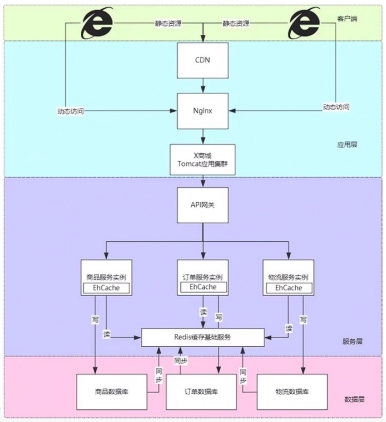
1、多级缓存架构？



客户端缓存（静态资源）：

在浏览器层面主要缓存css，js，字体等静态资源文件

例如：百度的图标，服务器在响应头中设置expires，浏览器在该段时间将图片以文件缓存在本地，客户端再次访问会看到

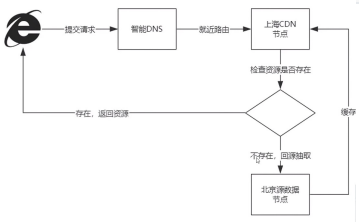
from disk cache的提示，浏览器不再请求服务器，而是直接在本地取图片。

可以很大程度上缓解浏览器重复请求静态资源的带宽损耗，客户端只需要缓存文件就可以。

应用层缓存（静态资源）：

CDN内容分发网络，是互联网静态资源分发的主要技术手段，是广域的互联网应用的基础设施，有效解决了带宽集中占用、资源分发的问题。

CDN的核心技术是智能DNS。



例如：大量上海用户访问北京资源。

但是投入较高，可以租用阿里云、腾讯云、华为云提供的CDN服务，进行租用。

对于企业级应用，工作人员往往分布在指定的工作区域，或相对固定的场所，再加上并发度并不是很高，

也就没必要去部署CDN这样重量级的解决方案，可以在架构部署Nginx，利用其自带的静态资源缓存的能力和压缩的功能，

就可以胜任绝大多数企业级应用场景。Nigix实际是在本地通过一个个目录的方式组织。

服务层缓存（缓存后端接口查出的数据）：

进程内缓存：应用中开辟内存空间，进程在运行中载入这块内存，通过本地内存的低延迟，高吞吐特性来提高程序的访问速度（ehCache）

进程外缓存：分布式缓存（redis），集中缓存。

先进到远、由快到慢的访问缓存。网络是不确定的，同时访问也是有上限的，所以不能上来就找Redis。

要在应用端要设计多级应用缓存，通过进程内缓存和进程外缓存结合分摊压力。先看进程内缓存，再看进程外缓存，都查不到再去db，然后写回缓存。

因为引入多级缓存，要考虑分布式缓存数据一致性问题，此时可以通过MQ主动推送给别的服务实例变更的数据。各实例收到后先删除本地缓存，再重新创建，以保存各实例的数据一致。

2、何时采用多级缓存架构？

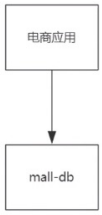
第一种情况，缓存的数据是稳定的。

第二种情况，瞬时可能会产生极高并发的场景。

第三种情况，一定程度上允许数据不一致。

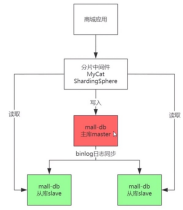
3、MySqL集群模式与应用场景

单库模式：



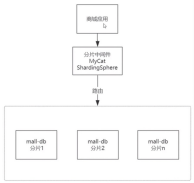
简单粗暴
适合数据量千万以下小型应用
企业网站，创业公司首选
不具备可用性与并发性

读写分离模式：



架构复杂度提升，成本提高
所有节点数据均保持同步
适用于读多写少，单表不过千万的互联网应用
配合MHA中间件方案实现高可用性

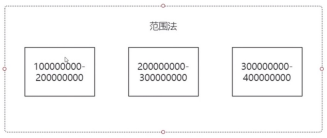
分库分表（分片）模式：



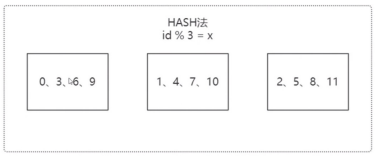
架构复杂度提升，成本提高
每个节点数据是所有数据的子集
适用于十亿级数据总量大型应用
不具备高可用特性

再大用大数据仓库 HBASE

分片算法：

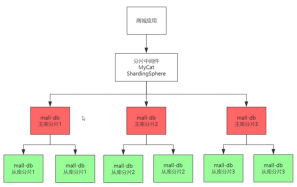


范围法结构简单，扩展容易
适合范围检索
数据分布不均匀，局部负载压力大
适用于流水账应用



Hash法分为取模与一致性Hash
数据分配均衡
节点扩展复杂，数据迁移难度大
建议提前部署足够的节点
适用于预算充足的大型互联网应用

互联网主流MYsql集群架构：



4、为什么要垂直分表？何时进行垂直分表？

innodb在1.0以后引入了压缩页，在跨页上解压缩和压缩的执行效率不高，所以表在设计的时候，要尽可能保证每一页内尽可能的多存储一些行数据。

通过将重要字段剥离，让一页能容纳更多的行，进而缩小数据扫描的页范围，减少跨页检索，检索效率变高。

数据总量很大且字段超过20，且包含了超长的varchar，CLOB，BLOB。

主表中主要存放：查询（skuid，商户id）、排序时需要的字段（品牌编号）、高频访问的小字段（商品名称、价格）。
从表中需要存放：低频访问字段、大字段（图文详情、图片BLOB）

5、为什么在大表严禁使用自增主键？

在分布式下有严重的问题。

1、每个分片能承载多少数据，只能靠猜，可能会产生比较大的浪费，不能在运行期间动态扩展的

2、因为自增主键是在数据库层面生成自增的序列，那么数据库集群必须按主键范围分片（不能hash分片）

3、范围分片会导致“尾部热点”效应。所有的操作都在一个热点分片上集中处理，该分片的数据库压力会非常大，而其他分片的数据库却没什么压力。

6、为什么不能使用uuid作为主键？

1、128位，比起整形、长整形浪费空间。

2、因为无序，作为主键会产生大量的索引重排（可见part1 -51）

7、简介雪花算法SnowFlake？

有序，且每个机器上序列唯一。

第一部分是 1 位的符号位，并没有实际用处，主要为了兼容长整型的格式。

第二部分是 41 位的时间戳用来记录本地的毫秒时间。

第三部分是机器 ID，这里说的机器就是生成 ID 的节点，用 10 位长度给机器做编码，那意味着最大规模可以达到 1024 个节点（2^10）。

最后是 12 位序列，序列的长度直接决定了一个节点 1 毫秒能够产生的 ID 数量，12 位就是 4096（2^12）。

但是雪花算法要注意时间回拨的问题。但是平常基本不会去时间回拨。

8、布隆过滤器原理？

二进制数组+n个hash函数。

在数组中通过hash函数将有效数据所映射的n个位置为1。

检验数据时，如果全部为1，则代表可能含有该数据。如果出现0，则代表肯定不含此数据。使用的时候只需maven引入redission，其中集成了bloomFilter。

问题：商品删除了该怎么办？

布隆过滤器因为某一位二进制可能被多个编号hash映射，因此布隆过滤器无法直接处理删除数据的情况。

方案1、定时异步重建布隆过滤器（例如每4小时）

方案2、计数bloomFilter

9、阿里开发规范解读：为啥禁用外键约束？

表的外键是另一表的主键

1、额外的数据一致性校验查询，会去另一张表查是否唯一等。

2、并发问题：外键约束会在主表启用行级锁（共享锁），主表写入或者更新时（开启独占锁），若独占锁一直不释放，详情表会一直阻塞状态。从外界来看，

所有详情表的写操作，都会因为主表的锁定进入阻塞，可能会导致大量的线程积压，甚至造成系统延迟，崩溃。

3、级联删除：多层级联删除会让数据变的不可控。禁用是为了数据健壮和可追溯。

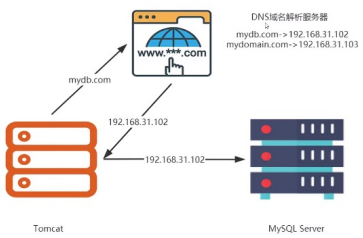
4、数据耦合：数据库层面数据关系产生耦合，数据迁移维护困难，例如：数据多了，将数据源迁移，迁移至大数据库HBASE上，主外键无意义，需要去掉，那么以前程序里未校验的代码可能就出现问题。

10、为什么开发要禁用数据库IP直连？

会造成两个模块之间强耦合。更换IP时费劲。

解决：

1、引入内部DNS：



简单粗暴

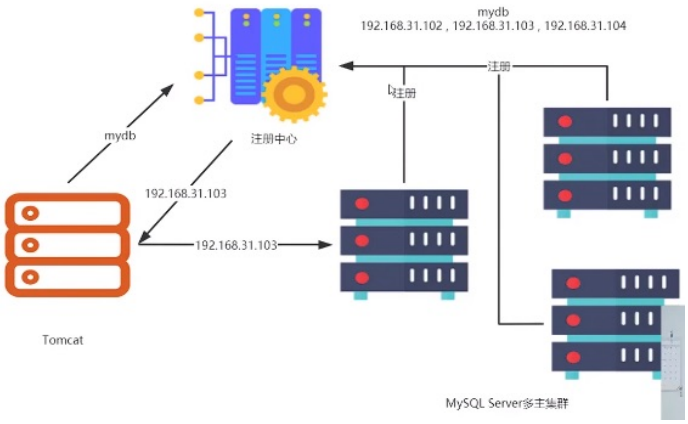
但没有故障发现与转移

多IP只有轮询规则

2、注册中心：

方法2：加入注册中心

Nacos / Eureka / Consul



支持故障发现与故障转移

多种负载均衡规则

架构复杂度增加