# 一、自我介绍(3-5 分钟)

**基本信息：** 姓名、年龄、籍贯、毕业院校、应聘什么岗位、曾经主要工作经历和时间节点

**个人优势：** 做成过什么工作、擅长什么技术、学习提高的经历（有助于工作岗位的）

**其他：**爱好，性格，结束语&感谢

**注意：**

1. 考察面试者口述自我介绍的内容和简历中的基本情况是否相冲突?以此来判断简历的真实性。

2. 考察面试者基本的逻辑思维能力、语言表达能力，总结概括能力。

3. 考察面试者是否聚焦，以及现场的感知能力与把控能力。

4、学历不具备优势的在自我介绍中不建议着重体现，一带而过即可。

**自我介绍参考：**

您好，我是来面试咱们公司 java 高级工程师的，我叫 XXX。我从事 java 行业已经两年多了。曾近做过三个项目，最近做的是宝亮网智的服务管理系统 和 心连心社交平台等。在这几年的项目开发中，我掌握了并发、微服务、多线程等技术的应用，以及我能够熟练使用 Oracle,MySQL 等关系型数据库以及现在比较获得 NoSQL（ MongoDB）非关系型数据库。我还掌握了一些有关于项目中的优化问题，如：缓存，SQL 优化，性能优化，业务优化，Tomcat 调优，负载均衡等。

# 二、人事话术

## 1、为什么离职？

**解析：**

【个人原因】正面 ： 家里有事导致、离开企业所在城市 、挑战自我。

回答范例 1：是想要来北京寻找更好的发展，提升自己的技术，寻找更好的平台。之前在天津上学 ,毕业后就留在那边啦 ,但是后来发现天津跟北京的区别还是很大的 ,之前就是过的比较安逸，工作上也很难有很大的突破，所以就想重新寻找更大一点的平台，跳槽来北京发展了。

回答范例 2：家里近期有一些事情需要处理，之前已经请假去处理了几天，但是未处理完毕，后期请假时间不确定，有可能时间会比较久，也担心时间太久会影响公司的项目进度，毕竟公司编制也是比较有限的，综合跟项目经理商量了一下最终决定离职，但是目前已经处理完毕，所以重新找工作。

【公司原因】正面 ：部门/公司架构调整（公司地点变化）。

回答范例：我离职是因为这家公司经营不善，我在公司工作了 2、3 年，有较深的感情，从去年始，由于市场形势突变，公司的局面急转直下，到眼下这一步我觉得很遗憾，但还要面对现实，重新寻找能发挥我能力的舞台。

【反面原因】：避免说以前的公司没有发展空间、跟同事关系处不好发生矛盾，工作中受委屈

或者对公司有很多怨言.

## 2、如何谈薪资？

**解析：**

-薪资定位：预期薪资涨幅 30%（相对于上一家），是一个数值而不是范围。

-压薪资时候，询问福利待遇（五险一金、年终奖、试用期是否有打折），不轻易降薪。

例如：A 公司　12k　　福利 500 元，　公积金 300 元；总计：12.8k

　　B 公司　10k　　福利 1200 元，公积金 2000 元；总计：13.2k

-考虑企业业务领域，实际面试情况，公司具体情况

例如：金融，电商，医疗行业相对工资会高一点

公司装修环境，地理位置，整体感受也可以衡量

公司技术平台也是很重要的考核项，一个好的平台很重要

公司的项目选型和技术栈跟自己是否匹配

-如何收尾？切记一定要把 offer 要过来

例如：面试成功达到预期，那没什么问题的话可以把 offer 发我邮箱吗？我之后准备一下入职资料，尽量早点来入职。

例如：面试没有达到预期，那如果我表现的好，进入公司之后可以给我加薪吗？（面试官肯定回答可以），然后接着说：“那我觉得是金子到哪里都会发光的，进入公司之后我一定会好好干，咱们公司 xx 地方我觉得我很满意 ,我也很想加入贵公司 ,那看什么时间方便 ,给我下一下 offer，我来办理入职呢？

所有的都等拿到 offer 之后在考虑入职与否，offer 在手，再考虑。

## 3、如何看待加班？（理解，提高工作效率）

**解析：**

实际上好多公司问这个问题，并不证明一定要加班，只是想测试你是否愿意为公司奉献。

回答范例：我做咱们 IT 这个行业也好几年了，有时候赶项目或者项目上线会涉及加班，这个都能理解，工作需要我会义不容辞加班，我现在也单身，可以全身心的投入工作。但我也会提高工作效率，推进项目周期进度，减少不必要的加班。

## 4、在之前的工作中最有意义的事情是什么？

**解析：**

一般平时会很难记录住这些事情，如何从记忆里面把这些事找出来

让你痛苦不已的，让你感激涕零的，让你欣喜若狂的这些形容词的背后有没有你对应的事呢？

例如：

1、你干的项目，干了三四个月了终于有一天上线了，你会不会很开心啊？

2、你碰到一个 bug 或者一个问题，让你彻夜难眠，这个也算，你从中又学到了什么呢？

3、在你的团队沟通过程中有几件事情是不是让你非常有启发呀，之前没有遇到过，通过跟你团

队或者你老大的沟通你会发现原来也可以这样，也会让你觉得很受用

## 5、为何转行来计算机专业？

**解析：**

-重点并不在于你没有选择原来的专业并从事相关工作，而在于你选择了跨行业到计算机行业来能不能很专业。

第一，你要表达出你对这个行业的认知。

第二 ,你要表达出你对这个行业是如何做到相应的专业的 ,首先要说出来这个行业的基本特点( 这也是你为什么选择这个行业的原因）。

-例如：人才缺口大，发展空间广阔一些，待遇还不错，最重要的是你对技术本身很感兴趣。

进入这个行业我如何让自己变得专业呢？一是因为我的兴趣 ,第二呢在这个行业当中正好也有我的朋友 ,我在学习技术的过程当中朋友也给予了很多技术以及经验上的支持 ,让我的技术得以快速的提升，让我有了从事本行业的一个资本，同时未来呢，我也希望我能在这个行业走的更长更远，我也希望我的能力能给咱们的公司或者团队带来更多的价值。

# 三、知识点（更多技术点可参考黑马面试宝典）

## 1、ThreadLocal 的原理（高薪常问）

ThreadLocal：为共享变量在每个线程中创建一个副本，每个线程都可以访问自己

内部的副本变量。通过 threadlocal 保证线程的安全性。

其实在 ThreadLocal 类中有一个静态内部类 ThreadLocalMap(其类似于 Map)，

用键值对的形式存储每一个线程的变量副本，ThreadLocalMap 中元素的 key 为当前 ThreadLocal 对象，而 value 对应线程的变量副本。

ThreadLocal 本身并不存储值，它只是作为一个 key 保存到 ThreadLocalMap 中，但是这里要注意的是它作为一个 key 用的是弱引用，因为没有强引用链，弱引用在 GC 的时候可能会被回收。这样就会在 ThreadLocalMap 中存在一些 key 为 null 的键值对（ Entry）。因为 key 变成 null 了，我们是没法访问这些 Entry 的，但是这些 Entry 本身是不会被清除的。如果没有手动删除对应 key 就会导致这块内存即不会回收也无法访问，也就是内存泄漏。

使用完 ThreadLocal 之后，记得调用 remove 方法。 在不使用线程池的前提下，即使不调用 remove 方法，线程的"变量副本"也会被 gc 回收，即不会造成内存泄漏的情况。

## 2、同步锁、死锁、乐观锁、悲观锁 （高薪常问）

**同步锁：**

当多个线程同时访问同一个数据时，很容易出现问题。为了避免这种情况出现，我们要保证线程同步互斥，就是指并发执行的多个线程，在同一时间内只允许一个线程访问共享数据。Java 中可以使用 synchronized 关键字来取得一个对象的同步锁。

8 **死锁：**

何为死锁，就是多个线程同时被阻塞，它们中的一个或者全部都在等待某个资源被释放。

**乐观锁：**

总是假设最好的情况，每次去拿数据的时候都认为别人不会修改，所以不会上锁，但是

在更新的时候会判断一下在此期间别人有没有去更新这个数据，可以使用版本号机制和 CAS 算法实现。乐观锁适用于多读的应用类型，这样可以提高吞吐量，像数据库提供的类似于 write_conditio 机制，其实都是提供的乐观锁。在 Java 中 java.util.concurrent.atomic 包下面的原子变量类就是使用了乐观锁的一种实现方式 CAS 实现的。

**悲观锁：**

总是假设最坏的情况，每次去拿数据的时候都认为别人会修改，所以每次在拿数据的时候都会上锁，这样别人想拿这个数据就会阻塞直到它拿到锁（共享资源每次只给一个线程使用，其它线程阻塞，用完后再把资源转让给其它线程）。传统的关系型数据库里边就用到了很多这种锁机制，比如行锁，表锁等，读锁，写锁等，都是在做操作之前先上锁。Java 中 synchronized 和 ReentrantLock 等独占锁就是悲观锁思想的实现。

# 3、反射

在 Java 中的反射机制是指在运行状态中，对于任意一个类都能够知道这个类所有的

属性和方法；并且对于任意一个对象，都能够调用它的任意一个方法；这种动态获取信息

以及动态调用对象方法的功能成为 Java 语言的反射机制。

获取 Class 对象的 3 种方法 ：

调用某个对象的 getClass()方法

```
Person p=new Person();
Class clazz=p.getClass();
```

调用某个类的 class 属性来获取该类对应的 Class 对象

Class clazz=Person.class;

使用 Class 类中的 forName()静态方法(最安全/性能最好)

Class clazz=Class.forName("类的全路径"); (最常用)

# 4、Hashmap 和 hashtable ConcurrentHashMap 区别（高薪常问）

**区别对比一(HashMap 和 HashTable 区别)：**

1、HashMap 是非线程安全的，HashTable 是线程安全的。

2、HashMap 的键和值都允许有 null 值存在，而 HashTable 则不行。

3、因为线程安全的问题，HashMap 效率比 HashTable 的要高。

4、Hashtable 是同步的，而 HashMap 不是。因此，HashMap 更适合于单线程环境，而 Hashtable 适合于多线程环境。一般现在不建议用 HashTable, ①是 HashTable 是遗留类，内部实现很多没优化和冗余。②即使在多线程环境下，现在也有同步的 ConcurrentHashMap 替代，没有必要因为是多线程而用 HashTable。

**区别对比二(HashTable 和 ConcurrentHashMap 区别)：**

HashTable 使用的是 Synchronized 关键字修饰，ConcurrentHashMap 是 JDK1.7 使用了锁分段技术来保证线程安全的。JDK1.8ConcurrentHashMap 取消了 Segment 分段锁，采用 CAS 和 synchronized 来保证并发安全。数据结构跟 HashMap1.8 的结构类似，数组+链表/红黑二叉树。

synchronized 只锁定当前链表或红黑二叉树的首节点，这样只要 hash 不冲突，就不会产生并发，效率又提升 N 倍。

## 5、线程池的分类（高薪常问）

1.

**newCachedThreadPool：**创建一个可进行缓存重复利用的线程池

2.

**newFixedThreadPool：**创建一个可重用固定线程数的线程池，以共享的无界队列方式来运行这些线程，线程池中的线程处于一定的量，可以很好的控制线程的并发量

3.

**newSingleThreadExecutor** ：创建一个使用单个 worker 线程的 Executor ，以无界队列方式来运行该线程。线程池中最多执行一个线程，之后提交的线程将会排在队列中以此执行

**4.**

**newSingleThreadScheduledExecutor：**创建一个单线程执行程序，它可安排在给定延迟后运行命令或者定期执行

**5.**

**newScheduledThreadPool：**创建一个线程池，它可安排在给定延迟后运行命令或者定期的执行

**6.**

**newWorkStealingPool：**创建一个带并行级别的线程池，并行级别决定了

同一时刻最多有多少个线程在执行，如不传并行级别参数，将默认为当前系统的 CPU 个数

## 6、JDK1.8 堆内存结构（高薪常问）

**Young 年轻区（代）: Eden+S0+S1, S0 和 S1 大小相等，新创建的对象都在年轻代**

**Tenured 年老区: 经过年轻代多次垃圾回收存活下来的对象存在年老代中.**

Jdk1.7 和 Jdk1.8 的区别在于, 1.8 将永久代中的对象放到了元数据区，不存永久代这一区域了.

## 7、Hashmap 的底层原理

HashMap 在 JDK1.8 之前的实现方式 **数组+链表,**

但是在 JDK1.8 后对 HashMap 进行了底层优化,改为了由 **数组+链表或者数值+红黑树** 实现,主要的目的是提高查找效率

1. **Jdk8 数组+链表或者数组+红黑树**实现，当链表中的元素超过了 8 个以后， 会将链表转换为红黑树，当红黑树节点 小于 等于 6 时又会退化为链表。

2. 当 new HashMap():底层没有创建数组，首次调用 put()方法示时，底层创建长度为 16 的数组，

jdk8 底层的数组是：Node[],而非 Entry[]，用数组容量大小乘以加载因子得到一个值，一旦数组中存储的元素个数超过该值就会调用 rehash 方法将数组容量增加到原来的两倍，专业术语叫做扩容，在做扩容的时候会生成一个新的数组，原来的所有数据需要重新计算哈希码值重新分配到新的数组，所以扩容的操作非常消耗性能.

默认的负载因子大小为 0.75，数组大小为 16。也就是说，默认情况下，那么当 HashMap 中元素个数超过 16*0.75=12 的时候，就把数组的大小扩展为 2*16=32，即扩大一倍。

3. 在我们 Java 中任何对象都有 hashcode，hash 算法就是通过 hashcode 与自己进行向右位移 16 的异或运算。这样做是为了计算出来的 hash 值足够随机，足够分散，还有产生的数组下标足够随机，

**map.put(k,v)实现原理**

（

1）首先将 k,v 封装到 Node 对象当中（节点）。

（

2）先调用 k 的 hashCode()方法得出哈希值，并通过哈希算法转换成数组的下标。

（

3）下标位置上如果没有任何元素，就把 Node 添加到这个位置上。如果说下标对应的位置上有链表。此时，就会拿着 k 和链表上每个节点的 k 进行 equal。如果所有的 equals 方法返回都是 false，那么这个新的节点将被添加到链表的末尾。如其中有一个 equals 返回了 true，那么这个节点的 value 将会被覆盖。

**map.get(k)实现原理**

(1)、先调用 k 的 hashCode()方法得出哈希值，并通过哈希算法转换成数组的下标。

(2)、在通过数组下标快速定位到某个位置上。重点理解如果这个位置上什么都没有 则返回 null。如果这个位置上有单向链表，那么它就会拿着参数 K 和单向链表上的每一个节点的 K 进行 equals，如果所有 equals 方法都返回 false，则 get 方法返回 null。如果其中一个节点的 K 和参数 K 进行 equals 返回 true，那么此时该节点的 value 就是我们要找的 value

了，get 方法最终返回这个要找的 value。

4. **Hash 冲突**

不同的对象算出来的数组下标是相同的这样就会产生 hash 冲突，当单线链表达到一定长度后效率会非常低。

5. 在链表长度大于 8 的时候，将链表就会变成红黑树，提高查询的效率。

# 8、存储引擎

**1.MyISAM 存储引擎**

主要特点：

MySQL5.5 版本之前的默认存储引擎

支持表级锁（表级锁是 MySQL 中锁定粒度最大的一种锁，表示对当前操作的整张表加锁）；

不支持事务，外键。

适用场景：对事务的完整性没有要求，或以 select、insert 为主的应用基本都可以选用 MYISAM。在 Web、数据仓库中应用广泛。

特点：

1、不支持事务、外键

2、每个 myisam 在磁盘上存储为 3 个文件，文件名和表名相同，扩展名分别是

.frm

-------存储表定义

.MYD --------MYData，存储数据

.MYI

--------MYIndex，存储索引

### 2.InnoDB 存储引擎

主要特点：

MySQL5.5 版本之后的默认存储引擎；

支持事务；

支持行级锁（行级锁是 Mysql 中锁定粒度最细的一种锁，表示只针对当前操作的行进行加锁）；

支持聚集索引方式存储数据。

## 9、如何避免索引失效

**(1) 范围查询，右边的列不能使用索引，否则右边的索引也会失效.**

索引生效案例

select * from tb_seller where name = "小米科技" and status = "1" and address = "北京市";

select * from tb_seller where name = "小米科技" and status >= "1" and address = "北京市";

索引失效案例

select * from tb_seller where name = "小米科技" and status > "1" and address = "北京市";

address 索引失效，因为 status 是大于号，范围查询.

**(2) 不要在索引上使用运算，否则索引也会失效.**

比如在索引上使用切割函数，就会使索引失效.

select * from tb_seller where substring(name, 3, 2) = "科技";

39**(3) 字符串不加引号，造成索引失效.**

如果索引列是字符串类型的整数，条件查询的时候不加引号会造成索引失效. Mysql 内置的优化会有隐式转换.

### 索引失效案例

select * from tb_seller where name = "小米科技" and status = 1

**(4) 尽量使用覆盖索引，避免 select *，这样能提高查询效率.**

如果索引列完全包含查询列，那么查询的时候把要查的列写出来，不使用 select *

select sellerid, name, status from tb_seller where name = "小米科技" and staus = "1" and address = "西安市";

**(5) or 关键字连接**

用 or 分割开的条件，如果 or 前面的列有索引, or 后面的列没有索引，那么查询的时候前后索引都会失效

如果一定要用 or 查询，可以考虑下 or 连接的条件列都加索引，这样就不会失效了.

# 10、SQL 语句调优

根据业务场景建立复合索引只查询业务需要的字段，如果这些字段被索引覆盖，将极大的提高查询效率.

多表连接的字段上需要建立索引，这样可以极大提高表连接的效率.

where 条件字段上需要建立索引，但 Where 条件上不要使用运算函数，以免索引失效.

排序字段上，因为排序效率低，添加索引能提高查询效率.

优化 insert 语句：批量列插入数据要比单个列插入数据效率高.

优化 order by 语句：在使用 order by 语句时，不要使用 select *, select 后面要查有索引的列，如果一条 sql 语句中对多个列进行排序，在业务允许情况下，尽量同时用升序或同时用降序.

优化 group by 语句：在我们对某一个字段进行分组的时候, Mysql 默认就进行了排序，但是排序并不是我们业务所需的，额外的排序会降低效率. 所以在用的时候可以禁止排序，使用 order by null 禁用.

select age, count(*) from emp group by age order by null

尽量避免子查询，可以将子查询优化为 join 多表连接查询

# 11、Spring 的事务传播行为

spring 事务的传播行为说的是，当多个事务同时存在的时候，spring 如何处理这些事务的行为。

**备注(方便记忆): propagation 传播**

**require 必须的/suppor 支持/mandatory 强制托管/requires-new 需要新建/not -supported 不支持/never 从不/nested 嵌套的**

① PROPAGATION_REQUIRED：如果当前没有事务，就创建一个新事务，如果当前存在事务，就加入该事务，该设置是最常用的设置。

② PROPAGATION_SUPPORTS：支持当前事务，如果当前存在事务，就加入该事务，如果当前不存在事务，就以非事务执行。

③ PROPAGATION_MANDATORY：支持当前事务，如果当前存在事务，就加入该事务，如果当前不存在事务，就抛出异常。

④ PROPAGATION_REQUIRES_NEW：创建新事务，无论当前存不存在事务，都创建新事务。

⑤ PROPAGATION_NOT_SUPPORTED：以非事务方式执行操作，如果当前存在事务，就把当前事务挂起。

⑥ PROPAGATION_NEVER：以非事务方式执行，如果当前存在事务，则抛出异常。

⑦ PROPAGATION_NESTED：如果当前存在事务，则在嵌套事务内执行。如果当前没有事务，则按 REQUIRED 属性执行。

# 12、 RocketMQ 如何保证消息不丢失

首先在如下三个部分都可能会出现丢失消息的情况：

Producer 端

Broker 端

Consumer 端

### 12.1、Producer 端如何保证消息不丢失

采取 send()同步发消息，发送结果是同步感知的。

发送失败后可以重试，设置重试次数。默认 3 次。

producer.setRetryTimesWhenSendFailed(10);

集群部署，比如发送失败了的原因可能是当前 Broker 宕机了，重试的时候会发送到其他 Broker 上。

**12.2、Broker 端如何保证消息不丢失**

修改刷盘策略为同步刷盘。默认情况下是异步刷盘的。

flushDiskType = SYNC_FLUSH

集群部署，主从模式，高可用。

**12.3 Consumer 端如何保证消息不丢失**

完全消费正常后在进行手动 ack 确认。

# 13、redis 的持久化

Redis 提供了两种持久化的方式，分别是 RDB（Redis DataBase）和 AOF（

Append

Only File）。

RDB，简而言之，就是在不同的时间点，将 redis 存储的数据生成快照并存储到磁盘

等介质上。

AOF，则是换了一个角度来实现持久化，那就是将 redis 执行过的所有写指令记录下

来，在下次 redis 重新启动时，只要把这些写指令从前到后再重复执行一遍，就可以实现

数据恢复了。

RDB 和 AOF 两种方式也可以同时使用，在这种情况下，如果 redis 重启的话，则会

优先采用 AOF 方式来进行数据恢复，这是因为 AOF 方式的数据恢复完整度更高。

## 14、SpringCloud 有哪些核心组件?

・Eureka: 注册中心, 服务注册和发现

・Ribbon: 负载均衡, 实现服务调用的负载均衡

・Hystrix: 熔断器

・Feign: 远程调用

・Gateway: 网关

・Spring Cloud Config: 配置中心

## 15、SpringBoot 常用注解

@SpringBootApplication: 它封装了核心的@SpringBootConfiguration +@EnableAutoConfiguration +@ComponentScan 这三个类,大大节省了程序员配置时间,这就是 SpringBoot 的核心设计思想.

@EnableScheduling 是通过@Import 将 Spring 调度框架相关的 bean 定义都加载到 IoC 容器

@MapperScan:spring-boot 支持 mybatis 组件的一个注解，通过此注解指定 mybatis 接口类的路径，即可完成对 mybatis 接口的扫描

@RestController 是 @Controller 和 @ResponseBody 的结合，一个类被加上 @RestController 注解,数据接口中就不再需要添加@ResponseBody,更加简洁。

@RequestMapping,我们都需要明确请求的路径.

@GetMappping,@PostMapping, @PutMapping, @DeleteMapping 结合

@RequestMapping 使用，是 Rest 风格的，指定更明确的子路径.

@PathVariable：路径变量注解，用{}来定义 url 部分的变量名.

@Service 这个注解用来标记业务层的组件，我们会将业务逻辑处理的类都会加上这个注解交给 spring 容器。事务的切面也会配置在这一层。当让 这个注解不是一定要用。有个泛指组件的注解，当我们不能确定具体作用的时候 可以用泛指组件的注解托付给 spring 容器

・@Component 和 spring 的注解功能一样，注入到 IOC 容器中.

・@ControllerAdvice 和 @ExceptionHandler 配合完成统一异常拦截处理.

# 四、项目（新闻资讯、电商、社交）

## 资讯项目

在技术选型时，按照黑马头条 TPS 为 1000 来计算，一个月数据量就是 3 千万，3 年可以达到 10 个亿，这是当时与产品商量出来的一个提前预估的数据量级，由于 mysql 单表可以存储大约 5 千万数据，需要 20 个表就可以满足需求，操作简单，学习成本低，所以我们选定了 mysql 做了其中一种持久化方案。

并且针对用户进行了合理的分库分表，来解决单表存储时数据量级过大的问题，按照用户行为类型做的分库，按照用户 ID 做的分表，其中对于关注来说，复杂一点，关注后就成为对方的粉丝，一条数据主要存储两份，一个是关注表，一个是粉丝表，关注表以粉丝 ID 为主体，同时记录被关注作者的信息；粉丝表以作者用户 ID 为主体，同时记录粉丝的信息，说白了就是一件事儿，

一个表存储的是主动语句，一个表存储的是被动语句，因为我们的查询场景主要是根据用户 ID 进行数据的查询，这样避免数据分散到多个表，避免一次查询多个表，保证性能，同时分为两个表的好处是针对不同的主体方便查询，避免了用一张表时查询量过大的问题。

另一种持久化方案是使用 mongodb，mongodb 可以轻松解决大数据量的问题，效率很高，并且 mongodb 面向 document 可以实现可扩展性并且支持丰富的数据类型和数据表达，算是对 MySQL 的一种辅助。最终我们结合使用两种持久化方案，使用 mysql 存储一份数据，用户页面的回显，例如关注或粉丝列表查询，使用 mongodb 记录用户行为，相当于流水的概念，用于后续进行大数据统计计算以及数据统计。

由于频繁的单条往 MySQL 与 mongodb 中插入删除数据比较耗性能，并且流入 MySQL 中的业务行为数据我们可以不允许数据丢失，所以引用 rocketmq 消息中间件来批量处理用户行为数据，进行异步解耦以及批量处理，topic 是按照库表加上用户的 ID 取模做的 topic，但是针对一些埋点的点击流行为数据我们选用 kafka 作为消息中间件，解决方案架构也是类似的，知识 MQ 替换成 kafka。

用户行为数据业务层应用接收到后，发送到 rocketMQ，下游我们实现了一个专门做持久化的服务，批量的把数据存储到 DB，双写到 MySQL 与 mongodb 中，用户查询走的是 MySQL，其中代码中使用了多数据源的方式，没有其他的中间件，根据业务场景，是在代码中做的路由，根据查询用户的 ID 做 hash 取模，找到对于的库表，用户行为服务采用 springboot 和 springcloud 搭建，并引入 nacos、feign、gateway 等组件，jdk 使用 1.8。

# 自动零售项目

## 1.立可得 1.0 到 2.0 升级哪些功能，为什么要做这些升级？（从商业模式、技术选型等方面进行阐述）

商业模式升级

（1）取消了售货机和人员的一对一绑定关系，添加了“运营区域”的概念，售货机和人员都归属与区域，这样有利于经营的规模化发展。

（2）强化了工单管理，新增智能派单，极大了缩减了企业的运营成本。

（3）对合作商管理更加规范化、系统化、新增了合作商后台，查看分账数据更清晰、更透明。

（4）C 端由原来的 h5 升级为小程序，新增了附近售货机搜索功能。

（5）运营后台新增智能排货，后台通过对同商圈的近期数据进行分析，给出最可能盈利的 10 个推荐上架商品。

技术选型升级：

（1）采用物联网领域常用的 MQTT 协议 ，使用 EMQ 作为消息中间件

（2）使用 Xxl-job 任务调度 ，分片任务可以将大任务进行切分

（3）logstach 进行数据的同步。

（4）使用 minIO 进行海量图片存储。

## 2.你们是使用 mybatisPlus 哪些功能简化了系统的开发?（你们为什么要使用 MybatisPlus）

（1）使用 IService 接口，基本的增删改查方法都不用写了，提高了开发的效率。

（2）使用自动填充功能可以使很多表都有的字段自动填充数据，我们在项目中每个表都有 create_time 和 update_time 字段，就是使用自动填充来实现的，在业务代码中并不需要对这两个属性进行处理。

### 3.请简述一下你所开发的工单模块。（需求、用到的技术）

工单是立可得比较核心的模块之一，工单有两大类：运维工单和运营工单。运维工单包括投放工单、撤机工单、维修工单三种；运营工单指的就是补货工单。

手动工单：管理员在后台创建工单，指派给相关人员（运营或运维）。运营或运维人员在运营管理 APP 中可以看到指派给自己的工单，点击接收或取消。当完成工单时运营或运维人员在运营管理 APP 点击完成。

自动工单：维修和补货工单可以由系统自动创建。当售货机发生故障时，会发送故障状态信息到服务端，服务端会自动创建维修工单，被指派人的选择算法是系统根据当前区域当天工单量最少的人。补货工单是每天定时进行的，会扫描所有的缺货的售货机自动计算补货量，创建补货工单。

> 延展问题：工单模块用到哪些技术？

> 这里采用的技术主要有，使用 redis 的 zset 存储每个区域的每个人的当天工单量（因为 zset 自动排序，非常方便），使用 xxl-job 进行每日工单量的初始化，使用 xxl-job 的分片广播，解决补货工单任务时间超长的问题，使用 emq 的共享订阅接收售货机发来的状态报文。

### 4.请简述一下从下单到发货是怎么实现的？

（1）支付成功后微信平台回调订单微服务

（2）订单微服务发送出货通知到 emq，主题为 vm/售货机编号/vendoutReq

（3）售货机订阅出货通知，收到消息后进行发货（调用串口）

（4）售货机出货成功后上报出货结果到 emq ，主题为 server/售货机编号/vendoutResp

（5）订单微服务和售货机微服务以群组方式共享订阅该主题，分别进行业务处理：订单微服务更改订单状态；售货机微服务更改货道库存。

# 电商项目

## 1、微服务网关中如何实现限流？

令牌桶算法是比较常见的限流算法之一，大概描述如下：

1）所有的请求在处理之前都需要拿到一个可用的令牌才会被处理；

2）根据限流大小，设置按照一定的速率往桶里添加令牌；

3）桶设置最大的放置令牌限制，当桶满时、新添加的令牌就被丢弃或者拒绝；

4）请求达到后首先要获取令牌桶中的令牌，拿着令牌才可以进行其他的业务逻辑，处理完业务逻辑之后，将令牌直接删

除；

5）令牌桶有最低限额，当桶中的令牌达到最低限额的时候，请求处理完之后将不会删除令牌，以此保证足够的限流

## 2、各个微服务中如何实现用户身份识别的？

1.用户登录成功后，会将令牌信息存入到 cookie 中(一般建议存入到头文件中)

2.用户携带 Cookie(或者 Header)中的令牌访问微服务网关

3.微服务网关先获取头文件中的令牌信息，如果 Header 中没有 Authorization 令牌信息，则取参数中找，参数中如果没

有，则取 Cookie 中找 Authorization，最后将令牌信息封装到 Header 中，并调用其他微服务

4.其他微服务会获取头文件中的 Authorization 令牌信息，然后匹配令牌数据是否能使用公钥解密，如果解密成功说明

用户已登录，解密失败，说明用户未登录