海量数据存储、查询和最佳实践

存储系统的技术选型

存储系统的一个特点是繁杂，可供选择的产品非常多。排除那些小众或目前还不太成熟的产品，真正广泛应用于生产系统的、可供选择的存储系统，仍然非常多。每种存储产品都有其擅长之处，有其适用的业务场景，当然也有各自的短板。

如果所选择的存储系统不能很好地匹配业务，那么不仅是开发的时候会很别扭、不顺畅，数据量稍大时，还可能会出现性能严重下降的问题，甚至出现存储慢到卡死以致不可用的情况。反过来，如果选择的是合适的存储系统，就会让你在构建和运营系统的时候感觉顺畅很多。

选择什么样的存储系统来保存数据，对系统的性能和稳定性来说都是非常重要的。要牢记一句话，在实际的业务开发中，我们所做的所有事情总结起来两件：存储和计算。选定了存储系统，往往决定了系统的上限；计算方式决定了能否发挥这个上限以及系统的下限。比如选择了 MySQL 数据库，决定了单机能达到的 QPS 在千级，业务中不适当的表的设计和 SQL 的编写，却只能达到一、两百甚至几十的 QPS。

那么，我们应该如何根据业务的特点，选择合适的存储来构建系统呢？我们把互联网大厂存储方面相关的内容做一个提炼，总结一下，看看我们如何做存储系统的技术选型。

技术选型时应该考虑哪些因素

我们需要根据业务的特点来选择合适的存储系统，是否有一些具体的、可操作的方法，让我们做参照呢？肯定是有的，我们来看看大厂多年的实践中总结出来的经验。

首先需要确定的是系统的类型，是一个在线业务系统，还是一个分析系统？在线业务系统对应的术语是 OLTP (On-line Transaction Processing，在线事务处理)，分析系统对应的术语是 OLAP(Off-line Analytical Processing，离线分析处理)。

由于这两种不同类型的系统对存储系统的要求完全不一样，因此在做存储技术选型的时候，需要先确定到底是哪种系统。

但现实情况是，大部分系统很难明确地归类为在线业务系统或是分析系统。比如，电商系统既包括在线业务部分，又要满足做报表分析的需求，像这种情况又该如何划分呢？

答案是主要取决于系统的规模，如果系统的规模不大，那么我们需要确认的是，系统的主要业务是在线业务还是分析业务，然后以这个主要业务作为划分的依据。

比如，创业公司的电商系统，其主要业务一定是在线交易部分，那就按照在线业务系统来处理。如果系统的规模足够大，那就把系统划分为在线业务系统和

分析系统两个部分，每个部分分别选择合适的存储，当然这样的架构成本会比较高，只有规模足够大才值得这样拆分。

第二个需要考量的维度是数据量，系统需要处理的数据在什么量级？这里的数据量不需要特别精确，能估计到量级就可以了。在估算系统数据量级的时候，需要考虑存量数据和增量数据两个部分，简单地说就是，现在有多少数据,未来还会新增多少数据。在估计系统数据量的时候，不必对未来做过多的预留，一般来说按照未来两年，最多三年来估计就足够了。

不用担心因为预估不足，而无法支撑两年之后业务的问题。一般来讲，很少有新系统在上线之后两三年内，业务没有发生重大改变的。既然系统在不到两三年的时间内就要进行重构，那么存储只需要在重构时，也跟着进行相应的调整即可。

退一步来讲，即使系统在两三年内没有进行重构，那么之前预估的两三年的数据量再撑个三四年问题也不大。因为大部分人在对未来业务和数据量做预估的时候，往往都会过于乐观。也就是说，两年后系统的实际数据量大概率要远少于两年前预估的数据量。

自然我们考量系统的数据量，就是系统现在的数据存量加上未来二到三年的数据量。然后再来看一下，我们预估的数据量级在下面哪个范围内。

1)1GB 以下量级，或者数据的条数在千万以下。对于这个量级内的数据来说，几乎所有的存储产品其性能都还不错，因此不需要过多考虑数据量和性能，重点考虑其他维度即可。

2） 1GB 以上、10GB 以下量级，或者数据的条数在一亿以内。这个量级基本上是单机存储系统能够处理的上限。

3）超过 10GB 量级，或者数据的条数超过一亿。这个量级的数据必须使用分布式存储，只有将数据分片，才能获得可以接受的性能。

第三个需要考虑的维度，非常重要，但也最容易被忽略，那就是总体拥有成本(Total Cost of Ownership，TOC)。总体拥有成本是指，选择该存储产品，所需要付出的成本。虽然现在大部分存储系统都是开源免费的，但是无论使用哪一款产品，都是有成本的。成本主要来自如下三个方面。

第一，也是最重要的，团队是否熟悉该产品？如果不熟悉则意味着使用过程中可能要踩坑，然后填坑。踩坑和填坑的代价可能是系统宕机、丢数据或者开发进度延期。

第二，需要考虑该产品是否简单、易于学习和使用。

第三，需要考虑系统上线后的运维成本，比如，Hadoop 生态的一系列产品，维护工作相对来说就比较困难，要想让它们持续正常地运转，一般都需要一个有经验的运维人员专门负责维护。

在线业务系统如何选择存储产品

首先来看一下在线业务系统如何选择存储产品。在线业务系统是指为在线业务提供服务的系统，比如，电商系统的交易部分，或者手机上使用的绝大多数 App，直接支撑这些 App 的后端系统，都是在线业务系统。通俗地说就是，那些主要对数据库执行增删改查操作的系统，都是在线业务系统。

那么在线业务系统对存储产品有什么样的要求呢?

1）由于需要频繁地对数据进行增删改的操作，因此存储产品需要有较好的写性能。

2）由于在线业务直接服务于前端，需要快速响应，因此每次存储访问必须要快，至少要达到毫秒级的响应。

3）另外，存储产品需要能够支撑足够多的并发请求，以满足大量用户同时访问的需求。

4)最后，很重要的一点是，由于在线业务系统的需求一直都在不停地变化，因此存储产品需要能够提供相对比较强大的查询能力，以便应对频繁变化的需求。否则，一旦业务需求稍微有一点儿变动，存储结构就不得不随之做出调整，这就非常麻烦了。

那么哪些存储产品可以满足上面列出的这些需求，支撑在线业务呢？答案是，没有这么完美的产品。

但是如果把要求放宽一点，最接近上述要求的就是我们最常用的以 MySQL 为代表的关系型数据库。除了我们常用的 MySOL 之外，Oracle、DB2、SOL Server，以及各大云厂商提供的 RDS 等都是关系型数据库。

由于各关系型数据库产品的存储结构和实现原理都是类似的，因此它们在功能等方面的差别并不大，是可以相互替代的。考虑到成本等诸方面的因素，MySOL 一般是我们的首选。

此外，一些 KV 存储也可以用于在线业务，比如，Redis、Memcached 等等。Redis 这种基于内存的存储，具有非常好的读写性能，能提供有限的查询功能，但是其并不能保证数据的可靠性，一般来说,Redis 都是配合 MySQL 数据库作为缓存来使用。所以，目前绝大多数的在线业务，仍然使用的是 MySQL(或者其他关系型数据库)加 Redis 这对经典组合，暂时还没有更好的选择。

还有一些存储产品也可以用于在线业务，但大多数局限于特定的业务场景中,不具备通用性,比如,用于存储文档型数据的 MongoDB，等等。

在线业务系统需要存储产品能够支持高性能写入、毫秒级响应以及高并发。MvSOL 加 Redis 的经典组合可以应对大部分的场景需求。而分析系统则需要存储产品能够促在海量数据，并且能够支持在海量数据上快速聚合、分析和查询，而对写入性能、响应时延和并发的要求并不高。

一般来说量级在 GB 以内的可以使田 MySQL ；量级超过 GB 的数据并且如果还是需要做实时的分析和查询，则可以优先考虑 ES，Hbase、Cassandra 和 ClickHouse 这些列式数据库也可以视情况选择。量级超过 TB 的数据，一般只能事先对数据做聚合计算，然后再在聚合计算的结果上进行实时查询，这种情况下，一般选择把数据保存在 HDFS 中。

常见问题：如何存储前端埋点之类的海量数据

对于大部分互联网公司来说，数据量最大的几类数据是：前端埋点数据、监控数据和日志数据。“前端埋点数据”也称为“点击流”，是指在 App、小程序和 Web 页面上的埋点数据，这些埋点数据主要用于记录用户的行为，比如打开了哪个页面，点击了哪个按钮，在哪个商品上停留了多久等信息。

这种记录用户的行为数据，为了从统计学上分析群体用户的行为，从而改进产品和运营。比如，浏览某件商品的人很多、在其上停留的时间也很长，最后下单购买的人却很少，那么采销人员就要考虑这件商品的定价是不是太高了。

除了点击流数据之外，监控和日志也是大家常用的数据。

上述三种数据都是真正的“海量”数据，相比于订单、商品之类的业务数据，点击流的数据量要多出 2～3 个数量级。在头部大厂，这类数据每天产生的数据量很有可能会超过 TB(1 TB = 1024 GB)级别，经过一段时间的累积，有些数据会达到 PB(1 PB =1024 TB）级别。

早期对于海量原始数据的存储方案，都倾向于先计算再存储。也就是说，在接收原始数据的服务中，先对数据进行过滤和聚合等初步的计算，将数据压缩收敛之后再进行存储。这样可以降低存储系统的写入压力，同时还能节省磁盘空间。

随着存储设备的成本越来越低，以及数据的价值被不断地重新挖掘，很多大型企业都倾向于先存储再计算。即直接保存海量的原始数据，再对数据进行实时或批量计算。这种方案成本较高但是优点很多，比如不需要二次分发，就可以同时为多个流和批计算任务提供数据；如果计算任务出现错误，则可以随时回滚，重新计算；如果对数据有新的分析需求，则上线之后，可以直接用历史数据计算出结果，而不用等待收集新的数据。

不过先存储再计算的方式，对保存原始数据的存储系统提出了更高的要求：不仅要有足够大的容量，能够水平扩容，而且要求读写速度足够快，要能跟得上数据生产的速度，同时环要为下游计算提供低延迟的读服务。一般都会选择 Kafka/RocketMQ 来存储。

这类产品能够提供“无限”的消息堆积能力，具有招高的吞吐量，与大部分大数据生态圈的开源软件都有非常好的兼容性和集成度。

如果是需要长时间(几个月到几年)保存的海量数据，适合用 HDFS 之类的分布式文件系统来存储。

还有一类是时序数据库(Time Series Databases)，比如 InfluxDB，不仅具有非常好的读写性能,还能提供简便的查询和聚合数据的能力。但是它们并不能存储所有类型的数据，而是专用于存储类似于监控之类的有时间特征，并且数据内容都是数值的数据。

面对海量数据,如何才能查得更快

前面说明了如何保存海量的原始数据，因为原始数据的数据量实在是太大了,能够存储下来已属不易，这个数据量是无法直接提供给业务系统进行查询和分析的。

其中有两个方面的原因：一是数据量太大了，二是目前没有很好的数据结构和查询能力可以支持业务系统的查询。

所以，目前的一般做法是，通过流计算或批计算（也就是 MapReduce)对原始数批进行二次或多次讨滤、汇聚和计算的处理,然后把计算结果保存到另外一个存储系统中由该存储系统为业务系统提供查询支持。

有的业务计算后的数据变得非常少，比如，一些按大进行汇总的数据，或者排行榜类的数据，无论使用哪种存储，都能满足要求。还有一些业务，无法通过

事先计算的方式解决所有的问题。比如，像点击流、监控和日志之类的原始数据，就属于“海量数据中的海量数据”，这些原始数据经过过滤汇总和计算之后，在大多数情况下，数据量会出现数量级的下降，比如，从 TB 级别的数据量，下降到 GB 级别，但仍然属于海量数据。除此之外，我们还要对这个海量数据，提供性能可以接受的查询服务。

面对这种数量级的海量数据，如何才能让查询变得更快一些?

常用的分析类系统应该如何选择存储

查询海量数据的系统大多是离线分析类系统，可以简单地将其理解为类似于做报表的系统，也就是那些主要功能是对数据做统计分析的系统。这类系统大多是重度依赖于存储的。选择什么样的存储系统、使用什么样的数据结构来存储数据，将直接决定数据查询、聚合和分析的性能。

分析类系统对存储的需求一般包含如下四点。

1）用于分析的数据量一般会比在线业务的数据量高出几个数量级，这就要求存储系统能够保存海量数据。

2）并且还要能在海量数据上快速进行聚合、分析和查询的操作。注意，这里所说的“快速”，前提是处理 GB、TB 甚至 PB 级别的海量数据，在这么大的数据量上做分析，几十秒甚至几分钟都算是快速的了,这一点与在线业务要求的毫秒级速度是不一样的。

3）由于在大多数情况下，数据都是异步写入，因此系统对于写入性能和响应时延，要求一般不高。

4）由于分析类系统不用直接支撑前端业务，因此也不要求高并发。接下来我们看一下，可供选择的存储产品有哪些。如果系统的数据量在 GB 量级以下，那么 MySQL 依然是可以考虑的，因为它的查询能力足以应付大部分分析系统的业务需求。而且可以与在线业务系统合用一个数据库，不用做 ETL(数据抽取),更简便而且实时性更好。当然最好能为分析系统配置单独的 MySQL 实例，以避免影响在线业务。

如果数据量级已经超过了 MySQL 的极限，则还可以选择一些列式数据库，比如 Hbase、Cassandra、ClickHouse 等。这些产品对海量数据都有非常好的查询性能，在正确使用的前提下，10GB 量级的数据查询基本上可以做到秒级返回。不过，这些数据库对数据的组织方式会有一些限制，在查询方式上也没有 MySQL 那么灵活。

还可以考虑 Elasticsearch (ES)，ES 本来是一个为了搜索而生的存储产品，，但是其也支持结构化数据的存储和查询，也支持分布式并行查询，因此其在海量结构化数据查询方面的性能也非常好。最重要的是 ES 对数据组织方式和查询方式的限制,不像其他列式数据库那么死板。也就是说，ES 的查询能力和灵活性是要强于上述这些列式数据库的。不,ES 也有一个缺点，那就是需要具有大内存的服务器，硬件成本比较高。

当数据量级超过 TB 级的时候，对这么大量级的数据做统计分析,无论使用哪种存储系统，速度都快不了，这里的性能瓶颈主要在于磁盘 IO 和网络带宽，这种情况下肯定做不了实时的查询和分析，这里可以采用的解决方案具定期对数据

讲行聚合和计算，然后把结果保存起来，在需要时再对结果做一次查询。这么大量级的数据，一般是选择存储在 HDFS 中，配合 Spark、 Hive 等大数据生态圈产品，对数据进行聚合和计算。

转变思想：根据查询选择存储系统

面对海量数据仅根据数据量级来选择存储系统是远远不够的。因为在过去几十年的时间里,存储技术和分布式技术，在基础理论方面并没有本质上的突破。技术发展更多的是体现在应用层面上，比如集群管理更加简单，查询更加自动化,像 MapReduce 之类的产品就是如此。

不同的存储系统之间并没有本质的差异。它们的区别只在于，存储引擎的数据结构、存储集群的构建方式，以及提供的查询能力等这些方面的差异。这些差异，使得不同的存储系统只有在它所擅长的那些领域或场景下，才会有很好的性能表现。

比如我们前面说过的 RocksDB 和 LevelDB，它们的存储结构 LSM-Tree 其实就是日志和跳表的组合，单从数据结构的时间复杂度上来说，相较于 MySQL 所采用的 B+树，LSM-Tree 并没有本质上的提升，它们的时间复杂度都是 O(log n)。但是，LSM-Tree 在某些情况下利用日志能有更好的写性能表现。

也就是说没有哪种存储能在所有情况下，都具有明显的性能优势，所以说，存储系统没有银弹，不要指望简单地更换一种数据库就可以解决数据量大、查询慢的问题。

不过在特定的场景下，通过一些优化方法，把查询性能提升几十倍甚至几百倍这一点还是有可能的。这里有个很重要的思想就是，**根据查询来选择存储系统和数据结构**。比如我们经常用 Elasticsearch 构建商品搜索系统，就是把这个思想实践得很好的一个例子。ES 采用的倒排索引的数据结构，并没有比 MySQL 的 B+树更快,或者说更先进，但是面对“全文搜索”这个查询需求，相较于使用其他的存储系统和数据结构，使用 ES 的倒排索引在性能上能高出几十倍。

说个大厂的实际例子，京东的物流速度是非常快的。京东的物流之所以能够做到这么快，有一个很重要的原因，那就是它有一套智能的补货系统。根据历史的物流数据，对未来的趋势做出预测，来为全国的每个仓库补货。这样京东就可以做到用户下单购买的商品，有很大概率就在离用户不远的京东仓库里，这样自然就能很快送达了。这个系统在后台需要分析每天几亿条的物流数据，每条物流数据又细分为几段到几十段，因此每天的物流数据就是几十亿的量级。

这份物流数据的用途非常大，比如智能补货系统要用；运力调度的系统也要用；评价每个站点、每个快递小哥的时效达成情况要用；物流规划人员同样也要用这个数据进行分析,并对物流网络做持续优化。

那么，采用什么样的存储系统来保存这些物流数据，才能满足这些查询的需求呢？显然，任何一种存储系统都无法满足这么多种查询的需求。需要根据每种需求的具体情况，专门为其选择适合的存储系统；定义适合的数据结构,解决各自的问题。而不是用一种数据结构及一个数据库去解决所有的问题。

对于智能补货和运力调度这两个系统，由于它们具有很强的区域性，因此京东把数据按照区域(省或地市)做分片，再汇总成一份全国的跨区域物流数据，这样绝大部分查询都可以落在一个分片上，查询性能就会很好。

对于站点和快递人员的时效达成情况，由于这种业务的查询方式大多以点查询为主，因此可以考虑在计算的时候，事先按照站点和快递人员把数据汇总好，存放到分布式 KV 存储中，基本上就可以达到毫秒级查询的性能。

而对于物流规划的查询需求，查询方式是多变的，把数据放到 Hive 表中，按照时间进行分片。物流规划人员可以在上面执行一些分析类的查询任务，这样的查询任务即使是花上数小时的时间，用于验证一个新的规划算法，也是可以接受的。

商品系统的存储架构设计

从某种程度来说，我们商城系统中商品系统的设计和实现不是太完善。我们来看如何设计一个快速、可靠的商品系统存储架构。

电商的商品系统所包含的主要功能就是增、删、改、查商品信息，业务逻辑比较简单，支撑的主要页面就是商品详情页。但是在设计商品系统的存储架构时，仍然需要着重考虑如下两个方面的问题。

第一，需要考虑高并发的问题。不管是哪种电商系统，商品详情页一定是整个系统中 DAU(Daily Active User，日均访问人数)最高的页面之一。商品详情页 DAU 高的原因与用户使用电商 App 的习惯息息相关，绝大部分用户浏览完商品详情页之后不一定会购买，但购买之前一定会浏览很多同类商品的详情页，正所谓“货比三家”。

所以商品详情页的浏览次数要远高于系统的其他页面。如果在设计商品系统的存储架构时，没有考虑到高并发的问题，那么在电商系统举办大促活动（不是秒杀）的时候，海量的浏览请求会在促销开启的那一刻同时涌向系统，支撑商品详情页的商品系统必然是第一个被流量冲垮的系统。

第二，需要考虑商品数据规模的问题。一般情况下，商品详情页的数据规模，对 B2C 的电商系统来说，数量多，体量大。

为什么说“数量多”？在国内一线的电商平台中，SKU 的数量大约在几亿到几十亿这个量级。商品数量级这么大的原因有很多，比如，同一个商品通常会有数种不同的版本型号，再比如商家为了促销需要，可能会反复上下架同一个商品，或者为同一个商品加上不同的“马甲”，这些原因都导致了 SKU 数量巨大。

为什么说“体量大”？看看淘宝或者京东的商品详情页，从上一直拉到底，页面会非常长。一般来说都在 10 个屏幕高度左右，并且这其中不仅包含了大量的文字，还会包含大量的图片和视频，甚至还包含了 AR/VR。

商品系统需要保存哪些数据

商品详情页需要保存哪些信息？

商品详情页一般展示的信息有：

基本信息：标题、副标题、原价、价格、促销价、颜色〔规格型号)……

详细信息：商品参数、商品介绍、图片视频……

其他信息：促销信息、推荐商品、评价、评论、配送信息、店铺信息

一般来说其他信息来自电商平台的其他系统，比如促销等等。我们暂且不讨论；基本信息和详细信息都是商品系统需要存储的内容。

应该如何存储这么多内容呢?能不能像保存订单数据那样，设计一张商品表，把这些数据全部存放进去？或者说，一张表存不下就再加几张子表，这样存储行不行？

其实并不是不可以。现今的一线电商企业，在发展的早期阶段采用的就是这种存储结构。而现今它们所采用的复杂的分布式存储架构，都是在发展的过程中逐步演进而来的。

用数据库表存储的好处就是“糙、快、猛”，简单、可靠而且容易实现，但是缺点是，表能支撑的数据量有限，以及无法满足高并发的需求。

如果只是低成本且快速构建一个小规模的电商，这可能会是相对比较合理的选择。

当然规模一旦变大，就不能再采用数据库表存储这种简单粗暴的方案了。如果不能用数据库,那么我们应该选择哪种存储系统来保存这么复杂的商品数据呢？在目前的情况下，任何一种存储方案都无法完全满足全部需求，最好的解决方案是分而治之，即把商品系统需要存储的数据，按照特点分成商品基本信息、商品参数、图片视频和商品介绍几个部分，分别进行存储。

如何存储商品的基本信息

首先我们分析一下商品的基本信息，其中主要包括商品的主副标题、价格、颜色等一些最基本、最主要的属性。这些属性都是固定的，不太可能会因为需求改变或不同的商品而变化，而且这部分数据不会太大，所以可以在数据库中建一张表来保存商品的基本信息。

然后，我们还需要在数据库前面加一个缓存，以帮助数据库抵挡绝大部分的读请求。

当然对于缓存数据的一致性，可以采用 Cache Aside 更新策略。

但是设计商品基本信息表的时候，需要特别注意的一点是，一定要记得保留商品数据的每一个历史版本。因为商品数据是随时变化的，但是订单中关联的商品数据，必须是下单那个时刻的商品数据。

解决方案则是，在对商品数据进行修改时，要为每个历史版本的商品数据保存一个快照，可以创建一个历史表保存到 MySQL 中，或者其他的存储系统都可以。

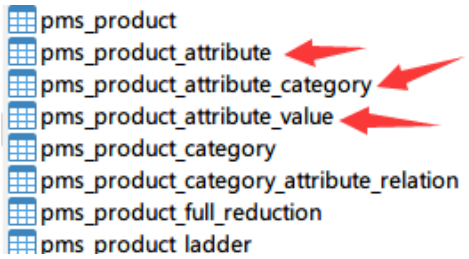
使用 MongoDB 保存商品参数

商品参数就是商品的特征,比如,电脑的内存大小、手机的屏幕尺寸、酒的度数、口红的色号等等。与商品的基本属性一样，参数也是结构化的数据。关于参数需要解决的一个难题是，不同类型的商品其参数是完全不一样的。

如果要设计一个商品参数表，那么这个表所要包含的字段就太多了，并且每增加一个品类的商品，这个表就要加入新的字段，所以这个方案行不通。

既然一个表不能解决问题，那就每个类别分别建一张表。比如建一个电脑参数表，其中包含的字段有 CPU 型号、内存大小、显卡型号、硬盘大小等等；

再建一个酒类参数表，其中包含的字段有酒精度数、香型、产地等等。如果品类比较少（在 100 个以内），那么用几十张表分别保存不同品类商品参数的做法也是可以的。比如我们的商城系统中，就是采用类似的设计



但是这并不是一个很好的解决方法。大多数数据库都要求数据表要有一个固定的结构，但 MongoDB 没有这个要求，特别适合用于保存像“商品参数”这种属性不固定的数据。

MongoDB 最大的特点是，它的“表结构”是不需要事先定义的或者说 MongoDB 中根本就没有表结构。由于没有表结构，因此 MongoDB 可以把任意数据都放在同一张表里。MongoDB 也支持按照数据的某个字段进行查询。

MongoDB 可以很好地满足商品参数信息数据量大、数据结构不统一等特性，而且我们也不需要对商品参数进行事务和多表联查，所以商品参数比较适宜存入 MongoDB。

使用对象存储保存商品图片和视频

图片和视频所占用的存储空间比较大，因此一般的存储方式是在数据库中只保存图片和视频的 ID 或 URL，实际的图片和视频则以文件的方式单独存储。

现今，图片和视频的存储技术已经非常成熟了，首选的方式是保存在对象存储（Object Storage）中。各大云厂商都提供了对象存储服务，对图片和视频都进行了大量有针对性的优化。比如，国内的七牛云、阿里云等等。这样上传图片和视频的时候，可以直接保存到对象存储中，然后把对应的键保存在商品系统。访问图片和视频的时候，真正的图片和视频文件，也不需要经过商品系统的后端服务进行读取，而是在 Web 页面上通过对象存储提供的 URL 直接访问，而且几乎所有的对象存储云服务都自带 CDN (Content Delivery Network，内容分发网络)加速服务，响应时间比直接请求业务的服务器更短。

商品介绍静态化

商品介绍在商品详情页中所占的比重是最大的，其中包含了大量的带格式文字、图片和视频。图片和视频自然要存放在对象存储中，而关于商品介绍的文本这部分内容基本上是不会频繁改变的，则一般是随着商品详情页一起静态化，保存在 HTML 文件中。

商品详情页静态化之后，不仅可以节省服务器资源，还可以利用 CDN 加速，把商品详情页放到离用户最近的 CDN 服务器上，让商品详情页的访问变得更快。

至于商品价格、促销等变动的信息，由于不能将其静态化到页面中因此可以在前端而面使用 AJAX 请求商品系统动态获取。

我们的秒杀系统就使用了商详页的静态化，在一般的大厂会有专门的 CMS（内容管理系统）负责商详页的静态化。

购物车系统的存储架构

我们来看看购物车系统的存储架构。

购物车系统的主要功能是什么？购物车系统主要用于在用户选购商品时暂存用户想要购买的商品。购物车系统对数据可靠性的要求不高，对性能也没有特别的要求，在整个电商系统中属于相对比较容易设计和实现的一个子系统。购物车系统主要包含如下功能。

把商品加入购物车。

展示购物车列表页，发起结算下单。

为了支撑购物车系统的这几个功能，存储模型应该如何设计呢？直观地分析，只要一个“购物车”实体就够了。那么，购物车实体包含的主要属性又有哪些呢？打开一个电商系统（以京东为例)的购物车界面进行对照分析。

购物车实体的主要属性包括 SKUID(商品 ID)、数量、加购时间和勾选状态。

其中，“勾选状态”属性，即购物车界面中每件商品前面的那个小对号，表示在结算下单时是否要包含这件商品。至于商品的价格和总价，以及商品介绍等信息，可以从电商的其他系统中实时获取，不需要购物车系统专门保存。

虽然购物车的功能很简单，但是在设计购物车系统的存储时，仍然有一些特殊的问题需要考虑。

设计购物车系统的存储架构时需要把握什么原则

请思考下面这几个问题。

1）如果用户没有登录网站，而是直接在浏览器中将商品加入购物车，那么关闭浏览器后再次打开购物网站，刚才加入购物车的商品是否还在？

2）如果用户没有登录网站，而是直接在浏览器中将商品加入购物车、然后再登录，那么刚才加入购物车的商品是否还在？

3）再打开手机 App，用相同的用户账号登录，第 2 步中加入购物车的商品是否还在？

一般情况下：

1）即使用户没有登录网站,加入购物车的商品信息也会保存在用户的电脑里，这样关闭浏览器后再打开，购物车的商品仍然存在。

2）如果用户先加入购物车，再登录购物网站，那么登录前加入购物车的商品就会自动归并到用户名下的购物车中，所以登录后购物车中仍然有登录前加入购物车的商品。

4）使用手机 App 登录相同的用户账号,看到的就是该用户的购物车，这时无论是在手机 App、电脑还是微信中登录,只要是相同的用户，看到的就是同一个购物车，所以第 2 步中加入购物车的商品是存在的。

所以总之，

1）如果用户未登录，则需要暂存购物车中的商品。

2）用户登录时，系统需要把暂存在购物车中的商品合并到用户的购物车中,并且清除暂存的购物车。

3）用户登录后,购物车中的商品需要在浏览器、手机 App 和微信等终端中保持同步。

也就是说，购物车系统需要保存两类购物车，一类是未登录情况下的“暂存购物车”，另一类是登录后的“用户购物车”。

当然如果系统业务约束用户必须登录后才能将物品放入购物车，那就没有暂存购物车一说了，早期的淘宝和京东都是允许用户未登录的暂存购物车，现在已经要求用户必须登录了。

如何设计“暂存购物车”的存储

暂存购物车的数据应该保存在客户端还是服务端？如果保存在服务端，那么每个暂存购物车都需要有一个全局唯一的标识，这个标识不太好设计，而且还会浪费服务端的资源。所以将暂存购物车的数据保存在客户端会更好，既可以节约服务器的存储资源，也不用考虑购物车标识的问题，因为各个客户端只需要保存自己唯一一个购物车就可以了，所以不需要额外标识。

浏览器客户端可以选择的存储并不多，只有 Session、Cookie 和 LocalStorage。暂存购物车的数据存在哪里最合适呢？保存在 Session 中是不太合适的，因为 Session 的保留时间较短,而且 Session 的数据实际上还是保存在服务端。剩余的两种存储 Cookie 和 LocalStorage 都可以用来保存购物车数据，选择哪种方式更好呢？答案是各有优劣。

在上述场景中，使用 Cookie 和 LocalStorage 最关键的区别是，客户端与服务端的每次交互都会自动带着 Cookie 数据往返，这样服务端就可以读写客户端 Cookie 中的数据了，而 LocalStorage 中的数据只能通过客户端访问。

使用 Cookie 存储实现起来比较简单。在加减购物车、合并购物车的过程中,由于服务端可以读写 Cookie，因此全部逻辑都可以在服务端实现，而且客户端和服务端请求的次数相对也少一些。

使用 LocalStorage 存储，实现相对复杂一些，客户端和服务端都要实现一些业务逻辑，但使用 LocalStorage 的好处是，它的存储容量比 Cookie 的 4KB 上限要大得多，而且不用像 Cookie 那样，无论用不用得上，每次请求都要带着 Cookie，因此 LocalStorage 更能节省带宽。

所以选择 Cookie 或 LocalStorage 来存储暂存购物车都是可以的，我们可以根据自己的需求来选择。如果是个小型电商系统，那么选择 Cookie 来存储，实现起来会更简单。但如果电商系统面对的用户需要或者喜欢加购大量的商品（比如批发的行业用户），这种情况下选择 LocalStorage 更合适。

用户购物车的存储

因为用户购物车必须保证多端的数据同步，所以其数据必须保存在服务端。常规的思路是设计一张购物车表，把数据存在 MySQL 中。

但是要注意要在 user_id 上建一个索引，因为查询购物车表时，是以 user_id 作为查询条件来进行操作的。

也可以选择更快的 Redis，以用户 ID 作为 Key，以一个 HASH 作为 Value 来保存购物车中的商品。

从读写性能上来说 Redis 比 MySQL 快很多，但是 MySQL 的数据可靠性是要好于 Redis 的。不过购物车里的数据，其对可靠性的要求并没有那么苛刻，丢失少量数据的后果也就是个别用户的购物车少了几件商品，问题通常不是很大。所以,在购物车的场景下，Redis 的数据可靠性不高这个缺点并不是不能接受的。

不过,每个电商系统都有其个性化的需求，如果需要以其他方式访问购物车的数据，比如，统计某一天加人购物车的商品总数，那么使用 MySOL.存储数据很容易就能实现,而使用 Redis 存储，查询起来就会非常麻烦且低效。

综合比较下来，考虑到需求总是会不断发生变化这个普遍情况，一般还是推荐使用 MySQL 来存储购物车的数据。如果追求高性能，或者支持高并发，则可以加入 Redis 来抗压力，比如在用户登录时将用户购物车数据加载入缓存，对购物车的变化可以直接操作缓存，异步的方式写入数据库。

总的来说，存储架构的设计过程就是一个不断做选择题的过程。很多情况下，可供选择的方案不止一套，选择的时候需要考虑实现的复杂度、性能、系统可用性、数据可靠性、可扩展性等多方面的影响因素。需要强调的是,这些因素中的每一个都是可以根据业务适当牺牲的。

比如我们一般都会认为数据是绝对不可以丢失的，也就是说不能牺牲数据的可靠性。但是像用户购物车的存储，使用 Redis 替代 MySQL 就是牺牲了数据的可靠性来换取高性能。即在购物车这样的场景下，很低概率地丢失少量数据是可以接受的。如果性能提升带来的收益远大于丢失少量数据所付出的代价，那么这个选择就是合理的。如果说不考虑需求变化的因素，牺牲一点点数据可靠性换取大幅度的性能提升，那么选择 Redis 是最优解。

文档分享地址：

http://note.youdao.com/noteshare?id=32760a0b500f4637136e4166aa0aa654&sub=FCE9D5B7C0A040278A2E5F15C30B6342