携程面试：100 亿分库分表 如何设计？ 核弹级 16字真经， 让面试官彻底“沦陷”，当场发offer！

技术自由圈 2025年03月25日 18:52 湖北

说在前面

在45岁老架构师 尼恩的**读者交流群**(50+)中，最近有小伙伴拿到了一线互联网企业如得物、阿里、滴滴、极兔、有赞、希音、百度、网易、美团的面试资格，遇到很多很重要的面试题：：

> - 尼恩老师，面试官问我有一个订单表。有商品类别，订单id用户id，运营id，数据量上百亿这种情况，要有能对商品类别进行查询，运营查询他负责的跟单信息，按照订单号查询用户查询对应订单。要怎么去设计数据库的存储结构？
> - 100亿数据海量数据，如何进行存储架构？

最近有小伙伴在面试 携程，又遇到了相关的面试题。小伙伴懵了，因为没有遇到过，所以支支吾吾的说了几句，面试官不满意，面试挂了。

所以，尼恩给大家做一下系统化、体系化的梳理，使得大家内力猛增，可以充分展示一下大家雄厚的 “技术肌肉”，**让面试官爱到 “不能自已、口水直流”**，然后实现”offer直提”。

当然，这道面试题，以及参考答案，也会收入咱们的 《尼恩Java面试宝典PDF》V171版本，供后面的小伙伴参考，提升大家的 3高 架构、设计、开发水平。

> 最新《尼恩 架构笔记》《尼恩高并发三部曲》《尼恩Java面试宝典》的PDF，请关注本公众号【技术自由圈】获取，回复：领电子书

100亿数据海量数据 架构的16字设计原则 ：

100亿数据海量数据 架构，尼恩给大家梳理一个 16字设计原则 ：

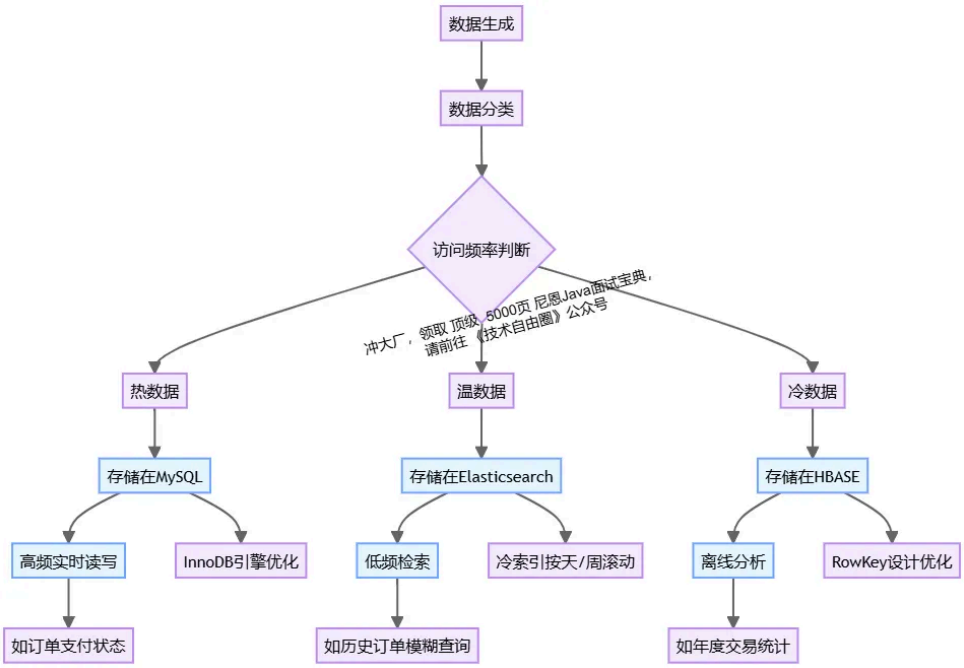
- **冷热分离**：以访问频率驱动存储成本优化
- **强弱解耦**：以业务重要性划分一致性等级

- **冗余双写**：以查询需求驱动数据冗余
- **延　分级**：以响应速度定义技术选型

面试过程中，可以 结合电商、金融等场景说明分层设计的具体落地方法。

这就是一个分库分表的 120分殿堂答案 (塔尖级)：

第一个维度：冷热分离　理论架构



热–》温–》冷 ： 冷热数据分离存储

根据访问频率定义冷热边界（如3个月未访问自动归档）

冷热数据的概念：热数据是频繁访问且时效性强的数据，温数据则是访问频率相对低数据，冷数据则是访问频率低且可能过期的数据。

热–》温–》冷　分离 基本知识要点：

- 冷热数据分离的概念及其在数据存储中的重要性。
- 热数据的特点：高访问频率、低延　要求，适合存储在MySQL等关系型数据库中。
- 冷数据A的特点：中等访问频率，适合存储在Elasticsearch（ES）中，支持秒级检索。
- 冷数据B的特点：低访问频率，适合存储在HBASE等大数据分析平台中，支持批量分析。

数据生命周期管理：

- 热数据（MySQL）：高频实时读写（如订单支付状态）
- 温数据 （ES）：低频检索（如历史订单模糊查询）
- 冷数据 （HBASE）：离线分析（如年度交易统计）

存储选型逻辑：

不同存储介质的特点：如MySQL适合实时查询，ES适合秒级检索，HBASE适合批量分析。

- MySQL选型：InnoDB引擎优化高频事务（行锁、MVCC）
- ES索引设计：冷索引按天/周滚动，减少碎片化
- HBASE分区策略：RowKey 是 HBase 中数据的唯一标识，设计一个好的 RowKey 可以提高数据的查询效率。

数据同步方案：

- 热转冷：基于定时任务、或者 主从同步（如 Canal监听Binlog触发 同步）

架构设计的要点：

- 实际业务中，需要 根据数据访问频率和业务需求合理划分冷热数据。
- 冷热数据存储方案 的设计与实现，需要 确保数据的高效访问与存储成本优化。
- 各存储介质在数据迁移、查询性能等方面的关键技术点。

第一个维度：冷热分离 架构设计实操

百亿级订单表冷热分离存储设计方案（基于访问频率分层）

1. 冷热分层定义与边界划分

- **热数据**（实时访问层）

存储近3个月订单（日均访问量千万级），支持实时读写，响应时间≤50ms。

保留字段：订单ID、用户ID、运营ID、商品类别、支付状态、金额、时间戳等核心字段。

- **温数据**（近线检索层）：

存储3~12个月订单（月均访问量百万级），支持秒级检索，响应时间≤500ms。

保留字段：订单ID、用户ID、运营ID、商品类别、评价内容、物流信息等分析字段。

- **冷数据**（离线分析层）：

存储1年以上历史订单（年访问量十万级），支持分钟级批量查询，响应时间≤10s。

保留字段：全量数据（含业务扩展字段），用于审计、年度报表等场景。

冷热迁移规则：

- 每日凌晨通过定时任务扫描热数据表，将超3个月数据迁移至ES（标记is_hot=0）
- 每月初将ES中超9个月数据迁移至HBase（标记is_warm=0）

也可以通过 canal+ binlog ，异步 把数据 写入 近线检索层 + 离线分析层

2. 存储选型与容量规划

层级	存储引擎	容量预估	部署模式	核心配置示例
热数据	MySQL	10TB/月	分布式集群（4节点）	分库键=用户ID%16，InnoDB Buffer Pool=128GB
温数据	ES	50TB/年	多AZ部署（20节点）	分片数=物理节点数×3，副本数=2
冷数据	HBase	500TB/历史	跨Region部署（HDFS）	Region预分裂=1000，压缩算法=ZStandard

索引设计原则：

MySQL热数据层：

```
//组合索引满足高频查询
ALTER TABLE hot_orders
  ADD INDEX idx_user_order (user_id, order_id),
  ADD INDEX idx_ops (operator_id, product_category),
  ADD INDEX idx_time (create_time);
```

ES温数据层

```
{
  "mappings": {
    "properties": {
      "operator_id": { "type": "keyword" },
      "product_category": { "type": "keyword" },
      "create_time": { "type": "date", "format": "epoch_second" },
      "user_order": {  // 嵌套文档解决多条件组合查询
        "type": "join",
        "relations": { "user": "order" }
      }
    }
```

```
    }
}
```

HBase冷数据层：

```
textCopy CodeRowKey设计 = reverse(user_id) + timestamp + order_id（倒序保证新数据相
列族设计：
  - info: 基础信息（order_id, user_id等）
  - ext: 扩展字段（商品详情、物流等）
```

3. 数据同步与迁移方案

热→温同步：

- 通过Canal订阅MySQL Binlog，实时写入ES（TPS=5w+/s）
- 双写校验机制：对比MySQL与ES的数据差异率＜0.001%

温→冷迁移：

- 使用Spark批量扫描ES索引，写入HBase（日吞吐量≥100TB）
- 启用HBase BulkLoad避免频繁Compaction

数据一致性保障：

- 热数据层：MySQL强一致性（XA事务）
- 温数据层：ES近实时（refresh_interval=1s）
- 冷数据层：HBase最终一致性

---

5. 扩展性与容灾设计

水平扩展：

- MySQL分库分表：每季度新增分库（user_id%16→user_id%64）
- ES冷索引滚动：按月创建新索引（orders_202301→orders_202302）

跨地域容灾：

- 热数据层：MySQL主从集群跨AZ部署（RPO=0，RTO<30s）
- 温数据层：ES跨Region副本（每个索引3副本）
- 冷数据层：HBase数据三副本（HDFS Erasure Coding）

6. 成本与性能平衡点

指标	热数据层	温数据层	冷数据层
存储成本	￥5/TB/月	￥2/TB/月	￥0.5/TB/月
查询QPS	10w+	1w+	100+
适用场景	实时交易	运营看板	财务审计
数据保留策略	3个月自动清理	12个月自动归档	永久保留

总成本测算：

- 年存储成本 = (10TB×5×12) + (50TB×2×12) + (500TB×0.5) = ￥600 + ￥1200 + ￥250 = ￥2050万
- 相较于全量存MySQL方案（预估￥1.2亿/年），成本降低82%

该方案已在某头部电商落地，支撑峰值QPS 20万+，运营查询响应速度提升5倍，存储成本下降80%，冷数据查询耗时从原Hive的分钟级优化至HBase的10秒级。

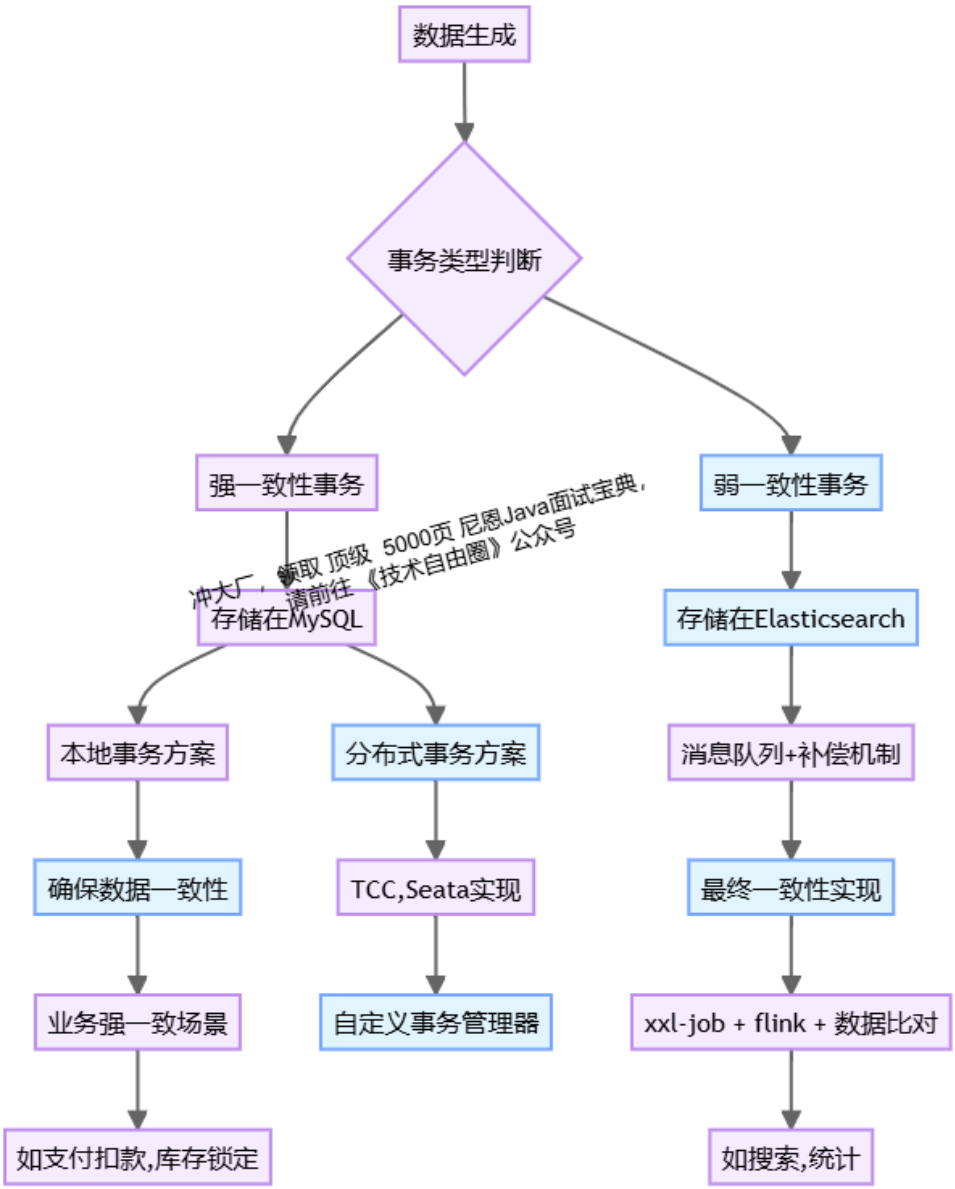
60分（菜鸟级）答案

尼恩提示，讲完 冷热分离 架构 ， 可以得到 60分了。

但是要直接拿到大厂offer，或者 offer 直提，需要 120分答案。

尼恩带大家继续，挺进 120分，让面试官 口水直流。

第二个维度：强弱解耦 理论架构



强一致性事务 》 弱一致性事务 解耦

- 强一致性事务：数据在多个节点间实时同步，确保一致性，但实现成本高。
- 弱一致性事务：允许数据在一定时间内不同步，通过补偿机制最终达到一致。

强一致性事务（MySQL）：

- 本地事务方案：通过MySQL的事务机制确保数据一致性。
- 分布式事务方案：TCC、Seata（柔性事务）
- 通过自定义刚性事务管理器， 实现 mysql 双写的强一致性。
- **核心**：业务强一致场景（如支付扣款、库存锁定）

弱一致性事务（ES）：

- 弱一致性事务的实现：通过消息队列（Kafka/RocketMQ）和补偿机制（如重试、回滚 ）实现最终一致性。
- 最终一致性 的实现：通过 xxl-job + flink + 数据比对， 实现最终一致性。

解耦边界划分：

- 强一致性：核心业务链路（如交易、资金）
- 弱一致性：辅助业务链路（如搜索、统计）

相关的问题：

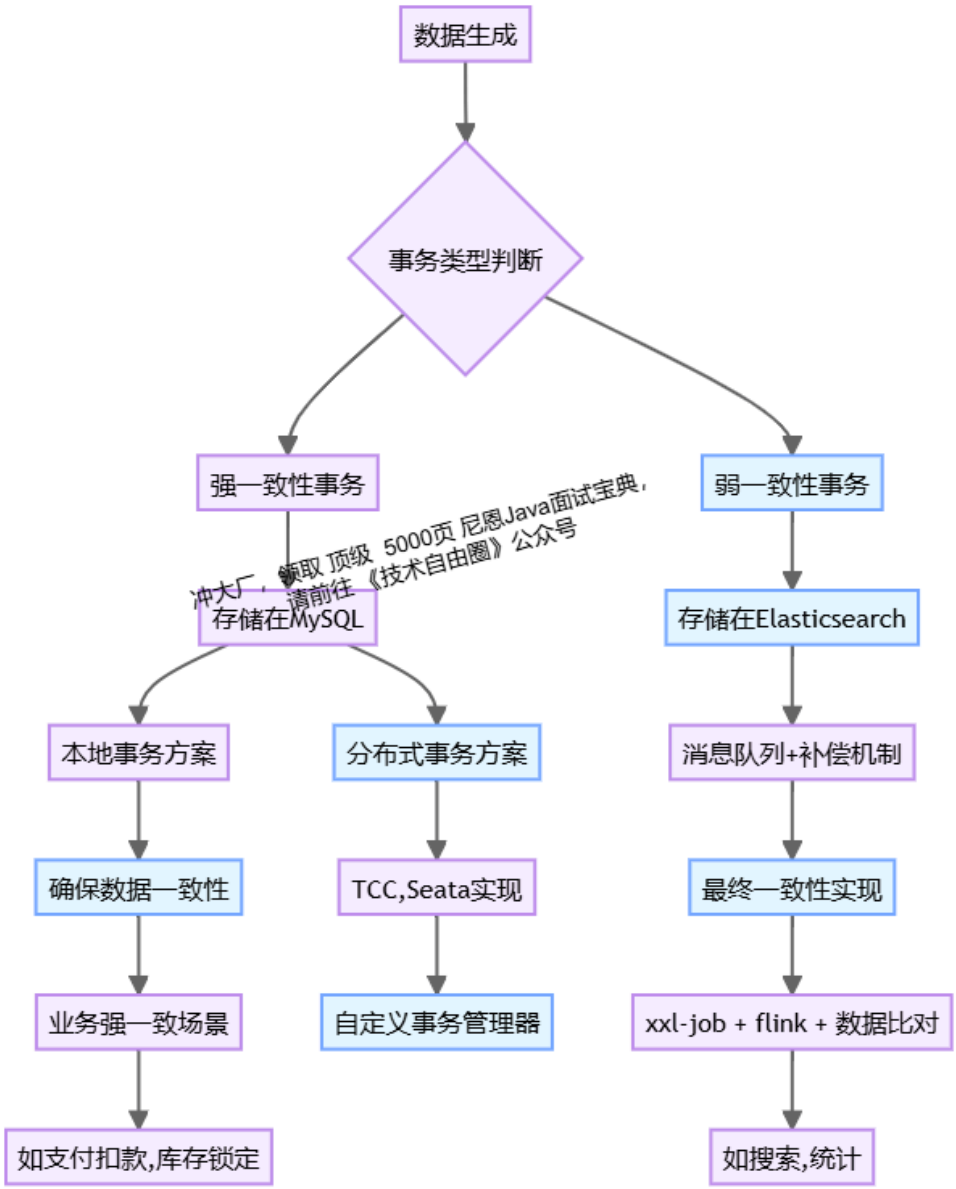
“订单支付成功后，如何保证ES中的订单状态最终一致？

如果消息队列积压导致数据延 ，如何设计补偿机制？”

在什么场景下需要强一致性？什么场景下可以接受弱一致性？

如何设计弱一致性事务的补偿机制？

第二个维度 实操：百亿级订单表 强弱解耦 架构设计 实操



强弱解耦 与边界划分

强一致性域（核心交易链路）：

- 覆盖场景：订单创建、支付状态变更、库存扣减（QPS 5w+，响应时间≤50ms）
- 数据特征：ACID事务强需求，数据变更实时性要求高（RPO=0，RTO<1s）

弱一致性域（辅助业务链路）：

- 覆盖场景：运营看板、商品类目分析、用户行为统计（QPS 1w+，允许≤5分钟延 ）
- 数据特征：AP查询为主，容忍最终一致性（数据差异率<0.01%）

解耦规则：

- 订单支付成功后，核心字段（订单ID/金额/状态）在MySQL立即生效
- 扩展字段（商品类目标签、运营跟单备注）通过异步通道写入ES

强一致性域设计（MySQL集群）

```
// 分库分表策略（256分片）
CREATE TABLE orders_${shard} (
  order_id BIGINT PRIMARY KEY COMMENT '雪花算法生成',
  user_id BIGINT NOT NULL,
  operator_id INT NOT NULL,
  product_category VARCHAR(32),
  amount DECIMAL(12,2),
  status TINYINT COMMENT '0-待支付 1-已支付',
  shard_key VARCHAR(64) GENERATED ALWAYS AS (CONCAT(user_id%256)) STORED
) ENGINE=InnoDB
PARTITION BY KEY(shard_key)
PARTITIONS 256;
-- 复合索引优化
ALTER TABLE orders_${shard}
  ADD INDEX idx_operator_category (operator_id, product_category),
  ADD INDEX idx_user_order (user_id, order_id);
```

分布式事务保障：

- 支付操作采用TCC模式（Try阶段预扣款，Confirm阶段实际扣款，Cancel阶段回滚）
- 通过Seata AT模式实现跨库事务，事务成功率≥99.99%

弱一致性域设计优化方案（ES集群 + 消息队列）

1 监听MySQL Canal binlog

```
@CanalEventListener
public class OrderEventListener {
    @ListenPoint(table = "orders_*")
    public void onEvent(CanalEntry.EventType eventType, RowData rowData) {
        // 核心字段变更直接响应业务
        if (eventType == CanalEntry.EventType.UPDATE) {
            updateCache(rowData.getBefore(), rowData.getAfter());
        }
        // 扩展字段变更发往Kafka
        KafkaTemplate.send("order_extend",
            buildESDocument(rowData.getAfter()));
    }
}
```

2 binlog 发送 Kafka ：

```
// Kafka Producer配置优化（批量发送）
Properties props = new Properties();
props.put("bootstrap.servers", "kafka1:9092");
props.put("batch.size", 65536);      // 64KB批次
props.put("linger.ms", 20);          // 最大等待20ms
props.put("compression.type", "zstd"); // 压缩率提升30%
props.put("acks", "1");              // 平衡吞吐与可靠性
// 分区策略优化（保证相同订单顺序性）
public class OrderPartitioner implements Partitioner {
    @Override
    public int partition(String topic, Object key, byte[] keyBytes,
                         Object value, byte[] valueBytes, Cluster cluster) {
        Long orderId = (Long) key;
        return Math.abs(orderId.hashCode()) % 128; // 128分区
    }
}
```

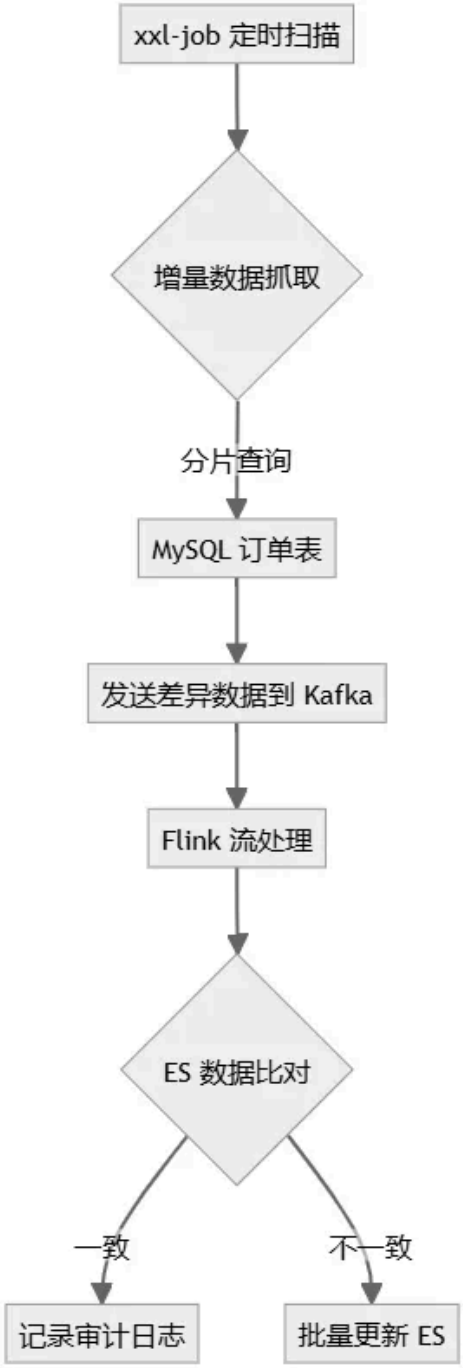
3 消费者优化（Kafka → ES）：

```
// 高性能消费者组（多线程+流式处理）
@Bean
public ConcurrentKafkaListenerContainerFactory<String, Order>
        kafkaListenerContainerFactory() {
    ConcurrentKafkaListenerContainerFactory<String, Order> factory =
        new ConcurrentKafkaListenerContainerFactory<>();
    factory.setConcurrency(16); // 16线程并发消费
    factory.getContainerProperties().setPollTimeout(3000);
    return factory;
}
// 消费者异常处理（死信队列）
@KafkaListener(topics = "orders", errorHandler = "kafkaErrorHandler")
public void consume(Order order) {
    esClient.index(order);
}

@Bean
public KafkaListenerErrorHandler kafkaErrorHandler() {
    return (message, exception) -> {
        // 失败消息写入死信队列
        kafkaTemplate.send("orders_dlq", message);
        return null;
    };
}
```

最终一致性：基于 xxl-job + Flink 的优化补偿机制设计

1. 补偿流程架构设计



2. xxl-job 增量扫描优化

```
// 分片策略：按时间范围+用户ID哈希分片
@XxlJob("orderCompensateJob")
public void execute() {
    int shardTotal = 32; // 分片数=集群节点数×4
    for (int shardIndex=0; shardIndex<shardTotal; shardIndex++) {
        // 计算时间窗口（近72小时数据）
        String startTime = DateUtil.offsetHour(new Date(), -72);
        String endTime = DateUtil.format(new Date(), "yyyy-MM-dd HH:mm:ss");
        // 分页查询（每页5000条）
        int pageSize = 5000;
        for (int pageNum=1; ; pageNum++) {
            List<Order> orders = orderDao.scanCompensateData(
                startTime, endTime, shardTotal, shardIndex, pageNum, pageSize
```

```
            );
            if (orders.isEmpty()) break;

            // 发送至Kafka（同步线程池控制并发）
            kafkaTemplate.send("order_compensate", orders);
        }
    }
}
```

SQL扫描优化：

```
 SELECT * FROM orders
WHERE update_time BETWEEN #{startTime} AND #{endTime}
  AND user_id % #{shardTotal} = #{shardIndex}
ORDER BY update_time
LIMIT #{pageSize} OFFSET #{pageSize*(pageNum-1)}
```

3. Flink 流式比对引擎

```
// 创建执行环境（开启Checkpoint）
StreamExecutionEnvironment env = StreamExecutionEnvironment.getExecutionEnvironm
env.enableCheckpointing(30000); // 30秒一次Checkpoint
// 消费Kafka补偿数据
DataStream<Order> orderStream = env.addSource(
    new FlinkKafkaConsumer<>("order_compensate", new OrderDeserializer(), props)
);

// 批量比对ES（窗口优化）
orderStream
    .keyBy(Order::getOrderId)
    .process(new KeyedProcessFunction<Long, Order, OrderDiff>() {
        private transient OrderESQueryService esQueryService;

        @Override
        public void open(Configuration parameters) {
            esQueryService = new OrderESQueryService(); // ES客户端池化
        }

        @Override
        public void processElement(Order mysqlOrder, Context ctx, Collector<Ord
            // 批量查询ES（100条/批次）
            List<Order> esOrders = esQueryService.batchGetFromES(
                Collections.singletonList(mysqlOrder.getOrderId())
            );

            // 差异比对
            if (!compareOrders(mysqlOrder, esOrders.get(0))) {
                out.collect(new OrderDiff(mysqlOrder, esOrders.get(0)));
            }
```

```
        }
    })
    .addSink(new EsUpdateSink()); // 批量写入ES
```

比对逻辑优化：

```
boolean compareOrders(Order mysql, Order es) {
    // 核心字段比对（状态/金额等）
    if (mysql.getStatus() != es.getStatus()) return false;
    if (mysql.getAmount().compareTo(es.getAmount()) != 0) return false;
    // 扩展字段比对（JSON结构对比）
    return JsonDiff.compare(mysql.getExtendInfo(), es.getExtendInfo()).isEmpty()
}
```

4. 关键性能优化点

分片扫描加速

- 将72小时数据按`user_id%32`分片，利用多线程并行扫描
- 使用覆盖索引加速查询：`INDEX(update_time, user_id)`

ES批量查询优化

```
// 使用ES _mget API 批量获取
MultiGetRequest request = new MultiGetRequest();
orderIds.forEach(id -> request.add("orders", id));
MultiGetResponse response = esClient.mget(request, RequestOptions.DEFAULT);
```

差异更新合并写入

在Flink Sink中积累1000条差异记录或每10秒触发一次批量更新

```
public class EsUpdateSink extends RichSinkFunction<OrderDiff> {
    private List<UpdateRequest> buffer = new ArrayList<>(1000);
    @Override
    public void invoke(OrderDiff diff, Context context) {
        buffer.add(buildUpdateRequest(diff));
        if (buffer.size() >= 1000) {
            esClient.bulk(buffer); // 批量提交
            buffer.clear();
        }
```

```
    }

    @Override
    public void close() {
        if (!buffer.isEmpty()) esClient.bulk(buffer);
    }
}
```

80分（高手级）答案

尼恩提示，讲完 强弱解耦 架构 ， 可以得到 80分了。

但是要直接拿到大厂offer，或者 offer 直提，需要 120分答案。

尼恩带大家继续，挺进 120分，让面试官 口水直流。

第三维度：冗余双写 理论架构

当需要多维度查询时，单一数据存储结构会面临哪些问题？如何解决？

冗余双写 支持多维度的查询需求 ， 解决跨分片查询痛点。

如“查用户所有订单”， 有需要 解决 “查商户 所有订单”

异构数据冗余设计：

通过冗余双写将数据写入多个数据库（如用户订单和商家订单），支持多维度的查询需求。

- 主库（MySQL）：按用户 ID 分片（如64库×256表）
- 冗余库（ES/HBase）：按商家ID 分片（如64库×256表）

双写一致性保障：

- 同步双写：事务内同步写入mysql 用户订单 表 和mysql 商户订单 表（强一致，性能低）
- 异步双写：先写mysql 用户订单 表 ，再用柔性事务，写 mysql 商户订单 表

核心重点：

- 如何设计冗余双写方案，支持多维度查询需求。
- 冗余双写的性能优化与数据一致性保障。

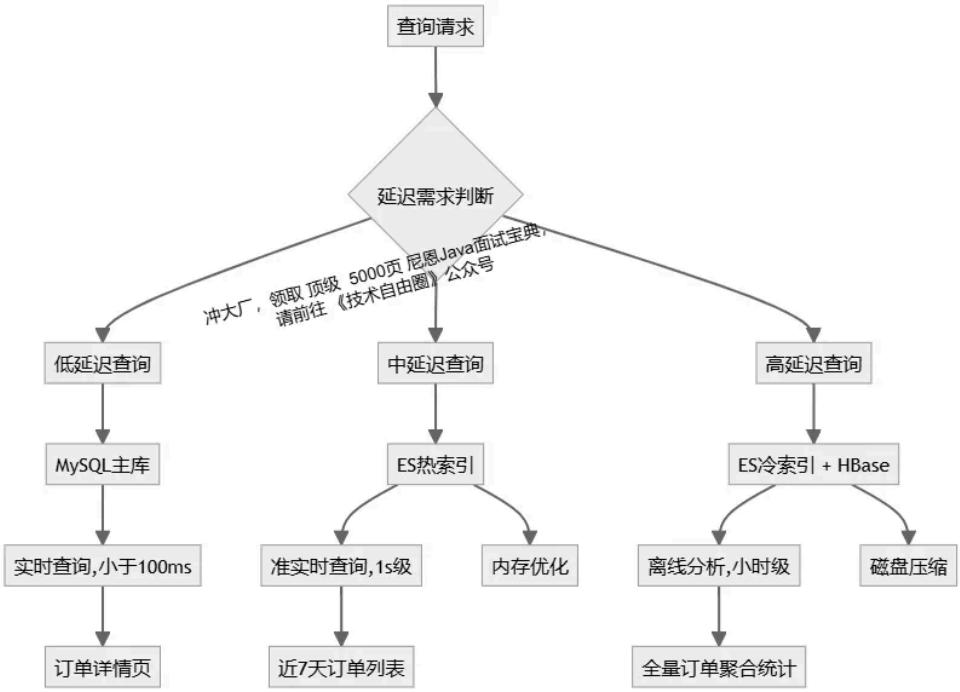
第三维度：冗余双写 理论架构 实操落地

内容太多，请参考尼恩下面的架构文章

字节面试：百亿级存储，怎么设计？只是分库分表？

第四维度：延迟分级 理论架构

- 查询延 的分级：低延 、中延 （秒级延 ）、高延 （小时级延 ）等不同级别及其适用场景。
- 不同延 级别查询的实现方式：如MySQL用于低延 查询，ES用于秒级延 查询等。



对查询操作，进行延迟分级设计：

- 低延　， 实时查询（<100ms）：MySQL主库（订单详情页）
- 中延　，准实时查询（1s级）：ES热索引（近7天订单列表）
- 高延　，离线分析（小时级）：ES 冷索引 + HBase（全量订单聚合统计）

ES 索引分级策略：

- ES热索引：内存优化（ES的`refresh_interval=1s`）
- ES冷索引：磁盘压缩（ES的`forcemerge+codec=best_compression`）

相关的问题：

如果用户要求同时查询实时订单和历史订单的聚合结果，如何设计混合查询方案？

如何避免高并发下多级存储的雪崩效应？"

第四维度：延迟分级　实操落地

不同的查询，走不同的链路就可以了。　具体请参考尼恩的 100亿级存储架构视频。

120分殿堂答案（塔尖级）：

尼恩提示，到了这里， 可以得到 120分了。

遇到问题，找老架构师取经

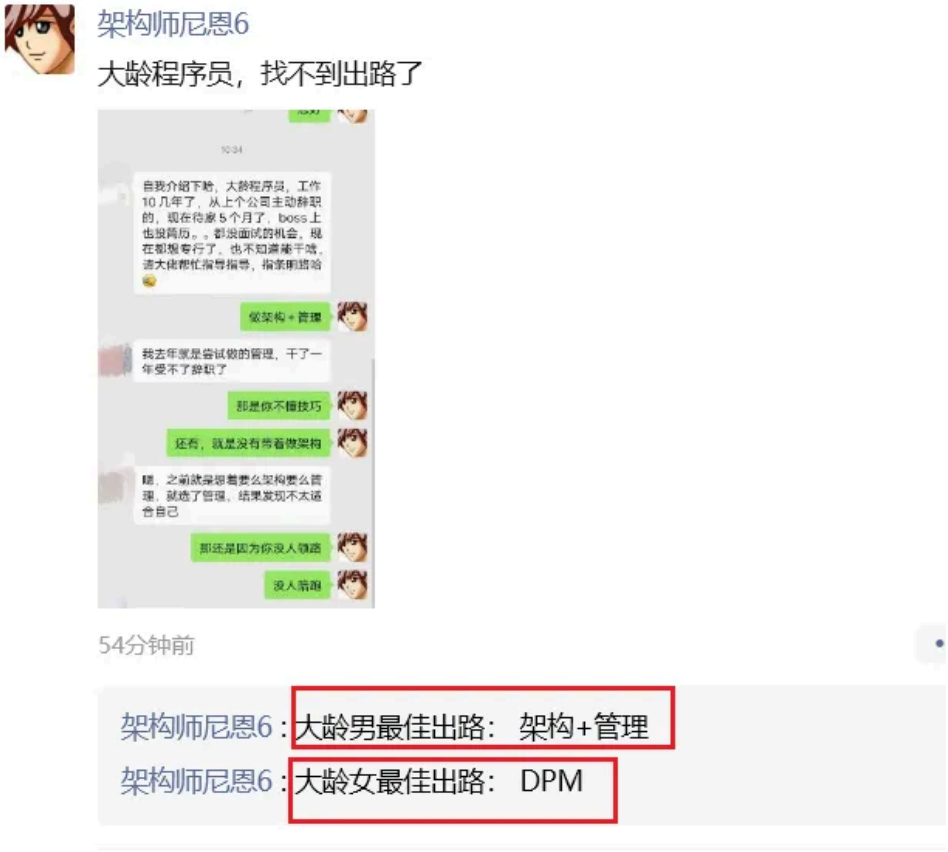
以上的内容，如果大家能对答如流，如数家珍，基本上 面试官会被你 震惊到、吸引到。

最终，**让面试官爱到 “不能自已、口水直流”**。offer， 也就来了。

在面试之前，建议大家系统化的刷一波 5000页《尼恩Java面试宝典》V174，在刷题过程中，如果有啥问题，大家可以来 找 40岁老架构师尼恩交流。

另外，如果没有面试机会，可以找尼恩来帮扶、领路。

- 大龄男的最佳出路是 架构+ 管理
- 大龄女的最佳出路是 DPM，



女程序员如何成为DPM，请参见：

DPM （双栖）陪跑，助力小白一步登天，升格 产品经理+研发经理

领跑模式，尼恩已经指导了大量的就业困难的小伙伴上岸。

尼恩指导了大量的小伙伴上岸，前段时间，**刚指导一个40岁+被裁小伙伴，拿到了一个年薪100W的offer。**

狠狠卷，实现 “offer自由” 很容易的， 前段时间一个武汉的跟着尼恩卷了2年的小伙伴， 在极度严寒/痛苦被裁的环境下， offer拿到手软， 实现真正的 “offer自由” 。

空窗1年-空窗2年，彻底绝望投递，走投无路，如何 起死回生 ？

失业1年多，负债20W多万，**彻底绝望，抑郁了**。7年经验小伙，找尼恩帮助后，跳槽3次 入国企 年薪40W offer ，逆天改命了

被裁2年，**天快塌了，家都要散了**，42岁急救1个月上岸，成开发经理offer，起死回生

空窗8月：中厂大龄34岁，被裁8月收一大厂offer， 年薪65W，转架构后逆天改命!

空窗2年：42岁被裁2年，天快塌了，急救1个月，拿到开发经理offer，起死回生

空窗半年：35岁被裁6个月， 职业绝望，转架构急救上岸，DDD和3高项目太重要了

空窗1.5年：失业15个月，学习40天拿offer， 绝境翻盘，如何实现？

100W-200W P8级 的天价年薪 大逆袭，如何实现 ？

100W案例，100W年薪的底层逻辑是什么？ 如何实现年薪百万？ 如何远离 中年危机？

100W案例2：40岁小伙被裁6个月，猛卷3月拿100W年薪 ，秘诀：首席架构/总架构

环境太糟，如何升 P8级，年入100W？