 = col + dx[i];
285.                        if (row_1 >= top && row <= bottom && col_1 >= left
286.                                        && col_1 <= right) {
287.                                if (pixarr[row_1][col_1] == 1) {
288.                                        have1 = true;
289.                                        break;
290.                                }
291.                        }
292.                }
293.                if (!have1)
294.                        pixarr[row][col] = 0;
295.        }
296.
297.        private static void cleared(int top, int left, int[][] matrix, char ch) {// 以
    (top,left)为起点，对比矩阵 matrix，将 matrix 为 1 的点与 pixarr 对应位置处的数值置为 0
298.                for (int row = 0; row < matrix.length; row++)
299.                        for (int col = 0; col < matrix[0].length; col++) {
```

```
300.                                if (ch == '8'
301.                                                && ((row == 18 && col == 7) || (row == 12
    && col == 9)))
302.                                        System.out.println();
303.                                if (matrix[row][col] == 1)
304.                                        pixarr[row + top][col + left] = 0;
305.                        }
306.
307.        }
308.
309.        private static int match(int top, int left, Template template) {// 以(top,left)
    为起点,匹配模板 template,返回吻合率
310.                int[][] matrix = template.getMatrix();
311.                double sum = 0.0;
312.                for (int row = 0; row < matrix.length; row++)
313.                        for (int col = 0; col < matrix[0].length; col++) {
314.                                if (matrix[row][col] == pixarr[top + row][left + col])
315.                                        sum++;
316.                        }
317.                // System.out.println("sum="+sum);
318.                return (int) (sum * 100 / (matrix.length * matrix[0].length));
319.        }
320.
321.        private static boolean have1(int col) {//
322.                int sum = 0;
323.                int firstrow = 0;
324.                for (int row = 0; row < pixarr.length; row++)
325.                        if (pixarr[row][col] == 1) { // return true;
326.                                if (firstrow == 0)
327.                                        firstrow = row;
328.                                sum++;
```

```
329.                        }
330.                if (sum == 0)
331.                        return false;
332.                else {
333.                        if (sum == 1 && firstrow > 9) {
334.                                pixarr[firstrow][col] = 0;
335.                                return false;
336.                        }
337.                        return true;
338.                }
339.        }
340.
341.        public static void main(String[] args) {
342.                String imgfile = "D:/58同城/8.gif";
343.                shibie(imgfile);
344.
345.        }
346.
347.}
348.
```

职场中：面试时如何谈薪金问题

前言

坐在面试官面前拉锯“价格战”。战战兢兢生怕说错一个字，既委屈了自己又失去了机会……工作就要得到应有回报，这本是天经地义的事情，但面对面试中的薪金问题，求职者总感到头疼发怵。

根据可锐职业顾问近期对求职者面试问题的调查发现，有 73%的职场人士在面试时会碰到“谈薪资”的

敏感问题。专家建议，应届毕业生刚走出校门，在面试前对关于薪金的敏感问题一定要有所准备，免得到时候手足无措。

先谈薪资是种浪费

在宝贵的面试机会中谈薪资是一种浪费，从某种意义上说，这是给别人一个拒绝你的理由。所以职业顾问不主张在面试时主动和老板谈薪水。但在有些面试中，即使你一再避免谈薪水，面试官还是会要求你正面回答这些问题。这个时候如果还一再推脱，恐怕就要使自己显得软弱了。

如何谈到点子上？

在回答薪金问题的时候，不能乘匹夫之勇乱答一气，要有准备，要有策略：

策略 1、把期望值放到行业发展的趋势去

考虑你的专业是什么？人才市场对你这类人才的需求有多大？留意一下你周围的人：你的同学、你的朋友、和你找同一个工作的人，他们能拿多少的薪水？结合公司的情况，取他们中间的一个平均值来考虑你的期望薪资，同时还应该多留意新闻中和本行业有关的报道。

策略 2、谈薪水的时候不要拘泥于薪资本身

在面试中谈薪水，是不能“就薪水谈薪水”，要把握适度合理的原则。告诉自己的面试官，薪水不是重要的，你更在乎的是职位本身，你喜欢这份工作；告诉公司你希望公司能了解自己的价值。这样，就能将薪金问题提升到另一个高度，将有助于你找到一份满意的工作。

学会给自己留后路

旅游专业的张毕业后来到一家大型的旅游会展公司面试，在业内人事看来，这是一家非常有名气和实力的公司。在面试中，张表现得非常出色，但当面试官问及她期望的薪资的时候，她开出了一个较高的薪水，和该公司提供给新员工的薪水差距较大。面试官明确表示：这样的薪水，本公司不能接受。眼看着面试陷入僵局，自己喜欢的工作就要失去，张又不想自贬身价，于是她一方面先是告诉面试官，薪水不是最重要的，重要的是自己希望能在公司学习、工作；另一方面，她又拿出自己以往的工作经历，并结合会展业的前景进行分析。这个“缓兵之计”很好地缓和了“谈判局势”，使即将结束的面试得到转机，也使张最后求职成功。

关键问题直接辅导

1、典型问题：在我们公司工作，你希望得到什么样的薪金待遇？

考前辅导：面试前要早做准备，在心里确定好自己希望的薪金范围。先了解该公司的所在地区、所属行业、公司规模，然后尽量了解本行业现在的工资水平。在告之对方自己希望的薪金待遇时，尽可能给出一个你希望的薪水范围，避免说出具体的数字，除非对方有这样的要求。

参考答案：工资并不是我决定是否加盟的唯一因素，如果您一定要我回答，那我当然希望自己的薪水符合我的学历水平和实践经验，我希望自己的工资不低于年薪 XX 万元。

2、典型问题：你觉得自己每年加薪的幅度是多少？

考前辅导：通常情况下，面试官可以接受的答案是“收入的增长和生活水平提高保持一致。”除此之外，

你还应该提到，自己工作业绩的提高是加薪的决定性因素。

参考答案：我想，自己薪水的提高取决于在公司的经营业绩和赢利状况，但我也希望自己收入的增长至少和我生活水平的提高保持一致。

3、典型问题：你愿意降低自己的薪水标准吗？

考前辅导：如果确实非常想得到眼前的这份工作，那开始工作时降低自己的薪金标准是可以考虑的。面对面试官，你要首先强调自己可以把工作做好，并且设法了解公司什么时候能够给你调整工资待遇。此外，对自己的能够承受的工资底限要心中有数，但是不要把这个底限告诉你的面试官。

参考答案：我对这个职位非常感兴趣，所以我可以考虑降低自己的薪金标准，但我也希望公司能给我时间让我证明自己的能力。我相信自己可以让公司满意我的工作，如果我出色地完成了自己的任务，您是否会考虑对我的薪水作一些调整呢？

4、典型问题：从现在开始的三年内，你的薪金目标是什么？

考前辅导：在面试前最好能了解一下同行业从业人员工资的增长情况，如果你能通过朋友打听到这家公司的薪金增长幅度更好。可以对面试官说出一个大概的数字范围或者百分比。

参考答案：我相信通过一段时间的实践，自己将成为这个行业中的佼佼者，我也希望自己以后的收入能和我的能力相符合。我希望自己的年收入在 XX 元到 XX 元之间。

5、典型问题：你认为我们给你提供的薪水如何？

考前辅导：首先确定面试官给出的薪水是公司正式员工的工资，如果是这种情况，要把这个数额和自己的薪金要求相比较，看自己能否接受这个工资待遇。在欧美公司，面试官大都希望应聘者就这个问题和他进行商量；而对于本土公司则相反。

参考答案：您给出的这个数字和我的期望值很接近，但是我心目中的期望稍微要高出一些。我了解公司对于新人工资数额有一定的限制，我们能否谈谈除了工资以外的其它福利待遇？

hadoop 面试题

国美在线面试题

1、hdfs 原理，以及各个模块的职责

2、mr 的工作原理

3、map 方法是如何调用 reduce 方法的

4、shell 如何判断文件是否存在，如果不存在该如何处理？

5、fsimage 和 edit 的区别？

6、hadoop1 和 hadoop2 的区别？

笔试：

1、hdfs 中的 block 默认报错几份？

2、哪个程序通常与 nn 在一个节点启动？并做分析

3、列举几个配置文件优化？

4、写出你对 zookeeper 的理解

5、datanode 首次加入 cluster 的时候，如果 log 报告不兼容文件版本，那需要 namenode 执行格式化操作，

这样处理的原因是？

6、谈谈数据倾斜，如何发生的，并给出优化方案

7、介绍一下 hbase 过滤器

8、mapreduce 基本执行过程

9、谈谈 hadoop1 和 hadoop2 的区别

10、hbase 集群安装注意事项

11、记录包含值域 F 和值域 G，要分别统计相同 G 值的记录中不同的 F 值的数目，简单编写过程。

程序员面试:简历、面试前、面试中该 注意什么

程序员找工作难，想要被成功聘用更难。最常见的办法是经历一次又一次的面试失败后自己琢磨出面试技巧，当然也可以花钱到一些培训机构去接受专业的书面简历和模拟面试的指导。这些方法可能都会奏效，但是却并不是时间和金钱利用率最高的。软件行业的工作期望比其他大多数岗位都要来得高。在这严格的选拔人才的机制下，只要我们花点心思，还是可以手到擒来，顺利地拿下心仪的工作岗位。

写简历

- 既简短又要突出重点。不超过两页。面试官不需要个人传记。
- 在描述每个项目的时候避免长篇大论。在每份工作和项目中注明用了什么技术，以便与面试官作深入的探讨。
- 简单说说以前的每份工作。
- 无需提供涉及非技术内容的信息。例如在快餐、零售和任何其他非技术岗位上的工作经验就没有说的必要。
- 在简历上不可说谎。面试官一问便知真与假。
- 如果可能的话，让在这一行的朋友给你的简历提提意见。有时候简历上面即使是一个很小的错误搞不好恰好就是某些雇主的雷区。
- 除了简历，还可以做一个可视的作品集。可以是软件的截图，也可以是你写的一些代码示例。关键是要展示你对自己的工作感到非常自豪。
- 将简历预存成多种格式的，包括 Word 和 PDF。

在面试之前

- 查看自己的上网记录，看看 Google、Bing 和 Yahoo，必要的话清理一下。
- 确定到达面试地点的方式和路线，避免迷路。
- 回顾以往的工作经验，详细谈论之前的项目。面试官会在讨论的过程中推断开发人员的能力如何。
- 搞一次模拟面试，请朋友来给你指出薄弱的区域。
- 不要太过紧张，也不要期望太高。
- 取得面试官的联系信息，以防万一（堵车、汽车故障等）。
- 研究此公司和职位。关键是公司和职位要求得与自己匹配。
- 额外打印一份简历。此外，还要带上笔记本和笔，

在面试过程中

- 如果不得不迟到，提前打个电话并对此道歉。没办法的话可以重新约定时间。
- 保持眼神交流。
- 注意肢体语言和姿势。不要分心。
- 保持笑容，展示出自己积极的心态。做一个积极的倾听者和参与者。不要骄傲自大也不要看上去无聊和不感兴趣。
- 有条不紊地回答问题。不要太急切。经过深思熟虑之后的回答总是比不经大脑脱口而出的要好。
- 可以直接说“I don’t know”，但是要流露出非常想了解这方面知识的想法。
- 不要主动暴露过多的信息。说话要简单明了，避免东拉西扯。还有不要透露太多的个人信息。
- 在面试过程中要时刻注意自己的行为举止，知道什么是能做的什么是不能做的。

- 不要讨论薪水，除非面试官主动提出来。
- 面试结束或者中场休息时间，将自己不熟悉的话题记下来，研究一下。

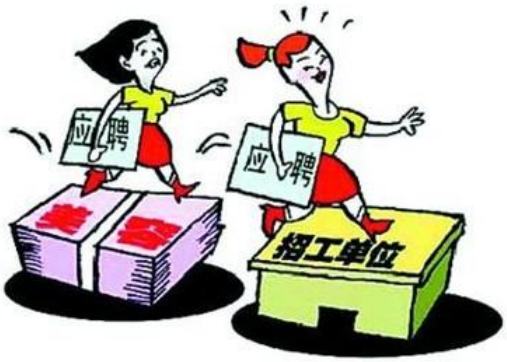
软件工程师面试前你要注意什么

此文是我作为一个面试官关于如何在技术面试中出奇制胜的经验总结。现今大多数的面试都遵循一定的模式。如果你能够理解面试官的问题，按照他的方式回答，那么任何面试都将不是问题。如果你觉得学这些技巧无关紧要，那我也无话可说，不过如果你各方面已经“各就各位”，那么此文必将助你发挥出所有的潜能。

如果你自认为技术娴熟，那么唯一会面试失败的原因就是准备不足。你可能无所不知，但是你仍然需要阅读书籍、论文文章等做好准备，这些资料可能并不能教给你什么新鲜的资讯，但是能让你有效地组织已经知道的知识和内容。而有效组织的内容，有助于面试官的理解。另外，我补充一下，阅读应该成为一种习惯，而不光光是一种面试的准备，这能让你将来的工作做得更好、职业生涯更上一层楼。

大多数的面试目的是面试官为了找到那种能一起工作的人，当然，空缺的岗位有可能是在其他团队，但是面试官还是会按照自己的评价标准来择取。本文主要涉及一些通用技巧，主要针对拥有 2 到 6 年工作经验的软件工程师。

面试前你要注意什么？



Top 1 诚实，不要虚张声势

自信地回答问题，如果你不知道，那就坦诚地说“我不知道，但是我认为.....”。知之为知之，不知为不知，不要理直气壮地讲述一个错误的答案，这会让面试官怀疑你前面那些正确的答案搞不好也是胡诌的，功亏一篑。虽然这句措辞也不是万金油，但是却能显示你积极思考的能力和永不放弃的态度。对面试官提出的所有问题都要做一番努力，不要一句“我不知道”就轻易打发。

Top 2 做好写代码的准备

如果要你写代码，那就要小心了，千万别忘记遵循基本规则。我常听那些应聘人员一脸茫然地说“呀，我忘记语法了.....”，我一看，原来是忘记 for 循环的语法了。我们不希望你在面试的时候记得所有的东西，但是一些基本的，像循环、if 条件、主方法、异常，这些要是忘记了，太不应该了。看到这里，如果你一时间也没法想起这些，不妨复习一下。写代码的时候要注重空格和缩进，那些字写的不好的可要注意了！

Top 3 做好解释项目的准备

软件工程师在写代码之前就应该对业务需求了如指掌。所以，你得能解释项目中一一对应的流程分别是什么。写个三四行代码就深层次解释一下，然后听听非团队人员的意见和建议。当局者迷，旁观者清。看看和客户交流的内部营销记录，找找线索。可以先找个朋友练习一下，确保自己能说到点子上。

一旦你解释完相关的业务需求，那么接下来面试官通常会问你关于这个项目的技术架构。所有你还必须准备架构图，以显示项目中的各个组件是如何相互作用的。架构图不需要任何特定的 UML 格式，但是你得确保画在上面的东西你都能解释得通。举个例子，如果你正在做一个可以显示数据是如何从用户界面到数据库的 Web 应用程序。那么你就得展示涉及的不同 layers 、使用的技术，等等。而最重要的是，你应该清楚你正在做什么，不要讲到后来，乱七八糟，答非所问。

Top 4 将争论转换为交谈

即使你知道对方错了，也不要争论，试着以"ok，不过我不是很确定这是否正确，我回去再查阅一下"这样的言语继续话题。这能让双方都有一个愉快的心情。在面试的时候要认真听对方的话中之意，回答的时候要以自己的经验为依据，不要天马行空。

Top 5 事先对各种 WHY 做好准备

大公司的面试关于问"Why?"的肯定很多，搞不好甚至是以"Why?"开头，以"Why?"结束，一路"Why?"下去。例如经典的 Java 问题"String 和 StringBuffer 的区别是什么？"，后面往往还会有"为什么 String……？"又或者"如何才能……？"事先做好如何回答这些"How？"和"Why？"问题的准备。

Top 6 讲述自己最大的成就

在自己以往的工作中，总会有个你认为是最棒的成就。你得学会将这个成就描绘得"天花乱坠"，让面试官觉得这是一个超级不平凡的成绩。所以准备一个励志生动又可信的故事以展示你的能力是如何一步步让你完成那个艰巨的任务的。之所以要事先准备好是因为担心事到临头你搞不好会忘东忘西有所遗漏不说，最怕就是语言不连贯，让面试官无法理解。

Top 7 你有什么问题要问的吗？

好吧，这个问题几乎每个面试都会出现。问问题并不是说你要知道这些答案，但是问某些“聪明”的问题能让面试官对你的印象加分。《The Five Best Questions a Job Candidate Can Ask》或许能为你开拓一些思路，助你面试成功。

面试资源：

程序员跳槽攻略

程序员面试宝典+剑指 Offer + 算法 100 题系列 + 15 个经典算法下载

java 面试题库(1)

java 面试题库(2)

java 面试题库(3)

找工作的一些思考

10 个面试题，问出真正优秀程序员

美国大数据工程师面试攻略

程序员面试宝典+剑指 **Offer +** 算法 **100** 题系列 **+ 15** 个经典算法下载

整理自下面内容：

hadoop、大数据笔试、面试都会问那些问题

腾讯面试题：有一千万条短信，有重复，以文本文件的形式保存，一行一条，有重复。

hive 面试题目：表大概有 **2T** 左右，对表数据转换

找工作的一些思考

面试题集锦与总结，及 **Github** 地址、**PDF** 下载

阿里四轮面试总结

Hadoop 阿里巴巴面试题目

面试题、经验分享及新手问答整理

优酷 **hadoop**，**mapred** 面试题及答案

面试过程中经常被问道的问题记录

机器学习、大数据面试问题及答题思路

Hadoop 面试题,看看书找答案，看看你能答对多少（**2**）答案公布

百度 **2015** 校园招聘面试题（成功拿到 **offer**）

去公司面试，记录下的最新 **hadoop** 面试题

大数据面试题

软件工程师面试前你要注意什么

hbase 40 道测试题

阿里 **2015** 校招面试回忆（成功拿到 **offer**）

面试 **hadoop** 可能被问到的问题，附部分参考答案

程序员生存定律**--**表达背后的力量

java 面试之大数据

hadoop 面试题

程序员面试**:**简历、面试前、面试中该 注意什么

如果想关注 about 云，可以通过下面方式：

1.加入 about 云群 371358502、322273151、90371779，云计算大数据爱好者群

2.关注腾讯认证空间

about 云腾讯认证空间

3.惯用手机的用户可关注 about 云微信地址：

搜索：

wwwaboutyuncom

4.关注微博：

新浪微博

5.邮件订阅

邮件订阅

2014 过去，2015 年来临，2014 年，about 云也有一年多，云技术、大数据迅速火了起来。过去的一年中，我们都尽其所能的为大家提供技术性实践资料、文章、视频。但是总的来说，还是不够。我们将一如既往的，出一些技术性文章。引导新手入门及学习，同时希望大数据、云技术爱好者多多支持 about 云，同时我们将会出一些视频，也欢迎捐助 about 云。

在 IT 行业，技术不断更新换代，不断抛陈出新，5 年不学习新知识，就会 out 了。about 云

创立之初，面对资料的缺乏，与大家共同学习，相互分享学习经验，帮助了很多初学者入门，about 云在 2015 年将会更努力，成为更多人的大数据、云技术学习分享基地。

同时希望接触 about 云，知道 about 云，了解 about 云的人，about 云能传播给大家一种学习精神----活到老，学到老。

2015 年，about 云为了目标将会不断的学习，努力、坚持，帮助更多的人，更多的大数据、云技术爱好者。