大数据面试吹牛草稿

V 2.0

本文档来自公众号：**五分钟学大数据**

微信扫码关注

目录

面试吹牛之前先打个草稿！

各位面试官好！

1. 我叫 xxx，毕业于 xxx，之前在 xxx 公司待了 1 年多，期间一直从事的是 IT 行业，刚开始的时候做的是 Java 开发后来转岗到大数据方向做大数据开发；刚转行到大数据开发的时候开始比较困难的，大数据并不像 Java 那样一套框架基本可以搞定所有的问题，而是不同的业务对于同一个问题有多种解决方案。

2. 我叫 xxx，毕业至今就职于 xx 公司，职位是数仓开发。
   参加工作以来，我先后参与筹备大数据服务器购买以及从 0 到 1 的搭建，离线数仓项目组，实时数仓项目组。这三个项目都得到了老板们的一致好评，大大增加了业务方需要客户数据的效率，能更快的做出实时决策。
   我对大数据各框架有浓厚的兴趣，工作之余经常钻研技术，例如 flink 的水位线，双流 join 等，对业务需求分析的比较透彻。
   工作三年，我已经连续两年被评为优秀员工，我觉得这足以说明我技术扎实，对待工作严谨，好学，当然也离不开原公司帮助过我的师傅们。

3. 我叫 xxx，毕业于 xxx，自己大学期间学了 C++就喜欢上了编程，然后大学期间利用业余时间学习了 JAVA。大四的时候又接触到了大数据，又对大数据产生兴趣，学校也开了大数据相关的专业，于是就在晚上找各种相关资料，在 B 站上搜一些比较好的大数据视频，把相关视频学习钻研了一遍。20 年毕业的时候，也从事的大数据行业的工作，参与了大数据架构的搭建，以及数仓的建模等相关的工作。又利用业余时间，进行补充知识，不断的提高自己。至于以后，还是会不断的学习来提高自己对大数据的开发能力，提高自己的业务水准。

4. 我叫 xxx，毕业于 xxx。之前 2 年多的时间里一直从事大数据开发工作。
   刚开始是在平台岗，主要负责数据平台的搭建以及维持整个框架的正常运行，从购买服务器，包括框架版本选型，以及服务器台数定制，这些都是从 0 到 1 搭建的；在平台岗工作了差不多半年时间，由于这段期间表现比较突出，公司想成立数仓组，就让我负责筹建搭建数仓的工作，我从数仓建模开始逐渐搭建元数据管理，数据质量监控，权限管理，指标分析，之后就一直数仓岗的工作；直到一年前，公司老大决定要做实时这块，要统计一个大屏的可视化展示，可

> 能觉得我的攻坚能力比较强，第一时间又想到了我，我就把这个活接过来了，我开始组建实时团队，也比较好的完成了任务。

我离职前主要是做平台的搭建以及各种指标的分析：实现和离线的都做；

我最近做的一个项目是商城平台，我们这个项目主要包含三个方面 ：

1. 数据仓库的搭建;
2. 实时计算系统;
3. 离线计算系统;

刚开始主要是负责做平台相关的工作，后来做了一段时间的实时指标，离职前主要负责离线 指标这块的内容以及一些维护优化的工作；

公司大数据部门这边刚开始有 5 个人，随后因为业务的增加又招了一些人离职前有 8 个人；

一、简要介绍项目

接下来我先介绍一下这个项目：

项目是一个高度定制化的商城平台，最近主要做的是数仓和离线指标计算这一块；

数仓初期

数仓搭建初期，由于公司数据量少，经验不足，数仓没有层级概念，过来的数据直接进行解析，每次计算一个指标的时候，都需要进行 ETL 操作，每次都需要进行 join，造成了大量的重复操作，效率十分低下，浪费大量人力。

数仓后期

数仓在搭建一段时间后，重复的计算操作困恼了我很久，后来我参考了阿里的离线数仓架构，我们对数仓进行了重新的架构搭建，对数仓进行了分层规划。

主要分为：

1. ods 层 ： 数据缓冲层;
2. dwd 层 ： 基础数据层;
3. dws 层 ： 数据汇总层;
4. app 层 ： 应用层;

分四层的原因主要是为了隔离数据然后还能复用上一层计算出来的数据，另一方面也是为了数据的备份；

二、介绍熟悉的框架

数据采集我们主要是采集**业务系统的数据**及**日志数据**两部分，业务系统数据存储在 Mysql 中，使用 Sqoop 将数据导入大数据平台。

Sqoop：

Sqoop 是在 Hadoop 生态体系和 RDBMS 体系之间传送数据的一种工具。它的工作机制是将导入或导出命令翻译成 mapreduce 程序来实现。在翻译出的 mapreduce 中主要是对 inputformat 和 outputformat 进行定制。

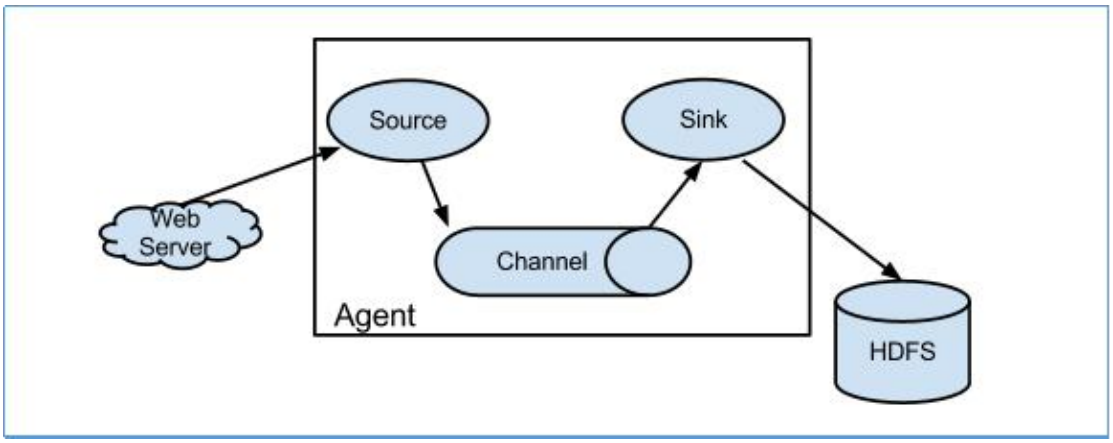
我们在使用 Sqoop 导入导出时出现了 Null 的存储一致性问题，Hive 中的 Null 在底层是以“\N”来存储，而 MySQL 中的 Null 在底层就是 Null。为了保证数据两端的一致性，在导出数据时采用--input-null-string 和 --input-null-non-string 两个参数。导入数据时采用--null-string 和 --null-non-string。

Flume：

对于日志采集我们当时选用的是 Flume，采集日志框架也有很多，之所以选择 Flume 主要是因为它采集数据的效果比较好，其次是对于 HDFS 和 Kafka 支持的也比较好；

Flume 主要包含三大组件：

Source、Channel、Sink



1. Flume 在 1.7 以后提供了一个 TailDirSource 用来支持多目录和断点续传功能;
    - 断点续传主要保证在服务器挂掉的情况下，再次启动服务数据不会丢失的问题;其原理就是在底层维护了一个 offset 偏移量(也就是每次读取文件的偏移量)Flume 会通过这个偏移量来找到上次文件读取的位置从而实现了断点续传的功能;
    - 在 1.6 以前这个实现断点续传是需要手工维护这个偏移量的会比较麻烦;
2. Channel 的种类比较多主要有：
    - MemoryChannel ： 数据放在内存中，会在 Flume 宕机的时候丢失数据，可以用在对数据安全性要求没有那么高的场景中比如日志数据;
    - FileChannel ： 不会丢失数据，因为数据是放在磁盘上边的而且支持多目录配置可以提高写入的性能，同时因为有落盘的操作所以效率比较低，适合用在对数据安全性要求比较高的场景比如金融类的数据;
    - KafkaChannel ： 主要是为了对接 Kafka，使用这个可以节省 Sink 组件也可以提升效率的，我们项目中使用的就是这个 Channel，因为下一层是使用 Kafka 来传递消息的;

3. 在 Flume 这一层我们还做了一个拦截器，主要是对收集到的日志做了一层过滤，因为有的日志没有 id 或一些关键字段，这些数据对我们数据分析来说是没有任何用的，所以在拦截器里边对这些数据进行了简单清洗;
4. 还做了一个分类型的拦截器，在这个拦截器里边我们对数据进行类型的区分，主要是做了一个打标签的功能对不同的日志数据打上不同的标签，然后通过后续的选择器 Multiplexing 将不同标签的数据放到不同的 topic 里边，方便下游对数据进行处理;

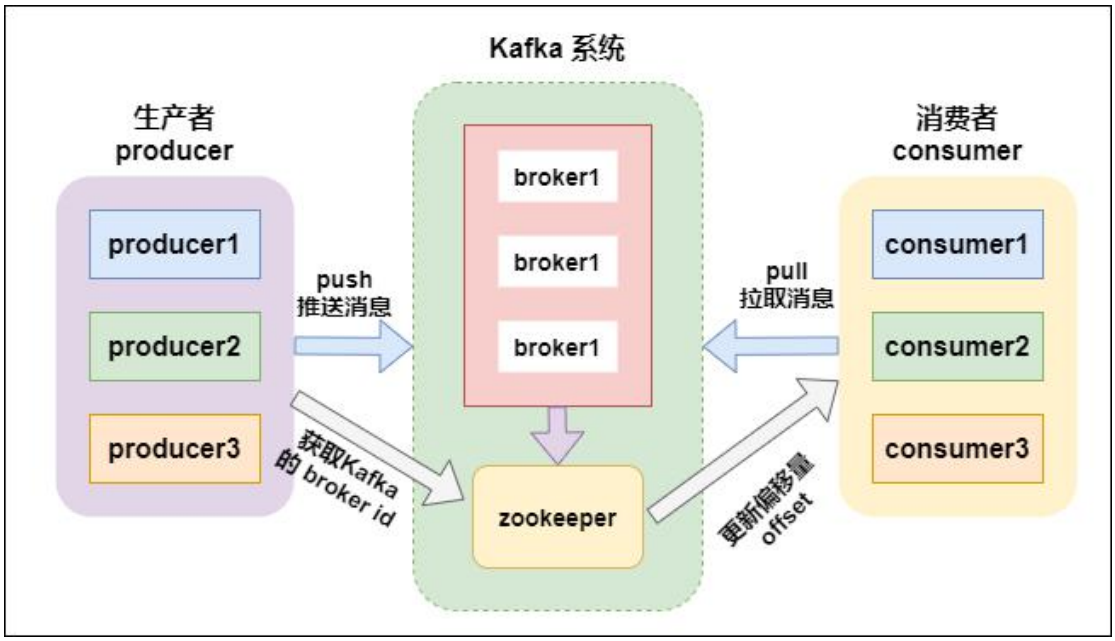
Kafka:

下游数据传输使用的是 Kafka 作为消息队列来传输消息，使用 Kafka 的主要原因是因为 Kafka 的高吞吐量以及可以对数据进行分类也就是不同的 topic，方便下一层的使用，**整体的架构是采用 Lamda 架构设计的**，实时和离线都会从 Kafka 中获取数据来进行处理，而且还有其他的业务线也是从 Kafka 中获取数据的，这样做以后可以有效的提高数据的复用减少数据的冗余，离线这块我们是在 Kafka 之后又做了一层 Flume 来作为消费者处理 Kafka 中的数据的，将消费到的数据直接放入 HDFS 中，实时这块使用的是 SparkStreaming 来消费 Kafka 中的数据;

为什么选择 Kafka 作为消息队列来处理数据

当时在做技术选型的时候我们也是做了大量的调研，因为消息队列的产品有很多比如 : ReactMQ Kafka 等;

我们当时调研考虑的主要指标就是吞吐量这一块，因为大数据流式处理对数据的吞吐量要求是非常高的，在这一块 ReactMQ 是比较厉害的，吞吐量可以达到 1 万多每秒;通过后来的调研以后发现 Kafka 的吞吐量比 ReactMQ 更高，如果使用恰当的话吞吐量甚至可以达到 10 万+每秒;



kafka 支持消息持久化，消费端是主动拉取数据，消费状态和订阅关系由客户端负责维护，**消息消费完后，不会立即删除，会保留历史消息**。因此支持多订阅时，消息只会存储一份就可以。

1. **broker**：kafka 集群中包含一个或者多个服务实例（节点），这种服务实例被称为 broker（一个 broker 就是一个节点/一个服务器）；
2. **topic**：每条发布到 kafka 集群的消息都属于某个类别，这个类别就叫做 topic；
3. **partition**：partition 是一个物理上的概念，每个 topic 包含一个或者多个 partition；
4. **producer**：消息的生产者，负责发布消息到 kafka 的 broker 中；
5. **consumer**：消息的消费者，向 kafka 的 broker 中读取消息的客户端；
6. **consumer group**：消费者组，每一个 consumer 属于一个特定的 consumer group（可以为每个 consumer 指定 groupName）；

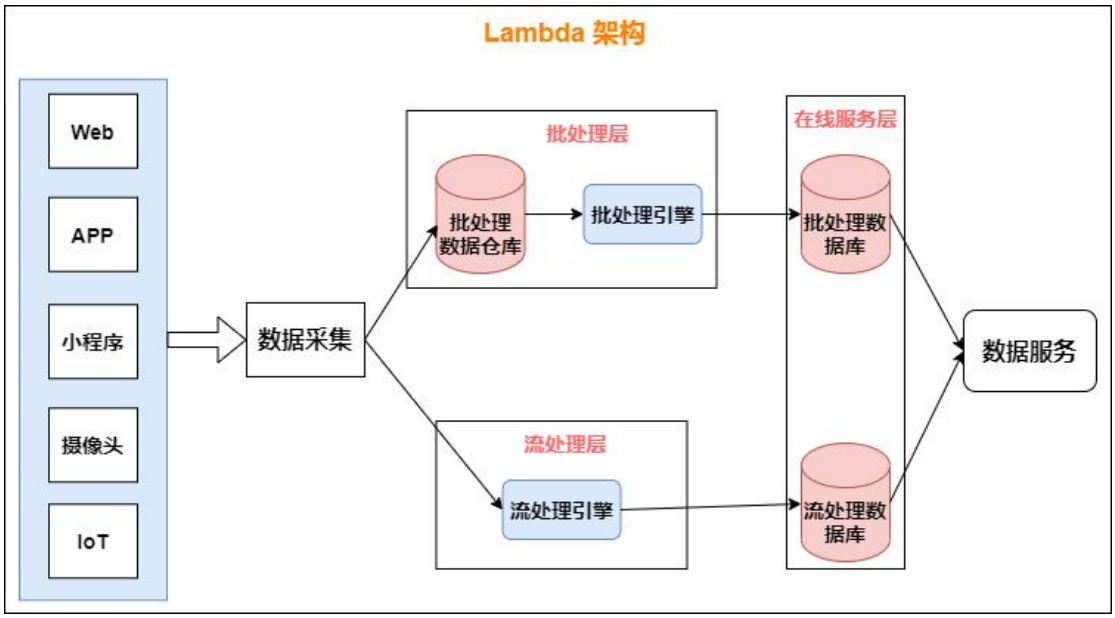
Kafka 为什么可以这么快?

1. 首先从生产者说起，生产者发送数据是按照批进行发送的并不是一条一条发送的，从这里就已经可以保证 Kafka 一个比较高的吞吐量了；
2. 生产者来一条消息以后会进入一个拦截器，在拦截器里边可以对数据进行一个整体的 修改操作一般这里是不做特殊的处理的，数据从拦截器出来以后就会进入到序列化器， 在序列化器里边将数据转换成一个二进制流的形式放入 Broker 里边；

3. 经过序列化器以后数据就会走到分区器，Kafka 使用的分区器是一个叫做 Hash 的分区器，Hasf 分区器我们可以对其进行重写；
4. 生产者将消息发送到 Broker 的过程可能会出现消息的重复或者丢失的情况，这个主要是靠 ACK 的配置来决定的，ack 的响应有三个状态值 0，1，-1
    - 0：生产者只负责发送数据，不关心数据是否丢失，丢失的数据，需要再次发送
    - 1：partition 的 leader 收到数据，不管 follow 是否同步完数据，响应的状态码为 1
    - -1：所有的从节点都收到数据，响应的状态码为-1
5. 这里还有一个很重要的概念就是 ISR 副本同步队列，在这个队列里边包含了 Leader 和 Follower，主要解决的问题就是 Leader 挂了以后谁来做 Leader 的问题：
    - 选举机制就是通过这个 ISR 来进行的，默认是有一个排序排序的一般都是选取第一 个，因为每一批的数据只有最快的那个才能达到第一个接收消息，其中 ISR 队列以及 Leader 的选举是由 Controller 来控制的，Zookeeper 来进行存储，关于 Controller 在 Kafka 中也是非常重要的 Controller 也有一个专门的选举机制，它是相当于是用 Zookeeper 来做了一个分布式锁，具体原理就是利用 Zookeeper 生成的临时节点生成一个分布式锁，谁先抢占到谁就是 Leader；

三、介绍项目采用的架构

上面也说到，我们整体的架构是采用 Lamda 架构设计的。



数据从底层的数据源开始，经过 Kafka、Flume 等数据组件进行收集，然后分成两条线进行计算：

- 一条线是进入流式计算平台（例如 Storm、Flink 或者 SparkStreaming），去计算实时的一些指标；
- 另一条线进入批量数据处理离线计算平台（例如 Mapreduce、Hive，Spark SQL），去计算 T+1 的相关业务指标，这些指标需要隔日才能看见。

在 Lambda 架构中，每层都有自己所肩负的任务。

1. 批处理层存储管理主数据集（不可变的数据集）和预先批处理计算好的视图：

批处理层使用可处理大量数据的分布式处理系统预先计算结果。它通过处理所有的已有历史数据来实现数据的准确性。这意味着它是基于完整的数据集来重新计算的，能够修复任何错误，然后更新现有的数据视图。输出通常存储在只读数据库中，更新则完全取代现有的预先计算好的视图。

2. 流处理层会实时处理新来的大数据：

流处理层通过提供最新数据的实时视图来最小化延迟。流处理层所生成的数据视图可能不如批处理层最终生成的视图那样准确或完整，但它们几乎在收到数据后立即可用。而当同样的数据在批处理层处理完成后，在速度层的数据就可以被替代掉了。

四、详细介绍数仓搭建

1. 数据各层作用

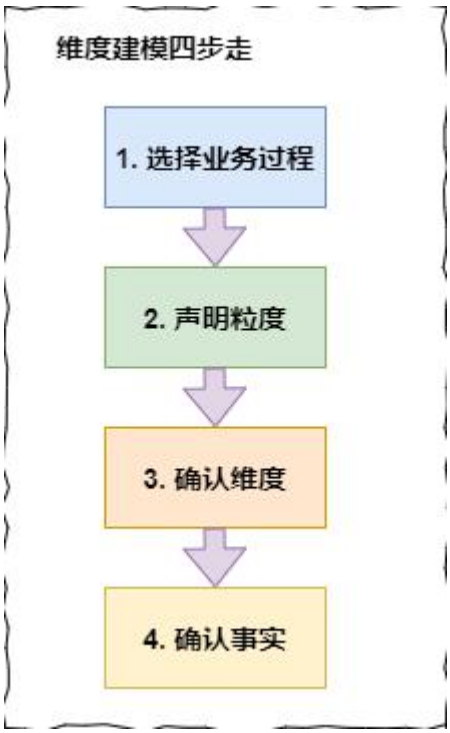
1. ODS（原始数据层）：日志数据和业务进入数仓后，首先放入该层，建立分区表，防止后续的全表扫描，使用 ORC 列式存储，同时对数据进行压缩，压缩格式采用 LZO，以减少存储空间。
    - 日志：商品列表、商品点击、商品详情;广告;故障;后台活跃、通知;启动表;点赞、评论、收藏等。
    - 业务数据：订单表、用户表、支付流水表、订单详情表、商品表、三级、二级、一级，物流信息(根据产品的来源，有两种，香港特快直送，闪电保税仓。一个从香港发货，一个从内地的保税仓发货)等。
2. DWD（明细数据层）：对 ODS 层数据清洗（去除空值，脏数据，超过极限范围的数据）。

    - 用户行为数据：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)，自定义方法的好处在于更加灵活以及方便调试 bug。在自定义函数解析字段时，我们一般建立中间表，存放解析后的表，最后通过 get_json_object 获的我们所需要的字段，建立最终所需表。
    - 业务数据：维度退化+数据清洗（where group by）
    - 脱敏：利用 spark 对手机号、身份证号、银行账号等敏感信息进行脱敏处理。
    - ETL：通过 HQL、Kettle 对数据进行清洗。清洗标准是核心字段满足业务逻辑要求，去除重复、空值、超过时限等数据。一般清洗率为万分之一，如果大于这个数，需要和前端、javaEE 人员进行沟通。
    - 维度退化：商品表+三级分类、二级分类、一级分类=>商品表，省份+地区表=>省份表，其中我们用到的维度建模理论是星型模型，事实表周围 1 级维度。
    - LZO 压缩：减少存储空间
    - 列式存储：ORC，增加压缩比
    - 分区表：防止后续的全表扫描

重点重点：DWD 层我们使用的是标准的数仓建模理论

数仓建模怎么建？

我们按照数仓工具箱中的维度建模四步走来建的：



- **选择业务过程**：由于我们公司当时数据量较小，我把 javaEE 涉及的业务表全部导入了，这些表包括**实体表，维度表，事务型快照事实表，周期性快照事实表、累积型事实表**。过来之后，将这些表作为矩阵的一个列。
- **声明粒度**：粒度一般有：一行信息代表一次、按天、按周、按月等，参考了很多架构之后，我们考虑到后期想要分析更多的指标，只能选择最小的粒度，一行信息代表一次消费。
- **确认维度**：采用标准数仓建模的思维，争取事实表周围都是 1 级维度。我们关系的就是什么时间、什么地点、什么人、具体什么活动、优惠券等主题的维度，同时将跟用户、商品相关的表进行维度退化，尽量把他们降成一级维度。
- **确认事实**：这里我们确定的不是事实表，而是**事实表的度量值**，我们用到的度量值有订单的个数、订单的金额、下单次数等可以累加的字段。

3. **DWS、DWT（每天的用户行为宽表）**：每天的用户行为宽表、商品宽表，相当于一个周期型快照事实表。每天记录用户做了那些事情，商品被下单了多少。
    - DWS 宽表的字段我们是站在维度的角度来取的，比如站在用户的维度去看待周围的对应事实表，取事实表对应的度量值，取出订单的次数、订单的金额、支付的次数、支付的金额、加入购物车的次数、

加入购物车的金额、评论的次数、点赞的次数、收藏的次数等等，将他们组合成为 DWS 层每天发生的事情。

- 后期我们为了统计的指标，加了一个 DWT 层，DWT 层还是站在维度的角度去看待对应事实表，但是它和 DWS 有略微的区别，现在关注的是这个用户什么时间开始创建的，最后一次登录是什么时候，累计登录多少次，最近 30 天登录多少次等信息。

4. DWS、DWT 统称为服务层：都是为后面的 ADS 层提供服务的，如果统计的是累积性指标，从 DWT 层拿取数据；如果统计的当天的指标，直接从 DWS 层取对应的数据。 DWS 层最大的行为宽表是用户行为宽表，其字段有互动日期、用户 id、用户昵称、注册日期、注册来源、细分渠道、注册省份、评论次数、打赏次数、添加收藏、取消收藏、关注商品、取消关注的商品、关注人、取消关注的人、点不值次数、点值次数、点赞次数、分享次数、爆料数、加购物车数、取消购物车次数等待。DWT 也是用户行为宽表，其字段有互动日期、用户 id、用户昵称、注册日期、注册来源、细分渠道、注册省份、最后一次登录日期、累计登录日期、最近 30 天登录日期等等。
5. ADS 层：分析了 100 多个指标：包括 日活、月活、周活、留存、留存率、新增（日、周、年）、转化率、流失、回流、七天内连续 3 天登录（点赞、收藏、评价、购买、加购、下单、活动）、连续 3 周（月）登录、GMV、复购率、复购率排行、点赞、评论、收藏、领优惠价人数、使用优惠价、沉默、值不值得买、退款人数、退款率 topN 热门商品、留转 G 复活等。

五、数仓业务详解

我们数据仓库是基于维度建模，主要使用星型模型。

1. 维度表：一般是对事实的描述信息。每一张维表对应现实世界中的一个对象或者概念。例如：用户、商品、日期、地区等。

维表的特征：

- 维表的范围很宽（具有多个属性、列比较多）
- 跟事实表相比，行数相对较小：通常< 10 万条
- 内容相对固定：编码表

2. **事实表**：分为事务型事实表（每个事务或事件为单位，一旦产生就固定）和周期型事实表（不会保留所有数据，只保留固定时间间隔的数据，比如每天、每月销售额）以及累积性事实表（累积型快照事实表用于跟踪业务事实的变化，比如订单的状态变化情况）。如果需要后面状态还会改变的就是周期型事实表，一旦确定了，就是事务性事实表。

事实表中的**每行数据代表一个业务事件（下单、支付、退款、评价等）**。“事实”这个术语表示的是业务事件的**度量值（可统计次数、个数、金额等）**，例如，订单事件中的下单金额。

每一个事实表的行包括：具有可加性的数值型的度量值、与维表相连接的外键、通常具有两个和两个以上的外键、外键之间表示维表之间多对多的关系。

事实表的特征：

- 非常的大
- 内容相对的窄：列数较少
- 经常发生变化，每天会新增加很多。

对于不同的表我们使用不同的同步策略：

同步策略包括全量表，增量表，新增及变化，拉链表

日志表：（商品点击，商品详情，商品详情页表，广告表，错误日志表，消息通知表等）

1. 商品点击：用户的基本信息字段，动作，商品 id，种类等。
2. 商品详情页：入口，上一页面来源，商品 id，加载时间，种类。
3. 广告表：入口，内容，行为，展示风格等。
4. 错误日志：错误详情
5. 消息通知表：通知类型，展示时间，通知内容等

这些记录性质的，都使用**每日增量**。

业务表：（购物车，评分，评论，订单表，订单详情表，退货表，用户表，商家表，商品分类表（一级，二级，三级），支付流水，物流信息等）

1. 购物车详情：用户 id ，商品 id，商品价格，商家 id ，商品型号，商品分类等 同步策略：这属于周期型事实表，因为它可能会随时改变，所以得用每日新增及变化。
2. 评分表：评分时间，评分用户，评分商品 ，分数等。

   同步策略：这是事务性事实表，一般可以用**每日增量**就可以了，因为评论只能增加，不能修改。
3. 评论表：评论时间，评论用户，评论商品，评论内容。

同步策略：这个跟评分差不多，用每日新增。

4. 订单表：订单状态，订单编号，订单金额，支付方式，支付流水，创建时间等

   同步策略：因为订单的状态会随时发生改变，比如下单，支付，商家发货，用户收到货，确认收货，等这一系列的状态会比较长，然后订单也比较多。所以，要做历史快照信息的话，最好使用拉链表。

5. 订单详情表：订单编号，订单号，用户 id，商品名称，商品价格，商品数量，创建时间等。

6. 用户表：用户 id，性别，等级，vip，注册时间等等。

   同步策略：因为表不是很大，每次做全量表。

7. 商家表：商家 id，商家地址，商家规模等级，商家注册时间，商家分类信息。

   同步策略：每次做每日全量。

总结：

1. 实体表，不大，就可以做每日全量。
2. 对于维度表，比如说商品分类，这种不是很大，也可以做每日全量，有一些不太会发生改变的维度，就可以固定保存一份值，比如说：地区，种族等。
3. 像事务型事实表，比如说交易流水，操作日志，出库信息，这种每日比较大，且需要历史数据的，就根据时间做每日新增，可以利用分区表，每日做分区存储。
4. 像周期型事实表的同步策略，比如订单表，有周期性变化，需要反应不同时间点的状态的，就需要做拉链表。记录每条信息的生命周期，一旦一条记录的生命周期结束，就开始下一条新的记录。并把当前的日期放生效开始日期。

六、离线指标

1. 日活/周活/月活统计：（每日的根据 key 聚合，求 key 的总数）
2. 用户新增：每日新增（每日活跃设备 left join 每日新增表，如果 join 后，每日新增表的设备 id 为空，就是新增）
3. 用户留存率：（一周留存）10 日新增设备明细 join 11 日活跃设备明细表，就是 10 日留存的。注意每日留存，一周留存
4. 沉默用户占比：只在当天启动过，且启动时间在一周前
5. 本周回流用户数
6. 用户在线时长统计
7. 区域用户订单数（根据区域分区，然后求订单数）
8. 区域订单总额（根据区域分区，求订单总额。）
9. 区域用户订单访问转化率（以区域分组成单数/访问数）
10. 区域客单价（订单总额度/下订单总人数）

11. 总退货率（退货商品数/购买商品总数）
12. 各区域退货率（根据区域分组）
13. GMV（成交总额）
14. 物流平均时长（用户收货时间-物流发货时间）求平均
15. 每周销量前十品类
16. 每周各品类热门商品销量前三
17. 各区域热门商品销量前五（有利于后期铺货）
18. 各区域漏斗分析
19. 商品评价人数占比（该商品的总评价人数/该商品的总购买人数）
20. 各品牌商家总销售额。
21. 各品类中销量前三的品牌
22. 购物车各品类占比（说明大家想买的东西，便于后期铺货。）
23. 每周广告点击率。看到这个广告的人数/点击这个广告商品的人数）
24. vip 用户每日，周订单总额
25. 每日限时特卖产品占比（限时特卖产品总额/每日交易总额）
26. 香港特快直送渠道总交易额占比（香港特快直送渠道总额/每日商品交易总额）
27. 香港特快直送渠道总交易单占比
28. 国内保税仓渠道总交易额占比（国内保税仓总额/每日商品交易总额）
29. 国内保税仓渠道总交易单占比
30. 各区域页面平均加载时长（考察各地区网络问题。后台访问是否稳定）
31. 页面单跳转化率统计
32. 获取点击下单和支付排名前 10 的品类
33. 各类产品季度复购率

七、实时指标

1. 每日日活实时统计
2. 每日订单量实时统计
3. 一小时内日活实时统计
4. 一小时内订单数实时统计
5. 一小时内交易额实时统计
6. 一小时内广告点击实时统计
7. 一小时内区域订单数统计
8. 一小时内区域订单额统计
9. 一小时内各品类销售 top3 商品统计
10. 用户购买明细灵活分析（根据区域，性别，品类等）

八、写出分析最难的两个指标

面试官说现场首先两个你分析过最难的两个指标：

最好不要选择最难的，除非你能完全写出来，并且还得让面试官理解你做的指标的含义，下面选择容易理解但不算简单的指标：

1. 活跃用户指标

我们经常会算活跃用户，活跃用户是指至少连续 5 天登录账户的用户，返回的结果表按照 id 排序。

```
+----+-----------+
| 7  | Jonathan  |
+----+-----------+
```

思路：

1. 去重：由于每个人可能一天可能不止登陆一次，需要去重
2. 排序：对每个 ID 的登录日期排序
3. 差值：计算登录日期与排序之间的差值，找到连续登陆的记录
4. 连续登录天数计算：select id， count(*) group by id， 差值（伪代码）
5. 取出登录 5 天以上的记录
6. 通过表合并，取出 id 对应用户名

参考代码：

```
SELECT DISTINCT b.id,  name
FROM
  (SELECT id,  login_date,
    DATE_SUB(login_date, ROW_NUMBER() OVER(PARTITION BY id ORDER BY login_date)) AS
diff
   FROM(SELECT DISTINCT id,  login_date FROM Logins) a) b
INNER JOIN Accounts ac
ON b.id = ac.id
GROUP BY b.id,  diff
HAVING COUNT(b.id) >= 5
```

注意点：

1. DATE_SUB 的应用：DATE_SUB (DATE， X)，注意，X 为正数表示当前日期的前 X 天；
2. 如何找连续日期：通过排序与登录日期之间的差值，因为排序连续，因此若登录日期连续，则差值一致；
3. GROUP BY 和 HAVING 的应用：通过 id 和差值的 GROUP BY，用 COUNT 找到连续天数大于 5 天的 id，注意 COUNT 不是一定要出现在 SELECT 后，可以直接用在 HAVING 中

2. 用户留存率

首先用户留存率一般是面向新增用户的概念，是指某一天注册后的几天还是否活跃，是以每天为单位进行计算的。
一般收到的需求都是一个时间段内的新增用户的几天留存

```
select '日期', '注册用户数', '次日留存率', '2日留存率', '3日留存率',  dim_date
      , total_cnt
      ,
concat_ws('% | ', cast(round(dif_1cnt*100/total_cnt, 2) as string), cast(dif_1cnt as
 string))
      ,
concat_ws('% | ', cast(round(dif_2cnt*100/total_cnt, 2) as string), cast(dif_2cnt as
 string))
      ,
concat_ws('% | ', cast(round(dif_3cnt*100/total_cnt, 2) as string), cast(dif_3cnt as
 string))
      ,
concat_ws('% | ', cast(round(dif_4cnt*100/total_cnt, 2) as string), cast(dif_4cnt as
 string))
     from
     (
      select p1.state dim_date
             , p1.device_os
             , count(distinct p1.user_id) total_cnt
             , count(distinct if(datediff(p3.state,
p1.state) = 1,  p1.user_id,  null)) dif_1cnt
             , count(distinct if(datediff(p3.state,
p1.state) = 2,  p1.user_id,  null)) dif_2cnt
             , count(distinct if(datediff(p3.state,
p1.state) = 3,  p1.user_id,  null)) dif_3cnt
             , count(distinct if(datediff(p3.state,
p1.state) = 4,  p1.user_id,  null)) dif_4cnt
          from
          (
           select
              from_unixtime(unix_timestamp(cast(partition_date as string), 'yyyyMMd
d'),  'yyyy-MM-dd') state,
              user_id
              from user_active_day
              where partition_date between date1 and date2
                    and user_is_new = 1
```

```
                group by 1, 2
            )p1 --日新增用户名单(register_date, user_id)
            left outer join
            (
                select
                    from_unixtime(unix_timestamp(cast(partition_date as str
ing), 'yyyyMMdd'), 'yyyy-MM-dd') state,
                    user_id
                from active_users
                where partition_date between date1 and date2
                group by 1, 2
            )p3 --期间活跃用户(active_date, user_id)
            on (p3.user_id = p1.user_id)
        group by 1, 2
    ) p4;
```

九、面试官问

自己说完项目之后面试官就开始发问了，注意接招：

1. 如何保证你写的 sql 正确性?

我一般是造一些特定的测试数据进行测试。
另外离线数据和实时数据分析的结果比较。

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. 测试环境什么样?

测试环境的配置是生产的一半

4. 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。

5. 你做的项目工作流程是什么？

1. 先与产品讨论，看报表的各个数据从哪些埋点中取
2. 将业务逻辑过程设计好，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

6. 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。

7. 说下 Spark 数据倾斜及解决？

数据倾斜以为着某一个或者某几个 partition 的数据特别大，导致这几个 partition 上的计算需要耗费相当长的时间。
在 spark 中同一个应用程序划分成多个 stage，这些 stage 之间是串行执行的，而 一个 stage 里面的多个 task 是可以并行执行，task 数目由 partition 数目决定，如 果一个 partition 的数目特别大，那么导致这个 task 执行时间很长，导致接下来 的 stage 无法执行，从而导致整个 job 执行变慢。
避免数据倾斜，一般是要选用合适的 key，或者自己定义相关的 partitioner，通 过加盐或者哈希值来拆分这些 key，从而将这些数据分散到不同的 partition 去执行。
如下算子会导致 shuffle 操作，是导致数据倾斜可能发生的关键点所在：
groupByKey；reduceByKey；aggregaByKey； join；cogroup

8. 为什么 Kafka 不支持读写分离？

在 Kafka 中，生产者写入消息、消费者读取消息的操作都是与 leader 副本进行交互的，从 而实现的是一种主写主读的生产消费模型。 Kafka 并不支持主写从读，因为主写从读有 2 个很明显的缺点:

1. 数据一致性问题:数据从主节点转到从节点必然会有一个延时的时间窗口，这个时间 窗口会导致主从节点之间的数据不一致。某一时刻，在主节点和从节点中 A 数据的值都为 X， 之后将主节点中 A 的值修改为 Y，那么在这个变更通知到从节点之前，应用读取从节点中的 A 数据的值并不为最新的 Y，由此便产生了数据不一致的问题。
2. 延时问题：类似 Redis 这种组件，数据从写入主节点到同步至从节点中的过程需要经历 网络 → 主节点内存 → 网络 → 从节点内存 这几个阶段，整个过程会耗费一定的时间。而在 Kafka 中，主从同步会比 Redis 更加耗时，它需要经历 网络 → 主节点内存 → 主节点磁盘 → 网络 → 从节 点内存 → 从节点磁盘 这几个阶段。对延时敏感的应用而言，主写从读的功能并不太适用。

而 kafka 的主写主读的优点就很多了：

1. 可以简化代码的实现逻辑，减少出错的可能;
2. 将负载粒度细化均摊，与主写从读相比，不仅负载效能更好，而且对用户可控;
3. 没有延时的影响;
4. 在副本稳定的情况下，不会出现数据不一致的情况。

十、最后的面试小技巧

最后给大家说一点面试小技巧：

一般来说，面试你的人都不是一个很好对付的人。别看他彬彬有礼，看上去笑眯眯的，很和气的样子。但没准儿一肚子坏水。

有些人待人特别客气，说话还稍稍有点结巴的，更容易让人上当。

所以，牢记一点，面试的时候保持高度警觉，对方不经意问出来的问题，很可能是他最想知道的。

1. 首先说话语速不要太快，有些人介绍自己时滔滔不绝，说话特快。其实这里面有个信息传递的问题，跟别人谈事情，语速太快，往往容易说错，对方接受起来也有问题。所以中等语速就可以了。

2. 问到期望薪金的时候，最好的回答是不回答，留到下一次面试再谈。或者可以反问，公司对于这个岗位定的薪金标准是多少。
3. 不要紧张，表现得自然些，要有礼貌，别忘记和主考人招呼，说句"早上好"等。
4. 举止要大方，不可闪缩，要保持自信。待主考人邀请你才可礼貌地坐下，不要太随便或左顾右盼；切忌装出懒洋洋和满不在乎的样子。
5. 微笑可以减轻你内心的不安，更可以令面试的气氛变得融洽愉快。
6. 让主考人知道你珍惜这次面试的机会。当主考人说话时，要眼望对方，并留心倾听。
7. 让主考人先打开话匣子。答问题要直接了当，无须太繁复，也不要单说"是"或"不是";否则，主考人会觉得你欠缺诚意。深入的谈话内容有助主考人对你作出确切的评估。
8. 假如有不太明白主考人的问题，应该礼貌地请他重复。不懂得回答的问题，不妨坦白承认。含糊其辞或乱吹牛会导致面试的失败。
9. 不要打断主考人的说话，被要求就相同的问题重复作答也不能表示不耐烦，更切忌与主考人争辩。
10. 主考人可能问你一些与面试或者申请的职位完全无关的问题，例如时人时事，目的在进一步了解你的思想及见识。
11. 紧记在适当时机带出自己的优点和特长。但切勿显得过份自信或浮夸。
12. 准备一些与该机构和申请的工作有关的问题在面试结束之前提出。这样能表现伙的积极，亦可给主考人留下良好印象。
13. 最后，问清楚多久才知道面试结果。不要忘记向主考人道谢及说声"再见"才离去。
14. 第一时间获取最新大数据技术，尽在公众号：五分钟学大数据
15. 搜索公众号：五分钟学大数据，学更多大数据技术！
16. 其他大数据技术文档可下方扫码关注获取：