离线数仓项目总结

如何向别人说明你的项目

首先说明项目背景：做的是一个电商项目，后台的数据，使用的是电商系统的**业务数据和日志数据。**

你对数据仓库的理解

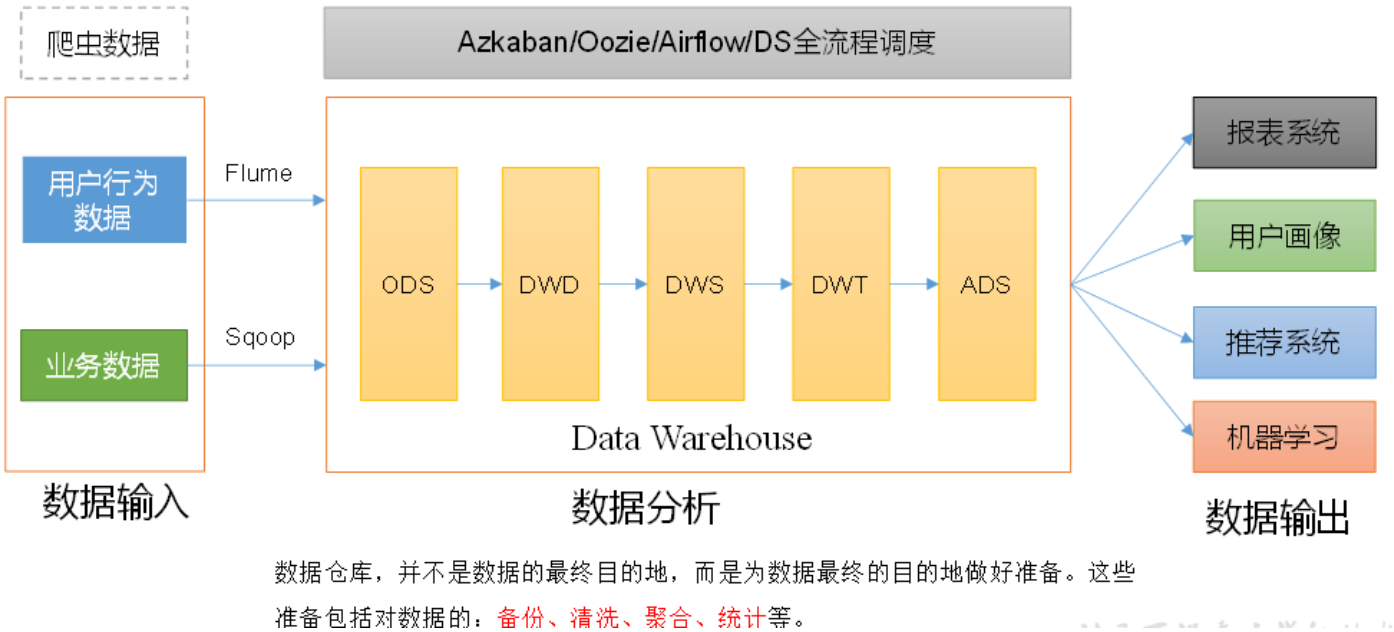
数据仓库是为企业提供决策支持的数据集合，数据仓库不像我们平时所使用的mysql业务数据库，业务数据库中的数据不是永久存储的，有固定的寿命，但是我们数据仓库中的数据是保存很长时间的，至少

保存半年之一年的历史数据，所以有了历史数据，我们就可以多一条分析数据的维度，**时间**，我们可以通过分析随着时间的推移，用户的行为，用户的喜好，等等很多信息，为我们的推广，决策提供支持。

并且通过这些数据的分析，我们可以分析用户画像，报表信息，或者进行机器学习等模型的训练。

数据仓库为什么要分层

如果要我一句话说明的话，我会说：复杂的问题简单化，如何理解呢？



数据仓库，并不是数据的最终目的地，而是为数据最终的目的地做好准备。这些准备包括对数据的：备份、清洗、聚合、统计等。

清晰的数据结构：

什么是清晰的数据结构，简单来说就是每一层的数据都有他自己的作用域，这样我们再使用数据的时候，可以更加方便的去定位。

数据血缘关系的追踪：

如果那我们的业务数据来举例，可能我们一张订单详情表中是由很多其他的维度表或者是事实表join得到的，如果我们不进行分层，那么随着数据量增长，我们很难分析表和表之间的关系，所以为了建立表之间的血缘关系，使用分层。

减少重复开发：

规范的数据分层，不但可以重用我们的中间数据，还可以减少计算，重用计算。

屏蔽原始数据的异常

这个很容易想到，采集到的数据不一定拿来就可以使用，再分层的过程中，我们会主键清洗掉异常的数据，有利于保护我们的数据安全性。

如何考虑系统架构的设计或者技术选型

对于离线数仓

通常采集日志数据使用flume.

采集业务数据使用sqoop。

使用消息队列kafka作为缓冲的组件。

数据的存储通常用分布式文件系统hdfs，可以存储大容量的数据。

计算引擎，通常使用hadoop或者spark。

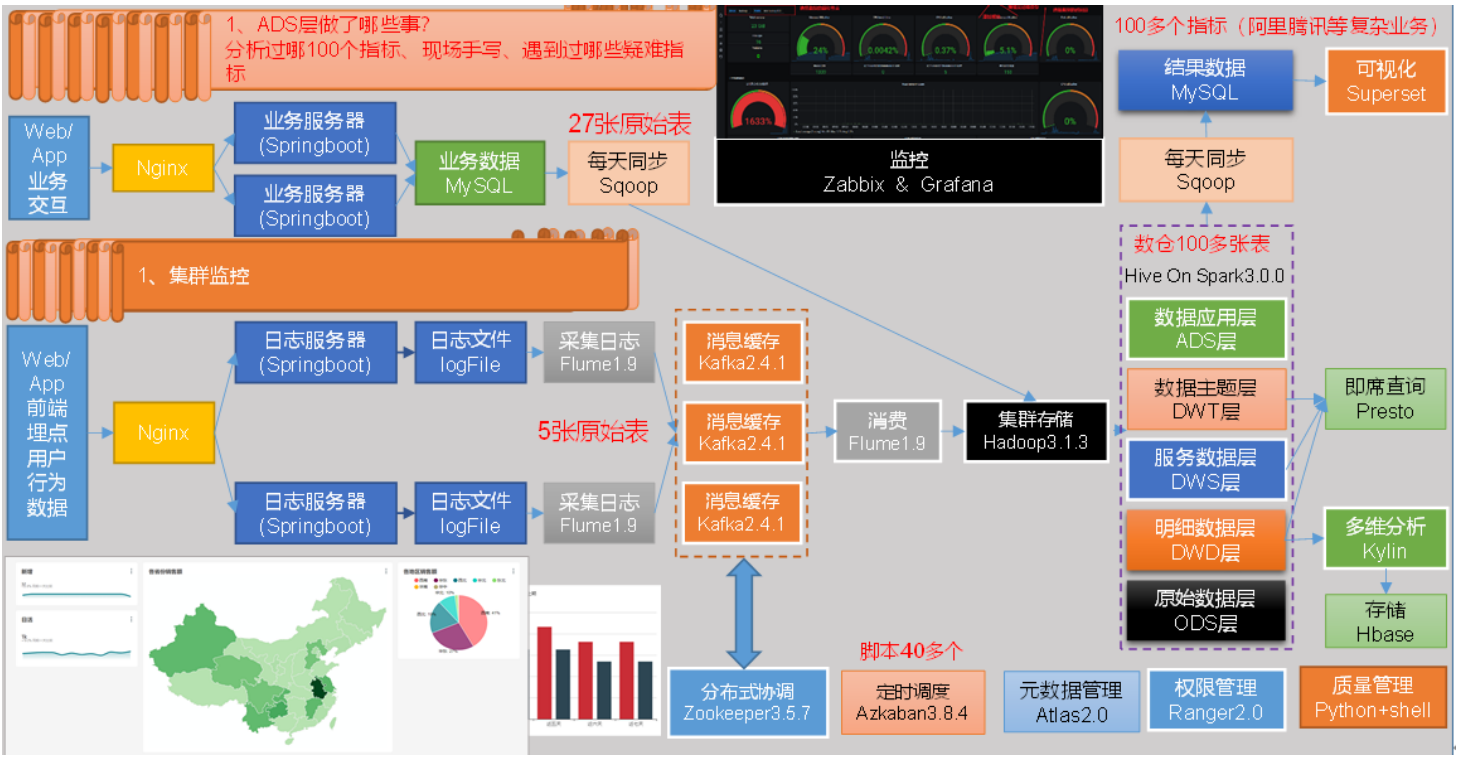
实时数仓

再实际中，企业通常使用一套采集系统来采集日志数据，中间通过kafaka组件对应离线系统和实时系统。

采集业务数据通常使用Flink cdc组件

计算组件使用Flink或者spark streamming

架构设计



Hadoop项目经验

hdfs多目录存储

在hdfs-site.xml文件中配置多目录，注意新挂载磁盘的访问权限问题。

HDFS的DataNode节点保存数据的路径由dfs.datanode.data.dir参数决定，其默认值为 `file://${hadoop.tmp.dir}/dfs/data` ，这个路径可以决定namenode和datanode存储数据的位置，若服务器有多个磁盘，必须对该参数进行修改。参数应修改为如下的值。

```
<property>
    <name>dfs.datanode.data.dir</name>
  <value>
    file:///dfs/data1,file:///hd2/dfs/data2,file:///hd3/dfs/data3,file:///hd4/dfs/data4
  </value>
</property>
file://代表是一种协议
```

注意：每台服务器挂载的磁盘不一样，所以每个节点的多目录配置可以不一致。单独配置即可。

使用多目录可以将数据的写操作分散到多个磁盘上，提高吞吐量。

集群数据均衡

节点间数据均衡

节点之间数据不均衡指的是，有的节点空间占用率达到80%，而有的节点空间占用率才30%，所以需要均衡节点之间的数据。

开启数据均衡命令：

```
start-balancer.sh -threshold 10
```

开启这一个进程，那么默认就会对数据进行跨界点数据的转移，直到数据均衡为止。

对于参数10，代表的是集群中各个节点的磁盘空间利用率相差不超过10%，可根据实际情况进行调整。

停止数据均衡命令： `stop-balancer.sh`

停止数据均衡后，已经均衡的数据不会恢复。

磁盘间数据均衡

这个是hadoop3.x之后的新特性，在这之前是没有的。

比如新加一块磁盘，那么就需要这个命令，让各个磁盘的数据均衡。

（1）生成均衡计划（我们只有一块磁盘，不会生成计划）

hdfs diskbalancer -plan hadoop103(新加磁盘的节点，这个计划其实就是一个json文件)

（2）执行均衡计划

hdfs diskbalancer -execute hadoop103.plan.json(生成的执行计划)

（3）查看当前均衡任务的执行情况

hdfs diskbalancer -query hadoop103

（4）取消均衡任务

hdfs diskbalancer -cancel hadoop103.plan.json

如果是取消执行计划，那么已经均衡的数据，是不会再恢复的。

Hadoop支持LZO压缩配置

1. hadoop本身并不支持lzo压缩，故需要使用twitter提供的hadoop-lzo开源组件。hadoop-lzo需依赖hadoop和lzo进行编译。
2. 将编译好后的hadoop-lzo-0.4.20.jar 放入hadoop-3.1.3/share/hadoop/common/
3. 同步hadoop-lzo-0.4.20.jar到集群中的所有节点。
4. core-site.xml增加配置支持LZO压缩

```
<configuration>
    <property>
        <name>io.compression.codecs</name>
        <value>
            org.apache.hadoop.io.compress.GzipCodec,
            org.apache.hadoop.io.compress.DefaultCodec,
            org.apache.hadoop.io.compress.BZip2Codec,
            org.apache.hadoop.io.compress.SnappyCodec,
            com.hadoop.compression.lzo.LzoCodec,
            com.hadoop.compression.lzo.LzopCodec
        </value>
    </property>

    <property>
        <name>io.compression.codec.lzo.class</name>
        <value>com.hadoop.compression.lzo.LzoCodec</value>
    </property>
</configuration>
```

同步配置文件到集群中其他的节点。

lzo压缩的长处是可以**支持切片**，但是是有条件的，需要建立索引。

建立索引

```
hadoop jar /path/to/your/hadoop-lzo.jar com.hadoop.compression.lzo.DistributedLzoIndexer big_fil
```

`com.hadoop.compression.lzo.DistributedLzoIndexer` 这个类是lzo中专门用来建立索引的类。

Hadoop基准测试

1） 测试HDFS写性能

测试内容：向HDFS集群写10个128M的文件

```
hadoop jar /opt/module/hadoop-3.1.3/share/hadoop/mapreduce/hadoop-mapreduce-client-jobclient-3.1

2020-04-16 13:41:24,724 INFO fs.TestDFSIO: ----- TestDFSIO ----- : write
2020-04-16 13:41:24,724 INFO fs.TestDFSIO:             Date & time: Thu Apr 16 13:41:24 CST 2020
2020-04-16 13:41:24,724 INFO fs.TestDFSIO:         Number of files: 10
2020-04-16 13:41:24,725 INFO fs.TestDFSIO:  Total MBytes processed: 1280
2020-04-16 13:41:24,725 INFO fs.TestDFSIO:       Throughput mb/sec: 8.88
2020-04-16 13:41:24,725 INFO fs.TestDFSIO:  Average IO rate mb/sec: 8.96
2020-04-16 13:41:24,725 INFO fs.TestDFSIO:   IO rate std deviation: 0.87
2020-04-16 13:41:24,725 INFO fs.TestDFSIO:      Test exec time sec: 67.61
```

2）测试HDFS读性能

测试内容：读取HDFS集群10个128M的文件

```
hadoop jar /opt/module/hadoop-3.1.3/share/hadoop/mapreduce/hadoop-mapreduce-client-jobclient-3.1

2020-04-16 13:43:38,857 INFO fs.TestDFSIO: ----- TestDFSIO ----- : read
2020-04-16 13:43:38,858 INFO fs.TestDFSIO:   Date & time: Thu Apr 16 13:43:38 CST 2020
2020-04-16 13:43:38,859 INFO fs.TestDFSIO:         Number of files: 10
2020-04-16 13:43:38,859 INFO fs.TestDFSIO:  Total MBytes processed: 1280
2020-04-16 13:43:38,859 INFO fs.TestDFSIO:       Throughput mb/sec: 85.54
2020-04-16 13:43:38,860 INFO fs.TestDFSIO:  Average IO rate mb/sec: 100.21
2020-04-16 13:43:38,860 INFO fs.TestDFSIO:   IO rate std deviation: 44.37
2020-04-16 13:43:38,860 INFO fs.TestDFSIO:      Test exec time sec: 53.61
```

3）删除测试生成数据

```
hadoop jar /opt/module/hadoop-3.1.3/share/hadoop/mapreduce/hadoop-mapreduce-client-jobclient-3.1
```

4）使用Sort程序评测MapReduce

（1）使用RandomWriter来产生随机数，每个节点运行10个Map任务，每个Map产生大约1G大小的二进制随机数

```
hadoop jar /opt/module/hadoop-3.1.3/share/hadoop/mapreduce/hadoop-mapreduce-examples-3.1.3.jar r
```

（2）执行Sort程序

```
hadoop jar /opt/module/hadoop-3.1.3/share/hadoop/mapreduce/hadoop-mapreduce-examples-3.1.3.jar s
```

（3）验证数据是否真正排好序了

```
hadoop jar /opt/module/hadoop-3.1.3/share/hadoop/mapreduce/hadoop-mapreduce-client-jobclient-3.1
```

> hadoop的基准测试：读，写，删除，性能

Hadoop参数调优

HDFS参数调优hdfs-site.xml

```
The number of Namenode RPC server threads that listen to requests from clients. If dfs.namenode.servicerpc-addres
```

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。当Datanode启动之后，会一直和namenode保持心跳，并且发送元数据信息。

对于大集群或者有大量客户端的集群来说，通常需要增大参数dfs.namenode.handler.count的默认值10。

```
<property>
    <name>dfs.namenode.handler.count</name>
    <value>10</value>
</property>
```

dfs.namenode.handler.count=20×log_e^(Cluster Size)，比如集群规模为8台时，此参数设置为41。

YARN参数调优yarn-site.xml

（1）情景描述：总共7台机器，每天几亿条数据，数据源->Flume->Kafka->HDFS->Hive

面临问题：数据统计主要用HiveSQL，没有数据倾斜，小文件已经做了合并处理，开启的JVM重用，而且IO没有阻塞，内存用了不到50%。但是还是跑的非常慢，而且数据量洪峰过来时，整个集群都会宕掉。基于这种情况有没有优化方案。

（2）解决办法：

内存利用率不够。这个一般是Yarn的2个配置造成的，**单个任务可以申请的最大内存大小，和Hadoop单个节点可用内存大小**。调节这两个参数能提高系统内存的利用率。

（a）yarn.nodemanager.resource.memory-mby

表示该节点上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。

（b）yarn.scheduler.maximum-allocation-mb

单个任务可申请的最多物理内存量，默认是8192（MB）。

Kafka

Kafka常用命令

查看Kafka Topic列表

```
bin/kafka-topics.sh --zookeeper hadoop102:2181/kafka --list
```

创建Kafka Topic

进入到/opt/module/kafka/目录下创建日志主题

```
bin/kafka-topics.sh --zookeeper hadoop102:2181,hadoop103:2181,hadoop104:2181/kafka  --create --r
```

删除Kafka Topic

```
bin/kafka-topics.sh --delete --zookeeper hadoop102:2181,hadoop103:2181,hadoop104:2181/kafka --tc
```

Kafka生产消息

```
bin/kafka-console-producer.sh \
--broker-list hadoop102:9092 --topic topic_log
>hello world
>rzf  rzf
```

Kafka消费消息

```
bin/kafka-console-consumer.sh \
--bootstrap-server hadoop102:9092
--from-beginning
--topic topic_log
--from-beginning:
```

会把主题中以往所有的数据都读取出来。根据业务场景选择是否增加该配置。

查看Kafka Topic详情

```
bin/kafka-topics.sh --zookeeper hadoop102:2181/kafka \
--describe --topic topic_log
```

kafka压力测试

用Kafka官方自带的脚本，对Kafka进行压测。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。

```
kafka-consumer-perf-test.sh
kafka-producer-perf-test.sh
```

Kafka Producer压力测试

1. 在/opt/module/kafka/bin目录下面有这两个文件。我们来测试一下

```
bin/kafka-producer-perf-test.sh
--topic test
--record-size 100
--num-records 100000
--throughput -1
--producer-props bootstrap.servers=hadoop102:9092,hadoop103:9092,hadoop104:9092
```

- record-size是一条信息有多大，单位是字节。
- num-records是总共发送多少条信息。
- throughput 是每秒多少条信息，设成-1，表示不限流，可测出生产者最大吞吐量。

> throughput设置为具体的值，测量的因为消息总数确定，所以测出的是延迟，但是如果值为-1，那么测出的是最高的速率。

kafka会打印下面消息

```
100000 records sent, 95877.277085 records/sec (9.14 MB/sec), 187.68 ms avg latency, 424.00 ms ma
```

参数解析：本例中一共写入10w条消息，吞吐量为9.14 MB/sec，每次写入的平均延迟为187.68毫秒，最大的延迟为424.00毫秒。

Kafka Consumer压力测试

Consumer的测试，如果这四个指标（IO，CPU，内存，网络）都不能改变，考虑增加分区数来提升性能。

```
bin/kafka-consumer-perf-test.sh
--broker-list hadoop102:9092,hadoop103:9092,hadoop104:9092
--topic test
--fetch-size 10000
--messages 10000000
--threads 1
```

参数说明：

--zookeeper 指定zookeeper的链接信息

--topic 指定topic的名称

--fetch-size 指定每次fetch的数据的大小

--messages 总共要消费的消息个数

测试结果说明：

```
start.time, end.time, data.consumed.in.MB, MB.sec, data.consumed.in.nMsg, nMsg.sec
2019-02-19 20:29:07:566, 2019-02-19 20:29:12:170, 9.5368, 2.0714, 100010, 21722.4153
```

开始测试时间，测试结束数据，共消费数据9.5368MB，吞吐量2.0714MB/s，共消费100010条，平均每秒消费21722.4153条。

Kafka机器数量计算

Kafka机器数量（经验公式）=2 * （峰值生产速度 * 副本数/100）+1

先拿到峰值生产速度，再根据设定的副本数，就能预估出需要部署Kafka的数量。

比如我们的峰值生产速度是50M/s。副本数为2。

Kafka机器数量=2*（50*2/100）+ 1=3台

> 峰值速度一般需要公司提供。

Kafka分区数计算

topic一般不需要计算，一个topic一般是一类数据。

1. 创建一个只有1个分区的topic
2. 测试这个topic的producer吞吐量和consumer吞吐量。
3. 假设他们的值分别是Tp和Tc，单位可以是MB/s。
4. 然后假设总的目标吞吐量是Tt，那么分区数=Tt / min（Tp，Tc）

例如：producer吞吐量=20m/s；consumer吞吐量=50m/s，期望吞吐量100m/s；

分区数=100 / 20 =5分区

```
https://blog.csdn.net/weixin_42641909/article/details/89294698
```

> 分区数一般设置为：3-10个

Flume

方案对比

本项目中，日志采集结构设计是：

flume--->kafka--->flume

第一种方案

第一个flume:

- source:Taildir Source采集日志文件
- channel:kafka channel,将数据直接写入kafka集群中，省掉了sink组件。

第二个flume:

- source:kafka source消费kafka中的数据
- channel:memory channel。
- sink:hdfs sink写入hdfs中。

第二种方案：

采用一个flume:

- source:采用Taildir Source采集日志文件。
- channle:采用的是kafka channel,将数据写入kafka集群当中。

- sink:采用hdfs sink方式，将数据写入hdfs中。

> 向kafka channel中写入数据其实是向kafka 集群中写。然后flume的sink在将数据从kafka集群中将数据读出来，写入hdfs中，这个过程需要经过网络的传输。

第三种方案

和第一种方案差不多，只不过再第一个flume使用kafka channel将数据写入kafka集群之后，第二个flume直接使用kafka channel消费kafka中的数据，然后将数据写入外部系统即可。此时的kafka channel相当于一个consumer。

> flume source-->kafka 集群---> flume sink---> hdfs

三种方案对比分析：

第一种方案简洁，向kafka集群中写入一次数据，然后再读取一次，很方便。

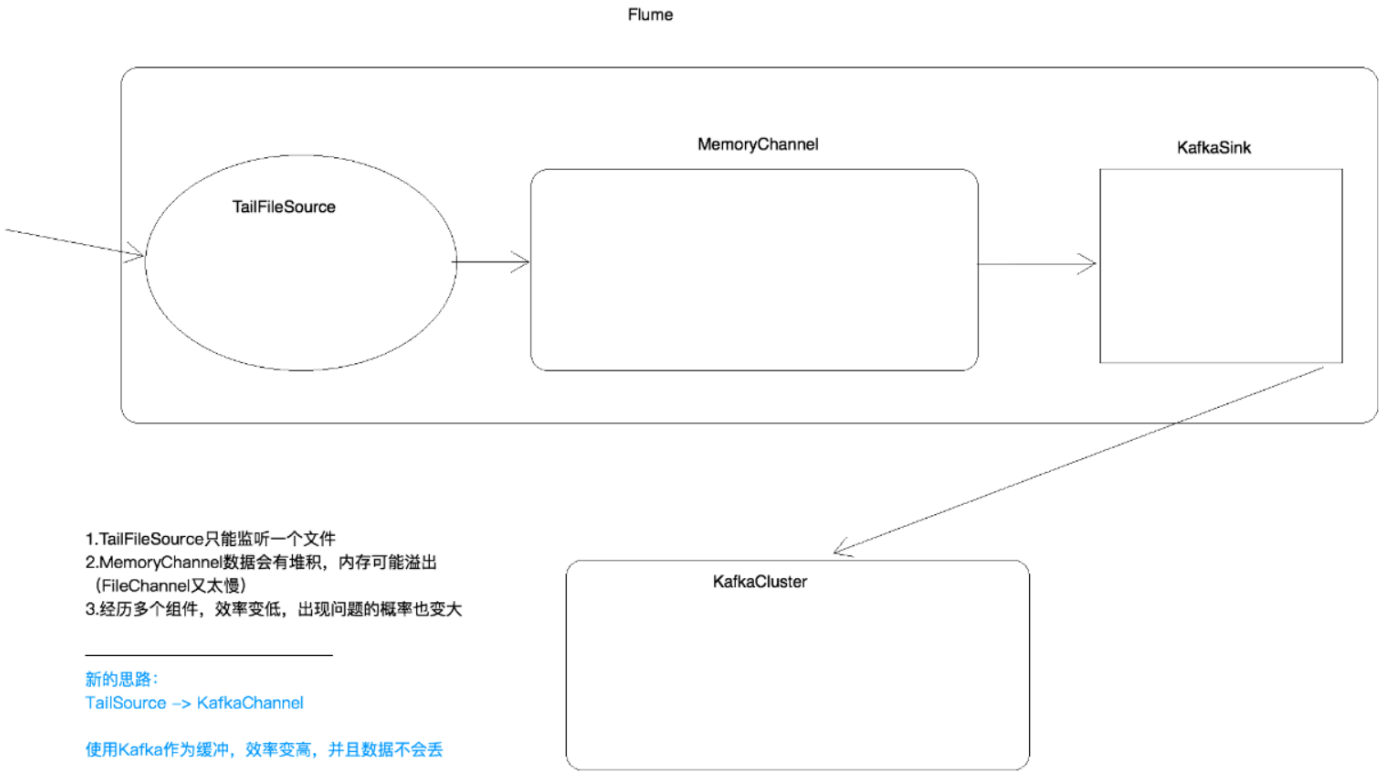
第二种设计结构很不好，压力比较大，首先我们的一个flume需要部署到日志服务器上面。日志服务器需要接收客户端埋点发送过来的日志数据，也就是所有数据先要写入日志服务器上，然后flume还需要读取日志向kafka集群写入一次，这个过程需要经过网络的传输，然后flume sink还需要去kafka集群读取一次数据，也就是数据还要从kafka集群写回flume一次，还要经过网络传输，最后再写入hdfs中，数据是两斤两处，对于日志服务器节点，压力比较大。

而第一种方案，把flume和kafka集群分散开，对日志服务器压力很小。

第三种方案和第一种方案都可以。

为什么需要kafka channel?

在使用flume对接Kafka时，我们往往使用TailFileSource–>MemoryChannel–>KafkaSink的这种方式，然后将数据输送到Kafka集群中。

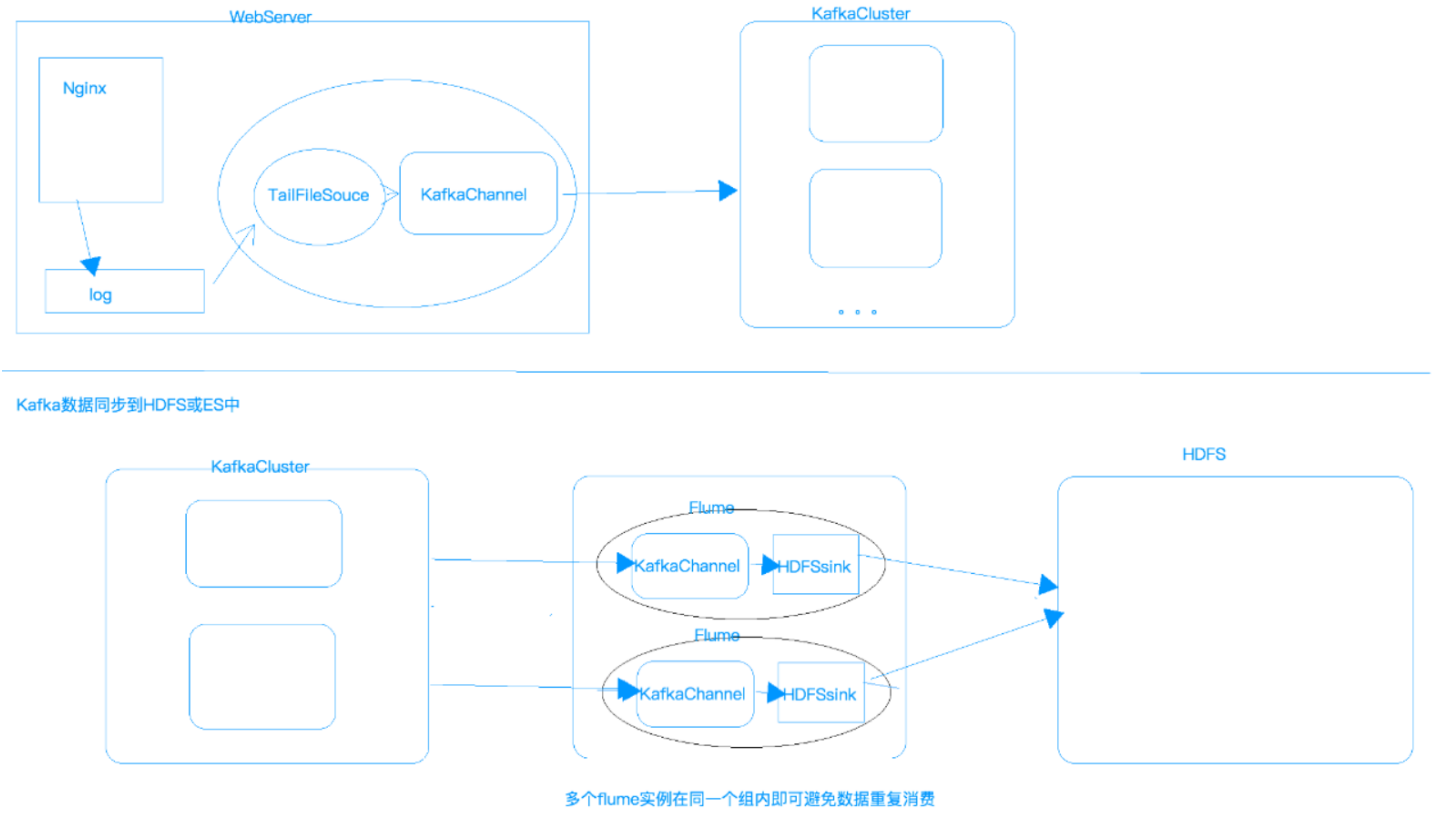


但是这种方式有弊端：

1. TailFileSource只能监听一个文件
2. MemoryChannel数据会有堆积，内存可能溢出（而FileChannel又比较慢）
3. 这种方式经历多个组件，效率变低，出现问题的概率也变大。

新的思路：

使用TailFileSource–>KafkaChannel这种方式，将KafkaChannel作为缓冲，效率变高，而且数据不会丢失。



这种方法就是使用TailFileSource来读取日志文件，然后将数据输送到KafkaChannel种，然后KafkaChannel直接将数据输送到Kafka集群中，此时不需要Sink，KafkaChannel相当于Kafka的生产者，这样就充分利用了Kafka集群的优点，当数据量很大的时候，也能hold得住。

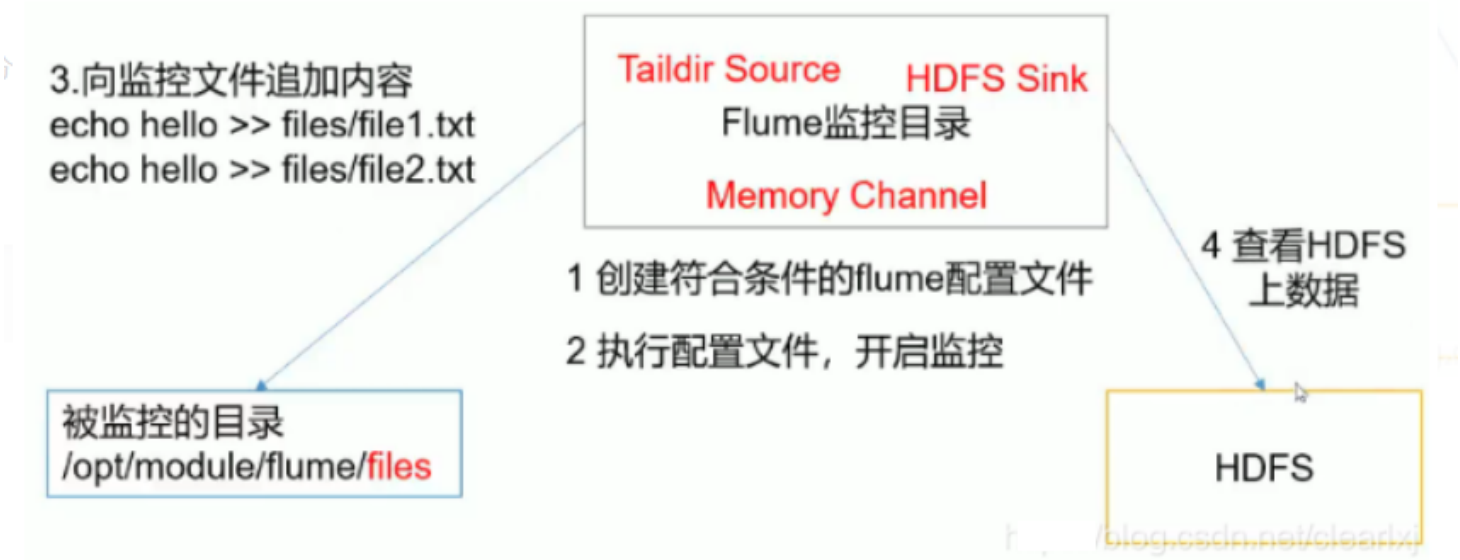
如果要将数据写入到HDFS或者ES中，要再创建一个flume集群，这个flume中只要有KafkaChannel和HDFSSink就可以了，此时的KafkaChannel相当于Kafka的消费者。但是要注意，为了避免多个flume消费同样的数据，要将多个flume实例放在同一个组内。

flume组件选型

Source

1. Taildir Source相比Exec Source、Spooling Directory Source的优势,这几个都是采集文件的source.

TailDir Source：**断点续传、保证数据不丢失，还可以监控数据。**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，继承了Exec Source实时传输数据和Spooling Directory Source断点续传的优点。



Exec Source：**可以实时监控数据，并且向文件中追加数据，但是在Flume不运行或者Shell命令出错的情况下，数据将会丢失**，也就是相当于重新执行了命令，但是再脚本挂掉期间，产生的数据会丢失。可以执行一个脚本，去事实的监控一个数据文件。

Spooling Directory Source：能够保证数据的不丢失，支持断点续传。原理是将写好的文件放到这个监控目录中，然后可以将文件中的内容一次性收集完。但是缺点是只要这个文件放到这个监控目录中之后，就不能修改文件的名字和追加内容，这是其局限性，所以不能事实的采集数据，所以再事实性监控，我们使用dlik cdc。

如果真要使用Spooling Directory Source实时监控数据，那么只能让文件小一点，比如几秒钟文东一个文件，尽可能的去模拟实时的场景，但是延迟高。

2. batchSize大小如何设置?

答：Event 1K左右时，500-1000合适（默认为100），也即是向channel中放数据的时候，一次性放多少个。

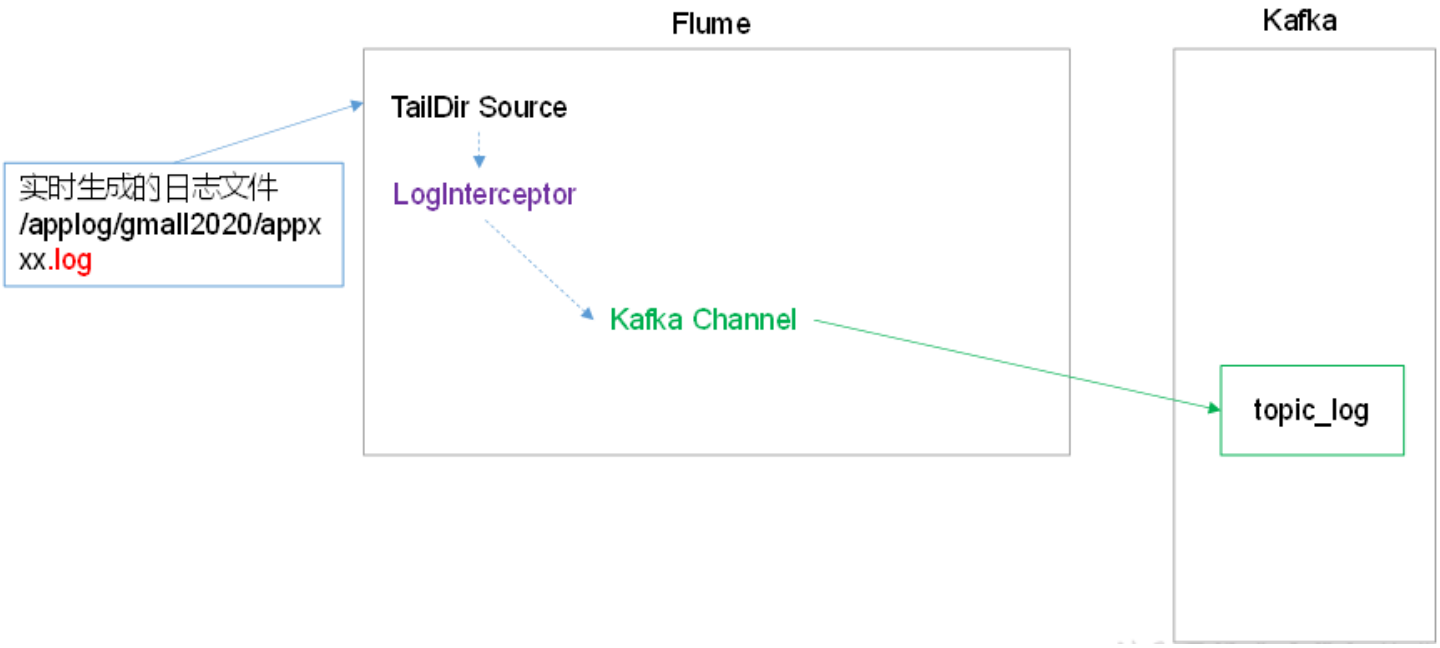
> 这里的批指的是什么，再put事务的时候，会首先将数据存储再一个putlist中，当批次满的时候，会提交一个事务，这个就是批次的大小。

Channel

采用Kafka Channel，直接将数据写入kafka集群，省去了Sink，提高了效率。KafkaChannel数据存储在Kafka里面，所以数据是存储在磁盘中。

> bug：注意在Flume1.7以前，Kafka Channel很少有人使用，因为发现parseAsFlumeEvent这个配置起不了作用。也就是无论parseAsFlumeEvent配置为true还是false，都会转为Flume Event。这样的话，造成的结果是，会始终都把Flume的headers中的信息混合着内容一起写入Kafka的消息中，这显然不是我所需要的，我只是需要把内容写入即可。

flume结构



Flume拦截器

flume拦截器属于flume的source中一个组件。使用拦截器，可以做日志的过滤或者分类。在这里做日志的清晰，清除不合法的日志。

1. 自定义类实现Interceptor 接口。
2. 实现initialize()初始化方法。
3. 实现Event intercept(Event event)拦截方法，判断一个事件是否是合法的。
4. 实现List intercept(List list)方法，这个方法是对上面第三个方法的封装，

在这里，对字符串的判断，使用的是阿里巴巴的fastJson对json字符串进行校验。

在这里创建拦截器对象使用的是builder创建者模式，自定义创建者实现Interceptor.Builder类，重写下面方法：

1. Interceptor build()创建一个拦截器对象。
2. void configure(Context context)配置信息。

消费kafka的Flume

方案选型

消费kafka中数据的Flume有两种配置方案：

1. 配置一个完整的flume，包括kafka source，channel和hdfs sink。

2. 配置一个kafka channel省去了source组件，然后再配置一个hdfs sink，这两种方案都可以。

但是这里也存在问题，flume sink将数据写入hdfs的时候，如何去写，因为我们离线数仓是一种批处理，那么这个一批指的是一天的数据，一次性需要计算一天的数据，所以再这里存储数据的时候，最好是按照分天进行存储，所以路径应该以天为单位。

但是实现这种方案，需要我们对event添加一个时间戳，由于flume默认会用linux系统时间（如果和flink类比的话，应该是摄入事件），作为输出到HDFS路径的时间。如果数据是23:59分产生的。Flume消费kafka里面的数据时，有可能已经是第二天了，那么这部门数据会被发往第二天的HDFS路径。我们希望的是根据日志里面的实际时间（也就是事件事件），发往HDFS的路径，所以下面拦截器作用是获取日志中的实际时间。

在这里我们依然使用一个拦截器解决：

拦截器属于source端的组件，所以再上面的两种方案中，我们就只能使用第一种方案了。自定义拦截器，提取日志中的事件事件，然后把事件事件添加到event的header中，根据header中的事件时间创建文件夹写出数据。

在这里我么也可以使用第二种方案，我们把提取时间戳的拦截器放到第一个flume中，形成一个拦截器链，这样做的话，再写入kafka的时候，我们就需要保留evevt的头部信息，不能过滤掉。

自定义拦截器步骤请参考上文。

组件选型

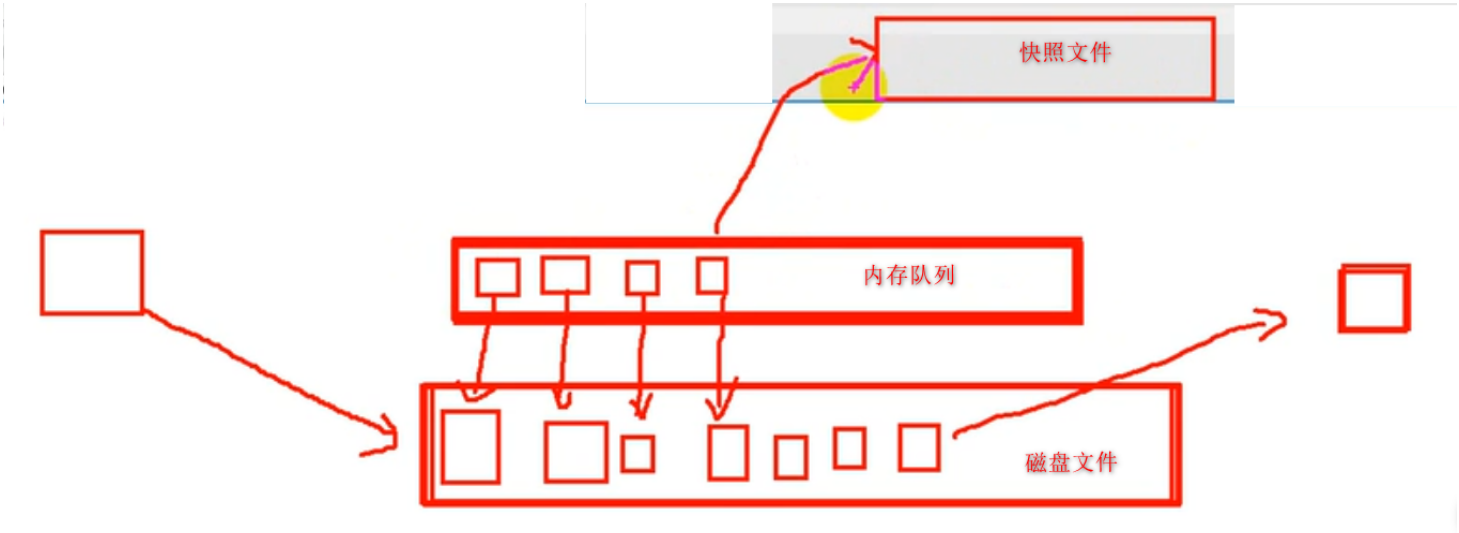
采用kafka source，再底层就是一个kafka的消费者，

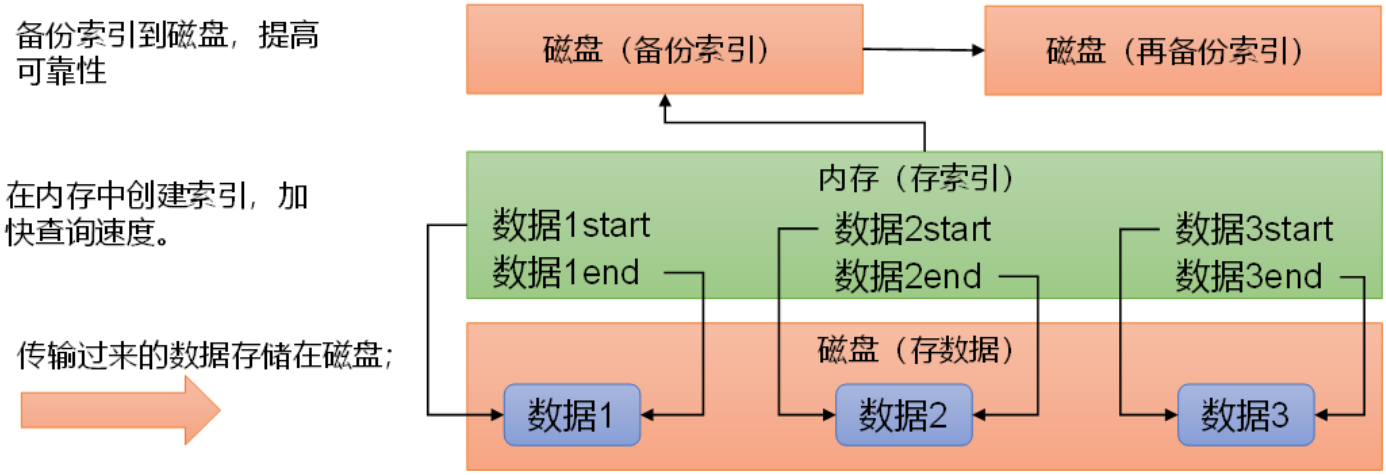
FileChannel和MemoryChannel区别

MemoryChannel传输数据速度更快，但因为数据保存在JVM的堆内存中，Agent进程挂掉会导致数据丢失，适用于对数据质量要求不高的需求。

FileChannel传输速度相对于Memory慢，但数据安全保障高,Agent进程挂掉也可以从失败中恢复数据。

为什么FileChannel不会发生数据的丢失：





可以看到底层的数据结构，真实的数据存储再磁盘文件中，但是还是有一个内存队列的，这个内存队列相当于是对磁盘上的文件建立一个索引，记录了消费到哪里。

虽然磁盘上的数据不会丢失，但是内存队列中的数据断电就会丢失，所以再内存队列之外，还有一个快照文件，这个快照文件可以看作是内存队列的快照，只要内存队列发生变化，这个快照文件就会跟着变化。这样就保证数据的安全性。

如果消费者挂点，那么就可以兄快照恢复，然后恢复读取的位置，重新消费。

选型

金融类公司、对钱要求非常准确的公司通常会选择FileChannel
传输的是普通日志信息（京东内部一天丢100万-200万条，这是非常正常的），通常选择MemoryChannel。

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(做快照的文件路径，保存内存索引)和backupCheckpointDir（检查点备份文件路径）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

批量指的是再将数据写入putlist事务列表的时候，事务列表的大小，再提交事务的时候，put事务会先检查一些channel是否有足够的空间存放数据，如果没有的话，就会立刻回滚数据，然后source会清空putlist列表，再次去拉去数据写入putlist中，但是如果立刻回滚事务，很消耗性能，所以就产生了keep_alive参数，等待几秒钟，再次提交事务，如果还没有成功，就进行回滚。

Sink

HDFS Sink

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=3600：1小时生成一个新文件

hdfs.rollSize=134217728：128m生成一个新文件

hdfs.rollCount =0：多少个event生成一个新文件

几个参数综合作用，效果如下：

1. 文件在达到128M时会滚动生成新文件
2. 文件创建超3600秒时会滚动生成新文件

数据压缩

再写入hdfs上的文件，我么也可以进行压缩处理，使用lzo压缩。

flume目前支持三种：

二进制文件：SequenceFile。

DataStream:不做任何处理。

Compress:压缩文件，目前支持：gzip,bzip2,lzo,snappy。

数据采集小结

web/app---> 日志服务器--->Flume--->kafka--->flume--->hdfs

再实际中，日志服务器可能部署再大数据集群中，也可能不在。

再第一个Flume中，有三种实现方案。

第二个flume中，有两种实现方案。

> 思考为什么中间需要一个kafka集群?

业务数据采集平台

业务数据是使用sqoop将数据导入到hdfs上面，对于业务数据，重要的是数据业务表之间的联系。

两个重要概念

电商常识（SKU、SPU）

SKU=Stock Keeping Unit（库存量基本单位）。现在已经被引申为产品统一编号的简称，每种产品均对应有唯一的SKU号。

SPU（Standard Product Unit）：是商品信息聚合的最小单位，是一组可复用、易检索的标准化信息集合。

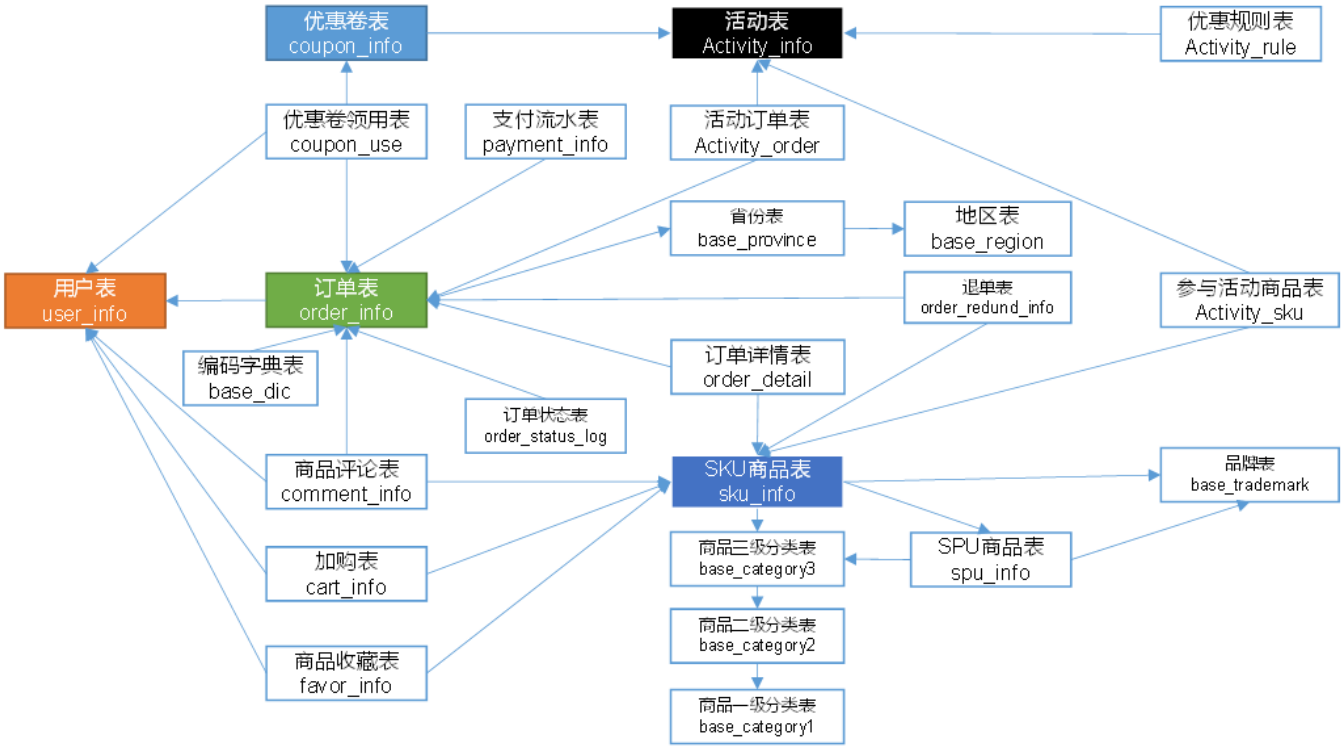
例如：iPhoneX手机就是SPU。一台银色、128G内存的、支持联通网络的iPhoneX，就是SKU。

SPU表示一类商品。好处就是：可以共用商品图片，海报、销售属性等。

spu描述的粒度更粗，sku描述一件具体商品，粒度细。

表ER模型图

本项目中设计24张表结构。



拿到表的脚本建立表之后，使用EZDML工具建立各个表之间的联系。

使用sqoop导入数据

sqoop可以实现双向传输，也就是从sqoop到mysql和从mysql到sqoop的双向传输。

sqoop的底层就是mapreduce任务，所以延迟很高，底层的mr任务只有map任务，没有reduce任务，这个可以从sqoop功能来看，sqoop的定位只是用来传输数据，不用来分析数据，所以只需要使用map将数据读出来然后写入hdfs即可。

那么sqoop做的工作就是自定义inoutFormat和outputFormat组件。自定义读取数据可以从数据库中读取。

sqoop基础操作

连接mysql

```
bin/sqoop list-databases --connect jdbc:mysql://hadoop102:3306/ --username root --password 00000
```

再导入的时候，sqoop支持将mysql数据库中的一张表导入到hdfs中的一个路径，或者是一张表或者是hbase中的一张表，但是到处的时候，只支持将hdfs上一个路径下的文件导出到mysql中的一张表中。

导入数据

```
bin/sqoop import --connect jdbc:mysql://hadoop102:3306/gmall
--user root
--password root
--table user_info  //全量表
--columns id,login_name//导入具体的列
--where "id >=10 and id<=30" //导入数据的过滤条件
--target-dir /test // 上传文件的路径
--delete-target-dir //如果目标路径存在就删除，这个参数可以保证数据的幂等性，因为如果任务失败，再次导入的
//优化相关参数
--num-mappers 2//指的是到数据的时候map任务个数
--fields-terminated-by //字段之间分隔符
--splits-by id //按照哪一个字段进行切片
```

关于这里的切片机制，自定义的inoutFormat会根据我们输入的配置进行切片，也就是根据我们输入的--num-mappers参数，将数据根据id进行切片，然后交给map处理。

使用sql方式导入数据

```
bin/sqoop import --connect jdbc:mysql://hadoop102:3306/gmall
--user root
--password root
--query "select id,login_name from user_info where id >=10 and id<=30"
--target-dir /test // 上传文件的路径
--delete-target-dir //如果目标路径存在就删除，这个参数可以保证数据的幂等性，因为如果任务失败，再次导入的
//优化相关参数
--num-mappers 2//指的是到数据的时候map任务个数
--fields-terminated-by //字段之间分隔符
--splits-by id //按照哪一个字段进行切片
```

同步策略

数据同步策略的类型包括：**全量同步、增量同步、新增及变化同步、特殊情况**

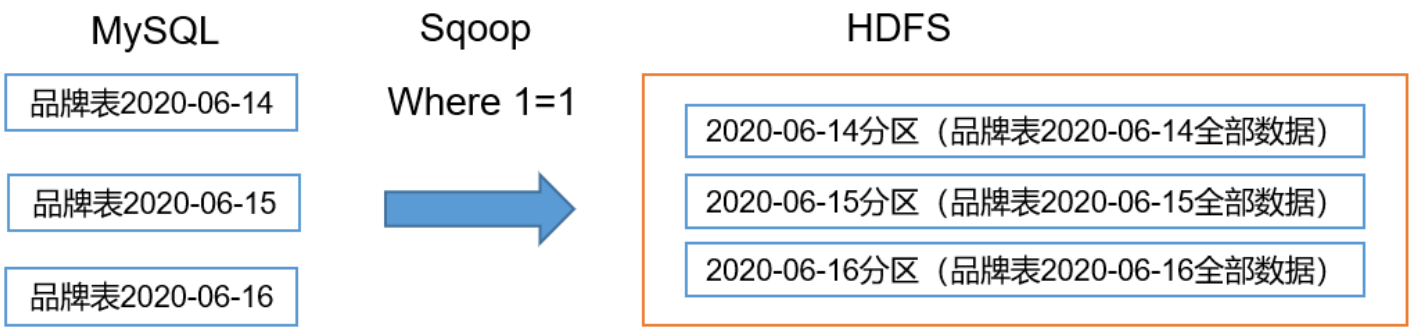
- 全量表：存储完整的数据。
- 增量表：存储新增加的数据。
- 新增及变化表：存储新增加的数据和变化的数据。
- 特殊表：只需要存储一次。

全量同步

每日全量，就是每天存储一份完整数据，作为一个分区，所以说全量表也是一个分区表，里面存储的是每天从mysql中导出来的全部数据。

适用于表数据量不大，且每天既会有新数据插入，也会有旧数据的修改的场景。

例如：编码字典表、品牌表、商品三级分类、商品二级分类、商品一级分类、优惠规则表、活动表、活动参与商品表、加购表、商品收藏表、优惠卷表、SKU商品表、SPU商品表

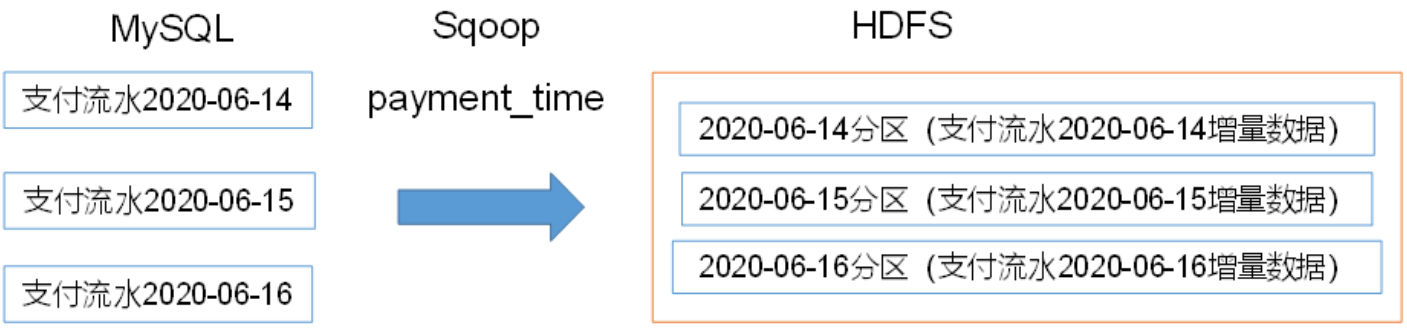


增量同步

每日增量，就是每天存储一份**增量数据**，作为一个分区，增量数据通常存储再**增量表**中，可以按照天使用一个**分区存储**。

增量同步适用于mysql中每天只会新增数据，不会发生修改的表，比流水表，支付流水，订单状态流水表。

适用于表数据量大，且每天只会有新数据插入的场景。例如：退单表、订单状态表、支付流水表、订单详情表、活动与订单关联表、商品评论表。



新增及变化策略

每日新增及变化，就是存储**创建时间和操作时间**都是今天的数据，这里使用的不是分区表。

查询数据主要有两种，获取最新数据，获取历史上某一天数据，但是再新增及变化表中查询上面两种数据很不方便。

适用场景为，表的数据量大，既会有新增，又会有变化。例如：用户表、订单表、优惠卷领用表。

trade_body	create_time	operate_time
小米（MI）电视 55英寸曲面4K智能WiFi网络液晶电视机4S L55M5-AQ 小米电	2020-06-14 08:37:45	2020-06-15 08:39:34
北纯精制黄小米（小黄米 月子米 小米粥 粗粮杂粮 大米伴侣）2.18kg等1件	2020-06-14 08:37:45	2020-06-15 08:39:34
Apple iPhoneXSMax (A2104) 256GB 深空灰色 移动联通电信4G手机 双卡双待	2020-06-14 08:37:45	2020-06-15 08:39:34
小米（MI） 小米路由器4 双千兆路由器 无线家用穿墙1200M高速双频wifi 千	2020-06-14 08:37:45	2020-06-14 08:37:45
迪奥（Dior）烈艳蓝金唇膏/口红 珊瑚粉 ACTRICE 028号 3.5g等2件商品	2020-06-14 08:37:45	2020-06-14 08:37:45
北纯精制黄小米（小黄米 月子米 小米粥 粗粮杂粮 大米伴侣）2.18kg等3件	2020-06-14 08:37:45	2020-06-14 08:37:45
Apple iPhoneXSMax (A2104) 256GB 深空灰色 移动联通电信4G手机 双卡双待	2020-06-14 08:37:45	2020-06-14 08:37:45
迪奥（Dior）烈艳蓝金唇膏/口红 珊瑚粉 ACTRICE 028号 3.5g等2件商品	2020-06-14 08:37:45	2020-06-14 08:37:45
荣耀10青春版 幻彩渐变 2400万AI自拍 全网通版4GB+64GB 渐变蓝 移动联通	2020-06-14 08:37:45	2020-06-14 08:37:45
小米（MI） 小米路由器4 双千兆路由器 无线家用穿墙1200M高速双频wifi 千	2020-06-15 08:39:33	2020-06-15 08:39:34
迪奥（Dior）烈艳蓝金唇膏/口红 珊瑚粉 ACTRICE 028号 3.5g等4件商品	2020-06-15 08:39:33	2020-06-15 08:39:34

特殊策略

某些特殊的维度表，可不必遵循上述同步策略。

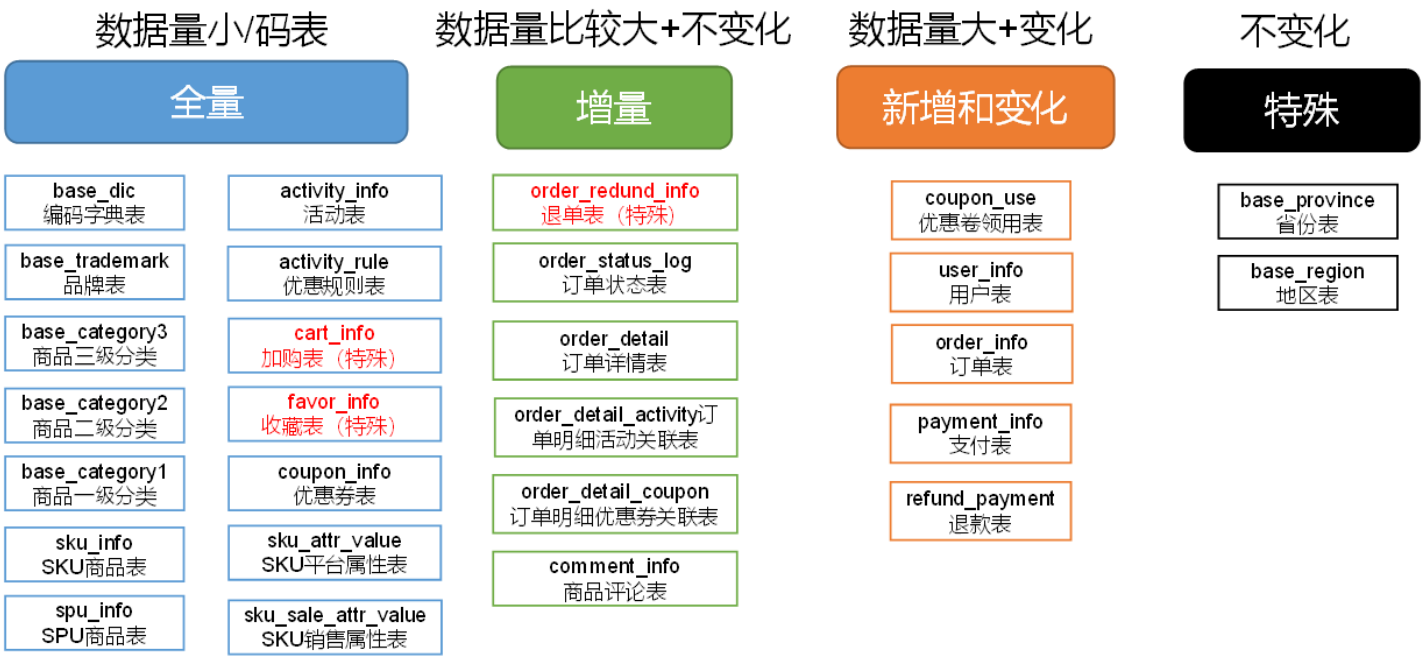
1）客观世界维度

没变化的客观世界的维度（比如性别，地区，民族，政治成分，鞋子尺码）可以只存一份固定值。

2）日期维度

日期维度可以一次性导入一年或若干年的数据。

本项目中表的导入策略



本项目中，全量表一般是一些描述性信息，有变化，但是数据量很小。

sqoop导入数据

使用脚本导入数据，一张表需要写一个sql命令。

这里注意一点就是再mysql中存储空值使用的是null，但是再HIve中空值存储再hdfs上面使用的是\N，

所以使用sqoop从mysql中导入数据到hive中需要将空值转换为\N，所以再sqoop的脚本中需要添加下面两个参数：

```
--null-string '\\N' \      表示字符串类型的空值存储为什么
--null-non-string '\\N'   非字符串类型空值存储为什么
```

同步脚本请参考文档。

如何同步增量数据：

```
import_data order_detail "select
                              id,
                              order_id,
                              sku_id,
                              sku_name,
                              order_price,
                              sku_num,
                              create_time,
                              source_type,
                              source_id,
                              split_total_amount,
                              split_activity_amount,
                              split_coupon_amount
                            from order_detail
                            where DATE_FORMAT(create_time,'%Y-%m-%d')='$do_date'"
}
```

增量数据，比如说订单详情表，如果我们需要获取前一天的新增数据，那么就是订单的支付时间==前一天的时间即可。

全量数据：

全量数据全部需要导入，所以where条件直接写为true即可，也就是1=1.

新增及变化

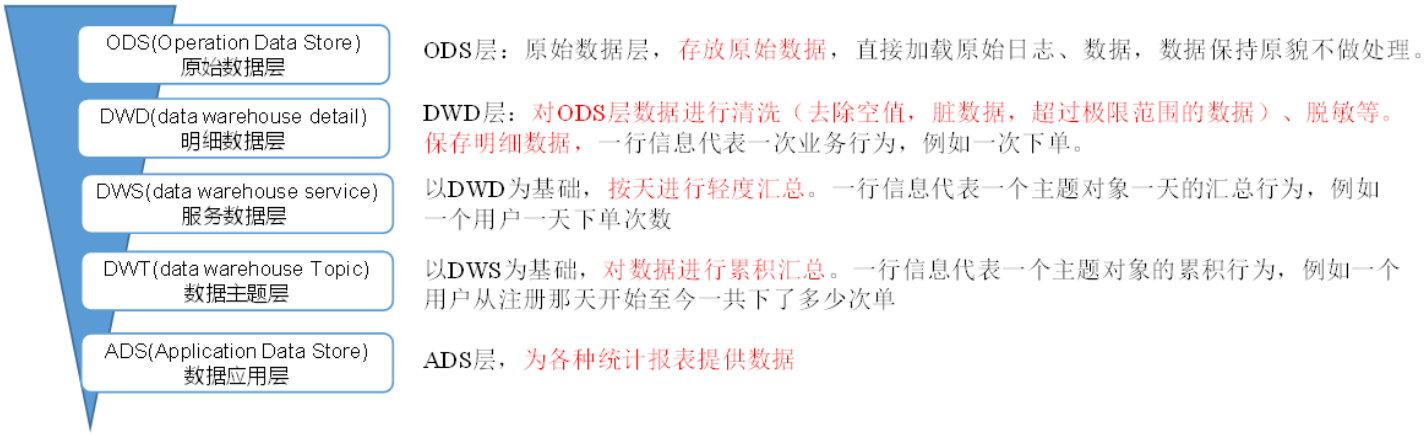
```
import_data "user_info" "select
                            id,
                            login_name,
                            nick_name,
                            name,
                            phone_num,
                            email,
                            user_level,
                            birthday,
                            gender,
                            create_time,
                            operate_time
                          from user_info
                          where (DATE_FORMAT(create_time,'%Y-%m-%d')='$do_date'
                          or DATE_FORMAT(operate_time,'%Y-%m-%d')='$do_date')"
}
```

需要两个条件：

1. 创建时间等于前一天，可以找到新增的数据。
2. 操作时间等于前一天，可以找出变化数据。

数仓理论

离线数仓分层



明细数据和汇总是一个相反的概念，明细表示最原始，最详细的数据，汇总，比如统计今天的支付金额，一条数据由多条数据汇总而来。

dwd层存储的数据是最明细的数据，比较重要的一层，是基础。对于业务数据来说，本来就是结构化的，但是对于日志数据来说，是一个json字符串，需要解析为一个一个的字段。

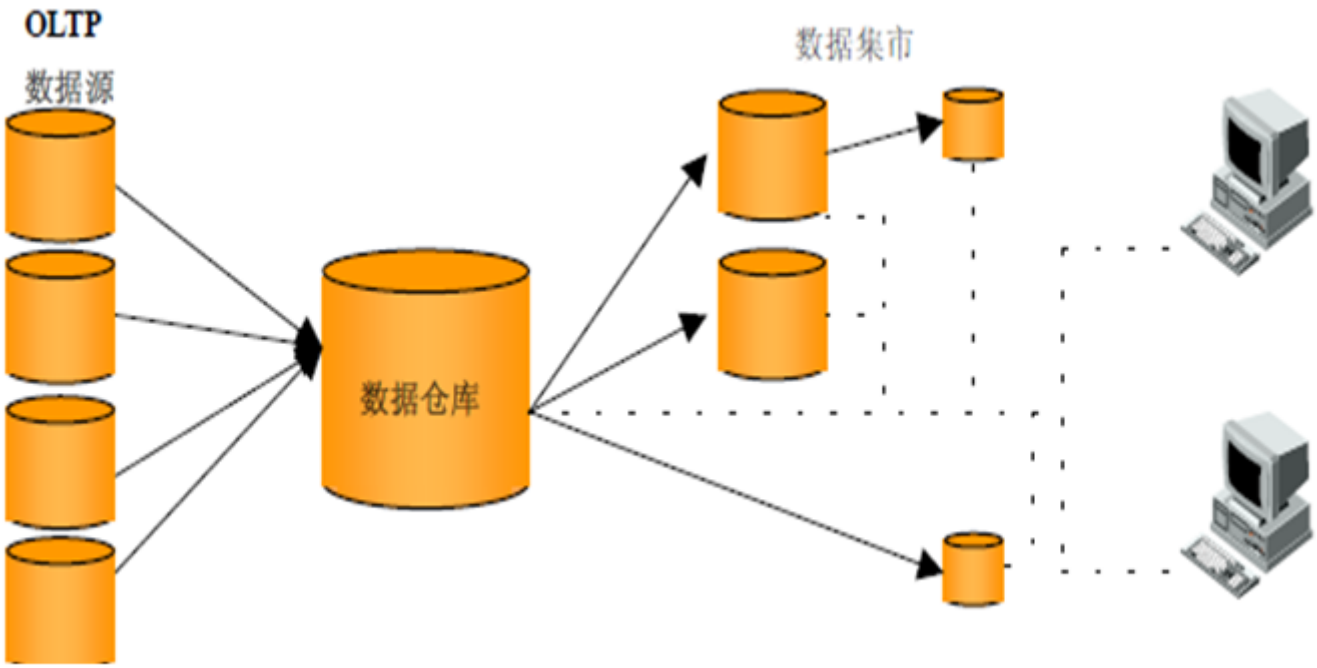
dws和dwt都是汇总的数据，不同的是dws是按照**天**进行汇总的，而dwt是按照**主题**进行汇总，汇总多少天或者某一个地区的数据，以主题为单位。汇总的粒度不一样。

ads是聚合好的数据，比如需要报表数据，那么ads层就把报表数据处理好，别人直接使用即可。

数据集市和数据仓库区别

数据集市则是一种微型的数据仓库，它通常有更少的数据，更少的主题区域，以及更少的历史数据，因此是部门级的，一般只能为某个局部范围内的管理人员服务。

数据仓库是企业级的，能为整个企业各个部门的运行提供决策支持手段。



范式理论

定义

范式可以理解为设计一张数据表的表结构，符合的标准级别、规范和要求。

优点

采用范式，可以降低数据的冗余性。

为什么要降低数据冗余性?

1. 十几年前，磁盘很贵，为了减少磁盘存储。
2. 以前没有分布式系统，都是单机，只能增加磁盘，磁盘个数也是有限的
3. 一次修改，需要修改多个表，很难保证数据一致性

缺点

范式的缺点是获取数据时，需要通过Join拼接出最后的数据。数据规范化，那么数据的粒度越细，但是会影响查询性能。

分类

目前业界范式有：

1. 第一范式(1NF)：消除完全函数依赖
2. 第二范式(2NF)：消除部分函数依赖
3. 第三范式(3NF)：消除传递依赖
4. 巴斯-科德范式(BCNF)
5. 第四范式(4NF)
6. 第五范式(5NF)

级别越高，数据的冗余度越小。

第一范式：

设X，Y是关系R的两个属性集合，X'是X的真子集，存在X→Y，但对每一个X'都有X'!→Y，则称Y完全函数依赖于X。记做：$X \xrightarrow{F} Y$.

比如通过，(学号，课程) 推出分数 ，但是单独用学号推断不出来分数，那么就可以说：分数 完全依赖于(学号，课程) 。

即：通过AB能得出C，但是AB单独得不出C，那么说C完全依赖于AB。

第一范式实际上要求所有的属性不可分割，实际上，1NF是所有关系型数据库的最基本要求，你在关系型数据库管理系统（RDBMS），例如SQL Server，Oracle，MySQL中创建数据表的时候，如果数据表的设计不符合这个最基本的要求，那么操作一定是不能成功的。也就是说，只要在RDBMS中已经存在的数据表，一定是符合1NF的。

第二范式

假如Y函数依赖于X，但同时Y并不完全函数依赖于X，那么我们就称Y部分函数依赖于X，记做：$X \xrightarrow{P} Y$.

比如通过，(学号，课程) 推出姓名，因为其实直接可以通过，学号推出姓名，所以：姓名部分依赖(学号，课程)

即：通过AB能得出C，通过A也能得出C，或者通过B也能得出C，那么说C部分依赖于AB。

如果存在部分函数依赖，我们可以把主键拆开，分为两张表即可。

第三范式

传递函数依赖：设X，Y，Z是关系R中互不相同的属性集合，存在$X \rightarrow Y(Y \not\rightarrow X), Y \rightarrow Z$，则称Z传递函数依赖于X。记做：$X \xrightarrow{T} Z$

比如：学号推出系名，系名推出系主任， 但是，系主任推不出学号，系主任主要依赖于系名。这种情况可以说：系主任传递依赖学号。

通过A得到B，通过B得到C，但是C得不到A，那么说C传递依赖于A。

同样，如果存在传递函数依赖，也需要对表进行拆分操作。

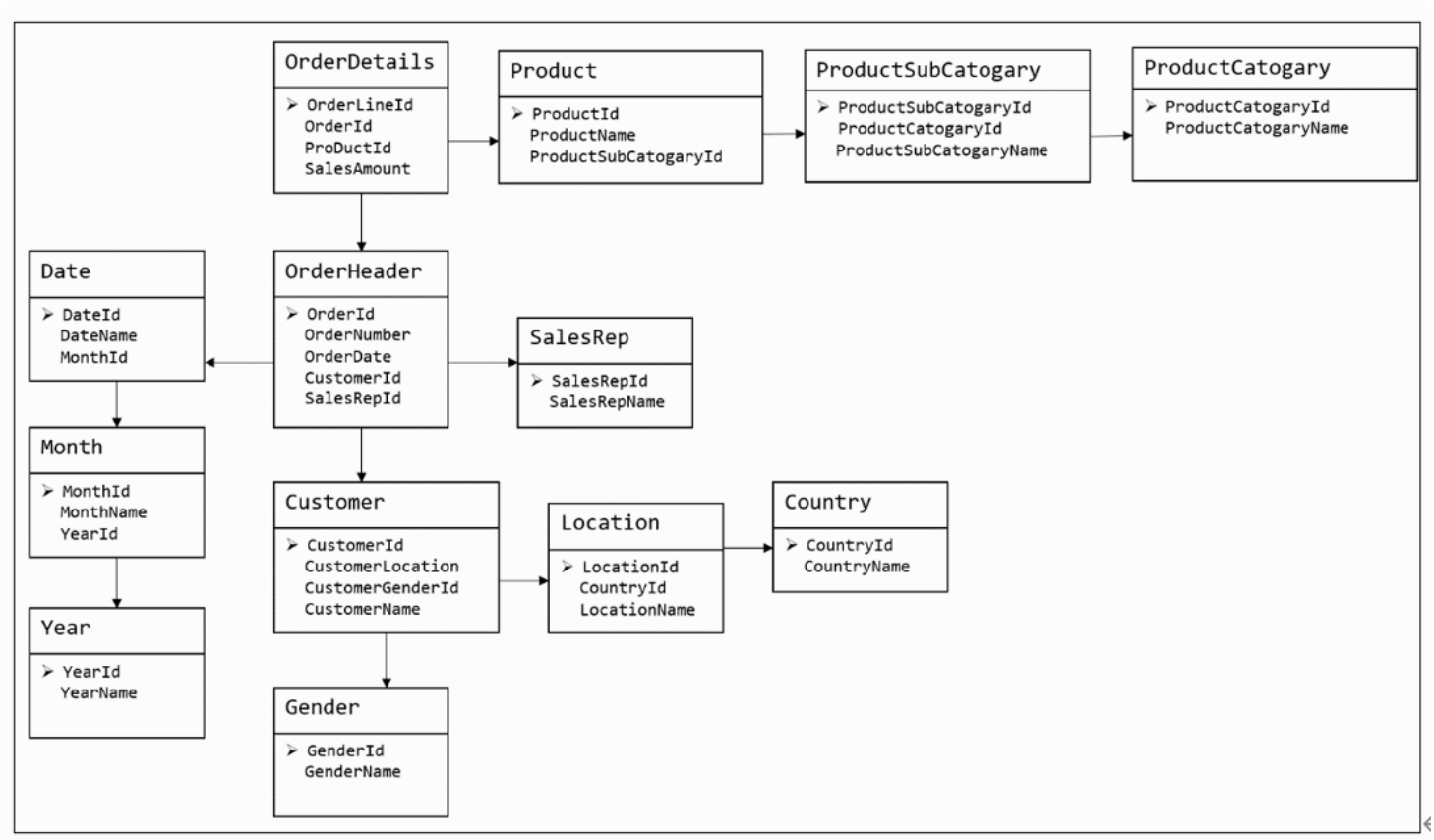
关系建模与维度建模

当今的数据处理大致可以分成两大类：**联机事务处理OLTP（on-line transaction processing）、联机分析处理OLAP（On-Line Analytical Processing）**。OLTP是传统的关系型数据库的主要应用，主要是基本的、日常的**事务处理**，例如银行交易。OLAP是数据仓库系统的主要应用，支持复杂的分析操作，侧重决策支持，并且提供直观易懂的查询结果。二者的主要区别对比如下表所示。

对比属性	OLTP	OLAP
读特性	每次查询只返回少量记录	对大量记录进行汇总
写特性	随机、低延时写入用户的输入	批量导入
使用场景	用户，Java EE 项目	内部分析师，为决策提供支持
数据表征	最新数据状态	随时间变化的历史状态
数据规模	GB	TB 到 PB

通常对于范式建模，我们使用er图建模，范式理论建立额模型表的数量比较多。

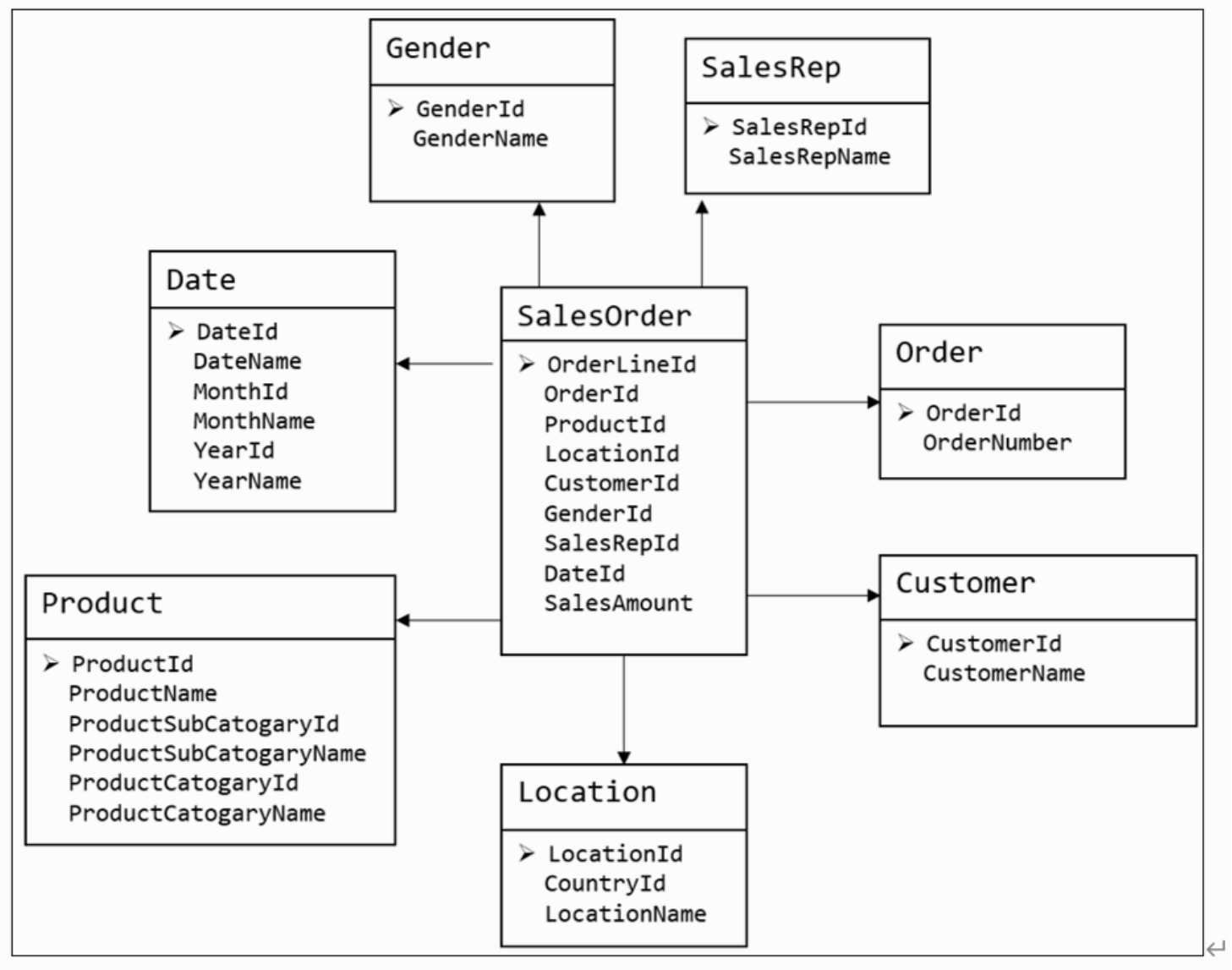
关系建模



关系模型如图所示，严格遵循第三范式（3NF），从图中可以看出，较为松散、零碎，物理表数量多，数据的粒度比较细，而数据冗余程度低。由于数据分布于众多的表中，这些数据可以更为灵活地被应用，功能性较强。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。缺点是查询数据的时候比较麻烦，需要进行很多的join操作，影响查询的性能。

因为范式建模需要去除数据的冗余性，所以需要对表进行切割，导致表比较多，比较分散。

维度建模



维度模型如图所示，主要应用于OLAP系统中，通常以某一个**事实表**（事实表一般在中心，并且有多张事实表，主要是一些操作）为中心进行表的组织，主要面向业务，特征是可能存在数据的冗余，但是能方便的得到数据。以业务为驱动，很方便理解。更适合做数据分析，更适合做聚合的数据分析。维度表一般是一些描述性的信息，而事实表中的属性就是维度表的外键组成的。维度表的属性一般有某一个事务的属性以及其他和该事物相关的属性组成的。

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。所以通常我们采用维度模型建模，把相关各种表整理成两种：事实表（中间的表）和维度表（两边的表）两种。维度表中存储的是对事实表的描述信息。事实表一般存储的是业务信息，动词。

维度模型通常是以一个主题为单位，通常围绕一个主题进行建模。这样可以完整的描述用户的一个动作，而如果使用范式理论建模，那多个表之间join才可以描述清除。

比如再上面的模型中，中间的事实表是销售订单，如果我想看各个地区的销售情况，那么我就可以将销售表和Location表进行关联，然后以LocationId进行分区聚合，类似的，还可以以Gender进行聚合操作。

事实表和维度表

维度表

维度表本质上是我们分析数据的角度，根据每一个维度去聚合分析事实表数据，也可以理解为我们后期写sql分组的字段。

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。可以看做是关系型数据库er图中的对象信息。

维表的特征：

1. 维表的范围很宽（具有多个属性、列比较多）对象本身的属性以及和对象相关联的其他对象的属性。
2. 跟事实表相比，行数相对较小：通常< 10万条，因为数对对象的描述性信息，不需要存储太多的数据
3. 内容相对固定：编码表

时间维度表

日期 ID	day of week	day of year	季度	节假日
2020-01-01	2	1	1	元旦
2020-01-02	3	2	1	无
2020-01-03	4	3	1	无
2020-01-04	5	4	1	无
2020-01-05	6	5	1	无

事实表

整个业务系统中，有很多的业务，那么每一个业务都有一张事实表和其对应，比如下单事实表，支付事实表中一行表示一个支付事件，订单事实表中一行表示一个订单事件。

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，2020年5月21日，宋宋老师在京东花了250块钱买了一瓶海狗人参丸。维度表：时间、用户、商品、商家。事实表：250块钱、一瓶。

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。

事实表的特征：

1. **非常的大，每天都会增加很多数据**

2. **内容相对的窄：列数较少（主要是外键id和度量值）**
3. **经常发生变化，每天会新增加很多。**

事务型事实表

以每个事务或事件为单位，例如一个销售订单记录，一笔支付记录等，作为事实表里的一行数据。**一旦事务被提交，事实表数据被插入，数据就不再进行更改，其更新方式为增量更新**。一行数据就是一个具体的事件。

事务性事实表对应mysql中的那张表是不会发生变化的，每一天只会新增数据，不会发生修改。

> 事务性事实表保留所有的数据。

周期型快照事实表

周期型快照事实表中**不会保留所有数据，只保留固定时间间隔的数据**，例如每天或者每月的销售额，或每月的账户余额等。离线数仓中一般周期是一天，也就是把mysql中一天的数据做一个快照。**周期型快照事实表就是一个全量表。**

例如购物车，有加减商品，随时都有可能变化，但是我们更关心每天结束时这里面有多少商品，方便我们后期统计分析。

累积型快照事实表

累计快照事实表用于跟踪业务事实的变化。

例如，数据仓库中可能需要累积或者存储订单从下订单开始，到订单商品被打包、运输、和签收的各个业务阶段的时间点数据来跟踪订单声明周期的进展情况。当这个业务过程进行时，事实表的记录也要不断更新。

订单 id	用户 id	下单时间	打包时间	发货时间	签收时间	订单金额
		3-8	3-8	3-9	3-10	

将事实表和同步策略对比：

事务性事实表：增量同步，分区表。事务性事实表存储的就是增量同步数据，按照分区表存储，每一天存储当天新增的数据。

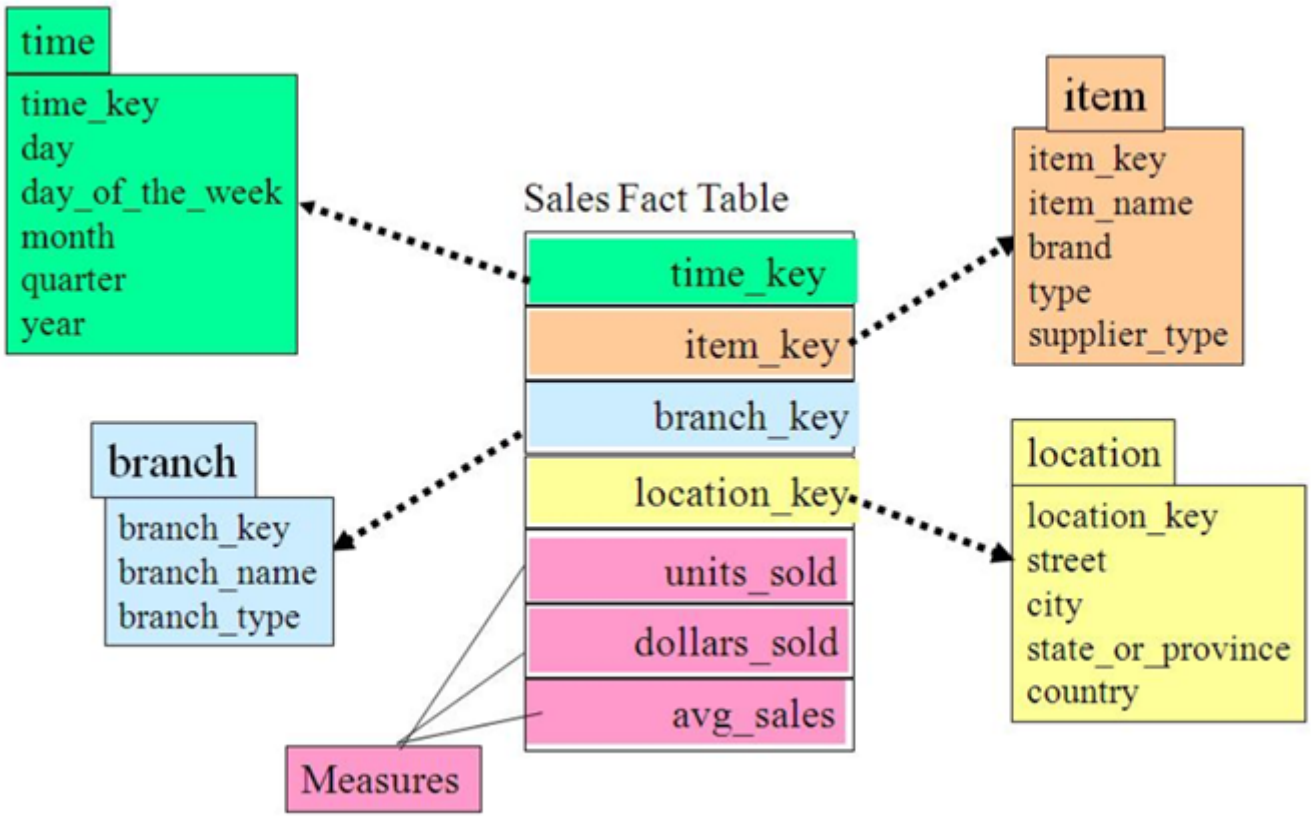
周期性快照事实表：对应全量同步，也是一个分区表，相当于每天做一个快照，每一个分区中存储mysql中一张表的快照。

累积性快照事实表：新增及变化同步，这一部分比较麻烦，因为要从Mysql中获取新增及变化然后和累积性快照事实表中的数据做一个整合。

维度模型分类

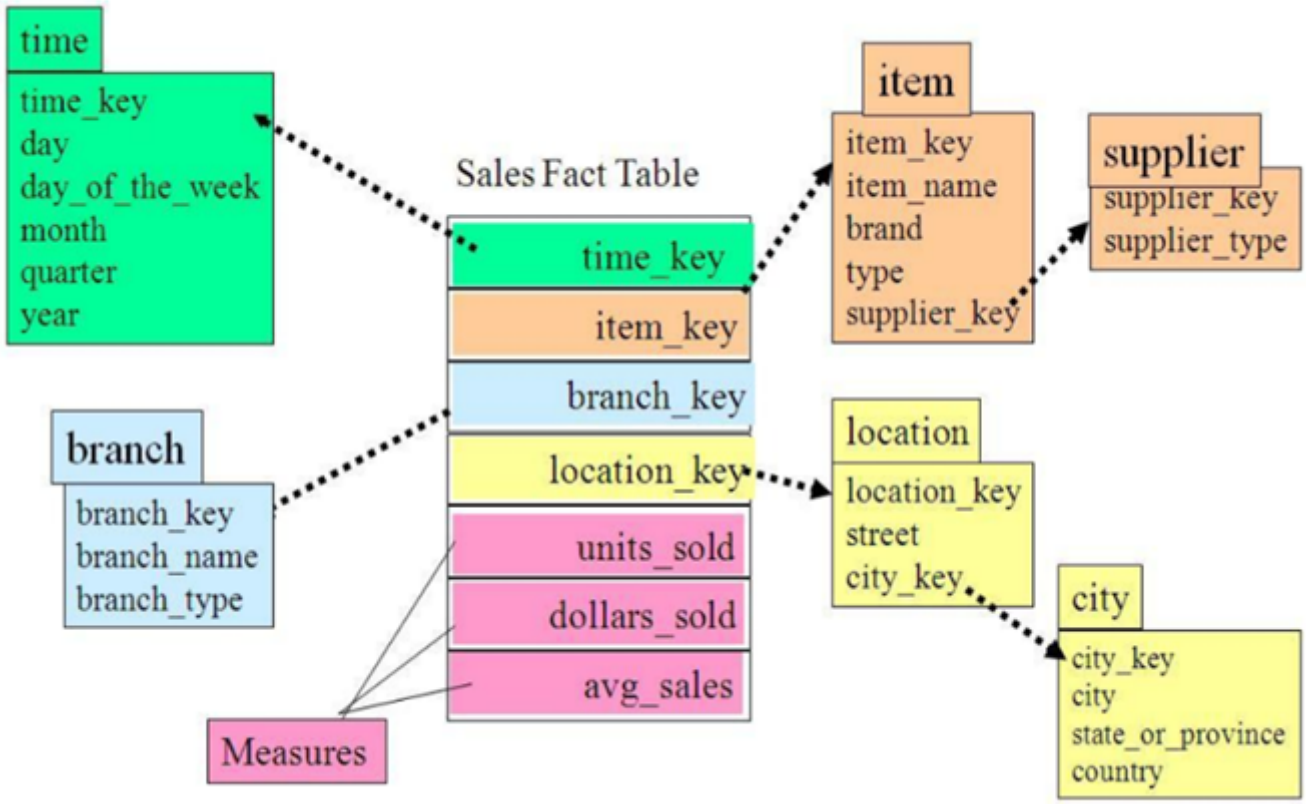
在维度建模的基础上又分为三种模型：星型模型、雪花模型、星座模型。

星型模型



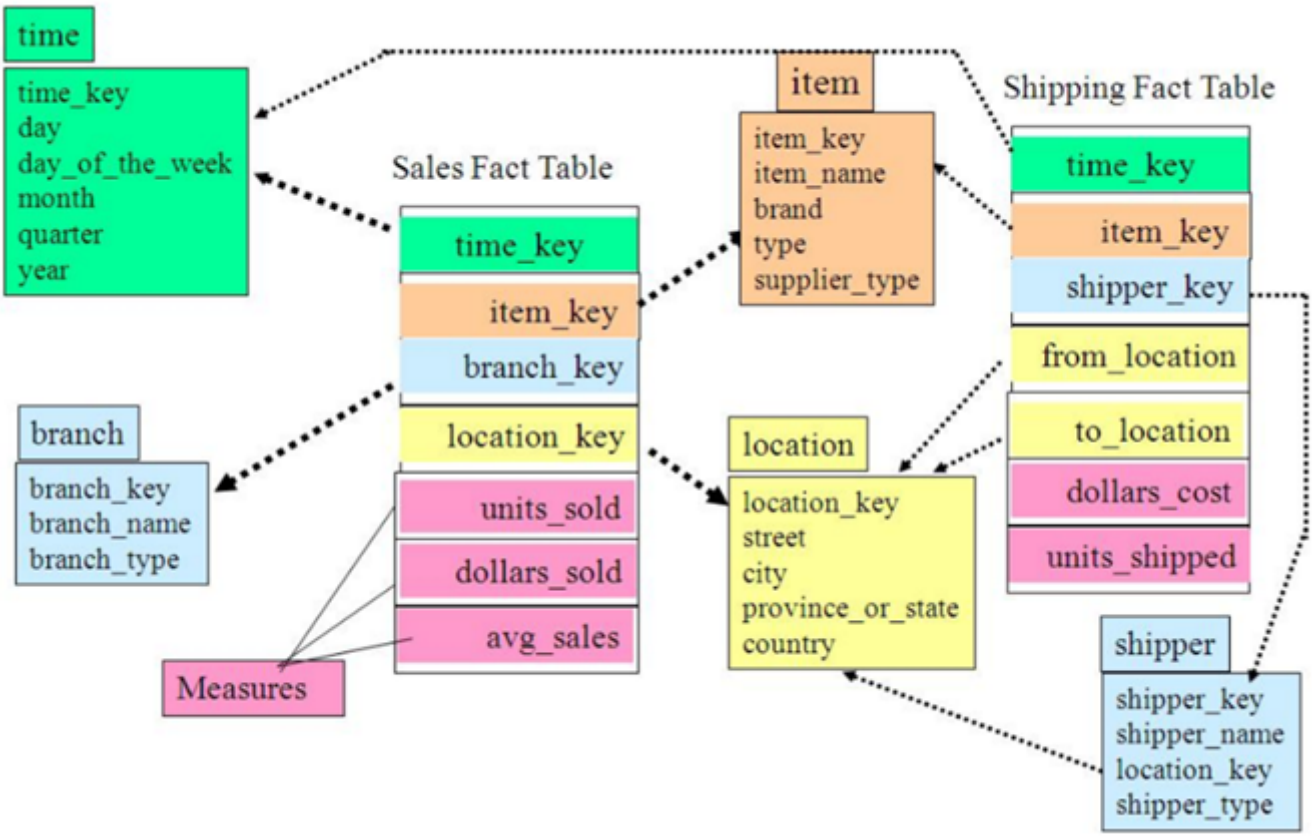
雪花模型与星型模型的区别主要在于维度的层级，标准的星型模型维度只有一层，而雪花模型可能会涉及多级。

雪花模型



雪花模型，比较靠近3NF，但是无法完全遵守，因为遵循3NF的性能成本太高。

星座模型



星座模型与前两种情况的区别是事实表的数量，星座模型是基于多个事实表。也就是说多个事实表可以公用一个维度表。

模型欸都选择

首先就是星座不星座这个只跟数据和需求有关系，跟设计没关系，不用选择。

基本上是很多数据仓库的常态,因为很多数据仓库都是多个
事实表的。所以星座不星座只反映是否有多个事实表,他们之间
是否共享一些维度表所以星座模型并不和前两个模型冲突

星型还是雪花，取决于性能优先，还是灵活更优先。目前实际企业开发中，不会绝对选择一种，根据情况灵活组合，甚至并存（一层维度和多层维度都保存）。但是整体来看，更倾向于维度更少的星型模型。尤其是Hadoop体系，减少Join就是减少Shuffle，性能差距很大。（关系型数据可以依靠强大的主键索引）

数据仓库建模

ODS层

HDFS用户行为数据

就是将hdfs上面的文件和表进行映射，中间做一个缓冲。对于日志，一般只建立**一张表**，将所有的日志放在一张表中(因为我们采集所有类型的日志都在一张表中)，表只有一个字段。在DWD层对日志进行解析。

HDFS业务数据

因为mysql业务数据本身就是结构化的数据，所以我们只需将mysql数据库中的表数据导入hdfs中即可，数据原封不动。

针对HDFS上的用户行为数据和业务数据，我们如何规划处理?

1. 保持数据原貌不做任何修改，起到备份数据的作用。
2. 数据采用压缩，减少磁盘存储空间（例如：原始数据100G，可以压缩到10G左右）
3. 创建分区表，防止后续的全表扫描

对于业务数据，hdfs上面有哪些数据文件，就建立那几张表，因为业务数据是从mysql中导入的，所以建立的表根据mysql表即可。

DWD层

DWD层需构建**维度模型**，一般采用星型模型，呈现的状态一般为星座模型。是最重要的一层。

维度建模一般按照以下四个步骤：

选择业务过程→声明粒度→确认维度→确认事实

1. 选择业务过程

在业务系统中，挑选我们感兴趣的业务线，比如**下单业务，支付业务，退款业务，物流业务**，一条业务线对应一张**事实表**。

如果是中小公司，尽量把所有业务过程都选择。

如果是大公司（1000多张表），选择和需求相关的业务线。

2. 声明粒度

数据粒度指数据仓库的数据中保存数据的**细化程度或综合程度**的级别。

声明粒度意味着精确定义事实表中的一行数据表示什么，应该尽可能选择最小粒度，以此来应各种各样的需求。

典型的粒度声明如下：

订单事实表中一行数据表示的是一个订单中的一个商品项。

支付事实表中一行数据表示的是一个支付记录。

确定维度

维度的主要作用是描述业务是事实，主要表示的是“谁，何处，何时”等信息。我们后期就是根据维度进行主题的统计。

确定维度的原则是：后续需求中是否要分析相关维度的指标。例如，需要统计，什么时间下的订单多，哪个地区下的订单多，哪个用户下的订单多。需要确定的维度就包括：**时间维度、地区维度、用户维度。**（也就是确认每一张事实表和哪一张维度表有关系）

确定事实

此处的“事实”一词，指的是业务中的**度量值**（次数、个数、件数、金额，可以进行累加），例如订单金额、下单次数等。（事实表中的字段有两类，第一类是维度表的外键，另一类是度量值，也就是说每一张事实表都有一个度量值，维度外键在第三步中确定，不同的业务，有不同的度量值）

在DWD层，以业务过程为建模驱动，基于每个具体业务过程的特点，构建最细粒度的明细层事实表。事实表可做适当的宽表化处理。

事实表和维度表的关联比较灵活，但是为了应对更复杂的业务需求，可以将能关联上的表尽量关联上。如何判断是否能够关联上呢？在业务表关系图中，只要两张表能通过中间表能够关联上，就说明能关联上。

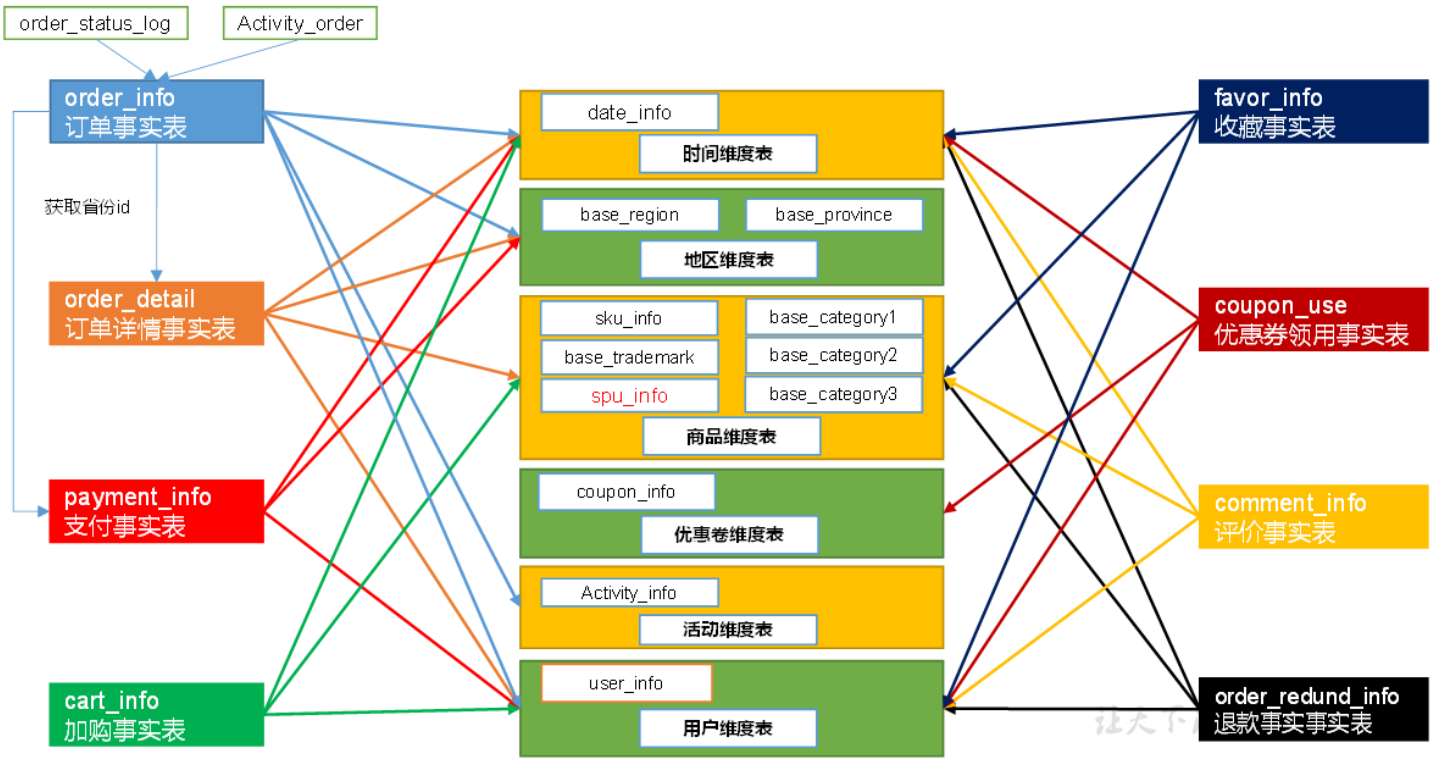
	时间	用户	地区	商品	优惠券	活动	编码	度量值
订单	√	√	√			√		件数/金额
订单详情	√	√	√	√				件数/金额
支付	√	√	√					金额
加购	√	√		√				件数/金额
收藏	√	√		√				个数
评价	√	√		√				个数
退款	√	√		√				件数/金额
优惠券领用	√	√			√			个数

横向描述的是事实表，纵向描述的是维度表。

至此，数据仓库的维度建模已经完毕，DWD层是以**业务过程**为驱动。

DWS层、DWT层和ADS层都是以**需求**为驱动，和维度建模已经没有关系了。

DWS和DWT都是建**宽表**，按照**主题**去建表。主题相当于观察问题的角度。对应着维度表。



对于日志数据，根据内容对日志文件进行解析。按照内容解析，每一类日志文件的字段类型一致，方便进行查询。（我们的日志一般分为，普通页面日志，启动日志，曝光日志等）

维度表一般是**名词**，可以从关系型数据库的er图中去选择。一般都是描述性的表。