尚硅谷大数据技术之高频面试题

(作者：尚硅谷大数据研发部)

版本：V5.1

尚硅谷大数据研发部

第 1 章 面试说明

1.1 面试的本质

面试就是一场秀。

1.2 面试过程最关键的是什么？

1）不是你说了什么，而是你怎么说

2）大大方方的聊，放松

1.3 面试时该怎么说？

1）语言表达清楚

（1）思维逻辑清晰，表达流畅

（2）一二三层次表达

2）所述内容不犯错

（1）不说前东家或者自己的坏话

（2）往自己擅长的方面说

（3）实质，对考官来说，内容听过，就是自我肯定；没听过，那就是个学习的过程。

1.4 面试技巧

1.4.1 六个常见问题

1）你的优点是什么？

大胆的说出自己各个方面的优势和特长

2）你的缺点是什么？

不要谈自己真实问题；用“缺点”衬托自己的优点

3）你的离职原因是什么？

- 不说前东家坏话，哪怕被伤过
- 合情合理合法
- 不要说超过 1 个以上的原因

4）您对薪资的期望是多少？

- 非终面不深谈薪资
- 只说区间，不说具体数字
- 底线是不低于当前薪资
- 非要具体数字，区间取中间值，或者当前薪资的+20%

5）您还有什么想问的问题？

- 这是体现个人眼界和层次的问题
- 问题本身不在于面试官想得到什么样的答案，而在于你跟别的应聘者的对比
- 标准答案：

  公司希望我入职后的 3-6 个月内，给公司解决什么样的问题

  公司（或者对这个部门）未来的战略规划是什么样子的？

  以你现在对我的了解，您觉得我需要多长时间融入公司？

6）您最快多长时间能入职？

一周左右，如果公司需要，可以适当提前

1.4.2 二个注意事项

1）职业化的语言

2）职业化的形象

1.4.3 自我介绍（控制在 4 分半以内，不超过 5 分钟）

1）个人基本信息

2）工作履历

时间、公司名称、任职岗位、主要工作内容、工作业绩、离职原因

3）深度沟通（也叫压力面试）

刨根问底下沉式追问（注意是下沉式，而不是发散式的）

基本技巧：往自己熟悉的方向说

第 2 章 手写代码

2.1快排

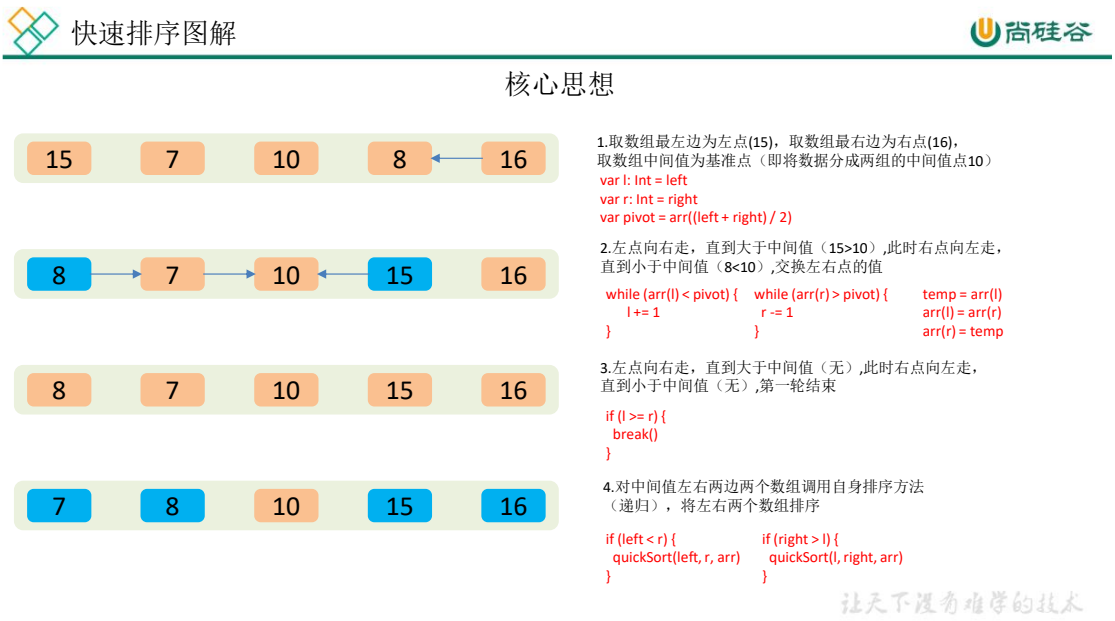


图 1-快速排序核心思想

代码实现：

```
def quickSort(left: Int, right: Int, arr: Array[Int]): Unit = {

    var l: Int = left
    var r: Int = right
    var pivot = arr((left + right) / 2)
    var temp = 0

    //Array(10, 11, 2, -1, 3)
    breakable {
      while (l < r) {

          //从左点向右遍历，直到找到比中间值大的
        while (arr(l) < pivot) {
          l += 1
        }

          //从右点向左遍历，直到找到比中间值小的
        while (arr(r) > pivot) {
          r -= 1
        }

          //判断是否已经越过中间值
        if (l >= r) {
          break()
        }

         //交换数据
        temp = arr(l)
```

```
        arr(l) = arr(r)
        arr(r) = temp
      }
    }

    if (l == r) {
      l += 1
      r -= 1
    }

    //向左递归
    if (left < r) {
      quickSort(left, r, arr)
    }

    //向右递归
    if (right > l) {
      quickSort(l, right, arr)
    }

}
```

2.2归并

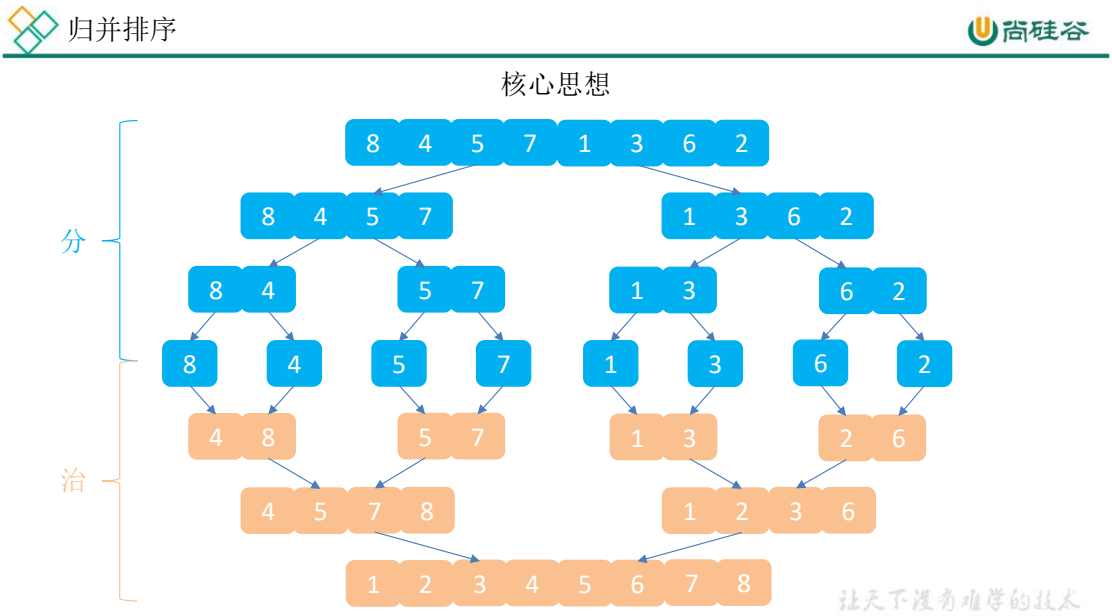


图 2-归并排序核心思想

核心思想：不断的将大的数组分成两个小数组，直到不能拆分为止，即形成了单个值。此时使用合并的排序思想对已经有序的数组进行合并，合并为一个大的数据，不断重复此过程，直到最终所有数据合并到一个数组为止。

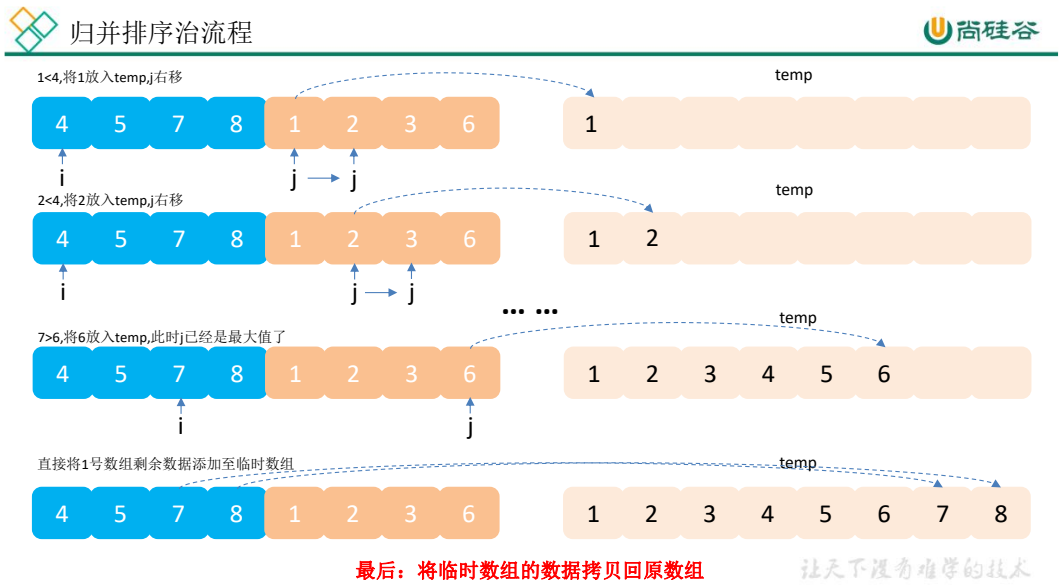


图 3-归并排序“治”流程

代码实现：

```
def mergeSort(arr: Array[Int], left: Int, right: Int, temp:
Array[Int]): Unit = {
    if (left < right) {

      //取中间值
     val mid: Int = (left + right) / 2

      //分过程：向左递归
     mergeSort(arr, left, mid, temp)

      //分过程：向右递归
     mergeSort(arr, mid + 1, right, temp)

      //合并两个有序数组
     merge(arr, left, mid, right, temp)
    }
  }

   //合并两个有序数组的方法
  def merge(arr: Array[Int], left: Int, mid: Int, right: Int, temp:
Array[Int]) {

    var i: Int = left
    var j: Int = mid + 1
    var t = 0
    while (i <= mid && j <= right) {
      if (arr(i) <= arr(j)) {
        temp(t) = arr(i)
        t += 1
        i += 1
      } else {
        temp(t) = arr(j)
        t += 1
```

```
      j += 1
    }
  }
  while (i <= mid) {
    temp(t) = arr(i)
    t += 1
    i += 1
  }
  while (j <= right) {
    temp(t) = arr(j)
    t += 1
    j += 1
  }
  t = 0
  var tempLeft: Int = left
  while (tempLeft <= right) {
    arr(tempLeft) = temp(t)
    t += 1
    tempLeft += 1
  }
}
```

2.3 手写 Spark-WordCount

```
val conf: SparkConf =
new SparkConf().setMaster("local[*]").setAppName("WordCount")

val sc = new SparkContext(conf)

sc.textFile("/input")
  .flatMap(_.split(" "))
  .map((_, 1))
  .reduceByKey(_ + _)
  .saveAsTextFile("/output")

sc.stop()
```

2.4 冒泡排序

```
/**
 * 冒泡排序 时间O(n^2) 空间O(1)
 */
public class BubbleSort {

   public static void bubbleSort(int[] data) {

      System.out.println("开始排序");
      int arrayLength = data.length;

      for (int i = 0; i < arrayLength - 1; i++) {

         boolean flag = false;

         for (int j = 0; j < arrayLength - 1 - i; j++) {
            if(data[j] > data[j + 1]){
               int temp = data[j + 1];
               data[j + 1] = data[j];
               data[j] = temp;
```

```
                flag = true;
            }
        }

        System.out.println(java.util.Arrays.toString(data));

        if (!flag)
            break;
    }
  }

  public static void main(String[] args) {

    int[] data = { 9, -16, 21, 23, -30, -49, 21, 30, 30 };

    System.out.println("    排    序    之    前    :    \n"    +
java.util.Arrays.toString(data));

    bubbleSort(data);

    System.out.println("    排    序    之    后    :    \n"    +
java.util.Arrays.toString(data));
  }
}
```

2.5 二分查找算法之 JAVA 实现

2.5.1 算法概念

二分查找算法也称为折半搜索、二分搜索，是一种在有序数组中查找某一特定元素的搜索算法。请注意这种算法是建立在有序数组基础上的。

2.5.2 算法思想

①搜索过程从数组的中间元素开始，如果中间元素正好是要查找的元素，则搜索过程结束；

②如果某一特定元素大于或者小于中间元素，则在数组大于或小于中间元素的那一半中查找，而且跟开始一样从中间元素开始比较。

③如果在某一步骤数组为空，则代表找不到。

这种搜索算法每一次比较都使搜索范围缩小一半。

2.5.3 实现思路

①找出位于数组中间的值，并存放在一个变量中（为了下面的说明，变量暂时命名为 temp）；

②需要找到的 key 和 temp 进行比较；

③如果 key 值大于 temp，则把数组中间位置作为下一次计算的起点；重复① ②。

④如果 key 值小于 temp，则把数组中间位置作为下一次计算的终点；重复① ② ③。

⑤如果 key 值等于 temp，则返回数组下标，完成查找。

2.5.4 实现代码

```
/**
 * @param <E>
 * @param array 需要查找的有序数组
 * @param from 起始下标
 * @param to 终止下标
 * @param key 需要查找的关键字
 * @return
 * @throws Exception
 */
public static <E extends Comparable<E>> int binarySearch(E[]
array, int from, int to, E key) throws Exception {

   if (from < 0 || to < 0) {
      throw new IllegalArgumentException("params from & length
must larger than 0 .");
   }

   if (from <= to) {

      int middle = (from >>> 1) + (to >>> 1); // 右移即除 2
      E temp = array[middle];

      if (temp.compareTo(key) > 0) {
         to = middle - 1;
      } else if (temp.compareTo(key) < 0) {
         from = middle + 1;
      } else {
         return middle;
      }
   }

   return binarySearch(array, from, to, key);
}
```

2.6 二叉树之 Java 实现

2.6.1 二叉树概念

二叉树是一种非常重要的数据结构，它同时具有数组和链表各自的特点：它可以像数组一样快速查找，也可以像链表一样快速添加。但是他也有自己的缺点：删除操作复杂。

二叉树：是每个节点最多有两个子树的有序树，在使用二叉树的时候，数据并不是随便插入到节点中的，一个节点的左子节点的关键值必须小于此节点，右子节点的关键值必须大于或者是等于此节点，所以又称二叉查找树、二叉排序树、二叉搜索树。

完全二叉树：若设二叉树的高度为 h，除第 h 层外，其它各层 (1～h-1) 的结点数都达

到最大个数，第 h 层有叶子结点，并且叶子结点都是从左到右依次排布，这就是完全二叉树。

满二叉树——除了叶结点外每一个结点都有左右子叶且叶子结点都处在最底层的二叉树。

深度——二叉树的层数，就是深度。

2.6.2 二叉树的特点

1）树执行查找、删除、插入的时间复杂度都是 O(logN)

2）遍历二叉树的方法包括前序、中序、后序

3）非平衡树指的是根的左右两边的子节点的数量不一致

4） 在非空二叉树中，第 i 层的结点总数不超过 , i>=1；

5）深度为 h 的二叉树最多有个结点(h>=1)，最少有 h 个结点；

6）对于任意一棵二叉树，如果其叶结点数为 N0，而度数为 2 的结点总数为 N2，则 N0=N2+1；

2.6.3 二叉树的 Java 代码实现

首先是树的节点的定义，下面的代码中使用的是最简单的 int 基本数据类型作为节点的数据，如果要使用节点带有更加复杂的数据类型，换成对应的对象即可。

```
public class TreeNode {

 // 左节点
 private TreeNode lefTreeNode;
 // 右节点
 private TreeNode rightNode;
 // 数据
 private int value;

 private boolean isDelete;

 public TreeNode getLefTreeNode() {
  return lefTreeNode;
 }

 public void setLefTreeNode(TreeNode lefTreeNode) {
  this.lefTreeNode = lefTreeNode;
 }

 public TreeNode getRightNode() {
  return rightNode;
 }

 public void setRightNode(TreeNode rightNode) {
  this.rightNode = rightNode;
```

```
 }

 public int getValue() {
  return value;
 }

 public void setValue(int value) {
  this.value = value;
 }

 public boolean isDelete() {
  return isDelete;
 }

 public void setDelete(boolean isDelete) {
  this.isDelete = isDelete;
 }

 public TreeNode() {
  super();
 }

 public TreeNode(int value) {
  this(null, null, value, false);
 }

 public TreeNode(TreeNode lefTreeNode, TreeNode rightNode, int
value,
   boolean isDelete) {
  super();
  this.lefTreeNode = lefTreeNode;
  this.rightNode = rightNode;
  this.value = value;
  this.isDelete = isDelete;
 }

 @Override
 public String toString() {
  return   "TreeNode   [lefTreeNode="   +   lefTreeNode   +   ",
rightNode="
    + rightNode + ", value=" + value + ", isDelete=" + isDelete
    + "]";
 }

}
```

下面给出二叉树的代码实现。由于在二叉树中进行节点的删除非常繁琐，因此，下面的代码使用的是利用节点的 isDelete 字段对节点的状态进行标识

```
public class BinaryTree {

 // 根节点
 private TreeNode root;

 public TreeNode getRoot() {
  return root;
 }
```

```
/**
 * 插入操作
 *
 * @param value
 */
public void insert(int value) {

 TreeNode newNode = new TreeNode(value);

 if (root == null) {
  root = newNode;
  root.setLefTreeNode(null);
  root.setRightNode(null);
 } else {

  TreeNode currentNode = root;
  TreeNode parentNode;

  while (true) {

   parentNode = currentNode;
   // 往右放
    if (newNode.getValue() > currentNode.getValue()) {

    currentNode = currentNode.getRightNode();

    if (currentNode == null) {
     parentNode.setRightNode(newNode);
     return;
    }
   } else {
    // 往左放
    currentNode = currentNode.getLefTreeNode();

    if (currentNode == null) {
     parentNode.setLefTreeNode(newNode);
     return;
    }
   }
  }
 }
}

/**
 * 查找
 *
 * @param key
 * @return
 */
public TreeNode find(int key) {

 TreeNode currentNode = root;

 if (currentNode != null) {

  while (currentNode.getValue() != key) {
```

```
    if (currentNode.getValue() > key) {
     currentNode = currentNode.getLefTreeNode();
    } else {
     currentNode = currentNode.getRightNode();
    }

    if (currentNode == null) {
     return null;
     }

   }

   if (currentNode.isDelete()) {
    return null;
   } else {
    return currentNode;
   }

  } else {
   return null;
  }
 }

 /**
  * 中序遍历
  *
  * @param treeNode
  */
 public void inOrder(TreeNode treeNode) {

  if (treeNode != null && treeNode.isDelete() == false) {

   inOrder(treeNode.getLefTreeNode());

   System.out.println("--" + treeNode.getValue());

   inOrder(treeNode.getRightNode());
  }
 }
}
```

在上面对二叉树的遍历操作中，使用的是中序遍历，这样遍历出来的数据是增序的。

下面是测试代码:

```
public class Main {

 public static void main(String[] args) {

  BinaryTree tree = new BinaryTree();

  // 添加数据测试
  tree.insert(10);
  tree.insert(40);
  tree.insert(20);
  tree.insert(3);
  tree.insert(49);
  tree.insert(13);
  tree.insert(123);
```

```
        System.out.println("root=" + tree.getRoot().getValue());

        // 排序测试
        tree.inOrder(tree.getRoot());

        // 查找测试
        if (tree.find(10) != null) {
            System.out.println("找到了");
        } else {
            System.out.println("没找到");
        }

        // 删除测试
        tree.find(40).setDelete(true);

        if (tree.find(40) != null) {
            System.out.println("找到了");
        } else {
            System.out.println("没找到");
        }
    }
}
```

第 3 章 项目架构

3.1 数仓概念

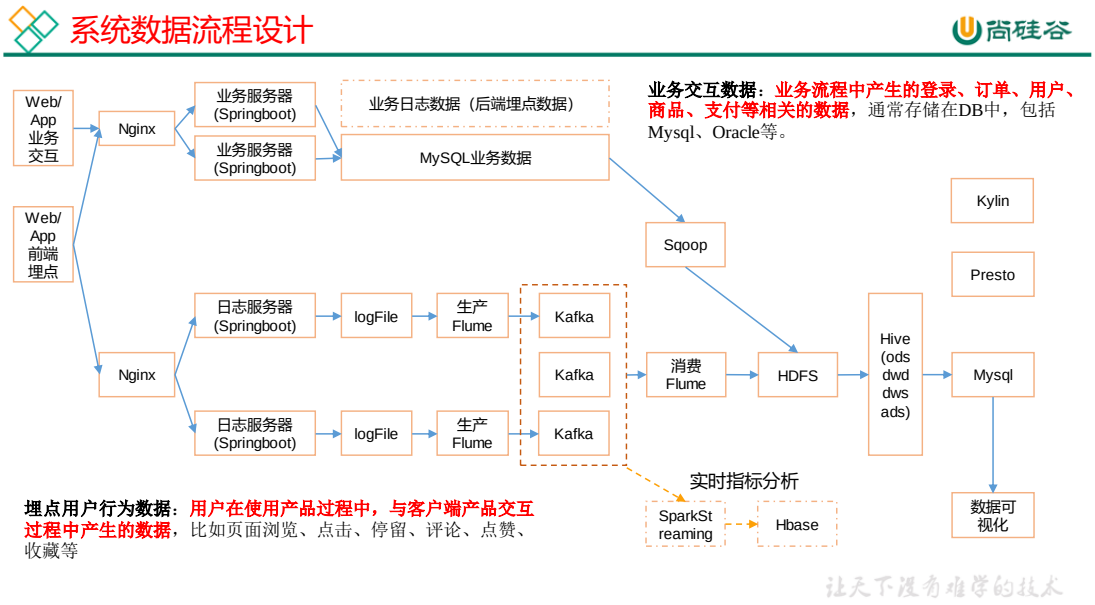
数据仓库的输入数据源和输出系统分别是什么？

输入系统：埋点产生的用户行为数据、JavaEE 后台产生的业务数据。

输出系统：报表系统、用户画像系统、推荐系统

评分标准：输入 2 分、输出 3 分

3.2 系统数据流程设计



评分标准：5 分 （用户行为采集+业务数据采集+数仓+查询+可视化），每条一分

3.3 框架版本选型

1）Apache：运维麻烦，组件间兼容性需要自己调研。（一般大厂使用，技术实力雄厚，有专业的运维人员）

2）CDH：国内使用最多的版本，但 CM 不开源，但其实对中、小公司使用来说没有影响（建议使用）

3）HDP：开源，可以进行二次开发，但是没有 CDH 稳定，国内使用较少

评分标准：5 分 任选其一，并说出理由，理由 4 分。

3.4 服务器选型

服务器使用物理机还是云主机？

1）机器成本考虑：

（1）物理机：以 128G 内存，20 核物理 CPU，40 线程，8THDD 和 2TSSD 硬盘，单台报价 4W 出头，需考虑托管服务器费用。一般物理机寿命 5 年左右

（2）云主机，以阿里云为例，差不多相同配置，每年 5W

2）运维成本考虑：

（1）物理机：需要有专业的运维人员

（2）云主机：很多运维工作都由阿里云已经完成，运维相对较轻松

评分标准：5 分 每条一分，全部答对 5 分

3.5 集群规模

集群规模

尚硅谷

1）如何确认集群规模？（假设：每台服务器8T磁盘，128G内存）

（1）每天日活跃用户100万，每人一天平均100条：10万*100条=10000万条

（2）每条日志1K左右，每天1亿条：100000000 / 1024 / 1024 = 约100G

（3）半年内不扩容服务器来算：100G*180天=约18T

（4）保存3副本：18T*3=54T

（5）预留20%~30%Buf=5.4T/0.7=77T

（6）算到这：约8T*10台服务器

2）如果考虑数仓分层？

服务器将近再扩容1-2倍，讲完数仓后详解

根据数据规模搭建集群

1	2	3	4	5	6	7	8	9	10
nn	nn	dn	dn	dn	dn	dn	dn	dn	dn
		rm	rm	nm	nm	nm	nm	nm	nm
		nm	nm						
							zk	zk	zk
				flume	flume	flume			
		flume	flume						
							kafka	kafka	kafka
hive	mysql								
spark	spark								
							Hbase	Hbase	Hbase

3.6 人员配置参考

3.6.1 整体架构。

属于研发部，技术总监下面有各个项目组，我们属于数据组，其他还有后端项目组，基础平台等。总监上面就是副总等级别了。其他的还有产品运营部等。

评分标准：5 分

3.6.2 你们部门的职级等级，晋升规则

职级就分初级，中级，高级。晋升规则不一定，看公司效益和职位空缺。

评分标准：5 分

3.6.3 人员配置参考

小型公司（3 人左右）：组长 1 人，剩余组员无明确分工，并且可能兼顾 javaEE 和前端。

中小型公司（3~6 人左右）：组长 1 人，离线 2 人左右，实时 1 人左右（离线一般多于实时），JavaEE 1 人（有或者没有人单独负责 JavaEE，有时是有组员大数据和 JavaEE 一起做，或者大数据和前端一起做）。

中型公司（5~10 人左右）：组长 1 人，离线 3~5 人左右（离线处理、数仓），实时 2 人左右，JavaEE 1 人左右（负责对接 JavaEE 业务），前端 1 人左右（有或者没有人单独负责前端）。

中大型公司（5~20 人左右）：组长 1 人，离线 5~10 人（离线处理、数仓），实时 5 人左右，JavaEE2 人左右（负责对接 JavaEE 业务），前端 1 人（有或者没有人单独负责前端）。（发展比较良好的中大型公司可能大数据部门已经细化拆分，分成多个大数据组，分别负责不同业务）

上面只是参考配置，因为公司之间差异很大，例如 ofo 大数据部门只有 5 个人左右，因此根据所选公司规模确定一个合理范围，在面试前必须将这个人员配置考虑清楚，回答时要非常确定。

评分标准：5 分

第 4 章 项目涉及技术

4.1 Linux&Shell 相关总结

4.1.1 Linux 常用命令

序号	命令	命令解释
1	top	查看内存
2	df -h	查看磁盘存储情况
3	iotop	查看磁盘 IO 读写(yum install iotop 安装)
4	iotop -o	直接查看比较高的磁盘读写程序
5	netstat -tunlp | grep 端口号	查看端口占用情况
6	uptime	查看报告系统运行时长及平均负载
7	ps aux	查看进程

评分标准：5 分 每条一分，全部答对 5 分

4.1.2 Shell 常用工具

awk、sed、cut、sort

评分标准：5 分 每条一分，全部答对 5 分

4.2 Hadoop 相关总结

4.2.1 Hadoop 常用端口号

- dfs.namenode.http-address:50070
- dfs.datanode.http-address:50075
- SecondaryNameNode 辅助名称节点端口号：50090
- dfs.datanode.address:50010
- fs.defaultFS:8020 或者 9000
- yarn.resourcemanager.webapp.address:8088
- 历史服务器 web 访问端口：19888

评分标准：5 分 每条一分，全部答对 5 分

4.2.2 Hadoop 配置文件以及简单的 Hadoop 集群搭建

（1）配置文件：

core-site.xml、hdfs-site.xml、mapred-site.xml、yarn-site.xml

hadoop-env.sh、yarn-env.sh、mapred-env.sh、slaves

（2）简单的集群搭建过程：

JDK 安装

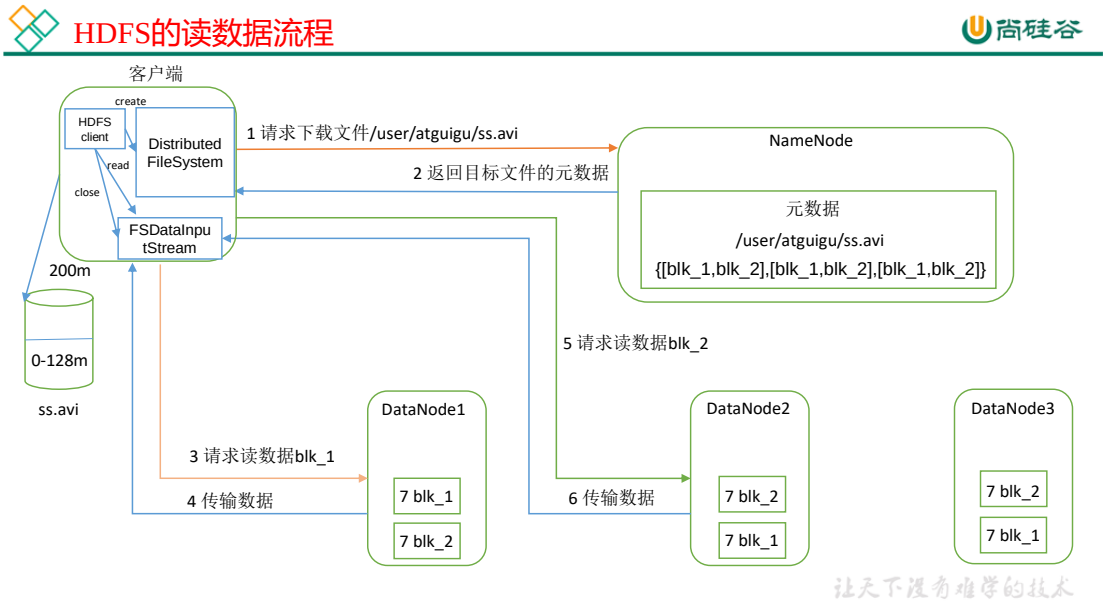
配置 SSH 免密登录

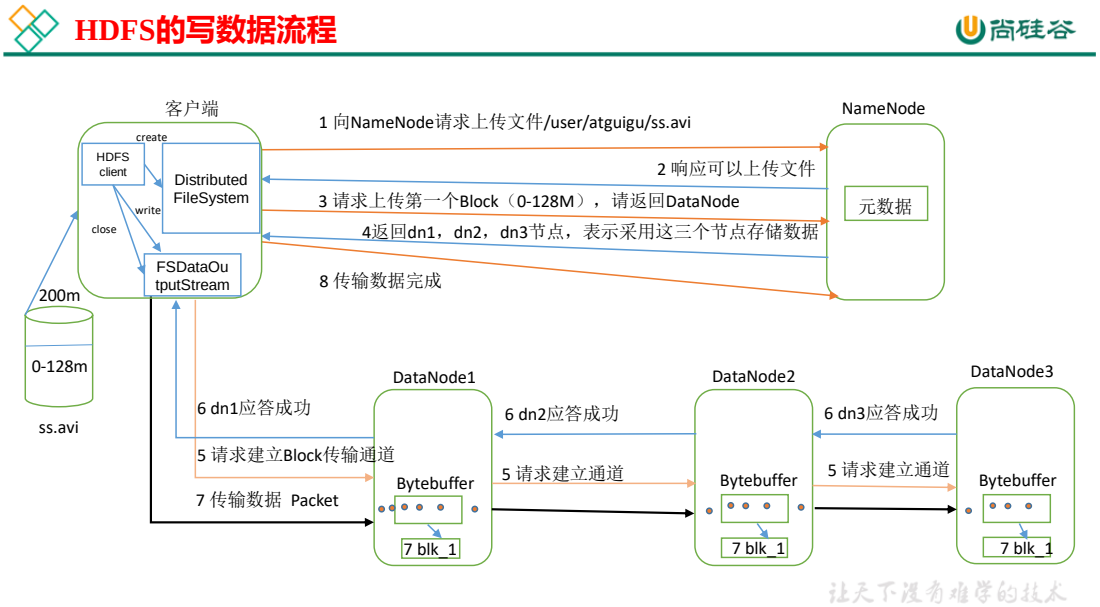
配置 hadoop 核心文件:

格式化 namenode

评分标准：5 分 配置文件 3 分；过程 2 分

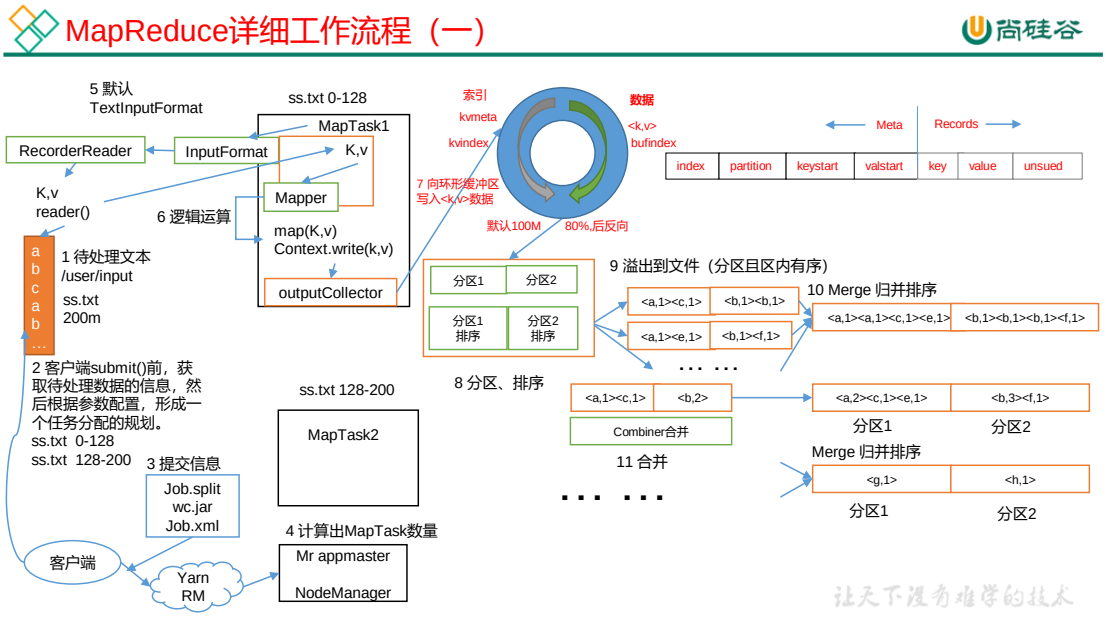
4.2.3 HDFS 读流程和写流程

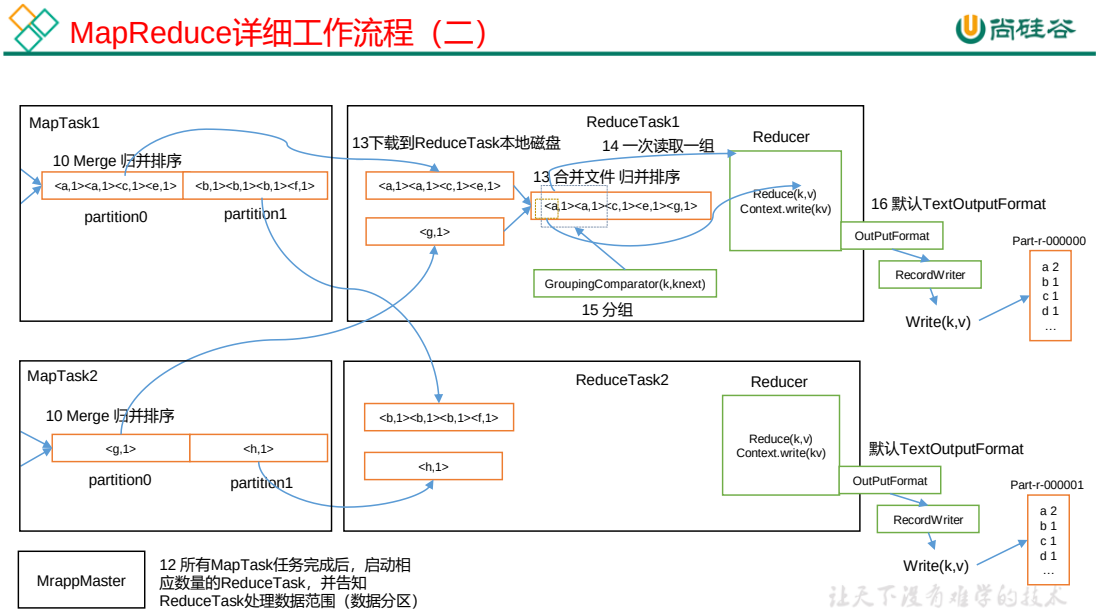




评分标准：5 分

4.2.4 MapReduce 的 Shuffle 过程及 Hadoop 优化（包括：压缩、小文件、集群优化）





评分标准：5 分 必须画图，不画图 0 分。

一、Shuffle 机制

1）Map 方法之后 Reduce 方法之前这段处理过程叫 Shuffle

2）Map 方法之后，数据首先进入到分区方法，把数据标记好分区，然后把数据发送到环形缓冲区；环形缓冲区默认大小 100m，环形缓冲区达到 80%时，进行溢写；溢写前对数据进行排序，排序按照对 key 的索引进行字典顺序排序，排序的手段快排；溢写产生大量溢写文件，需要对溢写文件进行归并排序；对溢写的文件也可以进行 Combiner 操作，前提是

汇总操作，求平均值不行。最后将文件按照分区存储到磁盘，等待 Reduce 端拉取。

3）每个 Reduce 拉取 Map 端对应分区的数据。拉取数据后先存储到内存中，内存不够了，再存储到磁盘。拉取完所有数据后，采用归并排序将内存和磁盘中的数据都进行排序。在进入 Reduce 方法前，可以对数据进行分组操作。

二、Hadoop 优化

0）HDFS 小文件影响

（1）影响 NameNode 的寿命，因为文件元数据存储在 NameNode 的内存中

（2）影响计算引擎的任务数量，比如每个小的文件都会生成一个 Map 任务

1）数据输入小文件处理：

（1）合并小文件：对小文件进行归档（Har）、自定义 Inputformat 将小文件存储成 SequenceFile 文件。

（2）采用 ConbinFileInputFormat 来作为输入，解决输入端大量小文件场景。

（3）对于大量小文件 Job，可以开启 JVM 重用。

2）Map 阶段

（1）增大环形缓冲区大小。由 100m 扩大到 200m

（2）增大环形缓冲区溢写的比例。由 80%扩大到 90%

（3）减少对溢写文件的 merge 次数。

（4）不影响实际业务的前提下，采用 Combiner 提前合并，减少 I/O。

3）Reduce 阶段

（1）合理设置 Map 和 Reduce 数：两个都不能设置太少，也不能设置太多。太少，会导致 Task 等待，延长处理时间；太多，会导致 Map、Reduce 任务间竞争资源，造成处理超时等错误。

（2）设置 Map、Reduce 共存：调整 slowstart.completedmaps 参数，使 Map 运行到一定程度后，Reduce 也开始运行，减少 Reduce 的等待时间。

（3）规避使用 Reduce，因为 Reduce 在用于连接数据集的时候将会产生大量的网络消耗。

（4）增加每个 Reduce 去 Map 中拿数据的并行数

（5）集群性能可以的前提下，增大 Reduce 端存储数据内存的大小。

4）IO 传输

（1）采用数据压缩的方式，减少网络 IO 的的时间。安装 Snappy 和 LZOP 压缩编码器。

（2）使用 SequenceFile 二进制文件

5）整体

（1）MapTask 默认内存大小为 1G，可以增加 MapTask 内存大小为 4-5g

（2）ReduceTask 默认内存大小为 1G，可以增加 ReduceTask 内存大小为 4-5g

（3）可以增加 MapTask 的 cpu 核数，增加 ReduceTask 的 CPU 核数

（4）增加每个 Container 的 CPU 核数和内存大小

（5）调整每个 Map Task 和 Reduce Task 最大重试次数

评分标准：10 分 每大点 2 分

三、压缩

压缩格式	Hadoop 自带？	算法	文件扩展名	支持切分	换成压缩格式后，原来的程序是否需要修改
DEFLATE	是，直接使用	DEFLATE	.deflate	否	和文本处理一样，不需要修改
Gzip	是，直接使用	DEFLATE	.gz	否	和文本处理一样，不需要修改
bzip2	是，直接使用	bzip2	.bz2	是	和文本处理一样，不需要修改
LZO	否，需要安装	LZO	.lzo	是	需要建索引，还需要指定输入格式
Snappy	否，需要安装	Snappy	.snappy	否	和文本处理一样，不需要修改

提示：如果面试过程问起，我们一般回答压缩方式为 Snappy，特点速度快，缺点无法切分（可以回答在链式 MR 中，Reduce 端输出使用 bzip2 压缩，以便后续的 map 任务对数据进行 split）

评分标准：5 分 必须有 LZO 和 Snappy

四、切片机制

1）简单地按照文件的内容长度进行切片

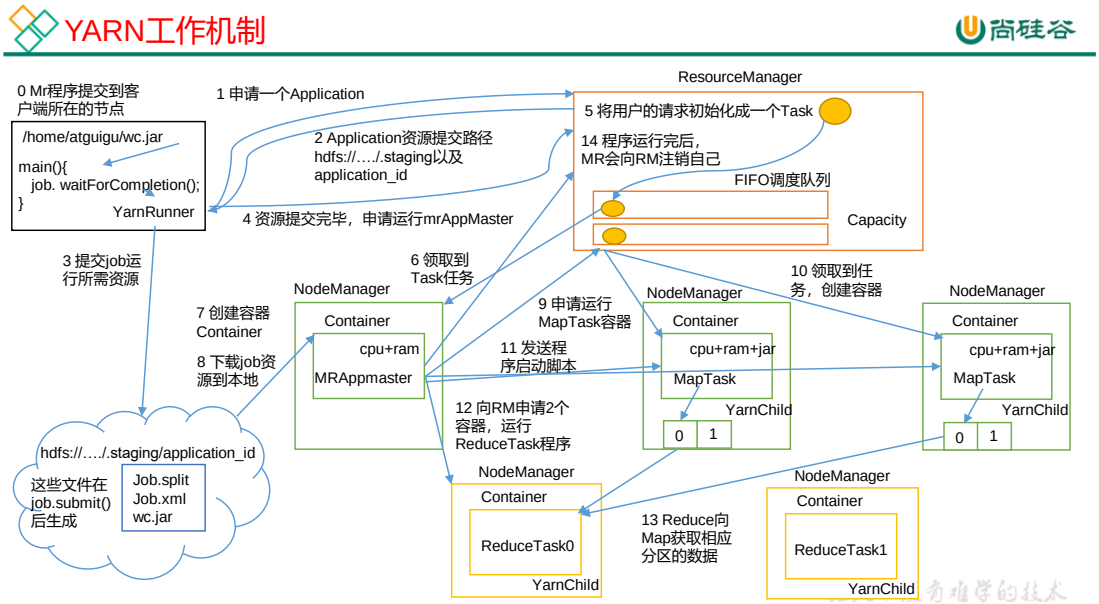
2）切片大小，默认等于 Block 大小

3）切片时不考虑数据集整体，而是逐个针对每一个文件单独切片

提示：切片大小公式：max(0,min(Long_max,blockSize))

评分标准：5 分

4.2.5 Yarn 的 Job 提交流程



评分标准：5 分

4.2.6 Yarn 的默认调度器、调度器分类、以及他们之间的区别

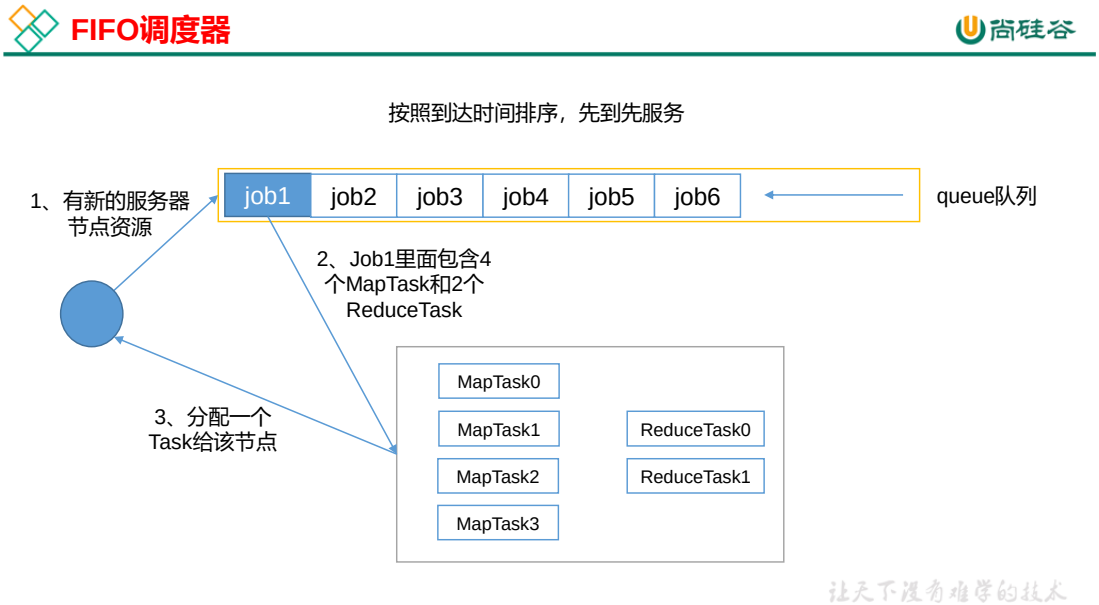
1）Hadoop 调度器重要分为三类：

FIFO 、Capacity Scheduler（容量调度器）和 Fair Sceduler（公平调度器）。

Hadoop2.7.2 默认的资源调度器是 容量调度器

2）区别：

FIFO 调度器：先进先出，同一时间队列中只有一个任务在执行。

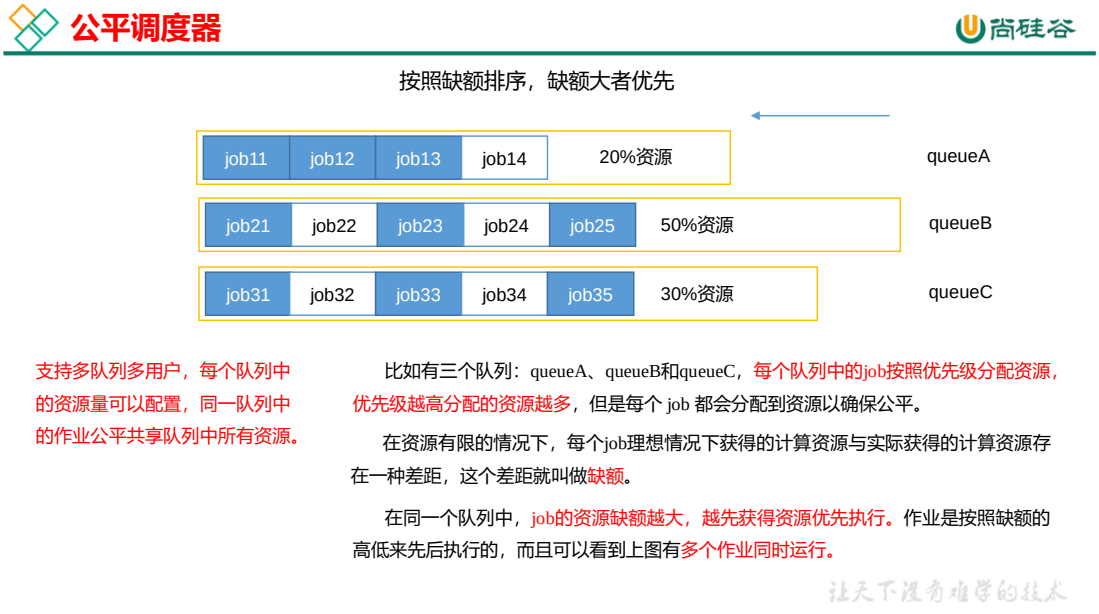


容量调度器：多队列；每个队列内部先进先出，同一时间队列中只有一个任务在执行。

队列的并行度为队列的个数。

公平调度器：多队列；每个队列内部按照缺额大小分配资源启动任务，同一时间队列中有多个任务执行。队列的并行度大于等于队列的个数。

3）一定要强调生产环境中不是使用的 FifoScheduler，面试的时侯会发现候选人大概了解这几种调度器的区别，但是问在生产环境用哪种，却说使用的 FifoScheduler（企业生产环境一定不会用这个调度的）



评分标准：5 分

4.2.7 项目经验之 LZO 压缩

Hadoop 默认不支持 LZO 压缩，如果需要支持 LZO 压缩，需要添加 jar 包，并在 hadoop

的 cores-site.xml 文件中添加相关压缩配置。

评分标准：5 分

4.2.8 Hadoop 参数调优

1）在 hdfs-site.xml 文件中配置多目录，最好提前配置好，否则更改目录需要重新启动集群

2）NameNode 有一个工作线程池，用来处理不同 DataNode 的并发心跳以及客户端并发的元数据操作。

dfs.namenode.handler.count=20 * log2(Cluster Size)，比如集群规模为 10 台时，此参数设置为 60

3）编辑日志存储路径 dfs.namenode.edits.dir 设置与镜像文件存储路径 dfs.namenode.name.dir 尽量分开，达到最低写入延迟

4）服务器节点上 YARN 可使用的物理内存总量，默认是 8192（MB），注意，如果你的节点内存资源不够 8GB，则需要调减小这个值，而 YARN 不会智能的探测节点的物理内存总量。yarn.nodemanager.resource.memory-mb

5）单个任务可申请的最多物理内存量，默认是 8192（MB）。yarn.scheduler.maximum-allocation-mb

评分标准：5 分 每条一分

4.2.9 项目经验之基准测试

搭建完 Hadoop 集群后需要对 HDFS 读写性能和 MR 计算能力测试。测试 jar 包在 hadoop 的 share 文件夹下。

评分标准：5 分

4.2.10 Hadoop 宕机

1）如果 MR 造成系统宕机。此时要控制 Yarn 同时运行的任务数，和每个任务申请的最大内存。调整参数：yarn.scheduler.maximum-allocation-mb（单个任务可申请的最多物理内存量，默认是 8192MB）

2）如果写入文件过量造成 NameNode 宕机。那么调高 Kafka 的存储大小，控制从 Kafka 到 HDFS 的写入速度。高峰期的时候用 Kafka 进行缓存，高峰期过去数据同步会自动跟上。

评分标准：5 分

4.3 Zookeeper 相关总结

4.3.1 选举机制

半数机制

评分标准：5 分

4.3.2 常用命令

ls、get、create

评分标准：5 分

4.4 Flume 相关总结

4.4.1 Flume 组成，Put 事务，Take 事务

1）Flume 组成，Put 事务，Take 事务

Taildir Source：断点续传、多目录。Flume1.6 以前需要自己自定义 Source 记录每次读取文件位置，实现断点续传。

File Channel：数据存储在磁盘，宕机数据可以保存。但是传输速率慢。适合对数据传输可靠性要求高的场景，比如，金融行业。

Memory Channel：数据存储在内存中，宕机数据丢失。传输速率快。适合对数据传输可靠性要求不高的场景，比如，普通的日志数据。

Kafka Channel：减少了 Flume 的 Sink 阶段，提高了传输效率。

Source 到 Channel 是 Put 事务

Channel 到 Sink 是 Take 事务

评分标准：5 分

4.4.2 Flume 拦截器

（1）拦截器注意事项

项目中自定义了：ETL 拦截器和区分类型拦截器。

采用两个拦截器的优缺点：优点，模块化开发和可移植性；缺点，性能会低一些

（2）自定义拦截器步骤

a）实现 Interceptor

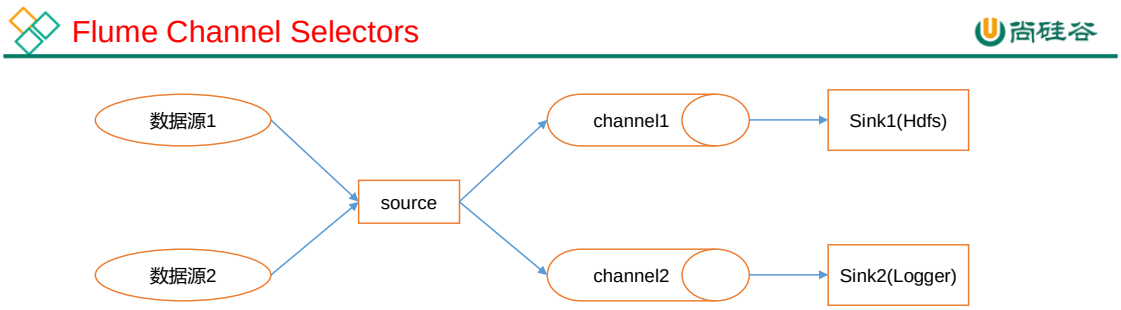
b）重写四个方法

- initialize 初始化
- public Event intercept(Event event) 处理单个 Event
- public List<Event> intercept(List<Event> events) 处理多个 Event，在这个方法中调用 Event intercept(Event event)
- close 方法

c）静态内部类，实现 Interceptor.Builder

评分标准：5 分

4.4.3 Flume Channel 选择器



Channel Selectors，可以让不同的项目日志通过不同的Channel到不同的Sink中去。官方文档上Channel Selectors 有两种类型:Replicating Channel Selector (default)和Multiplexing Channel Selector

这两种Selector的区别是:Replicating 会将source过来的events发往所有channel,而Multiplexing可以选择该发往哪些Channel。

评分标准：5 分

4.4.4 Flume 监控器

Ganglia

评分标准：5 分

4.4.5 Flume 采集数据会丢失吗?

不会，Channel 存储可以存储在 File 中，数据传输自身有事务。

评分标准：5 分

4.4.6 Flume 内存

开发中在 flume-env.sh 中设置 JVM heap 为 4G 或更高，部署在单独的服务器上（4 核 8

线程 16G 内存）

-Xmx 与-Xms 最好设置一致，减少内存抖动带来的性能影响，如果设置不一致容易导致频繁 fullgc。

评分标准：5 分

4.4.7 FileChannel 优化

通过配置 dataDirs 指向多个路径，每个路径对应不同的硬盘，增大 Flume 吞吐量。

官方说明如下：

```
Comma separated list of directories for storing log files. Using multiple directories on separate disks can improve file channel peformance
```

checkpointDir 和 backupCheckpointDir 也尽量配置在不同硬盘对应的目录中，保证 checkpoint 坏掉后，可以快速使用 backupCheckpointDir 恢复数据

评分标准：5 分

4.4.8 HDFS Sink 小文件处理

（1）HDFS 存入大量小文件，有什么影响？

元数据层面：每个小文件都有一份元数据，其中包括文件路径，文件名，所有者，所属组，权限，创建时间等，这些信息都保存在 Namenode 内存中。所以小文件过多，会占用 Namenode 服务器大量内存，影响 Namenode 性能和使用寿命

计算层面：默认情况下 MR 会对每个小文件启用一个 Map 任务计算，非常影响计算性能。同时也影响磁盘寻址时间。

（2）HDFS 小文件处理

官方默认的这三个参数配置写入 HDFS 后会产生小文件，hdfs.rollInterval、hdfs.rollSize、hdfs.rollCount

基于以上 hdfs.rollInterval=3600，hdfs.rollSize=134217728，hdfs.rollCount =0，hdfs.roundValue=10，hdfs.roundUnit= second 几个参数综合作用，效果如下：

（1）tmp 文件在达到 128M 时会滚动生成正式文件

（2）tmp 文件创建超 10 秒时会滚动生成正式文件

举例：在 2018-01-01 05:23 的时侯 sink 接收到数据，那会产生如下 tmp 文件：

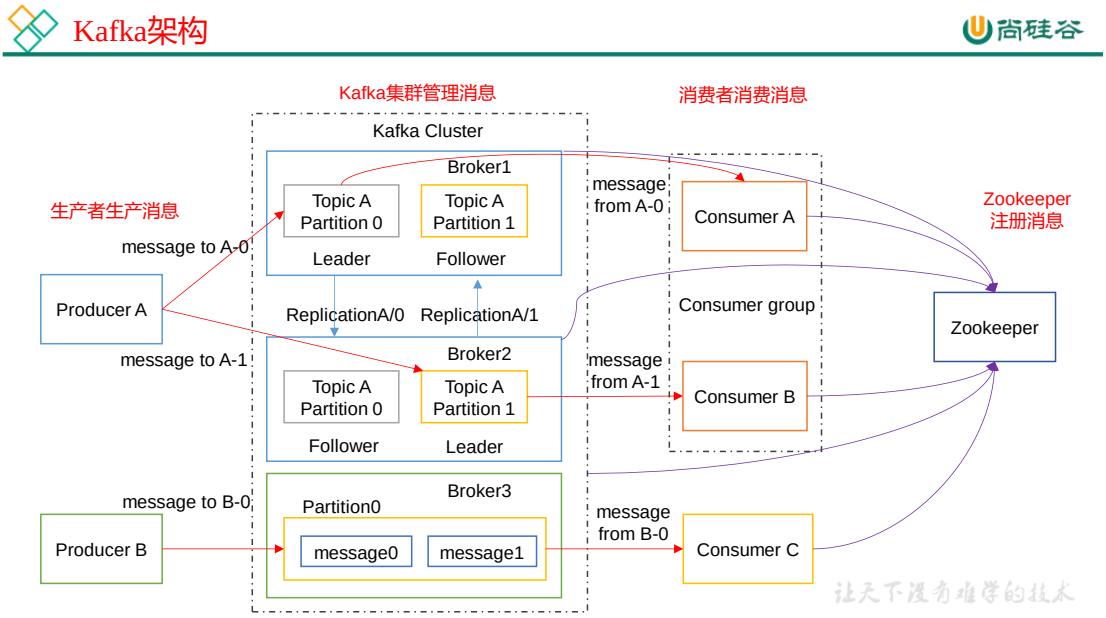
/atguigu/20180101/atguigu.201801010520.tmp

即使文件内容没有达到 128M，也会在 05:33 时滚动生成正式文件

评分标准：5 分

4.5 Kafka 相关总结*（以下每条 5 分）*

4.5.1 Kafka 架构



4.5.2 Kafka 压测

Kafka 官方自带压力测试脚本（kafka-consumer-perf-test.sh、kafka-producer-perf-test.sh）。Kafka 压测时，可以查看到哪个地方出现了瓶颈（CPU，内存，网络 IO）。一般都是网络 IO 达到瓶颈。

4.5.3 Kafka 的机器数量

Kafka 机器数量=2*（峰值生产速度*副本数/100）+1

4.5.4 Kafka 的日志保存时间

7 天

4.5.5 Kafka 的硬盘大小

每天的数据量*7 天

4.5.6 Kafka 监控

公司自己开发的监控器；

开源的监控器：KafkaManager、KafkaMonitor

4.5.7 Kakfa 分区数

分区数并不是越多越好，一般分区数不要超过集群机器数量。分区数越多占用内存越大（ISR 等），一个节点集中的分区也就越多，当它宕机的时候，对系统的影响也就越大。

分区数一般设置为：3-10 个

4.5.8 副本数设定

一般我们设置成 2 个或 3 个，很多企业设置为 2 个。

4.5.9 多少个 Topic

通常情况：多少个日志类型就多少个 Topic。也有对日志类型进行合并的。

4.5.10 Kafka 丢不丢数据

Ack=0，相当于异步发送，消息发送完毕即 offset 增加，继续生产。

Ack=1，leader 收到 leader replica 对一个消息的接受 ack 才增加 offset，然后继续生产。

Ack=-1，leader 收到所有 replica 对一个消息的接受 ack 才增加 offset，然后继续生产。

4.5.11 Kafka 的 ISR 副本同步队列

ISR（In-Sync Replicas），副本同步队列。ISR 中包括 Leader 和 Follower。如果 Leader 进程挂掉，会在 ISR 队列中选择一个服务作为新的 Leader。有 replica.lag.max.messages（延迟条数）和 replica.lag.time.max.ms（延迟时间）两个参数决定一台服务是否可以加入 ISR 副本队列，在 0.10 版本移除了 replica.lag.max.messages 参数，防止服务频繁的进去队列。

任意一个维度超过阈值都会把 Follower 剔除出 ISR，存入 OSR（Outof-Sync Replicas）列表，新加入的 Follower 也会先存放在 OSR 中。

4.5.12 Kafka 分区分配策略

在 Kafka 内部存在两种默认的分区分配策略：Range 和 RoundRobin。

Range 是默认策略。Range 是对每个 Topic 而言的（即一个 Topic 一个 Topic 分），首先对同一个 Topic 里面的分区按照序号进行排序，并对消费者按照字母顺序进行排序。然后用 Partitions 分区的个数除以消费者线程的总数来决定每个消费者线程消费几个分区。如果除不尽，那么前面几个消费者线程将会多消费一个分区。

例如：我们有 10 个分区，两个消费者（C1，C2），3 个消费者线程，10 / 3 = 3 而且除不尽。

C1-0 将消费 0, 1, 2, 3 分区

C2-0 将消费 4, 5, 6 分区

C2-1 将消费 7, 8, 9 分区

RoundRobin：前提：同一个 Consumer Group 里面的所有消费者的 num.streams（消费者消费线程数）必须相等；每个消费者订阅的主题必须相同。

第一步：将所有主题分区组成 TopicAndPartition 列表，然后对 TopicAndPartition 列表按照 hashCode 进行排序，最后按照轮询的方式发给每一个消费线程。

4.5.13 Kafka 中数据量计算

每天总数据量 100g，每天产生 1 亿条日志， 10000 万/24/60/60=1150 条/每秒钟

平均每秒钟：1150 条

低谷每秒钟：400 条

高峰每秒钟：1150 条*（2-20 倍）=2300 条-23000 条

每条日志大小：0.5k-2k

每秒多少数据量：2.3M-20MB

4.5.14 Kafka 挂掉

1）Flume 记录

2）日志有记录

3）短期没事

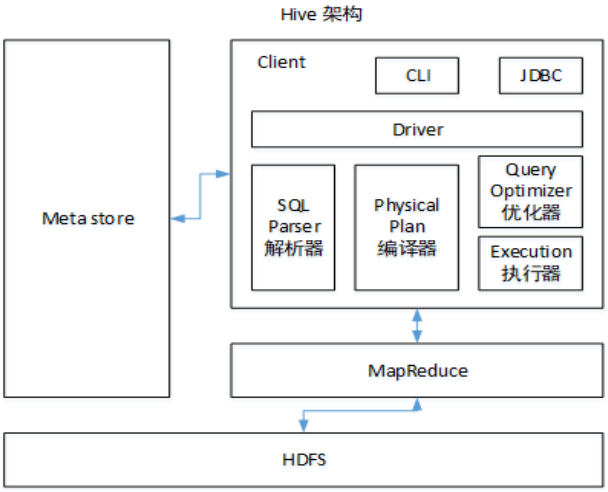
4.5.15 Kafka 消息数据积压，Kafka 消费能力不足怎么处理？

1）如果是 Kafka 消费能力不足，则可以考虑增加 Topic 的分区数，并且同时提升消费组的消费者数量，消费者数=分区数。（两者缺一不可）

2）如果是下游的数据处理不及时：提高每批次拉取的数量。批次拉取数据过少（拉取数据/处理时间<生产速度），使处理的数据小于生产的数据，也会造成数据积压。

4.6 Hive 总结

4.6.1 Hive 的架构



4.6.2 Hive 和数据库比较

Hive 和数据库除了拥有类似的查询语言，再无类似之处。

1）数据存储位置

Hive 存储在 HDFS 。数据库将数据保存在块设备或者本地文件系统中。

2）数据更新

Hive 中不建议对数据的改写。而数据库中的数据通常是需要经常进行修改的，

3）执行延迟

Hive 执行延迟较高。数据库的执行延迟较低。当然，这个是有条件的，即数据规模较小，当数据规模大到超过数据库的处理能力的时候，Hive 的并行计算显然能体现出优势。

4）数据规模

Hive 支持很大规模的数据计算；数据库可以支持的数据规模较小。

4.6.3 内部表和外部表

1）管理表：当我们删除一个管理表时，Hive 也会删除这个表中数据。管理表不适合和其他工具共享数据。

2）外部表：删除该表并不会删除掉原始数据，删除的是表的元数据

4.6.4 4 个 By 区别

1）Sort By：分区内有序；

2）Order By：全局排序，只有一个 Reducer；

3）Distrbute By：类似 MR 中 Partition，进行分区，结合 sort by 使用。

4）Cluster By：当 Distribute by 和 Sorts by 字段相同时，可以使用 Cluster by 方式。Cluster by 除了具有 Distribute by 的功能外还兼具 Sort by 的功能。但是排序只能是升序排序，不能指定排序规则为 ASC 或者 DESC。

4.6.5 窗口函数

RANK() 排序相同时会重复，总数不会变

DENSE_RANK() 排序相同时会重复，总数会减少

ROW_NUMBER() 会根据顺序计算

1）OVER()：指定分析函数工作的数据窗口大小，这个数据窗口大小可能会随着行的变而变化

2）CURRENT ROW：当前行

3）n PRECEDING：往前 n 行数据

4） n FOLLOWING：往后 n 行数据

5）UNBOUNDED：起点，UNBOUNDED PRECEDING 表示从前面的起点，UNBOUNDED FOLLOWING 表示到后面的终点

6） LAG(col,n)：往前第 n 行数据

7）LEAD(col,n)：往后第 n 行数据

8） NTILE(n)：把有序分区中的行分发到指定数据的组中，各个组有编号，编号从 1 开始，对于每一行，NTILE 返回此行所属的组的编号。注意：n 必须为 int 类型。

4.6.6 自定义 UDF、UDTF

在项目中是否自定义过 UDF、UDTF 函数，以及用他们处理了什么问题，及自定义步骤？

1）自定义过。

2）用 UDF 函数解析公共字段；用 UDTF 函数解析事件字段。

自定义 UDF：继承 UDF，重写 evaluate 方法

自定义 UDTF：继承自 GenericUDTF，重写 3 个方法：initialize(自定义输出的列名和类型)，process（将结果返回 forward(result)），close

为什么要自定义 UDF/UDTF，因为自定义函数，可以自己埋点 Log 打印日志，出错或

者数据异常，方便调试.

4.6.7 Hive 优化

- **1）MapJoin**

如果不指定 MapJoin 或者不符合 MapJoin 的条件，那么 Hive 解析器会将 Join 操作转换成 Common Join，即：在 Reduce 阶段完成 join。容易发生数据倾斜。可以用 MapJoin 把小表全部加载到内存在 map 端进行 join，避免 reducer 处理。

- **2）行列过滤**

列处理：在 SELECT 中，只拿需要的列，如果有，尽量使用分区过滤，少用 SELECT *。

行处理：在分区剪裁中，当使用外关联时，如果将副表的过滤条件写在 Where 后面，那么就会先全表关联，之后再过滤。

- **3）采用分桶技术**

- **4）采用分区技术**

- **5）合理设置 Map 数**

（1）通常情况下，作业会通过 input 的目录产生一个或者多个 map 任务。

主要的决定因素有：input 的文件总个数，input 的文件大小，集群设置的文件块大小。

（2）是不是 map 数越多越好？

答案是否定的。如果一个任务有很多小文件（远远小于块大小 128m），则每个小文件也会被当做一个块，用一个 map 任务来完成，而一个 map 任务启动和初始化的时间远远大于逻辑处理的时间，就会造成很大的资源浪费。而且，同时可执行的 map 数是受限的。

（3）是不是保证每个 map 处理接近 128m 的文件块，就高枕无忧了？

答案也是不一定。比如有一个 127m 的文件，正常会用一个 map 去完成，但这个文件只有一个或者两个小字段，却有几千万的记录，如果 map 处理的逻辑比较复杂，用一个 map 任务去做，肯定也比较耗时。

针对上面的问题 2 和 3，我们需要采取两种方式来解决：即减少 map 数和增加 map 数；

- **6）小文件进行合并**

在 Map 执行前合并小文件，减少 Map 数：CombineHiveInputFormat 具有对小文件进行合并的功能（系统默认的格式）。HiveInputFormat 没有对小文件合并功能。

- **7）合理设置 Reduce 数**

Reduce 个数并不是越多越好

（1）过多的启动和初始化 Reduce 也会消耗时间和资源；

（2）另外，有多少个 Reduce，就会有多少个输出文件，如果生成了很多个小文件，那么如果这些小文件作为下一个任务的输入，则也会出现小文件过多的问题；

在设置 Reduce 个数的时候也需要考虑这两个原则：处理大数据量利用合适的 Reduce 数；使单个 Reduce 任务处理数据量大小要合适；

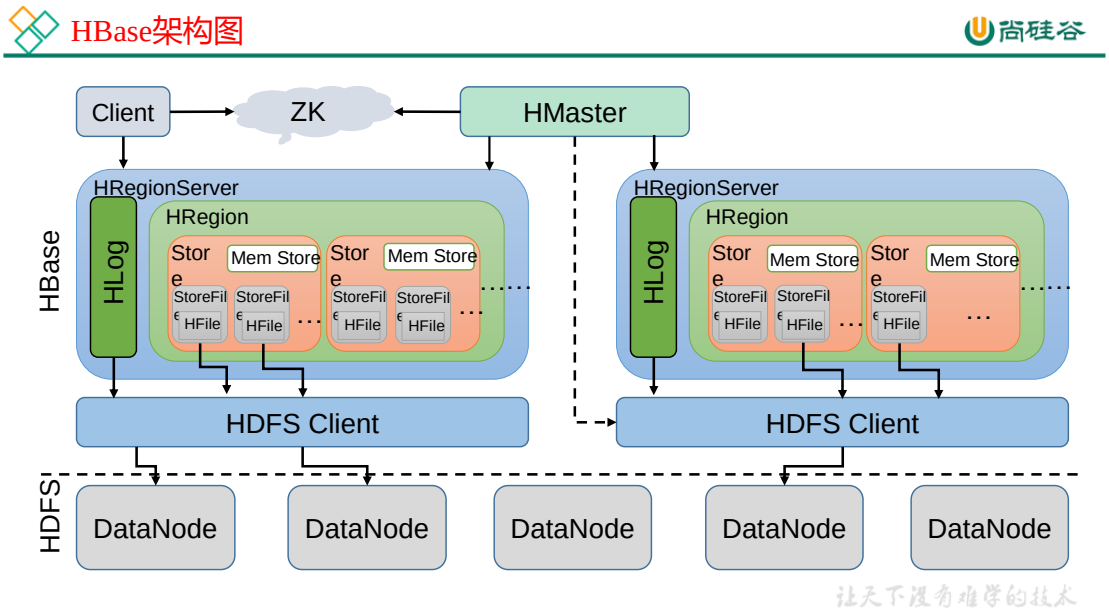
- **8）常用参数**

// 输出合并小文件

```
SET hive.merge.mapfiles = true; -- 默认 true，在 map-only 任务结束时合并小文件
SET hive.merge.mapredfiles = true; -- 默认 false，在 map-reduce 任务结束时合并小文件
SET hive.merge.size.per.task = 268435456; -- 默认 256M
SET hive.merge.smallfiles.avgsize = 16777216; -- 当输出文件的平均大小小于该值时，启动一个独立的 map-reduce 任务进行文件 merge
```

4.7 HBase 总结

4.7.1 HBase 存储结构



4.7.2 rowkey 设计原则

1）rowkey 长度原则

2）rowkey 散列原则

3）rowkey 唯一原则

4.7.3 RowKey 如何设计

（1）生成随机数、hash、散列值

（2）字符串反转

4.7.4 Phoenix 二级索引

phoenix的安装与
二级索引建立.doc

4.8 Sqoop 参数

```
/opt/module/sqoop/bin/sqoop import \
--connect \
--username \
--password \
--target-dir \
--delete-target-dir \
--num-mappers \
--fields-terminated-by   \
--query   "$2" ' and $CONDITIONS;'
```

4.8.1 Sqoop 导入导出 Null 存储一致性问题

Hive 中的 Null 在底层是以“\N”来存储，而 MySQL 中的 Null 在底层就是 Null，为了保证数据两端的一致性。在导出数据时采用--input-null-string 和--input-null-non-string 两个参数。导入数据时采用--null-string 和--null-non-string。

4.8.2 Sqoop 数据导出一致性问题

1）场景 1：如 Sqoop 在导出到 Mysql 时，使用 4 个 Map 任务，过程中有 2 个任务失败，那此时 MySQL 中存储了另外两个 Map 任务导入的数据，此时老板正好看到了这个报表数据。而开发工程师发现任务失败后，会调试问题并最终将全部数据正确的导入 MySQL，那后面老板再次看报表数据，发现本次看到的数据与之前的不一致，这在生产环境是不允许的。

官网：http://sqoop.apache.org/docs/1.4.6/SqoopUserGuide.html

```
Since Sqoop breaks down export process into multiple
transactions, it is possible that a failed export job may
result in partial data being committed to the database. This
can further lead to subsequent jobs failing due to insert
collisions in some cases, or lead to duplicated data in
others. You can overcome this problem by specifying a staging
table via the --staging-table option which acts as an
auxiliary table that is used to stage exported data. The
staged data is finally moved to the destination table in a
```

```
single transaction.
```

–staging-table 方式

```
sqoop export --connect
jdbc:mysql://192.168.137.10:3306/user_behavior --username root
--password 123456 --table app_cource_study_report --columns
watch_video_cnt,complete_video_cnt,dt --fields-terminated-by
"\t" --export-dir
"/user/hive/warehouse/tmp.db/app_cource_study_analysis_${day}"
--staging-table app_cource_study_report_tmp --clear-staging-
table --input-null-string '\N'
```

2）场景 2：设置 map 数量为 1 个（不推荐，面试官想要的答案不只这个）

多个 Map 任务时，采用–staging-table 方式，仍然可以解决数据一致性问题。

4.8.3 Sqoop 底层运行的任务是什么

只有 Map 阶段，没有 Reduce 阶段的任务。

4.8.4 Sqoop 数据导出的时候一次执行多长时间

Sqoop 任务 5 分钟-2 个小时的都有。取决于数据量。

4.9 Scala

4.9.1 元组

1）元组的创建

```
val tuple1 = (1, 2, 3, "heiheihei")
println(tuple1)
```

2）元组数据的访问，注意元组元素的访问有下划线，并且访问下标从 1 开始，而不是 0

```
val value1 = tuple1._4
println(value1)
```

3）元组的遍历

```
方式 1：
for (elem <- tuple1.productIterator ) {
  print(elem)
}
println()
方式 2：
tuple1.productIterator.foreach(i => println(i))
tuple1.produIterator.foreach(print(_))
```

4.9.2 隐私转换

隐式转换函数是以 implicit 关键字声明的带有单个参数的函数。这种函数将会自动应用，将值从一种类型转换为另一种类型。

```
implicit def a(d: Double) = d.toInt
//不加上边这句你试试
val i1: Int = 3.5
```

```
println(i1)
```

4.9.3 函数式编程理解

1）Scala 中函数的地位：一等公民

2）Scala 中的匿名函数(函数字面量)

3）Scala 中的高阶函数

4）Scala 中的闭包

5）Scala 中的部分应用函数

6）Scala 中的柯里化函数

4.9.4 样例类

case class Person(name:String,age:Int)

一 般使用在 ds=df.as[Person]

4.9.5 柯里化

函数编程中，接受多个参数的函数都可以转化为接受单个参数的函数，这个转化过程就叫柯里化，柯里化就是证明了函数只需要一个参数而已。其实我们刚的学习过程中，已经涉及到了柯里化操作，所以这也印证了，柯里化就是以函数为主体这种思想发展的必然产生的结果。

1）柯里化的示例

```
def mul(x: Int, y: Int) = x * y
println(mul(10, 10))
def mulCurry(x: Int) = (y: Int) => x * y
println(mulCurry(10)(9))
def mulCurry2(x: Int)(y:Int) = x * y
println(mulCurry2(10)(8))
```

2）柯里化的应用

比较两个字符串在忽略大小写的情况下是否相等，注意，这里是两个任务：

- 全部转大写（或小写）
- 比较是否相等

针对这两个操作，我们用一个函数去处理的思想，其实无意间也变成了两个函数处理的思想。示例如下：

```
val a = Array("Hello", "World")
val b = Array("hello", "world")
println(a.corresponds(b)(_.equalsIgnoreCase(_)))
其中 corresponds 函数的源码如下：
def corresponds[B](that: GenSeq[B])(p: (A,B) => Boolean): Boolean = {

val i = this.iterator
```

```
val j = that.iterator

while (i.hasNext && j.hasNex  t)

  if (!p(i.next(), j.next()    ))

    return fals
!i.hasNext && !j.hasNext
}
```

尖叫提示：不要设立柯里化存在义这样的命题，柯里化，是面向函数思想的必然产生结果。

4.9.6 闭包

一个函数把外部的那些不属于自己的对象也包含(闭合)进来。

案例 1：

```
def minusxy(x: Int) = (y: Int) => x - y
这就是一个闭包：
1）匿名函数(y: Int) => x -y 嵌套在 minusxy 函数中。
2）匿名函数(y: Int) => x -y 使用了该匿名函数之外的变量 x
3）函数 minusxy 返回了引用了局部变量的匿名函数
```

案例 2

```
def minusxy(x: Int) = (y: Int) => x - y
val f1 = minusxy(10)
val f2 = minusxy(10)
println(f1(3) + f2(3))
此处 f1,f2 这两个函数就叫闭包。
```

4.9.7 Some、None、Option 的正确使用

```
val map = Map("Tom"-> 23)
map("Jack") // 抛出异常 java.util.NoSuchElementException: key not found: Jack
map.get("Jack") // None
map("Tom") // 23
map.get("Tom") // Some(23)

使用模式匹配取出最后结果
val optionAge = map.get("Tom")
val age = optionAge match {
 case Some(x) => optionAge.get
 case None => 0
}
```

4.10 Spark

4.10.1 Spark 有几种部署方式？请分别简要论述

1）Local:运行在一台机器上，通常是练手或者测试环境。

2）Standalone:构建一个基于 Mster+Slaves 的资源调度集群，Spark 任务提交给 Master 运行。是 Spark 自身的一个调度系统。

3）Yarn: Spark 客户端直接连接 Yarn，不需要额外构建 Spark 集群。有 yarn-client 和 yarn-cluster 两种模式，主要区别在于：Driver 程序的运行节点。

4）Mesos：国内大环境比较少用。

4.10.2 Spark 任务使用什么进行提交，javaEE 界面还是脚本

Shell 脚本。

4.10.3 Spark 提交作业参数（重点）

参考答案：

https://blog.csdn.net/gamer_gyt/article/details/79135118

1）在提交任务时的几个重要参数

executor-cores —— 每个 executor 使用的内核数，默认为 1，官方建议 2-5 个，我们企业是 4 个

num-executors —— 启动 executors 的数量，默认为 2

executor-memory —— executor 内存大小，默认 1G

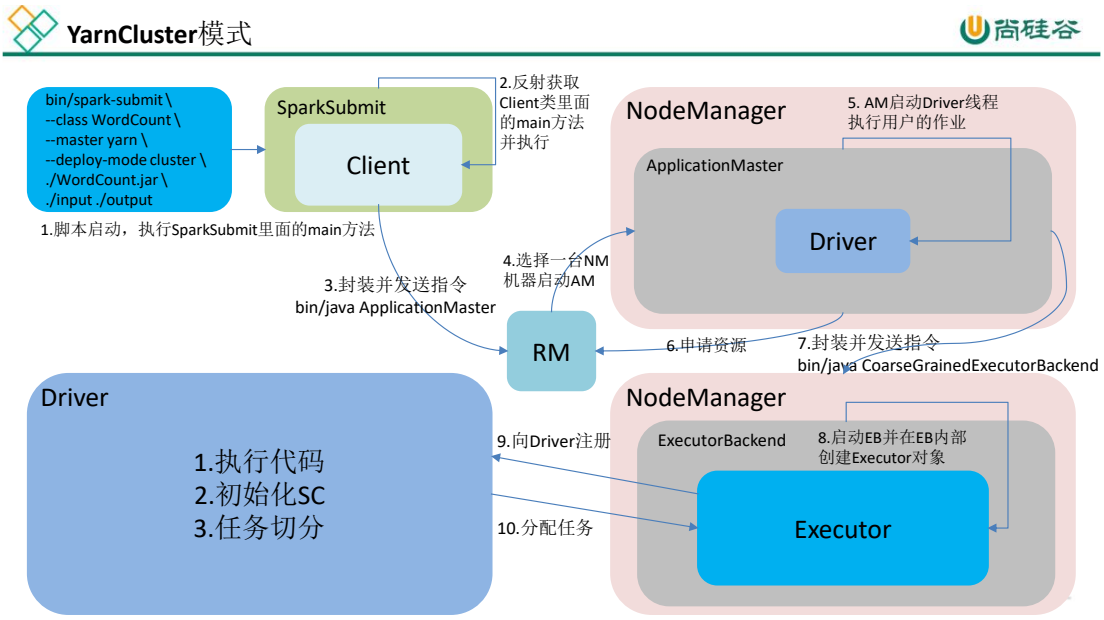
driver-cores —— driver 使用内核数，默认为 1

driver-memory —— driver 内存大小，默认 512M

2）边给一个提交任务的样式

```
spark-submit \
  --master local[5]   \
  --driver-cores 2    \
  --driver-memory 8g \
  --executor-cores 4 \
  --num-executors 10 \
  --executor-memory 8g \
  --class PackageName.ClassName XXXX.jar \
  --name "Spark Job Name" \
  InputPath          \
  OutputPath
```

4.10.4 简述 Spark 的架构与作业提交流程（画图讲解，注明各个部分的作用）（重点）



4.10.5 如何理解 Spark 中的血统概念（RDD）（笔试重点）

RDD 在 Lineage 依赖方面分为两种 Narrow Dependencies 与 Wide Dependencies 用来解决数据容错时的高效性以及划分任务时候起到重要作用。

4.10.6 简述 Spark 的宽窄依赖，以及 Spark 如何划分 stage，每个 stage 又根据什么决定 task 个数？（笔试重点）

Stage：根据 RDD 之间的依赖关系的不同将 Job 划分成不同的 Stage，遇到一个宽依赖则划分一个 Stage。

Task：Stage 是一个 TaskSet，将 Stage 根据分区数划分成一个个的 Task。

4.10.7 请列举 Spark 的 transformation 算子（不少于 8 个），并简述功能（重点）

1）map（func）：返回一个新的 RDD，该 RDD 由每一个输入元素经过 func 函数转换后组成.

2）mapPartitions(func)：类似于 map，但独立地在 RDD 的每一个分片上运行，因此在类型为 T 的 RD 上运行时，func 的函数类型必须是 Iterator[T] => Iterator[U]。假设有 N 个

元素，有 M 个分区，那么 map 的函数的将被调用 N 次,而 mapPartitions 被调用 M 次,一个函数一次处理所有分区。

3）reduceByKey（func，[numTask]）：在一个(K,V)的 RDD 上调用，返回一个(K,V)的 RDD，使用定的 reduce 函数，将相同 key 的值聚合到一起，reduce 任务的个数可以通过第二个可选的参数来设置。

4）aggregateByKey (zeroValue:U,[partitioner: Partitioner]) (seqOp: (U, V) => U,combOp: (U, U) => U: 在 kv 对的 RDD 中，，按 key 将 value 进行分组合并，合并时，将每个 value 和初始值作为 seq 函数的参数，进行计算，返回的结果作为一个新的 kv 对，然后再将结果按照 key 进行合并，最后将每个分组的 value 传递给 combine 函数进行计算（先将前两个 value 进行计算，将返回结果和下一个 value 传给 combine 函数，以此类推），将 key 与计算结果作为一个新的 kv 对输出。

5）combineByKey(createCombiner: V=>C, mergeValue: (C, V) =>C, mergeCombiners: (C, C) =>C):

对相同 K，把 V 合并成一个集合。

1.createCombiner: combineByKey() 会遍历分区中的所有元素，因此每个元素的键要么还没有遇到过，要么就和之前的某个元素的键相同。如果这是一个新的元素,combineByKey()会使用一个叫作 createCombiner()的函数来创建那个键对应的累加器的初始值

2.mergeValue: 如果这是一个在处理当前分区之前已经遇到的键，它会使用 mergeValue()方法将该键的累加器对应的当前值与这个新的值进行合并

3.mergeCombiners: 由于每个分区都是独立处理的， 因此对于同一个键可以有多个累加器。如果有两个或者更多的分区都有对应同一个键的累加器， 就需要使用用户提供的 mergeCombiners() 方法将各个分区的结果进行合并。

…

根据自身情况选择比较熟悉的算子加以介绍。

4.10.8 请列举 Spark 的 action 算子（不少于 6 个），并简述功能（重点）

1）reduce：

2）collect:

3）first：

4）take：

5）aggregate：

6）countByKey：

7）foreach：

8）saveAsTextFile：

4.10.9 请列举会引起 Shuffle 过程的 Spark 算子，并简述功能。

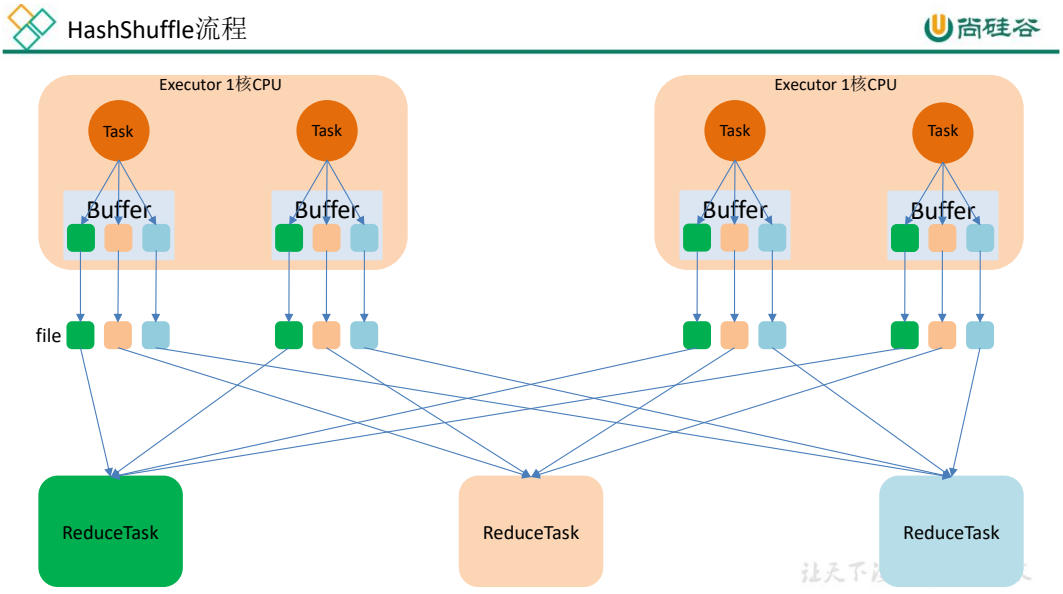
reduceBykey：

groupByKey：

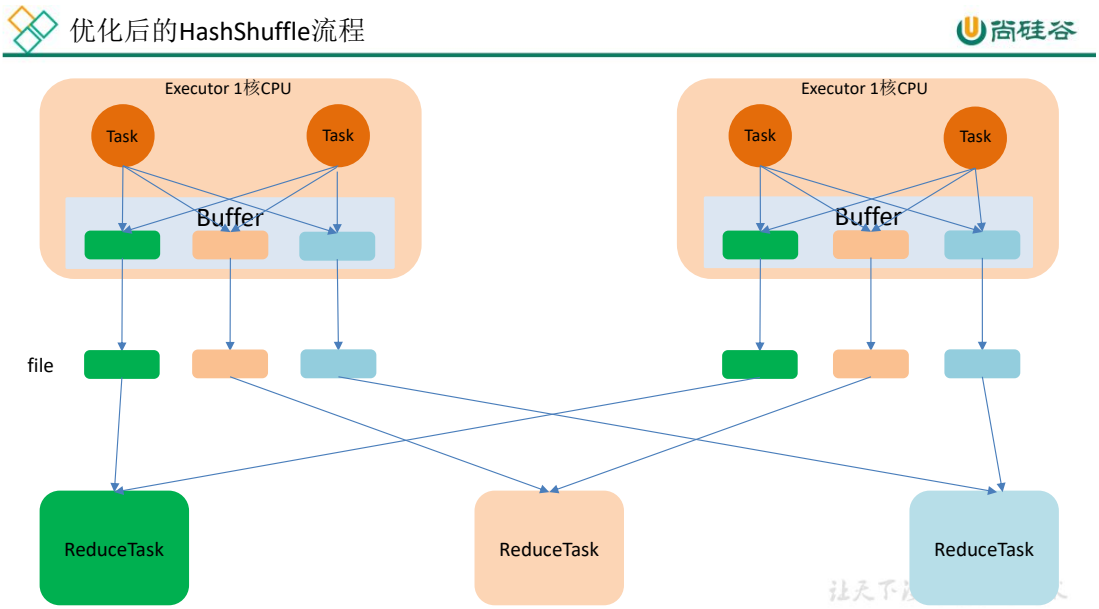
…ByKey:

4.10.10 简述 Spark 的两种核心 Shuffle（HashShuffle 与 SortShuffle）的工作流程（包括未优化的 HashShuffle、优化的 HashShuffle、普通的 SortShuffle 与 bypass 的 SortShuffle）（重点）

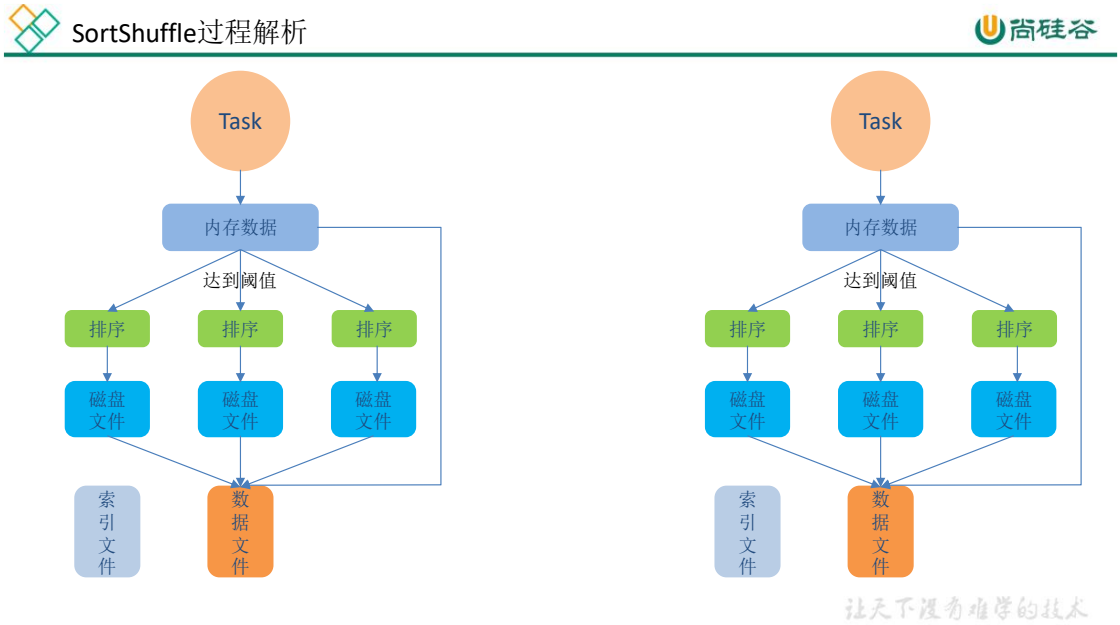
未经优化的 HashShuffle：



优化后的 Shuffle：



普通的 SortShuffle：



当 shuffle read task 的数量小于等于 spark.shuffle.sort。

bypassMergeThreshold 参数的值时（默认为 200），就会启用 bypass 机制。

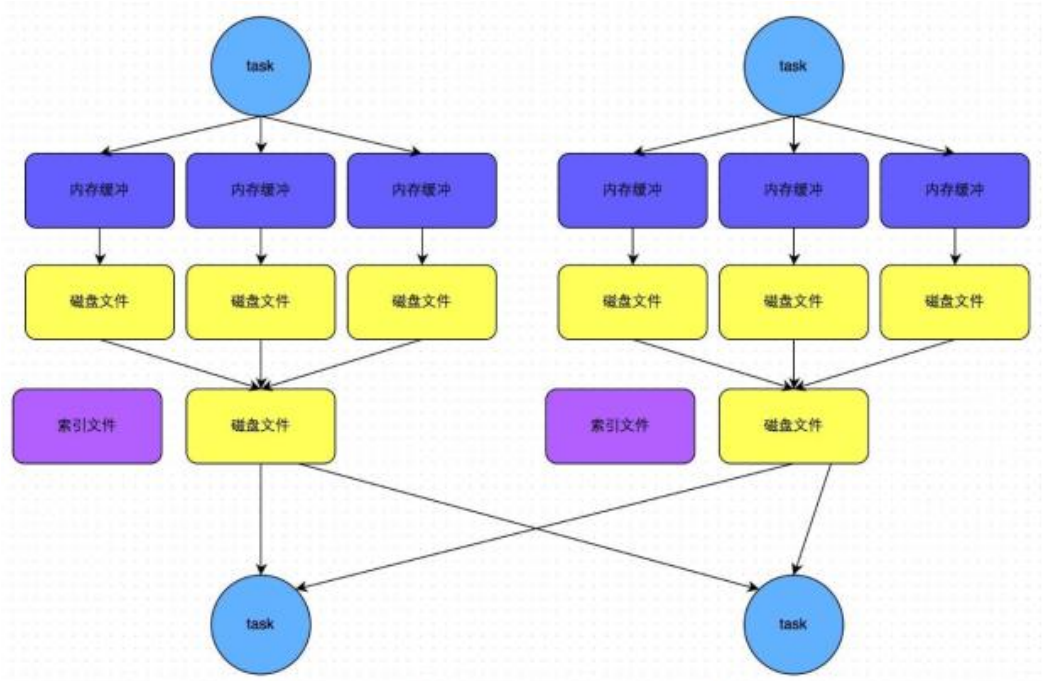


图 6-5 bypass 运行机制的 SortShuffleManager 工作原理

4.10.11 Spark 常用算子 reduceByKey 与 groupByKey 的区别，哪一种更具优势？（重点）

reduceByKey：按照 key 进行聚合，在 shuffle 之前有 combine（预聚合）操作，返回结果是 RDD[k,v]。

groupByKey：按照 key 进行分组，直接进行 shuffle。

开发指导：reduceByKey 比 groupByKey，建议使用。但是需要注意是否会影响业务逻辑。

4.10.12 Repartition 和 Coalesce 关系与区别

1）关系：

两者都是用来改变 RDD 的 partition 数量的，repartition 底层调用的就是 coalesce 方法：coalesce(numPartitions, shuffle = true)

2）区别：

repartition 一定会发生 shuffle，coalesce 根据传入的参数来判断是否发生 shuffle

一般情况下增大 rdd 的 partition 数量使用 repartition，减少 partition 数量时使用 coalesce

4.10.13 分别简述 Spark 中的缓存机制（cache 和 persist）与 checkpoint 机制，并指出两者的区别与联系

都是做 RDD 持久化的

cache:内存，不会截断血缘关系，使用计算过程中的数据缓存。

checkpoint：磁盘，截断血缘关系，在 ck 之前必须没有任何任务提交才会生效，ck 过程会额外提交一次任务。

4.10.14 简述 Spark 中共享变量（广播变量和累加器）的基本原理与用途。（重点）

累加器（accumulator）是 Spark 中提供的一种分布式的变量机制，其原理类似于 mapreduce，即分布式的改变，然后聚合这些改变。累加器的一个常见用途是在调试时对作业执行过程中的事件进行计数。而广播变量用来高效分发较大的对象。

共享变量出现的原因：

通常在向 Spark 传递函数时，比如使用 map() 函数或者用 filter() 传条件时，可以使用驱动器程序中定义的变量，但是集群中运行的每个任务都会得到这些变量的一份新的副本，更新这些副本的值也不会影响驱动器中的对应变量。

Spark 的两个共享变量，累加器与广播变量，分别为结果聚合与广播这两种常见的通信模式突破了这一限制。

4.10.15 当 Spark 涉及到数据库的操作时，如何减少 Spark 运行中的数据库连接数？

使用 foreachPartition 代替 foreach，在 foreachPartition 内获取数据库的连接。

4.10.16 简述 SparkSQL 中 RDD、DataFrame、DataSet 三者的区别与联系？（笔试重点）

1）RDD

优点:

编译时类型安全

编译时就能检查出类型错误

面向对象的编程风格

直接通过类名点的方式来操作数据

缺点:

序列化和反序列化的性能开销

无论是集群间的通信，还是 IO 操作都需要对对象的结构和数据进行序列化和反序列化。

GC 的性能开销 ，频繁的创建和销毁对象，势必会增加 GC

2）DataFrame

DataFrame 引入了 schema 和 off-heap

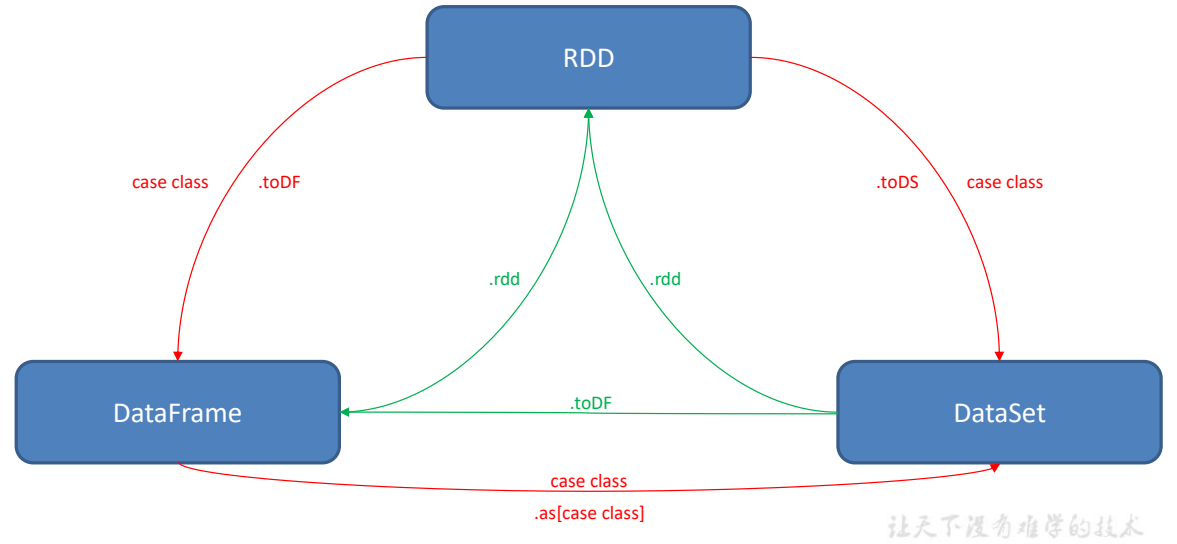
schema ： RDD 每一行的数据，结构都是一样的，这个结构就存储在 schema 中。 Spark 通过 schema 就能够读懂数据，因此在通信和 IO 时就只需要序列化和反序列化数据，而结构的部分就可以省略了。

3）DataSet

DataSet 结合了 RDD 和 DataFrame 的优点，并带来的一个新的概念 Encoder。

当序列化数据时，Encoder 产生字节码与 off-heap 进行交互，能够达到按需访问数据的效果，而不用反序列化整个对象。Spark 还没有提供自定义 Encoder 的 API，但是未来会加入。

三者之间的转换：



4.10.17 SparkSQL 中 join 操作与 left join 操作的区别？

join 和 sql 中的 inner join 操作很相似，返回结果是前面一个集合和后面一个集合中匹配成功的，过滤掉关联不上的。

leftJoin 类似于 SQL 中的左外关联 left outer join，返回结果以第一个 RDD 为主，关联不上的记录为空。

部分场景下可以使用 left semi join 替代 left join：

因为 left semi join 是 in(keySet) 的关系，**遇到右表重复记录，左表会跳过,性能更高**，而 left join 则会一直遍历。但是 left semi join 中最后 select 的结果中只许出现左表中的列名，因为右表只有 join key 参与关联计算了

4.10.18 SparkStreaming 有哪几种方式消费 Kafka 中的数据，它们之间的区别是什么？

一、基于 Receiver 的方式

这种方式使用 Receiver 来获取数据。Receiver 是使用 Kafka 的高层次 Consumer API 来实现的。receiver 从 Kafka 中获取的数据都是存储在 Spark Executor 的内存中的（如果突然数据暴增，大量 batch 堆积，很容易出现内存溢出的问题），然后 Spark Streaming 启动的 job 会去处理那些数据。

然而，在默认的配置下，这种方式可能会因为底层的失败而丢失数据。如果要启用高可靠机制，让数据零丢失，就必须启用 Spark Streaming 的预写日志机制（Write Ahead Log，WAL）。该机制会同步地将接收到的 Kafka 数据写入分布式文件系统（比如 HDFS）上的预写日志中。所以，即使底层节点出现了失败，也可以使用预写日志中的数据进行恢复。

需要注意的要点

二、基于 Direct 的方式

这种新的不基于 Receiver 的直接方式，是在 Spark 1.3 中引入的，从而能够确保更加健壮的机制。替代掉使用 Receiver 来接收数据后，这种方式会周期性地查询 Kafka，来获得每个 topic+partition 的最新的 offset，从而定义每个 batch 的 offset 的范围。当处理数据的 job 启动时，就会使用 Kafka 的简单 consumer api 来获取 Kafka 指定 offset 范围的数据。

优点如下：

简化并行读取：如果要读取多个 partition，不需要创建多个输入 DStream 然后对它们进行 union 操作。Spark 会创建跟 Kafka partition 一样多的 RDD partition，并且会并行从 Kafka 中读取数据。所以在 Kafka partition 和 RDD partition 之间，有一个一对一的映射关系。

高性能：如果要保证零数据丢失，在基于 receiver 的方式中，需要开启 WAL 机制。这种方式其实效率低下，因为数据实际上被复制了两份，Kafka 自己本身就有高可靠的机制，会对数据复制一份，而这里又会复制一份到 WAL 中。而基于 direct 的方式，不依赖 Receiver，不需要开启 WAL 机制，只要 Kafka 中作了数据的复制，那么就可以通过 Kafka 的副本进行恢复。

一次且仅一次的事务机制。

三、对比：

基于 receiver 的方式，是使用 Kafka 的高阶 API 来在 ZooKeeper 中保存消费过的 offset 的。这是消费 Kafka 数据的传统方式。这种方式配合着 WAL 机制可以保证数据零丢失的高可靠性，但是却无法保证数据被处理一次且仅一次，可能会处理两次。因为 Spark 和 ZooKeeper 之间可能是不同步的。

基于 direct 的方式，使用 kafka 的简单 api，Spark Streaming 自己就负责追踪消费的 offset，并保存在 checkpoint 中。Spark 自己一定是同步的，因此可以保证数据是消费一次且仅消费一次。

在实际生产环境中大都用 Direct 方式

4.10.19 简述 SparkStreaming 窗口函数的原理（重点）

窗口函数就是在原来定义的 SparkStreaming 计算批次大小的基础上再次进行封装，每次计算多个批次的数据，同时还需要传递一个滑动步长的参数，用来设置当次计算任务完成之后下一次从什么地方开始计算。

图中 time1 就是 SparkStreaming 计算批次大小，虚线框以及实线大框就是窗口的大小，必须为批次的整数倍。虚线框到大实线框的距离（相隔多少批次），就是滑动步长。

4.10.20 请手写出 wordcount 的 Spark 代码实现（Scala）（手写代码重点）

```
val conf: SparkConf = new SparkConf().setMaster("local[*]").setAppName("WordCount")
```

```
val sc = new SparkContext(conf)


 sc.textFile("/input")
.flatMap(_.split(" "))
.map((_,1))
.reduceByKey(_+_)
.saveAsTextFile("/output")
 sc.stop()
```

4.10.21 如何使用 Spark 实现 topN 的获取（描述思路或使用伪代码）（重点）

方法 1：

（1）按照 key 对数据进行聚合（groupByKey）

（2）将 value 转换为数组，利用 scala 的 sortBy 或者 sortWith 进行排序（mapValues）数据量太大，会 OOM。

方法 2：

（1）取出所有的 key

（2）对 key 进行迭代，每次取出一个 key 利用 spark 的排序算子进行排序

方法 3：

（1）自定义分区器，按照 key 进行分区，使不同的 key 进到不同的分区

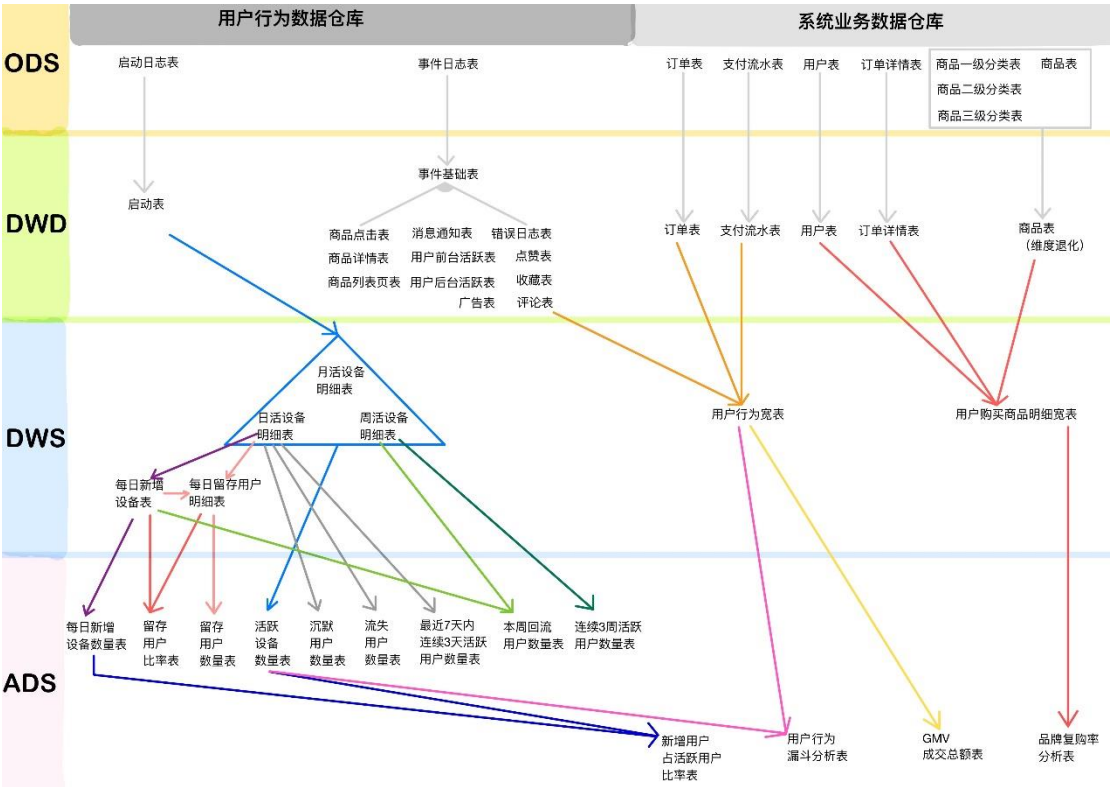
（2）对每个分区运用 spark 的排序算子进行排序

4.10.22 京东：调优之前与调优之后性能的详细对比（例如调整 map 个数，map 个数之前多少、之后多少，有什么提升）

这里举个例子。比如我们有几百个文件，会有几百个 map 出现，读取之后进行 join 操作，会非常的慢。这个时候我们可以进行 coalesce 操作，比如 240 个 map，我们合成 60 个 map，也就是窄依赖。这样再 shuffle，过程产生的文件数会大大减少。提高 join 的时间性能。

第 5 章 用户行为数据分析

5.1 数仓分层架构表



分层优点：复杂问题简单化、清晰数据结构(方便管理)、增加数据的复用性、隔离原始数据(解耦)

ods　　原始数据层　存放原始数据，保持原貌不做处理

dwd　　明细数据层　对 ods 层数据清洗（去除空值，脏数据，超过极限范围的数据）

dws　　服务数据层　轻度聚合

ads　　应用数据层　具体需求

数仓中各层建的表都是外部表

5.2 埋点行为数据基本格式(基本字段)

公共字段：基本所有安卓手机都包含的字段

业务字段：埋点上报的字段，有具体的业务类型

下面就是一个示例，表示业务字段的上传。

行为数据启动日志/事件日志表关键字段:

{

"ap":"xxxxx",//项目数据来源 app pc

```
"cm": {  //公共字段
        "mid": "",  // (String) 设备唯一标识
        "uid": "",  // (String) 用户标识
        "vc": "1",  // (String) versionCode，程序版本号
        "vn": "1.0",  // (String) versionName，程序版本名
        "l": "zh",  // (String) 系统语言
        "sr": "",  // (String) 渠道号，应用从哪个渠道来的。
        "os": "7.1.1",  // (String) Android 系统版本
        "ar": "CN",  // (String) 区域
        "md": "BBB100-1",  // (String) 手机型号
        "ba": "blackberry",  // (String) 手机品牌
        "sv": "V2.2.1",  // (String) sdkVersion
        "g": "",  // (String) gmail
        "hw": "1620x1080",  // (String) heightXwidth，屏幕宽高
        "t": "1506047606608",  // (String) 客户端日志产生时的时间
        "nw": "WIFI",  // (String) 网络模式
        "ln": 0,  // (double) lng 经度
        "la": 0  // (double) lat 纬度
    },
"et":  [  //事件
            {
                "ett": "1506047605364",  //客户端事件产生时间
                "en": "display",  //事件名称  启动和事件日志是根据事件名称的不同
                "kv": {  //事件结果，以 key-value 形式自行定义
                    "goodsid": "236",
                    "action": "1",
                    "extend1": "1",
                    "place": "2",
                    "category": "75"
                }
            }
        ]
}
```

根据事件标签的不同可以分成不同的日志表

5.3 项目经验总结

5.3.1 项目经验之元数据备份

元数据备份（重点，如数据损坏，可能整个集群无法运行，至少要保证每日零点之后备份到其它服务器两个复本）


MySQL基于Keepa
lived实现HA搭建文

5.3.2 日期处理函数

1）date_format 函数（根据格式整理日期）

2）date_add、date_sub 函数（加减日期）

3）next_day 函数

4）last_day 函数（求当月最后一天日期）

5）collect_set 函数

6）get_json_object 解析 json 函数

5.3.3 Union 与 Union all 区别

1）union 会将联合的结果集去重，效率较 union all 差

2）union all 不会对结果集去重，所以效率高

5.3.4 Shell 中单引号和双引号区别

1）在/home/atguigu/bin 创建一个 test.sh 文件

```
[atguigu@hadoop102 bin]$ vim test.sh
```

在文件中添加如下内容

```
#!/bin/bash
do_date=$1

echo '$do_date'
echo "$do_date"
echo "'$do_date'"
echo '"$do_date"'
echo `date`
```

2）查看执行结果

```
[atguigu@hadoop102 bin]$ test.sh 2019-02-10
$do_date
2019-02-10
'2019-02-10'
"$do_date"
2019年 05月 02日 星期四 21:02:08 CST
```

3）总结：

（1）单引号不取变量值
（2）双引号取变量值
（3）反引号`，执行引号中命令
（4）双引号内部嵌套单引号，取出变量值
（5）单引号内部嵌套双引号，不取出变量值

5.3.5 Tez 引擎优点？

Tez 可以将多个有依赖的作业转换为一个作业，这样只需写一次 HDFS，且中间节点较少，从而大大提升作业的计算性能。

5.4 ods 层

1）ods_start_log 启动日志表

只有一个字段 line（保存着 json），按照日期 dt 分区，表的格式：lzo

2）ods_event_log 事件日志表（格式同启动日志表）

只有一个字段 line ，按照日期 dt 分区，表的格式：lzo

5.5 dwd 层

1）dwd_start_log 启动表

关键字段：mid_id，user_id，dt(分区字段，按照日期分区) （其实这是启动表和事件表的公共字段）

从 ods_start_log 中的 line 用 get_json_object(line，'$.mid') mid_id 的方式获取字段

5.5.1 自定义 UDF/UDTF（项目中的应用）

自定义 UDF 函数（解析公共字段，一进一出）

自定义 UDTF 函数（解析具体事件字段，一进多出）

自定义 UDF：继承 UDF，重写 evaluate 方法

自定义 UDTF：继承自 GenericUDTF，重写 3 个方法：initialize(自定义输出的列名和类型)，process（将结果返回 forward(result)），close

为什么要自定义 UDF/UDTF，因为自定义函数，可以自己埋点 Log 打印日志，出错或者数据异常，方便调试。

5.5.2 事件日志基础明细表

dwd_base_event_log 事件日志基础明细表

1）关键字段：

公共字段：mid_id，user_id，dt(分区字段)以及 event_name、event_json、server_time

2）从 ods_event_log 的 line 中用 UDF 获取 公共字段 和 server_time，用 UDTF 获取 event_name , event_json 。

5.5.3 商品点击表

dwd_display_log 商品点击表

关键字段：公共字段 + 特有字段

从 dwd_base_event_log 中直接获取公共字段和 server_time，从 dwd_base_event_log 的 event_json 中获取特有字段，where event_name = "display"

get_json_object(event_json,'$.kv.action') action

5.5.4 其他的具体事件明细表

类似

dwd_newsdetail_log 商品详情页表

dwd_loading_log 商品列表页表

dwd_ad_log 广告表

dwd_notification_log 消息通知表

dwd_active_foreground_log 用户前台活跃表

dwd_active_background_log 用户后台活跃表

dwd_comment_log 评论表

dwd_favorites_log 收藏表

dwd_praise_log 点赞表

dwd_error_log 错误日志表

从一张事件基础明细表 dwd_base_event_log 一共可以获得 11 张具体事件明细表

5.6 需求一：用户活跃主题

5.6.1 DWS 层日活明细表

每日活跃设备分析

尚硅谷

1）建表语句

```
drop table if exists dws_uv_detail_day;
create  external  table dws_uv_detail_day(
    `mid_id` string COMMENT '设备唯一标识',
    `user_id` string COMMENT '用户标识',
    `version_code` string COMMENT '程序版本号',
    `version_name` string COMMENT '程序版本名',
    `lang` string COMMENT '系统语言',
    `source` string COMMENT '渠道号',
    `os` string COMMENT '安卓系统版本',
    `area` string COMMENT '区域',
    `model` string COMMENT '手机型号',
    `brand` string COMMENT '手机品牌',
    `sdk_version` string COMMENT 'sdkVersion',
    `gmail` string COMMENT 'gmail',
    `height_width` string COMMENT '屏幕宽高',
    `app_time` string COMMENT '客户端日志产生时的时间',
    `network` string COMMENT '网络模式',
    `lng` string COMMENT '经度',
    `lat` string COMMENT '纬度'
) COMMENT '活跃用户按天明细'
PARTITIONED BY ( `dt` string)
stored as  parquet
location '/warehouse/gmall/dws/dws_uv_detail_day/'
;
```

2）导入数据

```
insert       overwrite       table       dws_uv_detail_day
partition(dt='2019-02-10')
select
    mid_id,
    concat_ws('|', collect_set(user_id)) user_id,
    concat_ws('|', collect_set(version_code)) version_code,
    concat_ws('|', collect_set(version_name)) version_name,
    concat_ws('|', collect_set(lang))lang,
    concat_ws('|', collect_set(source)) source,
    concat_ws('|', collect_set(os)) os,
    concat_ws('|', collect_set(area)) area,
    concat_ws('|', collect_set(model)) model,
    concat_ws('|', collect_set(brand)) brand,
    concat_ws('|', collect_set(sdk_version)) sdk_version,
    concat_ws('|', collect_set(gmail)) gmail,
    concat_ws('|', collect_set(height_width)) height_width,
    concat_ws('|', collect_set(app_time)) app_time,
    concat_ws('|', collect_set(network)) network,
    concat_ws('|', collect_set(lng)) lng,
    concat_ws('|', collect_set(lat)) lat
from dwd_start_log
where dt='2019-02-10'
group by mid_id;
```


5.6.2 DWS 层周活明细表

每周活跃设备分析

尚硅谷

1）建表语句

```
drop table if exists dws_uv_detail_wk;
create external table dws_uv_detail_wk(
    `mid_id` string COMMENT '设备唯一标识',
    `user_id` string COMMENT '用户标识',
    `version_code` string COMMENT '程序版本号',
    `version_name` string COMMENT '程序版本名',
    `lang` string COMMENT '系统语言',
    `source` string COMMENT '渠道号',
    `os` string COMMENT '安卓系统版本',
    `area` string COMMENT '区域',
    `model` string COMMENT '手机型号',
    `brand` string COMMENT '手机品牌',
    `sdk_version` string COMMENT 'sdkVersion',
    `gmail` string COMMENT 'gmail',
    `height_width` string COMMENT '屏幕宽高',
    `app_time` string COMMENT '客户端日志产生时的时间',
    `network` string COMMENT '网络模式',
    `lng` string COMMENT '经度',
    `lat` string COMMENT '纬度',
    `monday_date` string COMMENT '周一日期',
    `sunday_date` string COMMENT  '周日日期'
) COMMENT '活跃用户按周明细'
PARTITIONED BY (`wk_dt` string)
stored as  parquet
location '/warehouse/gmall/dws/dws_uv_detail_wk/'
;
```

2）导入数据

```
insert  overwrite table dws_uv_detail_wk partition(wk_dt)
select
    mid_id,
    concat_ws('|', collect_set(user_id)) user_id,
    concat_ws('|', collect_set(version_code)) version_code,
    concat_ws('|', collect_set(version_name)) version_name,
    concat_ws('|', collect_set(lang)) lang,
    concat_ws('|', collect_set(source)) source,
    concat_ws('|', collect_set(os)) os,
    concat_ws('|', collect_set(area)) area,
    concat_ws('|', collect_set(model)) model,
    concat_ws('|', collect_set(brand)) brand,
    concat_ws('|', collect_set(sdk_version)) sdk_version,
    concat_ws('|', collect_set(gmail)) gmail,
    concat_ws('|', collect_set(height_width)) height_width,
    concat_ws('|', collect_set(app_time)) app_time,
    concat_ws('|', collect_set(network)) network,
    concat_ws('|', collect_set(lng)) lng,
    concat_ws('|', collect_set(lat)) lat,
    date_add(next_day('2019-02-10','MO'),-7),
    date_add(next_day('2019-02-10','MO'),-1),
    concat(date_add( next_day('2019-02-10','MO'),-7),  '_'  ,
date_add(next_day('2019-02-10','MO'),-1)
)
from dws_uv_detail_day
where    dt>=date_add(next_day('2019-02-10','MO'),-7)    and
dt<=date_add(next_day('2019-02-10','MO'),-1)
group by mid_id;
```

5.6.3 DWS 层月活明细表

每月活跃设备分析

1）建表语句

```
drop table if exists dws_uv_detail_mn;
create  external table dws_uv_detail_mn(
    `mid_id` string COMMENT '设备唯一标识',
    `user_id` string COMMENT '用户标识',
    `version_code` string COMMENT '程序版本号',
    `version_name` string COMMENT '程序版本名',
    `lang` string COMMENT '系统语言',
    `source` string COMMENT '渠道号',
    `os` string COMMENT '安卓系统版本',
    `area` string COMMENT '区域',
    `model` string COMMENT '手机型号',
    `brand` string COMMENT '手机品牌',
    `sdk_version` string COMMENT 'sdkVersion',
    `gmail` string COMMENT 'gmail',
    `height_width` string COMMENT '屏幕宽高',
    `app_time` string COMMENT '客户端日志产生时的时间',
    `network` string COMMENT '网络模式',
    `lng` string COMMENT '经度',
    `lat` string COMMENT '纬度'
) COMMENT '活跃用户按月明细'
PARTITIONED BY (`mn` string)
stored as  parquet
location '/warehouse/gmall/dws/dws_uv_detail_mn/'
;
```

2）导入数据

```
insert  overwrite table dws_uv_detail_mn  partition(mn)
select
    mid_id,
    concat_ws('|', collect_set(user_id)) user_id,
    concat_ws('|', collect_set(version_code)) version_code,
    concat_ws('|', collect_set(version_name)) version_name,
    concat_ws('|', collect_set(lang)) lang,
    concat_ws('|', collect_set(source)) source,
    concat_ws('|', collect_set(os)) os,
    concat_ws('|', collect_set(area)) area,
    concat_ws('|', collect_set(model)) model,
    concat_ws('|', collect_set(brand)) brand,
    concat_ws('|', collect_set(sdk_version)) sdk_version,
    concat_ws('|', collect_set(gmail)) gmail,
    concat_ws('|', collect_set(height_width)) height_width,
    concat_ws('|', collect_set(app_time)) app_time,
    concat_ws('|', collect_set(network)) network,
    concat_ws('|', collect_set(lng)) lng,
    concat_ws('|', collect_set(lat)) lat,
    date_format('2019-02-10','yyyy-MM')
from dws_uv_detail_day
where  date_format(dt,'yyyy-MM')  =  date_format('2019-02-10','yyyy-MM')
group by mid_id;
```


5.6.4 ADS 层日周月活跃设备数表

活跃设备分析

1）建表语句

```
drop table if exists ads_uv_count;
create  external table ads_uv_count(
    `dt` string COMMENT '统计日期',
    `day_count` bigint COMMENT '当日用户数量',
    `wk_count`  bigint COMMENT '当周用户数量',
    `mn_count`  bigint COMMENT '当月用户数量',
    `is_weekend` string COMMENT 'Y,N是否是周末,用于得到本周最终结果',
    `is_monthend` string COMMENT 'Y,N是否是月末,用于得到本月最终结果'
) COMMENT '每日活跃用户数量'
row format delimited fields terminated by '\t'
location '/warehouse/gmall/ads/ads_uv_count_day/'
;
```

2）导入数据

```
insert  into table ads_uv_count
select
  '2019-02-10' dt,
   daycount.ct,
   wkcount.ct,
   mncount.ct,
   if(date_add(next_day('2019-02-10','MO'),-1)='2019-02-10','Y','N') ,
   if(last_day('2019-02-10')='2019-02-10','Y','N')
from
(
   select
      '2019-02-10' dt,
       count(*) ct
   from dws_uv_detail_day
   where dt='2019-02-10'
)daycount   join
(
   select
     '2019-02-10' dt,
     count (*) ct
   from dws_uv_detail_wk
   where             wk_dt=concat(date_add(next_day('2019-02-10','MO'),-7),'_' ,date_add(next_day('2019-02-10','MO'),-1) )
)  wkcount  on daycount.dt=wkcount.dt
join
(
   select
     '2019-02-10' dt,
     count (*) ct
   from dws_uv_detail_mn
   where mn=date_format('2019-02-10','yyyy-MM')
)mncount on daycount.dt=mncount.dt ;
```

5.7 需求二：用户新增主题

5.7.1 DWS 层日新增明细表

每日新增设备分析

1）建表语句

```
drop table if exists  dws_new_mid_day;
create external  table  dws_new_mid_day
(
    `mid_id` string COMMENT '设备唯一标识',
    `user_id` string COMMENT '用户标识',
    `version_code` string COMMENT '程序版本号',
    `version_name` string COMMENT '程序版本名',
    `lang` string COMMENT '系统语言',
    `source` string COMMENT '渠道号',
    `os` string COMMENT '安卓系统版本',
    `area` string COMMENT '区域',
    `model` string COMMENT '手机型号',
    `brand` string COMMENT '手机品牌',
    `sdk_version` string COMMENT 'sdkVersion',
    `gmail` string COMMENT 'gmail',
    `height_width` string COMMENT '屏幕宽高',
    `app_time` string COMMENT '客户端日志产生时的时间',
    `network` string COMMENT '网络模式',
    `lng` string COMMENT '经度',
    `lat` string COMMENT '纬度',
    `create_date`  string  comment '创建时间'
)  COMMENT '每日新增设备信息'
stored as  parquet
location '/warehouse/gmall/dws/dws_new_mid_day/';
```

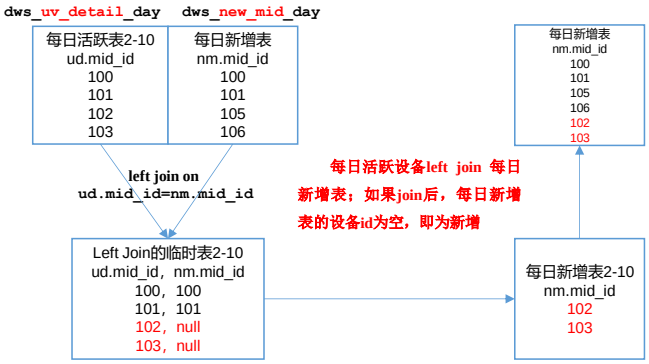
2）导入数据

```
insert into table dws_new_mid_day
select
    ud.mid_id,
    ud.user_id ,
    ud.version_code ,
    ud.version_name ,
    ud.lang ,
    ud.source,
    ud.os,
    ud.area,
    ud.model,
    ud.brand,
    ud.sdk_version,
    ud.gmail,
    ud.height_width,
    ud.app_time,
    ud.network,
    ud.lng,
    ud.lat,
    '2019-02-10'
from dws_uv_detail_day ud left join dws_new_mid_day nm
on ud.mid_id=nm.mid_id
where ud.dt='2019-02-10' and nm.mid_id is null;
```


每日新增设备分析

2）导入数据

```
insert into table dws_new_mid_day
select
    ud.mid_id,
    ud.user_id ,
    ud.version_code ,
    ud.version_name ,
    ud.lang ,
    ud.source,
    ud.os,
    ud.area,
    ud.model,
    ud.brand,
    ud.sdk_version,
    ud.gmail,
    ud.height_width,
    ud.app_time,
    ud.network,
    ud.lng,
    ud.lat,
    '2019-02-10'
from dws_uv_detail_day ud left join dws_new_mid_day nm
on ud.mid_id=nm.mid_id
where ud.dt='2019-02-10' and nm.mid_id is null;
```



5.7.2 ADS 层每日新增设备数表

每日新增设备表

1）建表语句

```
drop table if exists  ads_new_mid_count;
create external  table  ads_new_mid_count
(
    `create_date`     string comment '创建时间',
    `new_mid_count`   BIGINT comment '新增设备数量'
)  COMMENT '每日新增设备信息数量'
row format delimited  fields terminated by '\t'
location  '/warehouse/gmall/ads/ads_new_mid_count/';
```

2）导入数据

```
insert into table ads_new_mid_count
select
    create_date ,
    count(*)
from dws_new_mid_day
where create_date='2019-02-10'
group by create_date ;
```


5.8 需求三：用户留存主题

用户留存

留存用户：某段时间内的新增用户（活跃用户），经过一段时间后，又继续使用应用的被认作是留存用户；

留存率：留存用户占当时新增用户（活跃用户）的比例即是留存率。

例如，2月10日新增用户100，这100人在2月11日启动过应用的有30人，2月12日启动过应用的有25人，2月13日启动过应用的有32人；

则2月10日新增用户次日的留存率是30/100 = 30%，两日留存率是25/100=25%，三日留存率是32/100=32%。

时间	新增用户	1天后	2天后	3天后
2019-02-10	100	30% (2-11)	25% (2-12)	32% (2-13)
2019-02-11	200	20% (2-12)	15% (2-13)	
2019-02-12	100	25% (2-13)		
2019-02-13				

用户留存率分析

需求：每天计算前1、2、3、4、7、14天的留存率

分析：假设今天是11日，要统计前1天也就是10日新增设备的留存率，公式如下：

10日新增设备的留存率= 10日的新增设备 且 11日活跃的 / 10日的新增设备

1）分母获取

10日活跃表left join 每日新增表，新增表id为null的为10日新增设备

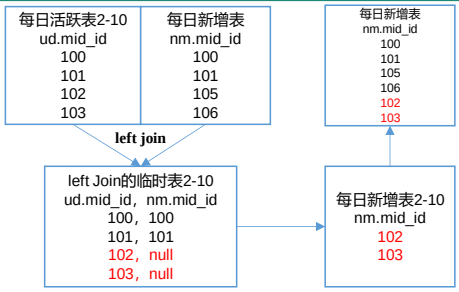
2）分子获取

10的新增join 11日的活跃，且新增日期是10日，活跃日期是11日

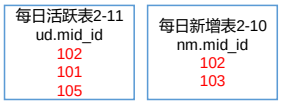
3）留存率计算

10日的新增设备 且 11日活跃的用户表 与10日新增设备join，算出留存率

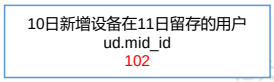
1）10日新增设备明细表



2）10日新增设备在11日留存用户



join on ud.mid_id =nm.mid_id
where ud.dt='2019-02-11' and
nm.create_date=date_add('2019-02-11',-1)



5.8.1 DWS 层日留存明细表

每日用户留存明细

1）创建表

```
drop table if exists  dws_user_retention_day;
create external  table  dws_user_retention_day
(
    `mid_id` string COMMENT '设备唯一标识',
    `user_id` string COMMENT '用户标识',
    `version_code` string COMMENT '程序版本号',
    `version_name` string COMMENT '程序版本名',
    `lang` string COMMENT '系统语言',
    `source` string COMMENT '渠道号',
    `os` string COMMENT '安卓系统版本',
    `area` string COMMENT '区域',
    `model` string COMMENT '手机型号',
    `brand` string COMMENT '手机品牌',
    `sdk_version` string COMMENT 'sdkVersion',
    `gmail` string COMMENT 'gmail',
    `height_width` string COMMENT '屏幕宽高',
    `app_time` string COMMENT '客户端日志产生时的时间',
    `network` string COMMENT '网络模式',
    `lng` string COMMENT '经度',
    `lat` string COMMENT '纬度',
    `create_date`       string  comment '设备新增时间',
    `retention_day`     int comment '截止当前日期留存天数'
)  COMMENT '每日用户留存情况'
PARTITIONED BY ( `dt` string)
stored as  parquet
location '/warehouse/gmall/dws/dws_user_retention_day/'
;
```

2）导入数据

```
insert    overwrite  table  dws_user_retention_day
partition(dt="2019-02-11")
select
    nm.mid_id,
    nm.user_id ,
    nm.version_code ,
    nm.version_name ,
    nm.lang ,
    nm.source,
    nm.os,
    nm.area,
    nm.model,
    nm.brand,
    nm.sdk_version,
    nm.gmail,
    nm.height_width,
    nm.app_time,
    nm.network,
    nm.lng,
    nm.lat,
    nm.create_date,
    1 retention_day
from  dws_uv_detail_day ud join dws_new_mid_day nm
on ud.mid_id =nm.mid_id
where            ud.dt='2019-02-11'            and
nm.create_date=date_add('2019-02-11',-1);
```

5.8.2 ADS 层留存用户数表

留存用户数

尚硅谷

1）创建表

```
drop table if exists  ads_user_retention_day_count;
create external  table  ads_user_retention_day_count
(
    `create_date`       string  comment '设备新增日期',
    `retention_day`     int comment '截止当前日期留存天数',
    `retention_count`   bigint comment  '留存数量'
)  COMMENT '每日用户留存情况'
row format delimited fields terminated by '\t'
location '/warehouse/gmall/ads/ads_user_retention_day_count/';
```

2）导入数据

```
insert into table ads_user_retention_day_count
select
    create_date,
    retention_day,
    count(*) retention_count
from dws_user_retention_day
where dt='2019-02-11'
group by create_date,retention_day;
```


5.8.3 ADS 层留存用户率表

留存比率

尚硅谷

1）创建表

```
drop table if exists  ads_user_retention_day_rate;
create external  table  ads_user_retention_day_rate
(
     `stat_date`          string comment '统计日期',
     `create_date`        string  comment '设备新增日期',
     `retention_day`      int comment '截止当前日期留存天数',
     `retention_count`    bigint comment  '留存数量',
     `new_mid_count`      bigint  comment '当日设备新增数量',
     `retention_ratio`    decimal(10,2) comment '留存率'
)  COMMENT '每日用户留存情况'
row format delimited fields terminated by '\t'
location '/warehouse/gmall/ads/ads_user_retention_day_rate/';
```

2）导入数据

```
insert into table ads_user_retention_day_rate
select
    '2019-02-11' ,
    ur.create_date,
    ur.retention_day,
    ur.retention_count ,
    nc.new_mid_count,
    ur.retention_count/nc.new_mid_count*100
from
(
    select
        create_date,
        retention_day,
        count(*) retention_count
    from dws_user_retention_day
    where dt='2019-02-11'
    group by create_date,retention_day
)       ur    join    ads_new_mid_count    nc    on
nc.create_date=ur.create_date;
```

5.9 需求四：沉默用户

沉默用户数

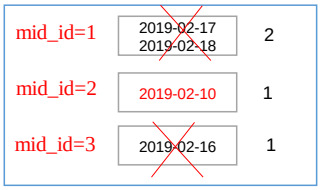
沉默用户：指的是只在安装当天启动过，且启动时间是在一周前

1）创建表

```
drop table if exists ads_slient_count;
create external table ads_slient_count(
    `dt` string COMMENT '统计日期',
    `slient_count` bigint COMMENT '沉默设备数'
)
row format delimited fields terminated by '\t'
location '/warehouse/gmall/ads/ads_slient_count';
```

2）导入数据

```
insert into table ads_slient_count
select
    '2019-02-20' dt,
    count(*) slient_count
from
(
    select
        mid_id
    from dws_uv_detail_day
    where dt<='2019-02-20'
    group by mid_id
    having count(*)=1
    and min(dt)<date_add('2019-02-20',-7)
) t1;
```



1、按照设备id对日活表分组

2、只在安装当天启动过

3、启动时间是在一周前

5.10 需求五：本周回流用户数

本周回流用户数

本周回流（上周以前活跃过，上周没活跃，本周活跃了）=本周活跃-本周新增-上周活跃

1）创建表

```
drop table if exists ads_back_count;
create external table ads_back_count(
    `dt` string COMMENT '统计日期',
    `wk_dt` string COMMENT '统计日期所在周',
    `wastage_count` bigint COMMENT '回流设备数'
)
row format delimited fields terminated by '\t'
location '/warehouse/gmall/ads/ads_back_count';
```

1、本周活跃

100
101
102
103
104

2、本周新增

101
102

3、上周活跃

101
102
103

本周回流用户=本周活跃left join本周新增 left join 上周活跃，且本周新增id为null，，上周活跃id为null

100
104

2）导入数据

```
insert into table ads_back_count
select
  '2019-02-20' dt,
  concat(date_add(next_day('2019-02-20','MO'),-7),'_',date_add(next_day('2019-02-20','MO'),-1)) wk_dt,
  count(*)
from
(
  select  t1.mid_id
  from
  (
    select  mid_id
    from dws_uv_detail_wk
    where wk_dt=concat(date_add(next_day('2019-02-20','MO'),-7),'_',date_add(next_day('2019-02-20','MO'),-1))
  )t1
  left join
  (
    select  mid_id
    from dws_new_mid_day
    where create_date<=date_add(next_day('2019-02-20','MO'),-1) and create_date>=date_add(next_day('2019-02-20','MO'),-7)
  )t2  on t1.mid_id=t2.mid_id
  left join
  (
    select  mid_id
    from dws_uv_detail_wk
    where wk_dt=concat(date_add(next_day('2019-02-20','MO'),-7*2),'_',date_add(next_day('2019-02-20','MO'),-7-1))
  )t3  on t1.mid_id=t3.mid_id
  where t2.mid_id is null and t3.mid_id is null
)t4;
```

1、本周活跃

2、本周新增

3、上周活跃

5.11 需求六：流失用户数

流失用户数

流失用户：最近7天未登录用户

1）创建表

```
drop table if exists ads_wastage_count;
create external table ads_wastage_count(
    `dt` string COMMENT '统计日期',
    `wastage_count` bigint COMMENT '流失设备数'
)
row format delimited fields terminated by '\t'
location  '/warehouse/gmall/ads/ads_wastage_count';
```

2）导入数据

```
insert into table ads_wastage_count
select
     '2019-02-20',
     count(*)
from
(
    select mid_id
    from dws_uv_detail_day
    group by mid_id
    having max(dt)<=date_add('2019-02-20',-7)
)t1;
```

mid_id=1	2019-02-17 2019-02-18
mid_id=2	2019-02-10 2019-02-11
mid_id=3	2019-02-16

5.12 需求七：最近连续 3 周活跃用户数

最近连续3周活跃用户数

最近3周连续活跃的用户：通常是周一对前3周的数据做统计，该数据一周计算一次。

1）创建表

```
drop table if exists ads_continuity_wk_count;
create external table ads_continuity_wk_count(
    `dt` string COMMENT '统计日期,一般用结束周周日日期,如果每天计算一次,可用当天日期',
    `wk_dt` string COMMENT '持续时间',
    `continuity_count` bigint
)
row format delimited fields terminated by '\t'
location '/warehouse/gmall/ads/ads_continuity_wk_count';
```

前一周	前二周	前三周
100	101	101
101	102	104
102		

2）导入数据

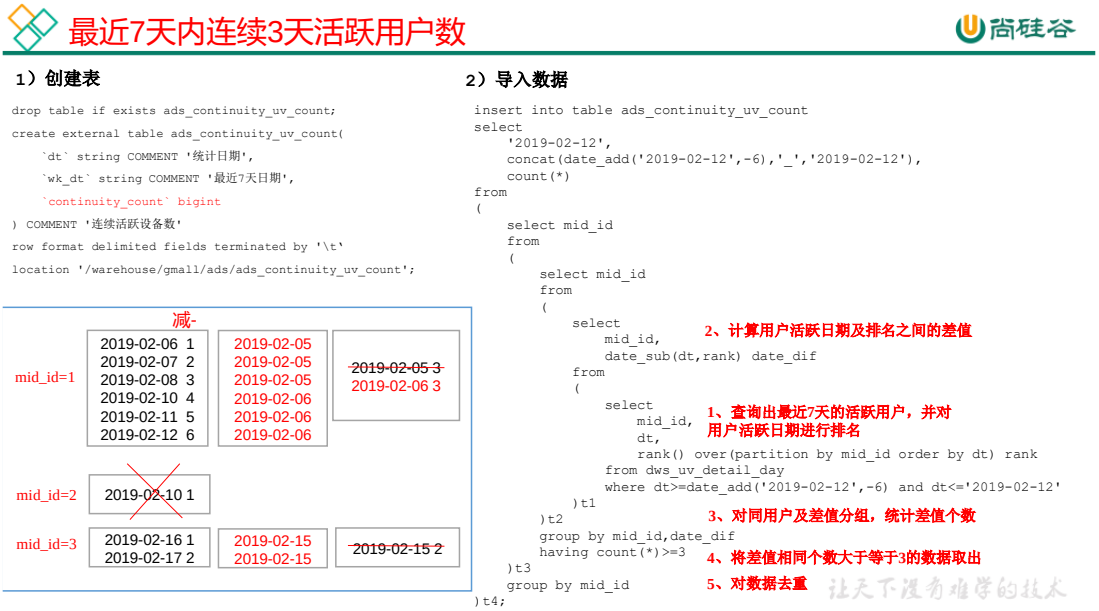
```
insert into table ads_continuity_wk_count
select
     '2019-02-20',
     concat(date_add(next_day('2019-02-20','MO'),-7*3),'_',date_add(next_day('2019-02-20','MO'),-1)),
     count(*)
from
(
    select mid_id
    from dws_uv_detail_wk
    where wk_dt>=concat(date_add(next_day('2019-02-20','MO'),-7*3),'_',date_add(next_day('2019-02-20','MO'),-7*2-1))
    and wk_dt<=concat(date_add(next_day('2019-02-20','MO'),-7),'_',date_add(next_day('2019-02-20','MO'),-1))
    group by mid_id
    having count(*)=3
)t1;
```

按照设备id对每周活跃表分组

统计次数等于3

100	1
101	3
102	2
104	1

5.13 需求八：最近七天内连续三天活跃用户数



5.14 需求逻辑

5.14.1 如何分析用户活跃？

在启动日志中统计不同设备 id 出现次数。

5.14.2 如何分析用户新增？

用活跃用户表 left join 用户新增表，用户新增表中 mid 为空的即为用户新增。

5.14.3 如何分析用户 1 天留存？

留存用户=前一天新增 join 今天活跃

用户留存率=留存用户/前一天新增

5.14.4 如何分析沉默用户？

(登录时间为 7 天前,且只出现过一次)

按照设备 id 对日活表分组，登录次数为 1，且是在一周前登录。

5.14.5 如何分析本周回流用户？

本周活跃 left join 本周新增 left join 上周活跃，且本周新增 id 和上周活跃 id 都为 null

5.14.6 如何分析流失用户？

(登录时间为 7 天前)

按照设备 id 对日活表分组，且七天内没有登录过。

5.14.7 如何分析最近连续 3 周活跃用户数？

按照设备 id 对周活进行分组，统计次数大于 3 次。

5.14.8 如何分析最近七天内连续三天活跃用户数？

1）查询出最近 7 天的活跃用户，并对用户活跃日期进行排名

2）计算用户活跃日期及排名之间的差值

3）对同用户及差值分组，统计差值个数

4）将差值相同个数大于等于 3 的数据取出，然后去重(去的是什么重???)，即为连续 3 天及以上活跃的用户

第 6 章 业务交互数据分析

第三份文档(电商业务数据的分析和处理)

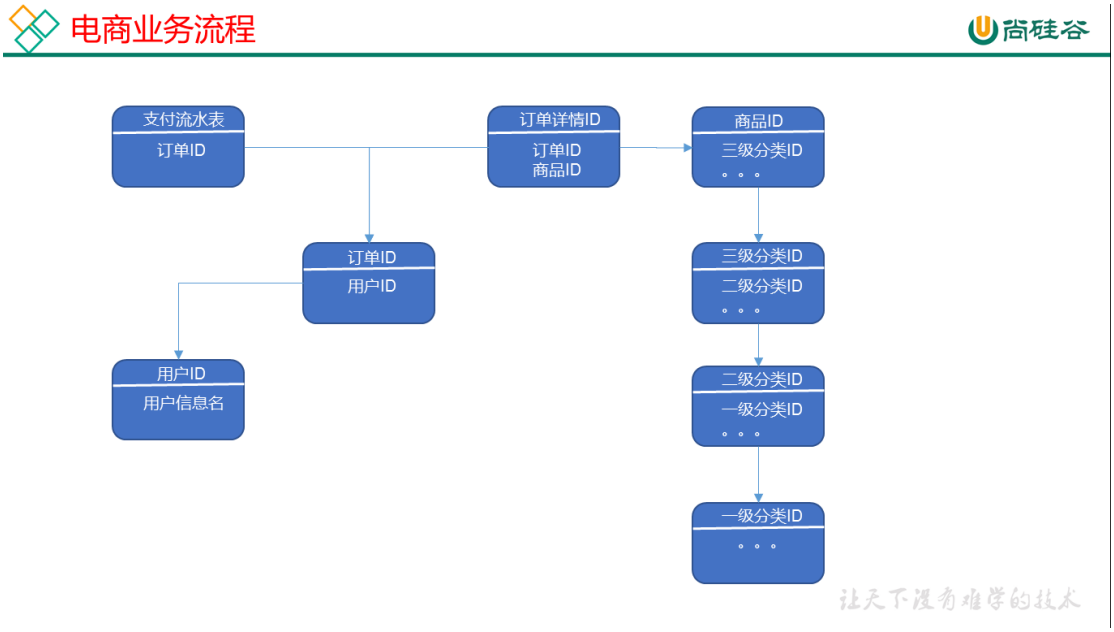
6.1 电商常识

SKU：一台银色、128G 内存的、支持联通网络的 iPhoneX

SPU：iPhoneX

Tm_id：品牌 Id 苹果，包括 IPHONE，耳机，mac 等

6.2 电商业务流程



6.3 业务表关键字段

6.3.1 订单表（order_info）

标签	含义
id	订单编号
total_amount	订单金额
order_status	订单状态
user_id	用户 id
payment_way	支付方式
out_trade_no	支付流水号
create_time	创建时间
operate_time	操作时间

6.3.2 订单详情表（order_detail）

标签	含义
id	订单编号
order_id	订单号
user_id	用户 id
sku_id	商品 id
sku_name	商品名称
order_price	商品价格
sku_num	商品数量
create_time	创建时间

6.3.3 商品表

标签	含义
id	skuId
spu_id	spuid
price	价格
sku_name	商品名称
sku_desc	商品描述
weight	重量
tm_id	品牌 id
category3_id	品类 id
create_time	创建时间

6.3.4 用户表

标签	含义
id	用户 id
name	姓名
birthday	生日
gender	性别
email	邮箱

user_level	用户等级
create_time	创建时间

6.3.5 商品一级分类表

标签	含义
id	id
name	名称

6.3.6 商品二级分类表

标签	含义
id	id
name	名称
category1_id	一级品类 id

6.3.7 商品三级分类表

标签	含义
id	id
name	名称
Category2_id	二级品类 id

6.3.8 支付流水表

标签	含义
id	编号
out_trade_no	对外业务编号
order_id	订单编号
user_id	用户编号
alipay_trade_no	支付宝交易流水编号
total_amount	支付金额
subject	交易内容
payment_type	支付类型
payment_time	支付时间

订单表跟订单详情表有什么区别？

订单表的订单状态会变化，订单详情表不会，因为没有订单状态。

订单表记录 user_id，订单 id 订单编号，订单的总金额 order_status，支付方式，订单状态等。

订单详情表记录 user_id，商品 sku_id ,具体的商品信息（商品名称 sku_name，价格 order_price，数量 sku_num）

6.4 MySql 中表的分类

实体表，维度表，事务型事实表，周期性事实表

其实最终可以把事务型事实表，周期性事实表统称实体表，实体表，维度表统称维度表

订单表（order_info）（周期型事实表）

订单详情表（order_detail）(事务型事实表)

商品表(实体表)

用户表(实体表)

商品一级分类表(维度表)

商品二级分类表(维度表)

商品三级分类表(维度表)

支付流水表(事务型实体表)

6.5 同步策略

序号	数据库中表名	表中文名称	同步策略
1	order_info	订单表	新增及变化
2	order_detail	订单详情	增量
3	sku_info	商品表	全量
4	user_info	用户表	全量
5	base_category1	商品一级分类表	全量
6	base_category2	商品二级分类表	全量
7	base_category3	商品三级分类表	全量
8	payment_info	支付流水表	增量

实体表，维度表统称维度表，每日全量或者每月（更长时间）全量

事务型事实表：每日增量

周期性事实表：拉链表

6.6 关系型数据库范式理论

1NF：属性不可再分割（例如不能存在 5 台电脑的属性，坏处：表都没法用）

2NF：不能存在部分函数依赖（例如主键（学号+课名）-->成绩，姓名，但学号--》姓名，

所以姓名部分依赖于主键（学号+课名），所以要去除，坏处：数据冗余）

3NF：不能存在传递函数依赖（学号--》宿舍种类--》价钱，坏处：数据冗余和增删异常）

Mysql 关系模型：关系模型主要应用与 OLTP 系统中，为了保证数据的一致性以及避免冗余，所以大部分业务系统的表都是遵循第三范式的。

Hive 维度模型：维度模型主要应用于 OLAP 系统中，因为关系模型虽然冗余少，但是在大规模数据，跨表分析统计查询过程中，会造成多表关联，这会大大降低执行效率。所以 HIVE 把相关各种表整理成两种：事实表和维度表两种。所有维度表围绕着事实表进行解释。

6.7 数据模型

雪花模型、星型模型和星座模型

（在维度建模的基础上又分为三种模型：星型模型、雪花模型、星座模型。）

星型模型（一级维度表），雪花（多级维度），星座模型（星型模型+多个事实表）

6.8 业务数据数仓搭建

sqoop

导数据的原理是 mapreduce,

import　把数据从关系型数据库 导到 数据仓库，自定义 InputFormat，

export　把数据从数据仓库 导到 关系型数据库，自定义 OutputFormat，

用 sqoop 从 mysql 中将八张表的数据导入数仓的 ods 原始数据层

全量无条件，增量按照创建时间，增量+变化按照创建时间或操作时间。

origin_data

sku_info 商品表（每日导全量）

user_info 用户表（每日导全量）

base_category1 商品一级分类表（每日导全量）

base_category2 商品二级分类表（每日导全量）

base_category3 商品三级分类表（每日导全量）

order_detail 订单详情表（每日导增量）

payment_info 支付流水表（每日导增量）

order_info 订单表（每日导增量+变化）

6.8.1 ods 层

（八张表，表名，字段跟 mysql 完全相同）

从 origin_data 把数据导入到 ods 层，表名在原表名前加 ods_

6.8.2 dwd 层

对 ODS 层数据进行判空过滤。对商品分类表进行维度退化(降维)。其他数据跟 ods 层一模一样

订单表 dwd_order_info

订单详情表 dwd_order_detail

用户表 dwd_user_info

支付流水表 dwd_payment_info

商品表 dwd_sku_info

其他表字段不变，唯独商品表，通过关联 3 张分类表，增加了

```
category2_id` string COMMENT '2id',
`category1_id` string COMMENT '3id',
`category3_name` string COMMENT '3',
`category2_name` string COMMENT '2',
`category1_name` string COMMENT '1',
```

小结：

1）维度退化要付出什么代价？或者说会造成什么样的需求处理不了？

如果被退化的维度，还有其他业务表使用，退化后处理起来就麻烦些。

还有如果要删除数据，对应的维度可能也会被永久删除。

2）想想在实际业务中还有那些维度表可以退化

城市的三级分类（省、市、县）等

6.8.3 dws 层

用户行为宽表

1）创建用户行为宽表

```
drop table if exists dws_user_action;
create  external table dws_user_action
(
    user_id         string      comment '用户 id',
    order_count     bigint      comment '下单次数 ',
    order_amount    decimal(16,2)  comment '下单金额 ',
    payment_count   bigint      comment '支付次数',
    payment_amount  decimal(16,2) comment '支付金额 ',
    comment_count   bigint      comment '评论次数'
) COMMENT '每日用户行为宽表'
PARTITIONED BY ( `dt` string)
location '/warehouse/gmall/dws/dws_user_action/'
```

2）导入数据

```
with
tmp_order as(
    select
        user_id,
        count(*)  order_count,
        sum(oc.total_amount) order_amount
    from dwd_order_info  oc
    where date_format(oc.create_time,'yyyy-MM-dd')='2019-02-10'
    group by user_id
)  ,
tmp_payment as(
    select
        user_id, sum(pi.total_amount) payment_amount, count(*) payment_count
    from dwd_payment_info pi
    where date_format(pi.payment_time,'yyyy-MM-dd')='2019-02-10'
    group by user_id
),
tmp_comment as(
    select  user_id, count(*) comment_count
    from dwd_comment_log c
    where date_format(c.dt,'yyyy-MM-dd')='2019-02-10'
    group by user_id
)
insert overwrite table dws_user_action partition(dt='2019-02-10')
select
    user_actions.user_id,
    sum(user_actions.order_count),
    sum(user_actions.order_amount),
    sum(user_actions.payment_count),
    sum(user_actions.payment_amount),
    sum(user_actions.comment_count)
from
(
    select
        user_id, order_count, order_amount , 0 payment_count ,
        0 payment_amount, 0 comment_count
    from tmp_order

    union all
    select  user_id, 0, 0, payment_count, payment_amount, 0
    from tmp_payment

    union all
    select  user_id, 0, 0, 0, 0, comment_count
    from tmp_comment
 ) user_actions
group by user_id;
```


从订单表 dwd_order_info 中获取 下单次数 和 下单总金额

从支付流水表 dwd_payment_info 中获取 支付次数 和 支付总金额

从事件日志评论表 dwd_comment_log 中获取评论次数

最终按照 user_id 聚合，获得明细，跟之前的 mid_id 聚合不同

6.9 需求一：GMV 成交总额

GMV分析

1）建表语句

```
drop table if exists ads_gmv_sum_day;
create external table ads_gmv_sum_day(
    `dt` string COMMENT '统计日期',
    `gmv_count`  bigint COMMENT '当日gmv订单个数',
    `gmv_amount`  decimal(16,2) COMMENT '当日gmv订单总金额',
    `gmv_payment`  decimal(16,2) COMMENT '当日支付金额'
) COMMENT '每日活跃用户数量'
row format delimited  fields terminated by '\t'
location '/warehouse/gmall/ads/ads_gmv_sum_day/'
;
```

2）数据导入

```
insert into table ads_gmv_sum_day
select
    '2019-02-10' dt ,
    sum(order_count)  gmv_count ,
    sum(order_amount) gmv_amount ,
    sum(payment_amount) payment_amount
from dws_user_action
where dt ='2019-02-10'
group by dt;
```


从用户行为宽表中 dws_user_action，根据统计日期分组，聚合，直接 sum 就可以了。

6.10 需求二：转化率

6.10.1 新增用户占日活跃用户比率表

1）建表语句

```
drop table if exists ads_user_convert_day;
create external table ads_user_convert_day(
    `dt` string COMMENT '统计日期',
    `uv_m_count`  bigint COMMENT '当日活跃设备',
    `new_m_count`  bigint COMMENT '当日新增设备',
    `new_m_ratio`  decimal(10,2) COMMENT '当日新增占日活的比率'
) COMMENT '每日活跃用户数量'
row format delimited  fields terminated by '\t'
location
'/warehouse/gmall/ads/ads_user_convert_day/'
;
```

2）数据导入

```
insert into table ads_user_convert_day
select
    '2019-02-10',
    sum( uc.dc) sum_dc,
    sum( uc.nmc) sum_nmc,
    cast(sum(  uc.nmc)/sum(  uc.dc)*100  as decimal(10,2))  new_m_ratio
 from
(
    select
        day_count dc,
        0 nmc
    from ads_uv_count
    where  dt='2019-02-10'
    union all
    select
        0 dc,
        new_mid_count nmc
    from ads_new_mid_count
    where create_date='2019-02-10'
)uc;
```


从日活跃数表 ads_uv_count 和 日新增设备数表 ads_new_mid_count 中取即可。

6.10.2 用户行为转化率表

1）建表语句

```
drop table if exists ads_user_action_convert_day;
create external table ads_user_action_convert_day (
    `dt` string COMMENT '统计日期',
    `total_visitor_m_count`  bigint COMMENT '总访问人数',
    `order_u_count` bigint   COMMENT '下单人数',
    `visitor2order_convert_ratio`   decimal(10,2)  COMMENT '访问到下单转化率',
    `payment_u_count` bigint     COMMENT '支付人数',
    `order2payment_convert_ratio` decimal(10,2) COMMENT '下单到支付的转化率'
 ) COMMENT '每日用户行为转化率统计'
row format delimited  fields terminated by '\t'
location '/warehouse/gmall/ads/ads_user_convert_day/'
;
```

2）数据导入

```
insert into table ads_user_action_convert_day
select
    '2019-02-10',
    uv.day_count,
    ua.order_count,
    cast(ua.order_count/uv.day_count as  decimal(10,2)) visitor2order_convert_ratio,
    ua.payment_count,
    cast(ua.payment_count/ua.order_count           as decimal(10,2)) order2payment_convert_ratio
from
(
    select
        dt,
        sum(if(order_count>0,1,0)) order_count,
        sum(if(payment_count>0,1,0)) payment_count
    from dws_user_action
    where  dt='2019-02-10'
    group by dt
)ua join ads_uv_count uv on  uv.dt=ua.dt
;
```


从用户行为宽表 dws_user_action 中取，下单人数（只要下单次数>0）,支付人数（只要支付次数>0）

从日活跃数表 ads_uv_count 中取活跃人数，然后对应的相除就可以了。

6.11 需求三：品牌复购率

需求：以月为单位统计，购买 2 次以上商品的用户

6.11.1 用户购买商品明细表（宽表）

1）创建用户购买商品明细表

```
create external table  dws_sale_detail_daycount
(   user_id   string  comment '用户 id',
    sku_id    string comment '商品 Id',
    user_gender  string comment '用户性别',
    user_age string  comment '用户年龄',
    user_level string comment '用户等级',
    order_price decimal(10,2) comment '商品价格',
    sku_name string   comment '商品名称',
    sku_tm_id string   comment '品牌id',
    sku_category3_id string comment '商品三级品类id',
    sku_category2_id string comment '商品二级品类id',
    sku_category1_id string comment '商品一级品类id',
    sku_category3_name string comment '商品三级品类名称',
    sku_category2_name string comment '商品二级品类名称',
    sku_category1_name string comment '商品一级品类名称',
    spu_id  string comment '商品 spu',
    sku_num  int comment '购买个数',
    order_count string comment '当日下单单数',
    order_amount string comment '当日下单金额'
) COMMENT '用户购买商品明细表'
PARTITIONED BY ( `dt` string)
```

2）导入数据

```
with
tmp_detail as
(
    select user_id, sku_id,
        sum(sku_num) sku_num , count(*) order_count ,
        sum(od.order_price*sku_num)  order_amount
    from dwd_order_detail od
    where od.dt='2019-02-10' group by user_id, sku_id
)
insert overwrite table  dws_sale_detail_daycount partition(dt='2019-02-10')
select
    tmp_detail.user_id,
    tmp_detail.sku_id,
    u.gender,
    months_between('2019-02-10', u.birthday)/12  age,
    u.user_level,
    price,
    sku_name,
    tm_id,
    category3_id ,
    category2_id ,
    category1_id ,
    category3_name ,
    category2_name ,
    category1_name ,
    spu_id,
    tmp_detail.sku_num,
    tmp_detail.order_count,
    tmp_detail.order_amount
from tmp_detail
left join dwd_user_info u on tmp_detail.user_id=u.id  and u.dt='2019-02-10'
left join dwd_sku_info s on tmp_detail.sku_id =s.id  and s.dt='2019-02-10';
```

6.11.2 品牌复购率表

1）建表语句

```
drop  table ads_sale_tm_category1_stat_mn;
create external  table ads_sale_tm_category1_stat_mn
(
    tm_id string comment '品牌id ' ,
    category1_id string comment '1级品类id ',
    category1_name string comment '1级品类名称 ',
    buycount   bigint comment  '购买人数',
    buy_twice_last bigint  comment '两次以上购买人数',
    buy_twice_last_ratio decimal(10,2) comment '单次复购率',
    buy_3times_last   bigint comment  '三次以上购买人数',
    buy_3times_last_ratio decimal(10,2) comment '多次复购率' ,
    stat_mn string comment '统计月份',
    stat_date string comment '统计日期'
)   COMMENT '复购率统计'
row format delimited  fields terminated by '\t'
location /warehouse/gmall/ads/ads_sale_tm_category1_stat_mn/'
;
```

2）导入数据

```
insert into table ads_sale_tm_category1_stat_mn
select
    mn.sku_tm_id,
    mn.sku_category1_id,
    mn.sku_category1_name,
    sum(if(mn.order_count>=1,1,0)) buycount,
    sum(if(mn.order_count>=2,1,0)) buyTwiceLast,
    sum(if(mn.order_count>=2,1,0))/sum( if(mn.order_count>=1,1,0)) buyTwiceLastRatio,
    sum(if(mn.order_count>3,1,0))  buy3timeLast  ,
    sum(if(mn.order_count>=3,1,0))/sum( if(mn.order_count>=1,1,0)) buy3timeLastRatio ,
    date_format('2019-02-10' ,'yyyy-MM') stat_mn,
    '2019-02-10' stat_date
from
(
    select
        user_id,
        od.sku_tm_id,
        od.sku_category1_id,
        od.sku_category1_name,
        sum(order_count) order_count
    from  dws_sale_detail_daycount  od
    where date_format(dt,'yyyy-MM')=date_format('2019-02-10' ,'yyyy-MM')
    group by user_id ,od.sku_tm_id, od.sku_category1_id, od.sku_category1_name
) mn
group by mn.sku_tm_id, mn.sku_category1_id, mn.sku_category1_name
;
```


从用户购买商品明细宽表 dws_sale_detail_daycount 中，根据品牌 id--sku_tm_id 聚合，计算每个品牌购买的总次数，购买人数 a=购买次数>=1,两次及以上购买人数 b=购买次数>=2，三次及以上购买人数 c=购买次数>=3,

单次复购率=b/a，多次复购率=c/a

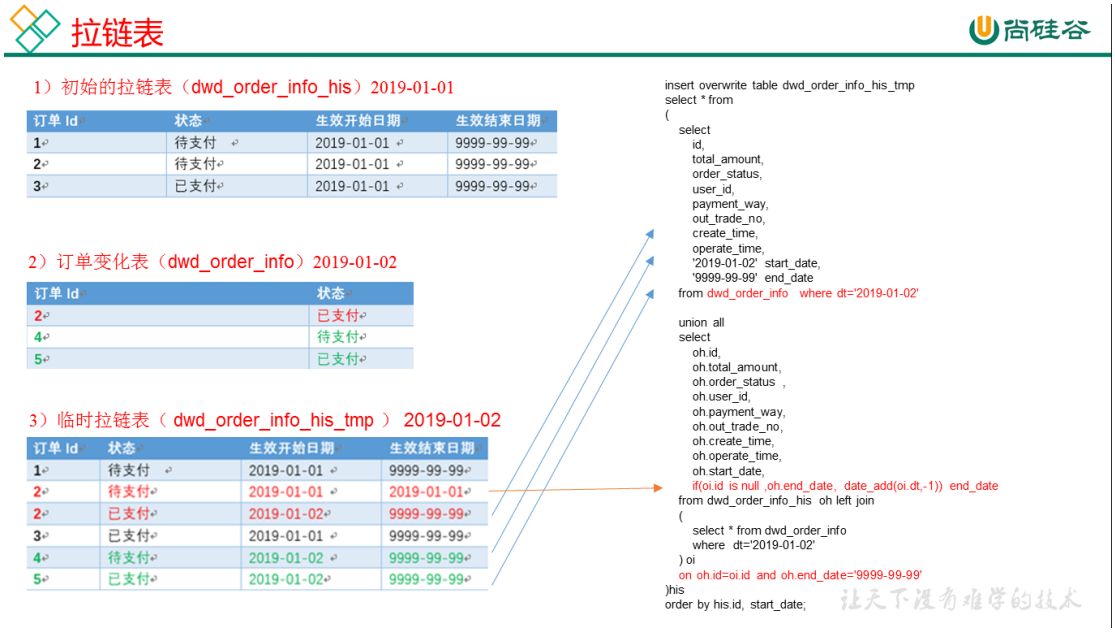
6.12 项目中有多少张宽表

宽表要 3-5 张，用户行为宽表，用户购买商品明细行为宽表，商品宽表，购物车宽表，物流宽表、登录注册、售后等。

1）为什么要建宽表

需求目标，把每个用户单日的行为聚合起来组成一张多列宽表，以便之后关联用户维度信息后进行，不同角度的统计分析。

6.13 拉链表



订单表拉链表 dwd_order_info_his

`id` string COMMENT '订单编号',

`total_amount` decimal(10,2) COMMENT '订单金额',

`order_status` string COMMENT '订单状态',

`user_id` string COMMENT '用户 id' ,

`payment_way` string COMMENT '支付方式',

`out_trade_no` string COMMENT '支付流水号',

`create_time` string COMMENT '创建时间',

`operate_time` string COMMENT '操作时间' ,

`start_date` string COMMENT '有效开始日期',

`end_date` string COMMENT '有效结束日期'

1）创建订单表拉链表，字段跟拉链表一样，只增加了有效开始日期和有效结束日期

初始日期，从订单变化表 ods_order_info 导入数据，且让有效开始时间=当前日期，有效结束日期=9999-99-99

（从 mysql 导入数仓的时候就只导了新增的和变化的数据 ods_order_info，dwd_order_info 跟 ods_order_info 基本一样，只多了一个 id 的判空处理）

2）建一张拉链临时表 dwd_order_info_his_tmp，字段跟拉链表完全一致

3）新的拉链表中应该有这几部分数据，

（1）增加订单变化表 dwd_order_info 的全部数据

（2）更新旧的拉链表左关联订单变化表 dwd_order_info，关联字段：订单 id, where 过滤出 end_date 只等于 9999-99-99 的数据，如果旧的拉链表中的 end_date 不等于 9999-99-99，说明已经是终态了，不需要再更新

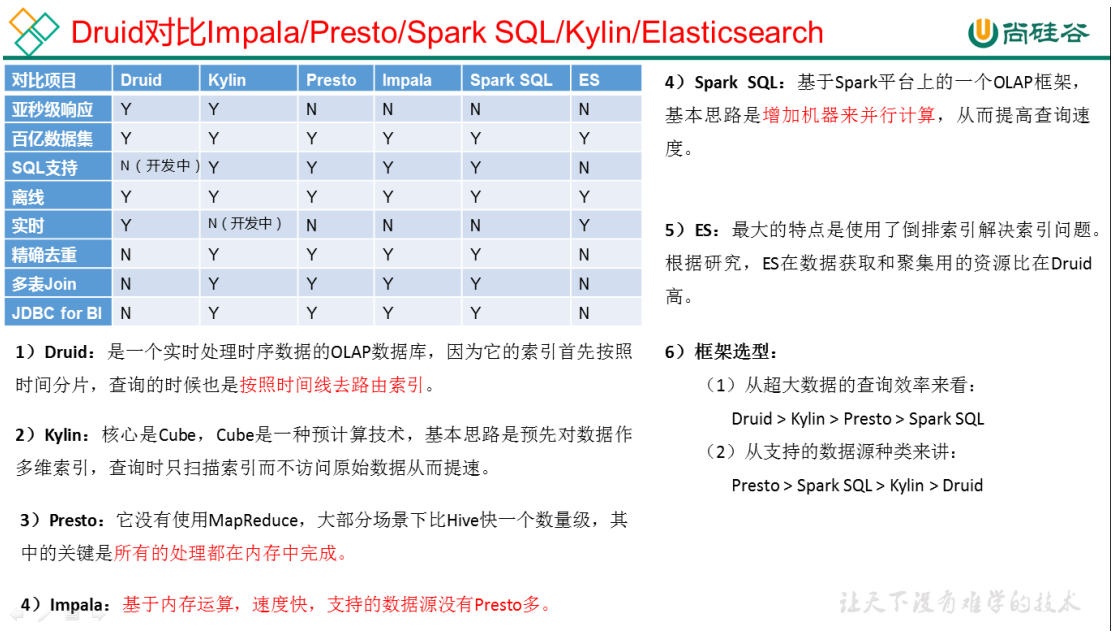
如果 dwd_order_info.id is null，没关联上，说明数据状态没变，让 end_date 还等于旧的 end_date

如果 dwd_order_info.id is not null，关联上了，说明数据状态变了，让 end_date 等于当前日期-1

把查询结果插入到拉链临时表中

4）把拉链临时表覆盖到旧的拉链表中

第 7 章 即席查询数据仓库



Kylin: T+1

Impala: CDH

Presto: Apache 版本框架

第 8 章 项目中遇到过哪些问题

8.1 Hadoop 宕机

（1）如果 MR 造成系统宕机。此时要控制 Yarn 同时运行的任务数，和每个任务申请的最大内存。调整参数：yarn.scheduler.maximum-allocation-mb（单个任务可申请的最多物理内存量，默认是 8192MB）

（2）如果写入文件过量造成 NameNode 宕机。那么调高 Kafka 的存储大小，控制从 Kafka 到 HDFS 的写入速度。高峰期的时候用 Kafka 进行缓存，高峰期过去数据同步会自动跟上。

8.2 Ganglia 监控

Ganglia 监控 Flume 发现发现尝试提交的次数大于最终成功的次数

（1）增加 Flume 内存

（2）增加 Flume 台数

8.3 Flume 小文件

Flume 上传文件到 HDFS 时参数大量小文件?

调整 hdfs.rollInterval、hdfs.rollSize、hdfs.rollCount 这三个参数的值。

8.4 Kafka 挂掉

（1）Flume 记录

（2）日志有记录

（3）短期没事

8.5 Kafka 消息数据积压，Kafka 消费能力不足怎么处理？

（1）如果是 Kafka 消费能力不足，则可以考虑增加 Topic 的分区数，并且同时提升消费组的消费者数量，消费者数=分区数。（两者缺一不可）

（2）如果是下游的数据处理不及时：提高每批次拉取的数量。批次拉取数据过少（拉取数据/处理时间<生产速度），使处理的数据小于生产的数据，也会造成数据积压。

8.6 Kafka 数据重复

在下一级消费者中去重。（redis、SparkStreaming）

8.7 Mysql 高可用

Hive 的 metadata 存储在 MySql 中（配置 MySql 的高可用（主从复制和读写分离和故障转移））

8.8 自定义 UDF 和 UDTF 解析和调试复杂字段

自定义 UDF（extends UDF 实现 evaluate 方法） 解析公共字段

自定义 UDTF(extends Genertic UDTF->实现三个方法 init(指定返回值的名称和类型)、process(处理字段一进多出)、close 方法) -> 更加灵活以及方便定义 bug

8.9 Sqoop 数据导出 Parquet

Ads 层数据用 Sqoop 往 MySql 中导入数据的时候，如果用了 orc（Parquet）不能导入，需转化成 text 格式

8.10 Sqoop 数据导出控制

Sqoop 中导入导出 Null 存储一致性问题：

Hive 中的 Null 在底层是以“\N”来存储，而 MySQL 中的 Null 在底层就是 Null，为了保证数据两端的一致性。在导出数据时采用--input-null-string 和--input-null-non-string 两个参数。导入数据时采用--null-string 和--null-non-string。

8.11 Sqoop 数据导出一致性问题

当 Sqoop 导出数据到 MySql 时，使用 4 个 map 怎么保证数据的一致性

因为在导出数据的过程中 map 任务可能会失败，可以使用—staging-table –clear-staging

```
sqoop export --connect
jdbc:mysql://192.168.137.10:3306/user_behavior --username root
--password 123456 --table app_cource_study_report --columns
watch_video_cnt,complete_video_cnt,dt --fields-terminated-by
"\t" --export-dir
"/user/hive/warehouse/tmp.db/app_cource_study_analysis_${day}"
--staging-table app_cource_study_report_tmp --clear-staging-
table --input-null-string '\N'
```

任务执行成功首先在 tmp 临时表中，然后将 tmp 表中的数据复制到目标表中（这个时

候可以使用事务，保证事务的一致性）

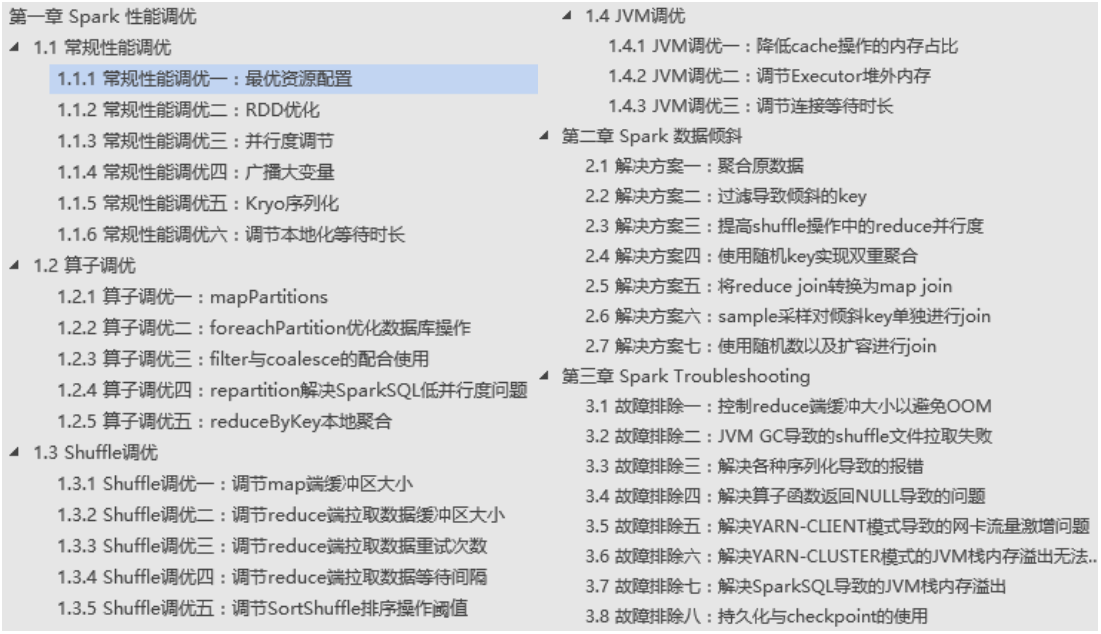
8.12 SparkStreaming 优雅关闭

如何优雅的关闭 SparkStreaming 任务（将写好的代码打包，Spark-Submit）

Kill -9 xxx ?

开启另外一个线程每 5 秒监听 HDFS 上一个文件是否存在。如果检测到存在，调用 ssc.stop()方法关闭 SparkStreaming 任务（当你要关闭任务时，可以创建你自定义监控的文件目录）

8.13 Spark OOM、数据倾斜解决



第 9 章 项目经验

9.1 框架经验

9.1.1 Hadoop

1）Hadoop 集群基准测试（HDFS 的读写性能、MapReduce 的计算能力测试）

2）一台服务器一般都有很多个硬盘插槽（插了几个插槽）

如果不配置 datanode.data.dir 多目录，每次插入一块新的硬盘都需要重启服务器

配置了即插即用

3）Hdfs 参数调优

Namenode 有一个工作线程池，用来处理与 datanode 的心跳（报告自身的健康状况和文

件恢复请求）和元数据请求　　dfs.namenode.handler.count=20 * log2(Cluster Size)

4）编辑日志存储路径 dfs.namenode.edits.dir 设置与镜像文件存储路径 dfs.namenode.name.dir 尽量分开，达到最低写入延迟（提高写入的吞吐量）

5）YARN 参数调优 yarn-site.xml

（1）服务器节点上 YARN 可使用的物理内存总量，默认是 8192（MB）

（2）单个任务可申请的最多物理内存量，默认是 8192（MB）。

6）HDFS 和硬盘空闲控制在 70%以下。

9.1.2 Flume

1）Flume 内存配置为 4G（flume-env.sh 修改）

2）FileChannel 优化

通过配置 dataDirs 指向多个路径，每个路径对应不同的硬盘，增大 Flume 吞吐量。

checkpointDir 和 backupCheckpointDir 也尽量配置在不同硬盘对应的目录中，保证 checkpoint 坏掉后，可以快速使用 backupCheckpointDir 恢复数据

3）Sink：HDFS Sink 小文件处理

这三个参数配置写入 HDFS 后会产生小文件，hdfs.rollInterval、hdfs.rollSize、hdfs.rollCount

9.1.3 Kafka

1）Kafka 的吞吐量测试（测试生产速度和消费速度）

2） Kafka 内存为 6G（不能超过 6G）

3）Kafka 数量确定：2 * 峰值生产速度（m/s）* 副本数 / 100 + 1 = ?

4）Kafka 中的数据量计算

每天数据总量 100g(1 亿条)　　10000 万/24/60/60 = 1150 条/s

平均每秒钟：1150 条

低谷每秒：400 条

高峰每秒钟：1150 * 10 = 11000 条

每条日志大小： 1K 左右

每秒多少数据量：20MB

5） Kafka 消息数据积压，Kafka 消费能力不足怎么处理？

（1）如果是 Kafka 消费能力不足，则可以考虑增加 Topic 的分区数，并且同时提升消

费组的消费者数量，消费者数=分区数。（两者缺一不可）

（2）如果是下游的数据处理不及时：提高每批次拉取的数量。批次拉取数据过少（拉取数据/处理时间<生产速度），使处理的数据小于生产的数据，也会造成数据积压。

9.1.4 MySql 之元数据备份

元数据备份（重点，如数据损坏，可能整个集群无法运行，至少要保证每日零点之后备份到其它服务器两个复本）

MySQL基于Keepalived实现HA搭建文

9.1.5 Tez 引擎优点？

Tez 可以将多个有依赖的作业转换为一个作业，这样只需写一次 HDFS，且中间节点较少，从而大大提升作业的计算性能。

9.1.6 Sqoop 参数

1）Sqoop 导入导出 Null 存储一致性问题

2）Sqoop 数据导出一致性问题

–staging-table 方式 --clear-staging

3）Sqoop 数据导出的时候一次执行多长时间

Sqoop 任务 5 分钟-2 个小时的都有。取决于数据量。

9.1.7 Azkaban 每天执行多少个任务

1）每天集群运行多少 job?

2）多个指标（200）*6=1200（1000-2000 个 job）

3）每天集群运行多少个 task? 1000*（5-8）=5000 多个

4）任务挂了怎么办？运行成功或者失败都会发邮件

Zip a.job b.job c.job　job.zip　把压缩的 zip 包放到 azkaban 的 web 界面上提交（指定 sechduler）

9.2 业务经验

9.2.1 ODS 层采用什么压缩方式和存储格式？

压缩采用 Snappy，存储采用 orc，压缩比是 100g 数据压缩完 10g 左右。

9.2.2 DWD 层做了哪些事？

1）数据清洗

（1）空值去除

（2）过滤核心字段无意义的数据，比如订单表中订单 id 为 null，支付表中支付 id 为空

（3）对手机号、身份证号等敏感数据脱敏

（4）对业务数据传过来的表进行维度退化和降维。

（5）将用户行为宽表和业务表进行数据一致性处理

select case when a is null then b else a end as JZR,

...

from A

2）清洗的手段

Sql、mr、rdd、kettle、Python（项目中采用 sql 进行清除）

3）清洗掉多少数据算合理

1 万条数据清洗掉 1 条。

9.2.3 DWS 层做了哪些事？

1）DWS 层有 3-5 张宽表（处理 100-200 个指标　70%以上的需求）

具体宽表名称：用户行为宽表，用户购买商品明细行为宽表，商品宽表，购物车宽表，物流宽表、登录注册、售后等。

2）哪个宽表最宽？大概有多少个字段？

最宽的是用户行为宽表。大概有 60-100 个字段

3）具体用户行为宽表字段名称

评论、打赏、收藏、关注--商品、关注--人、点赞、分享、好价爆料、文章发布、活跃、签到、补签卡、幸运屋、礼品、金币、电商点击、gmv

```
CREATE TABLE `app_usr_interact`(
  `stat_dt` date COMMENT '互动日期',
  `user_id` string COMMENT '用户 id',
  `nickname` string COMMENT '用户昵称',
  `register_date` string COMMENT '注册日期',
  `register_from` string COMMENT '注册来源',
  `remark` string COMMENT '细分渠道',
  `province` string COMMENT '注册省份',
  `pl_cnt` bigint COMMENT '评论次数',
```

```
    `ds_cnt` bigint COMMENT '打赏次数',
    `sc_add` bigint COMMENT '添加收藏',
    `sc_cancel` bigint COMMENT '取消收藏',
    `gzg_add` bigint COMMENT '关注商品',
    `gzg_cancel` bigint COMMENT '取消关注商品',
    `gzp_add` bigint COMMENT '关注人',
    `gzp_cancel` bigint COMMENT '取消关注人',
    `buzhi_cnt` bigint COMMENT '点不值次数',
    `zhi_cnt` bigint COMMENT '点值次数',
    `zan_cnt` bigint COMMENT '点赞次数',
    `share_cnts` bigint COMMENT '分享次数',
    `bl_cnt` bigint COMMENT '爆料数',
    `fb_cnt` bigint COMMENT '好价发布数',
    `online_cnt` bigint COMMENT '活跃次数',
    `checkin_cnt` bigint COMMENT '签到次数',
    `fix_checkin` bigint COMMENT '补签次数',
    `house_point` bigint COMMENT '幸运屋金币抽奖次数',
    `house_gold` bigint COMMENT '幸运屋积分抽奖次数',
    `pack_cnt` bigint COMMENT '礼品兑换次数',
    `gold_add` bigint COMMENT '获取金币',
    `gold_cancel` bigint COMMENT '支出金币',
    `surplus_gold` bigint COMMENT '剩余金币',
    `event` bigint COMMENT '电商点击次数',
    `gmv_amount` bigint COMMENT 'gmv',
    `gmv_sales` bigint COMMENT '订单数')
PARTITIONED BY ( `dt` string)
```

5）商品详情 ----- 购物车 ----- 订单 ------ 付款的转换比率

5% 30% 70%

6）每天的 GMV 是多少，哪个商品卖的最好？每天下单量多少？

（1）100 万的日活每天大概有 10 万人购买，平均每人消费 100 元，一天的 GMV 在 1000 万

（2）面膜，每天销售 5000 个

（3）每天下单量在 10 万左右

9.2.4 分析过哪些指标（一分钟至少说出 30 个指标）

1 离线指标

网站流量指标 独立访问数 UV 页面访客数 PV

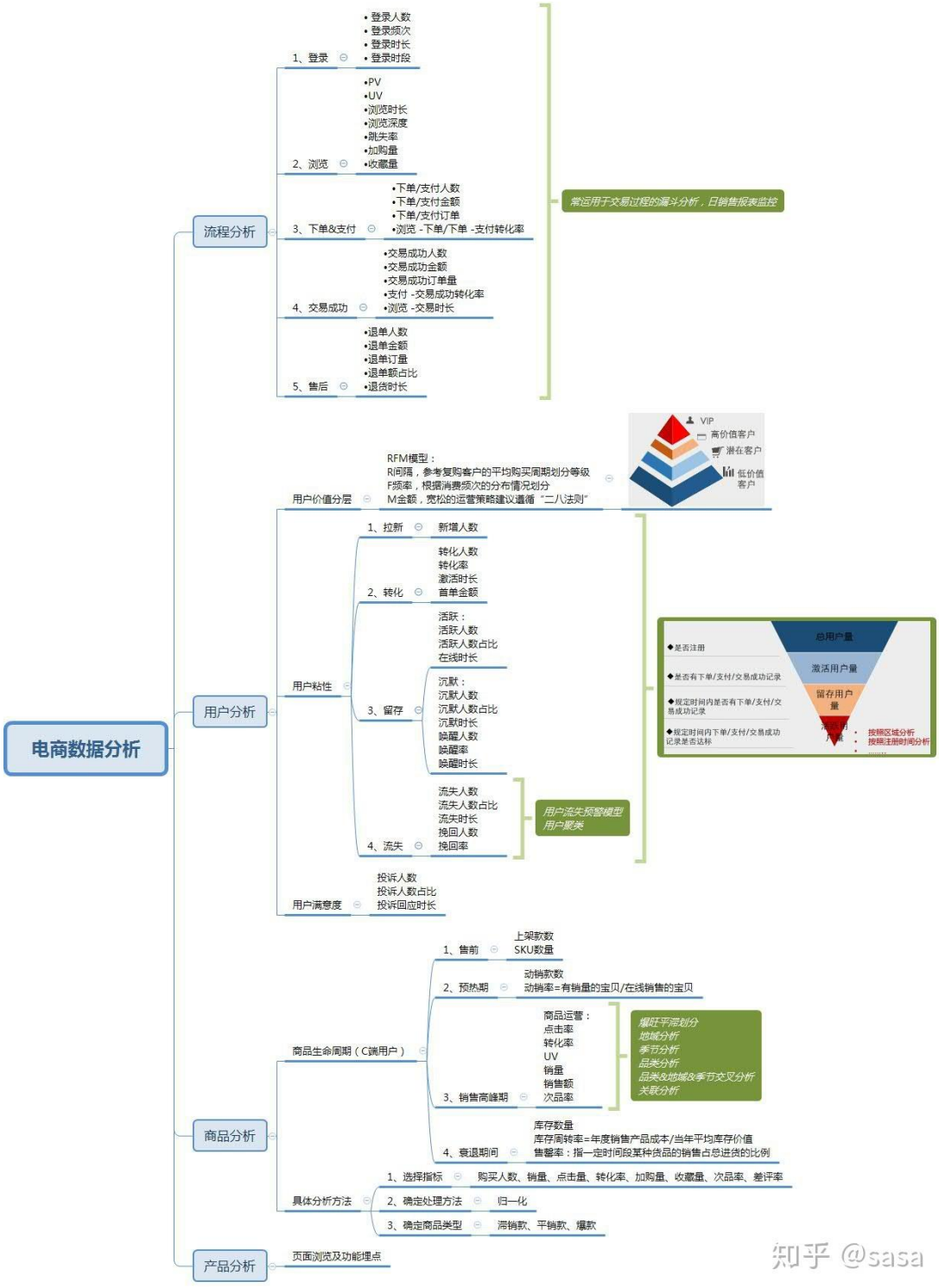
流量质量指标类 跳出率 平均页面访问时长 人均页面访问数

2 购物车类指标 加入购物车次数 加入购物车买家次数 加入购物车商品数

购物车支付转化率

3 下单类指标 下单笔数 下单金额 下单买家数 浏览下单转化率

4 支付类指标 支付金额 支付买家数 支付商品数 浏览-支付买家转化率
下单-支付金额转化率 下单-支付买家数转换率
5 交易类指标 交易成功订单数 交易成功金额 交易成功买家数 交易成功商品数
交易失败订单数 交易失败订单金额 交易失败买家数
交易失败商品数 退款总订单量 退款金额 退款率
6 市场营销活动指标 新增访问人数 新增注册人数 广告投资回报率 UV 订单转化率
7 风控类指标 买家评价数 买家上传图片数 买家评价率 买家好评率 买家差评率
物流平均配送时间
8 投诉类指标 发起投诉数 投诉率 撤销投诉(申诉数)
9 商品类指标 产品总数 SKU 数 SPU 数
上架商品 SKU 数 上架商品 SPU 数 上架商品数



日活跃用户，
月活跃用户，
各区域 Top10 商品统计，
季度商品品类点击率 top10，
用户留存，
月 APP 的用户增长人数，
广告区域点击数 top3，

活跃用户每天在线时长，
投诉人数占比，
沉默用户占比，
用户的新鲜度，
商品上架的 sku 数，
同种品类的交易额排名，
统计买家的评价率，
用户浏览时长，
统计下单的数量，
统计支付的数量，
统计退货的数量，
用户的（日活、月活、周活），
统计流失人数

日活，周活，月活，沉默用户占比，增长人数，活跃用户占比，在线时长统计，歌曲访问数，歌曲访问时长，各地区 Top10 歌曲统计 ,投诉人数占比，投诉回应时长，留存率，月留存率，转化率，GMV，复购 vip 率，vip 人数，歌榜，挽回率，粉丝榜，打赏次数，打赏金额，发布歌曲榜单，歌曲热度榜单，歌手榜单，用户年龄组，vip 年龄组占比，收藏数榜单，评论数

1.用户活跃数统计（日活，月活，周活）
2.某段时间的新增用户/活跃用户数
3.页面单跳转化率统计
4.活跃人数占比（占总用户比例）
5.在线时长统计（活跃用户每天在线时长）
12.统计本月的人均在线时长
6.订单产生效率（下单的次数与访问次数比）
7.页面访问时长（单个页面访问时长）
8.统计本季度付款订单
9.统计某广告的区城点击数 top3
10.统计本月用户的流失人数
11.统计本月流失人数占用户人数的比例
13.统计本月 APP 的用户增长人数
14.统计本月的沉默用户
15.统计某时段的登录人数
16.统计本日用户登录的次数平均值
17.统计用户在某类型商品中的浏览深度（页面转跳率）
18.统计用户从下单开始到交易成功的平均时长
19.Top10 热门商品的统计
20.统计下单的数量
21.统计支付的数量
22.统计退货的数量
23.统计动销率（有销量的商品/在线销售的宝贝）
24.统计支付转化率
25.统计用户的消费频率

26.统计商品上架的 SKU 数
27.统计同种品类的交易额排名
28.统计按下单退款排序的 top10 的商品
29.统计本 APP 的投诉人数占用户人数的比例
30.用户收藏商品

9.2.5 分析过最难的两个指标，现场手写

最近连续3周活跃用户数

最近3周连续活跃的用户：通常是周一对前3周的数据做统计，该数据一周计算一次。

1）创建表

```
drop table if exists ads_continuity_wk_count;
create external table ads_continuity_wk_count(
    `dt` string COMMENT '统计日期,一般用结束周周日日期,如果每天计算一次,可用当天日期',
    `wk_dt` string COMMENT '持续时间',
    `continuity_count` bigint
)
row format delimited fields terminated by '\t'
location '/warehouse/gmall/ads/ads_continuity_wk_count';
```

前一周	前二周	前三周
100	101	101
101	102	104
102		

2）导入数据

```
insert into table ads_continuity_wk_count
select
     '2019-02-20',
     concat(date_add(next_day('2019-02-20','MO'),-7*3),'_',date_add(next_day('2019-02-20','MO'),-1)),
     count(*)
from
(
    select mid_id
    from dws_uv_detail_wk
    where wk_dt>=concat(date_add(next_day('2019-02-20','MO'),-7*3),'_',date_add(next_day('2019-02-20','MO'),-7*2-1))
    and wk_dt<=concat(date_add(next_day('2019-02-20','MO'),-7),'_',date_add(next_day('2019-02-20','MO'),-1))
    group by mid_id
    having count(*)=3
)t1;
```

按照设备id对每周活跃表分组

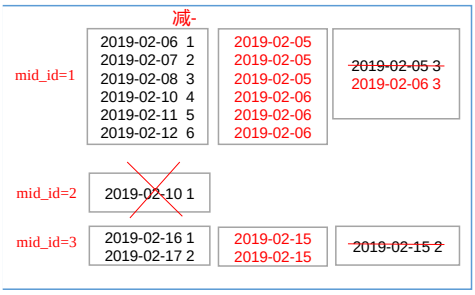
统计次数等于3

100	1
101	3
102	2
104	1

最近7天内连续3天活跃用户数

1）创建表

```
drop table if exists ads_continuity_uv_count;
create external table ads_continuity_uv_count(
    `dt` string COMMENT '统计日期',
    `wk_dt` string COMMENT '最近7天日期',
    `continuity_count` bigint
) COMMENT '连续活跃设备数'
row format delimited fields terminated by '\t'
location '/warehouse/gmall/ads/ads_continuity_uv_count';
```



2）导入数据

```
insert into table ads_continuity_uv_count
select
    '2019-02-12',
    concat(date_add('2019-02-12',-6),'_','2019-02-12'),
    count(*)
from
(
    select mid_id
    from
    (
        select mid_id
        from
        (
            select
                mid_id,
                date_sub(dt,rank) date_dif
            from
            (
                select
                    mid_id,
                    dt,
                    rank() over(partition by mid_id order by dt) rank
                from dws_uv_detail_day
                where dt>=date_add('2019-02-12',-6) and dt<='2019-02-12'
            )t1
        )t2
        group by mid_id,date_dif
        having count(*)>=3
    )t3
    group by mid_id
)t4;
```

2、计算用户活跃日期及排名之间的差值

1、查询出最近7天的活跃用户，并对用户活跃日期进行排名

3、对同用户及差值分组，统计差值个数

4、将差值相同个数大于等于3的数据取出

5、对数据去重

9.2.6 数据仓库每天跑多少张表，大概什么时候运行，运行多久？

基本一个项目建一个库，表格个数为初始的原始数据表格加上统计结果表格的总数。（一般 70-100 张表格）

每天 0：30 开始运行。

所有离线数据报表控制在 8 小时之内

大数据实时处理部分控制在 5 分钟之内。

评分标准：5 分

9.2.7 数仓中使用的哪种文件存储格式

常用的包括：textFile， ORC，Parquet，一般企业里使用 ORC 或者 Parquet，因为是列式存储，且压缩比非常高，所以相比于 textFile，查询速度快，占用硬盘空间少

9.2.8 数仓中用到过哪些 Shell 脚本及具体功能

1）集群启动停止脚本（Hadoop、Flume、Kafka、Zookeeper）

2）Sqoop 和数仓之间的导入导出脚本

3）数仓层级之间的数据导入脚本。

9.2.9 项目中用过的报表工具

Echarts、kibana

9.2.10 测试相关

1）公司有多少台测试服务器？

测试服务器一般三台

2）测试数据哪来的？

一部分自己写 Java 程序自己造，一部分从生产环境上取一部分。

3）如何保证写的 sql 正确性

需要造一些特定的测试数据，测试。

离线数据和实时数据分析的结果比较。

4）测试环境什么样？

测试环境的配置是生产的一半

5）测试之后如何上线？

上线的时候，将脚本打包，提交 git。先发邮件抄送经理和总监，运维。通过之后跟运维一起上线。

9.2.11 项目实际工作流程

1）先与产品讨论，看报表的各个数据从哪些埋点中取

2）将取得逻辑过程设计好，与产品确定后开始开发

3）开发出报表 SQL 脚本，并且跑几天的历史数据，观察结果

4）将报表放入调度任务中，第二天给产品看结果。

5）周期性将表结果导出或是导入后台数据库，生成可视化报表

9.2.12 项目中实现一个需求大概多长时间

刚入职第一个需求大概需要 7 天左右。

对业务熟悉后，平均一天一个需求。

影响时间的因素：开会讨论需求、表的权限申请、测试等

9.2.13 项目在 3 年内迭代次数，每一个项目具体是如何迭代的。

差不多一个月会迭代一次。就产品或我们提出优化需求，然后评估时间。每周我们都会开会做下周计划和本周总结。

有时候也会去预研一些新技术。

9.2.14 项目开发中每天做什么事

新需求比如埋点或是报表来了之后，需要设计做的方案，设计完成之后跟产品讨论，再开发。

数仓的任何步骤出现问题，需要查看问题，比如日活，月活下降等。

第 10 章 JavaSE

10.1 hashMap 底层源码，数据结构

底层结构：jdk7：数组+链表 jdk8：数组+链表+红黑树

HashMap 中维护了 Node 类型的数组 table,初始为 null

1）创建对象时，将加载因子 loadFactor 初始化为 0.75,其他成员保持默认值

2）添加元素时，相当于 putVal 方法,需要先将元素的 key 哈希值获取出来，并且运算得出在数组中存放索引。

如果该索引处没有其他元素，则可以直接存放

如果该索引处有其他元素，则需要先判断是否相等，

如果相等，则覆盖

如果不相等，则继续判断，是否为树结构或链表结构，根据不同结构进行不同处理

3）如果需要扩容，则进行对应的扩容。

如果第一次添加，则扩容 table 的 capacity 为 16，临界值 threshold 为 12.

如果其他次扩容，则扩容 table 的 capacity 为 2 倍，临界值 threshold 为 2 倍.

4）当链表中节点数>=7 && capacity>=64 则将链表变成树结构

jdk7 和 jdk8 的对比：

	结构	table 数据类型	初始容量
jdk7	数组+链表	Entry	16
jdk8	数组+链表+红黑数	Node	0

jdk7 中创建对象时，则初始化 table 容量为 16（饿汉式）

jdk8 中创建对象时，并没有初始化，而是第一次添加元素初始化 table 容量为 16（懒汉式）

10.2 Java 自带有哪几种线程池

1）newCachedThreadPool

创建一个可缓存线程池，如果线程池长度超过处理需要，可灵活回收空闲线程，若无可回收，则新建线程。这种类型的线程池特点是：

工作线程的创建数量几乎没有限制(其实也有限制的,数目为 Interger. MAX_VALUE)，这样可灵活的往线程池中添加线程。

如果长时间没有往线程池中提交任务，即如果工作线程空闲了指定的时间(默认为 1 分钟)，则该工作线程将自动终止。终止后，如果你又提交了新的任务，则线程池重新创建一个工作线程。

在使用 CachedThreadPool 时，一定要注意控制任务的数量，否则，由于大量线程同时运行，很有会造成系统瘫痪。

2）newFixedThreadPool

创建一个指定工作线程数量的线程池。每当提交一个任务就创建一个工作线程，如果工作线程数量达到线程池初始的最大数，则将提交的任务存入到池队列中。FixedThreadPool 是一个典型且优秀的线程池，它具有线程池提高程序效率和节省创建线程时所耗的开销的优点。但是，在线程池空闲时，即线程池中没有可运行任务时，它不会释放工作线程，还会占用一定的系统资源。

3）newSingleThreadExecutor

创建一个单线程化的 Executor，即只创建唯一的工作者线程来执行任务，它只会用唯一的工作线程来执行任务，保证所有任务按照指定顺序(FIFO, LIFO, 优先级)执行。如果这个线程异常结束，会有另一个取代它，保证顺序执行。单工作线程最大的特点是可保证顺序地执行各个任务，并且在任意给定的时间不会有多个线程是活动的。

4）newScheduleThreadPool

创建一个定长的线程池，而且支持定时的以及周期性的任务执行，支持定时及周期性任务执行。延迟 3 秒执行。

10.3 HashMap 和 HashTable 区别

HashMap:线程不安全的，key 和 value 可以是 null

Hashtable:线程安全的，key 和 values 不可以是 null，否则会报空指针异常

10.4 TreeSet 和 HashSet 区别

HashSet 是采用 hash 表来实现的。其中的元素没有按顺序排列，add()、remove()以及 contains()等方法都是复杂度为 O(1)的方法。

TreeSet 是采用树结构实现(红黑树算法)。元素是按顺序进行排列，但是 add()、remove()以及 contains()等方法都是复杂度为 O(log (n))的方法。它还提供了一些方法来处理排序的 set，如 first(), last(), headSet(), tailSet()等等。

10.5 String buffer 和 String build 区别

1、StringBuffer 与 StringBuilder 中的方法和功能完全是等价的，

2、只是 StringBuffer 中的方法大都采用了 synchronized 关键字进行修饰，因此是线程安全的，而 StringBuilder 没有这个修饰，可以被认为是线程不安全的。

3、在单线程程序下，StringBuilder 效率更快，因为它不需要加锁，不具备多线程安全而 StringBuffer 则每次都需要判断锁，效率相对更低

10.6 Final、Finally、Finalize

final：修饰符（关键字）有三种用法：如果一个类被声明为 final，意味着它不能再派生出新的子类，即不能被继承，因此它和 abstract 是反义词。将变量声明为 final，可以保证它们在使用中不被改变，被声明为 final 的变量必须在声明时给定初值，而在以后的引用中只能读取不可修改。被声明为 final 的方法也同样只能使用，不能在子类中被重写。

finally：通常放在 try…catch 的后面构造总是执行代码块，这就意味着程序无论正常执行还是发生异常，这里的代码只要 JVM 不关闭都能执行，可以将释放外部资源的代码写在 finally 块中。

finalize：Object 类中定义的方法，Java 中允许使用 finalize() 方法在垃圾收集器将对象从内存中清除出去之前做必要的清理工作。这个方法是由垃圾收集器在销毁对象时调用的，通过重写 finalize() 方法可以整理系统资源或者执行其他清理工作。

10.7 ==和 Equals 区别

==：如果比较的是基本数据类型，那么比较的是变量的值

== ：如果比较的是引用数据类型，那么比较的是地址值（两个对象是否指向同一块内存）

equals:如果没重写 equals 方法比较的是两个对象的地址值。

如果重写了 equals 方法后我们往往比较的是对象中的属性的内容

equals 方法是从 Object 类中继承的，默认的实现就是使用==

```
public boolean equals(Object obj) {
    return (this == obj);
}
```

第 11 章 Redis

11.1 缓存穿透

1）缓存系统定义：

按照 KEY 去查询 VALUE，当 KEY 对应的 VALUE 一定不存在的时候并对 KEY 并发请求量很大的时候，就会对后端造成很大的压力。（查询一个必然不存在的数据。比如文章表，查询一个不存在的 id，每次都会访问 DB，如果有人恶意破坏，很可能直接对 DB 造成影响。）

由于缓存不命中，每次都要查询持久层。从而失去缓存的意义。

2）解决方法：

（1）缓存层缓 存空值。

缓存太多空值，占用更多空间。（优化：给个空值过 期时间）

存储层更新代码了，缓存层还是空值。（优化：后台设置时主动删除空值，并缓存把值进去）

（2）将数据库中所有的查询条件，放布隆过滤器中。当一个查询请求来临的时候，先经过布隆过滤器进行查，如果请求存在这个条件中，那么继续执行，如果不在，直接丢弃。

3）备注 ：

比如数据库中有 10000 个条件，那么布隆过滤器的容量 size 设置的要稍微比 10000 大一些，比如 12000.

对于误判率的设置，根据实际项目，以及硬件设施来具体定。但一定不能设置为 0，并且误判率设置的越小，哈希函数跟数组长度都会更多跟更长，那么对硬件，内存中间的要求就会相应 的高

private st atic BloomFilter<Inte ger> bloomFi lt er = BloomFilter.create(Funnels.integerFuel(), s ize, 000 01) ;

有了 siz 跟误判率，那么布隆过滤器会产相应的哈希函数跟数组。

综上：我们可以利用布隆过滤器，将 redis 缓存击穿制在一个可容的范围内。

11.2 哨兵模式

如果 Master 异常，则会进行 Master-Slave 切换，将其中一 Slae 作为 Master，将之前的 Master 作为 Slave

下线：

①主观下线：Subjectively Down，简称 SDOWN，指的是当前 Sentinel 实例对某个 redis 服务器做出的下线判断。

②客观下线：Objectively Down， 简称 ODOWN，指的是多个 Sentinel 实例在对 Master Server 做出 SDOWN 判断，并且通过 SENTINEL is-master-down-by-addr 命令互相交流之后，得出的 Master Server 下线判断，然后开启 failover.

工作原理：

①每个 Sentinel 以每秒钟一次的频率向它所知的 Master，Slave 以及其他 Sentinel 实例发送一个 PING 命令 ；

②如果一个实例（instance）距离最后一次有效回复 PING 命令的时间超过 down-after-milliseconds 选项所指定的值， 则这个实例会被 Sentinel 标记为主观下线；

③如果一个 Master 被标记为主观下线，则正在监视这个 Master 的所有 Sentinel 要以每秒一次的频率确认 Master 的确进入了主观下线状态；

④当有足够数量的 Sentinel（大于等于配置文件指定的值）在指定的时间范围内确认 Master 的确进入了主观下线状态， 则 Master 会被标记为客观下线 ；

⑤在一般情况下， 每个 Sentinel 会以每 10 秒一次的频率向它已知的所有 Master，Slave 发送 INFO 命令

⑥当 Master 被 Sentinel 标记为客观下线时，Sentinel 向下线的 Master 的所有 Slave 发送 INFO 命令的频率会从 10 秒一次改为每秒一次 ；

⑦若没有足够数量的 Sentinel 同意 Master 已经下线， Master 的客观下线状态就会被移除；

若 Master 重新向 Sentinel 的 PING 命令返回有效回复， Master 的主观下线状态就会被移除；

11.3 数据类型

string	字符串
list	可以重复的集合
set	不可以重复的集合
hash	类似于 Map<String,String>
zset(sorted set）	带分数的 set

11.4 持久化

1）RDB 持久化：

每隔一段时间，将内存中的数据集写到磁盘

Redis 会单独创建（fork）一个子进程来进行持久化，会先将数据写入到个临时文件中，待持久化过程都结束了，再用这个临时文件替换上次持久化好的文件。整个过程中，主进程是不进行任何 IO 操作的，这就确保了极高的性能如果需要进行大规模数据的恢复，且对于数据恢复的完整性不是非常敏感，那 RDB 方式要比 AOF 方式更加的高效。

保存策略：

save 900 1 900 秒内如果至少有 1 个 key 的值变化，则保存

save 300 10 300 秒内如果至少有 10 个 key 的值变化，则保存

save 60 1 0000　60 秒内如果至 10000 个 key 的值变化，则保存

2）AOF： 以日志形式记录每个更新（(总结、改）操作

Redis 重新启动时读取这个文件，重新执行新建、修改数据的命令恢复数据。

保存策略：

（1）appendfsync always：每次产生一条新的修改数据的命令都执行保存操作；效率低，但是安全！

（2）appendfsync everysec：每秒执行一次保存操作。如果在未保存当前秒内操作时发生了断电，仍然会导致一部分数据丢失（即 1 秒钟的数据）。

（3）appendfsync no：从不保存，将数据交给操作系统来处理。更快，也更不安全的选择。

推荐（并且也是默认）的措施为每秒 fsync 一次， 这种 fsync 策略可以兼顾速度和安全性。

缺点：

1　比起 RDB 占用更多的磁盘空间

2 恢复备份速度要慢

3 每次读写都同步的话，有一定的性能压力

4 存在个别 Bug，造成恢复不能

选择策略:

可读的日志文本，通过操作 AOF

官方推荐：

如果对数据不敏感，可以选单独用 RDB；不建议单独用 AOF，因为可能出现 Bug;如果只是做纯内存缓存，可以都不用

11.5 悲观锁

执行操作前假设当前的操作肯定（或有很大几率）会被打断（悲观）。基于这个假设，我们在做操作前就会把相关资源锁定，不允许自己执行期间有其他操作干扰。

Redis 不支持悲观锁。Redis 作为缓存服务器使用时，以操作为主，很少写操作，相应的操作被打断的几率较少。不采用悲观锁是为了防止降低性能。

11.6 乐观锁

执行操作前假设当前操作不会被打断（乐观）。基于这个假设，我们在做操作前不会锁定资源，万一发生了其他操作的干扰，那么本次操作将被放弃。

第 12 章 MySql

12.1 行锁，表锁

	MyISAM	InnoDB
行表锁	表锁，即使操作一条记录也会锁住整个表，不适合高并发的操作	行锁,操作时只锁某一行，不对其它行有影响，适合高并发的操作

12.2 索引

数据结构：B+Tree

一般来说能够达到 range 就可以算是优化了

口诀：

全职匹配我最爱，最左前缀要遵守；

带头大哥不能死，中间兄弟不能断；

索引列上少计算，范围之后全失效；

LIKE 百分写最右，覆盖索引不写*；

不等空值还有 OR，索引影响要注意；

VAR 引号不可丢，SQL 优化有诀窍。

第 13 章 JVM

13.1 GC

1）引用计数法 应用于：微软的 COM/ActionScrip3/Python 等

a） 如果对象没有被引用，就会被回收，缺点：需要维护一个引用计算器

2）复制算法 年轻代中使用的是 Minor GC，这种 GC 算法采用的是复制算法(Copying)

a） 效率高，缺点：需要内存容量大，比较耗内存

b） 使用在占空间比较小、刷新次数多的新生区

3）标记清除 老年代一般是由标记清除或者是标记清除与标记整理的混合实现

a） 效率比较低，会差生碎片。

4）**标记压缩**　老年代一般是由标记清除或者是标记清除与标记整理的混合实现

a)　效率低速度慢，需要移动对象，但不会产生碎片。

5）**标记清除**压缩标记清除-标记压缩的集合，多次 GC 后才 Compact

a)　使用于占空间大刷新次数少的养老区，是 3 4 的集合体

13.2 GC 的简单理解

JVM 的区域分为两类 堆、非堆（Perm Gen 永久代 / Java 8 改为 元空间 Metaspace），堆区：Eden Space（伊甸园）、Survivor Space(幸存者区 0/1)、Tenured Gen（养老区）新生代： Eden Space（伊甸园）+ Survivor Space 0 + Survivor Space 1 养老代：Tenured Gen

a)　一个对象被创建后，就会出现在 Eden 区，每过一段时间 GC 就会过来进行回收，当对象没有引会被回收，有引用就会进入 Survivor Space 幸存者区

b)　在幸存者区里面有两个区域，一个是有对象的区域，另外一个是没对象的区域，有且仅有一个区域有对象。当 GC 到来时，有引用的对象都会转移到另外一个幸存者区，剩下没有引用的全部回收，当达对象到 16 次的没回收时候进入养老区

c)　在养老区内，GC 来的次数比较少，但是当 GC 来的时候，没有引用的对象一样会被清除。

d)　总结：新生代空间比较小，GC 频率高，养老代空间比较大，GC 频率低

第 14 章 模拟考试

14.1 选择题

14.1.1 HDFS

1.下面哪个程序负责 HDFS 数据存储？

a)NameNode　b)Jobtracker　c)Datanode　d)secondaryNameNode　e)tasktracker

2. HDfS 中的 block 默认保存几份？

a)3 份　b)2 份　c)1 份　d)不确定

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

解析：

JobTracker 对应于 NameNode

TaskTracker 对应于 DataNode

4. HDFS 默认 Block Size

a)32MB b)64MB c)128MB

注：旧版本是 64MB

5. 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

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。

6.下面与 HDFS 类似的框架是？C

A NTFS B FAT32 C GFS D EXT3

14.1.2 集群管理

1.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

2.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

3.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

4.下列哪个是 Hadoop 运行的模式

a)单机版 b)伪分布式 c)分布式

5. Cloudera 提供哪几种安装 CDH 的方法

a)Cloudera manager b)Tarball c)Yum d)Rpm

14.3.1 Zookeeper 基础

1. 下面与 Zookeeper 类似的框架是？D

AProtobuf

B Java

C Kafka

D Chubby

14.2 判断题

14.2.1 集群管理

1. Ganglia 不仅可以进行监控，也可以进行告警。（正确）

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

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

4. Cloudera CDH 是需要付费使用的。（错误）

5. 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）

6. DataNode 通过长连接与 NameNode 保持通信。错误

7. 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）

8. Slave 节点要存储数据，所以它的磁盘越大越好。（错误）

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

10. Hadoop 默认调度器策略为 FIFO（错误 ）

11. 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）

12. 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）

13. 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－Hadoopnamenode -format ‖操作格式化磁盘。（错误 ）

14.2.2 HDFS

1. Block Size 是不可以修改的。（错误）

2. Hadoop 支持数据的随机读写。（错）

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

14.2.3 MapReduce

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

2. 每个 map 就是一个线程。（错误）

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