网站架构的演变

应哥出品，必属精品！

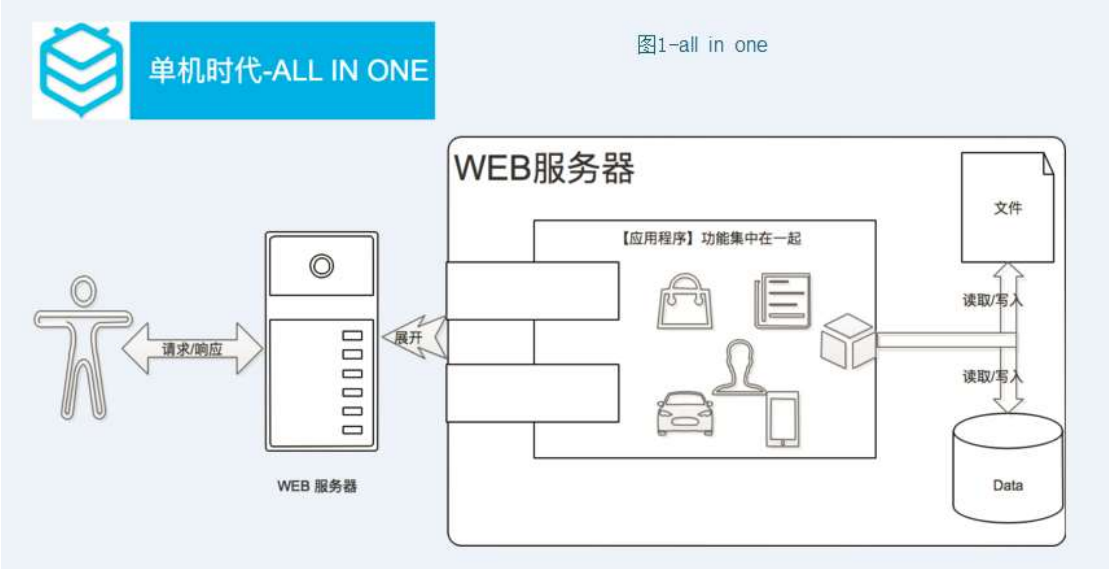
0 序言

互联网在变，架构也在变，架构的变迁亦是互联网的变迁。所以，我们有必要来聊聊互联网的架构及其变迁。何为架构？举个简单的例子，造一座大厦，首先得绘制出这座大厦的结构图纸，这样我们的施工人员才可以按照图纸施工建造。那么架构就相当于盖大厦的那张图纸，我们就是搬砖的码农。这里我们所讨论的架构，是指网站的的架构（狭义的理解）。这里按照我的理解这里把互联网的架构的发展分为三个阶段，第一阶段，单机时代；第二阶段，集群时代；第三阶段，分布式时代。接下来我们讨论一下这几个阶段架构的演变，以及各自阶段的优缺点。

1 单机时代

这里我们不得不讲一段背景，早期开发服务器等硬件资源很稀有也很昂贵，这就要求我们尽可能的节省资源，与此同时我们的用户数量并不是很多，业务也不是很复杂。在这种背景下单一的应用架构能够满足我们的正常使用需求。互联网早期，好比某个产品团队初创之时，资源有限，人力不足，为了快速开发一个产品，或上线一个网站，单机往往是一个不错的选择，此时会将应用程序、文件服务、数据库服务等资源集中在一台 Server 上。其中应用程序通常整体打包和部署，具体格式依赖于应用的语言和框架，例如 Java 的 WAR 文件、Rails 的目录文件，此种架构通常称为单体架构。

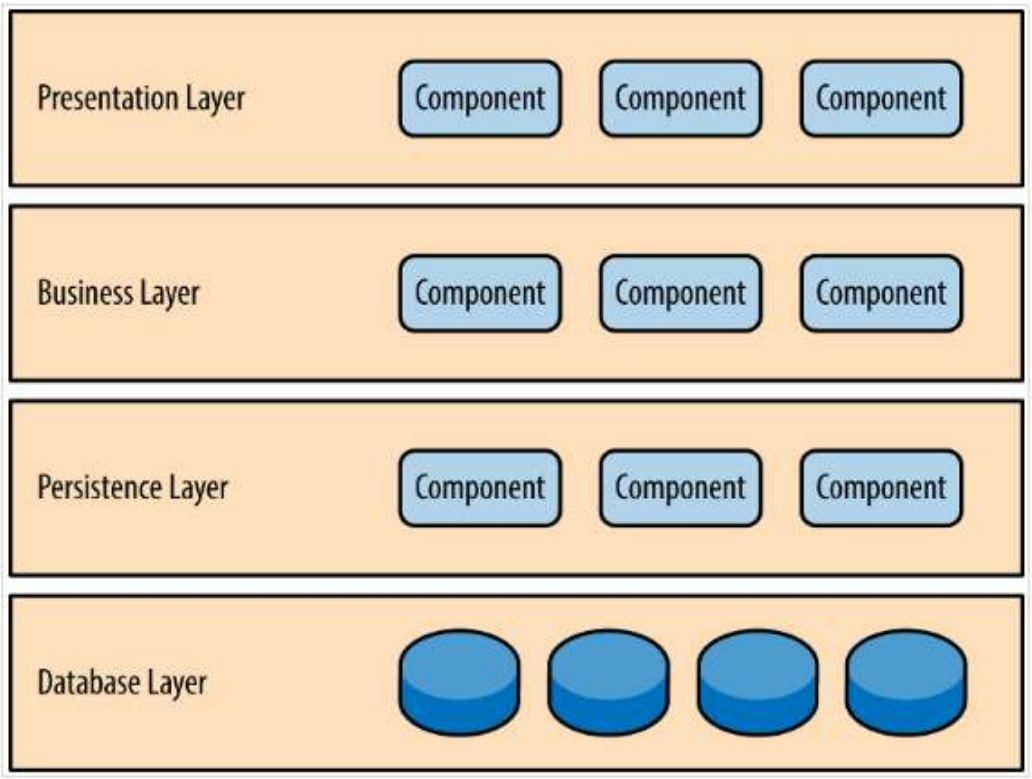
1.1 单体架构



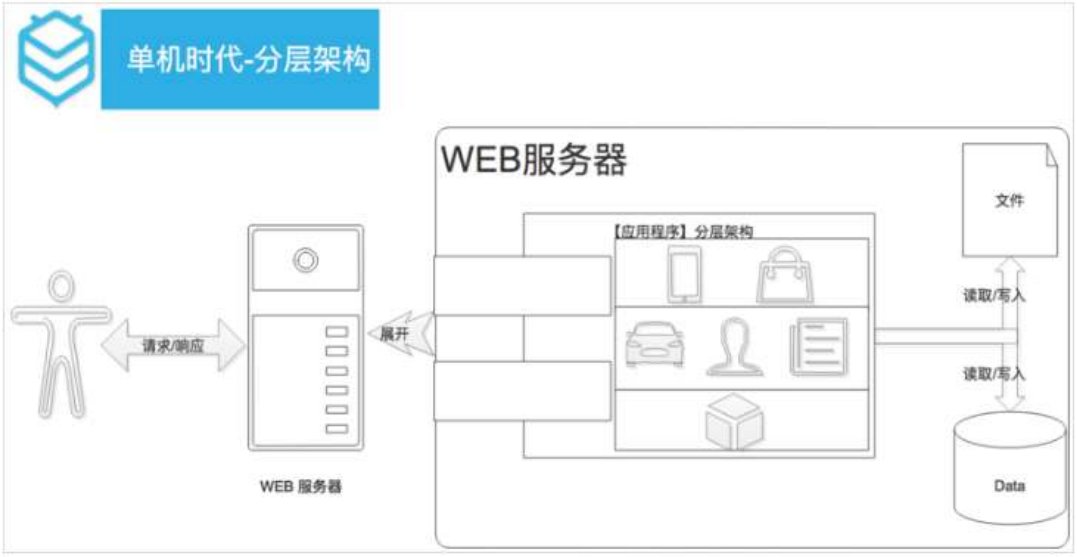
优点：简单快速，易于开发，易于测试，易于部署
缺点：也非常显著，只适合早期项目，变大后不易维护，且存在单点，升级需要停服

1.2 分层架构

不难发现，此时的应用程序架构显得杂乱无章，这在早期的 Web 开发中可能存在，比如使用 JSP+JDBC，ASP+ADO，但这显然不是一个友好的标准架构，于是分层架构应运而生，分层架构如下图所示，一般分为表现层（presentation）、业务层（business）、持久层（persistence）和数据库（database）。这其实也是最常见的 MVC 架构了。



改造之后的架构如下：

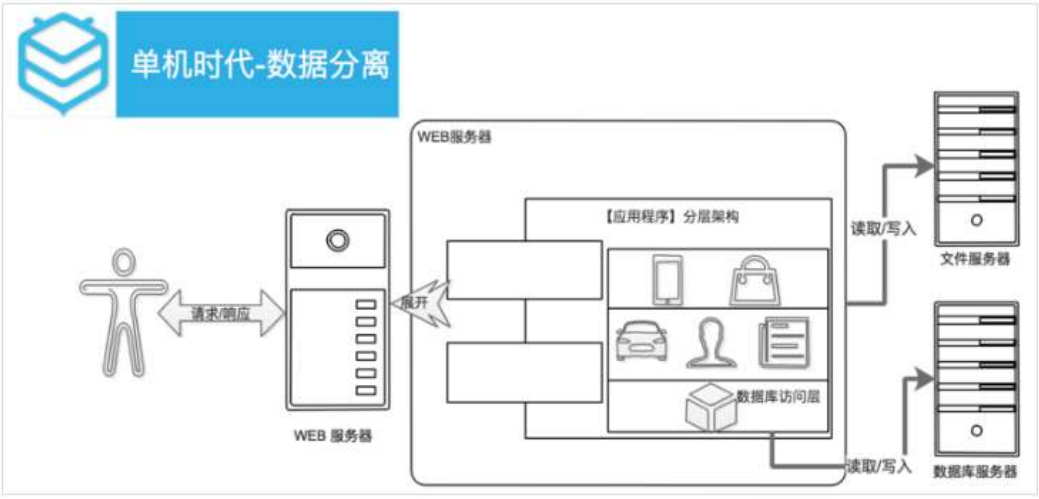


优点：结构简单，分工明确，分层测试，如果你不知道用什么软件架构时，推荐用它
缺点：扩展性差，迭代开发效率低，有时候层次过多导致流程复杂

1.3 数据分离

添加了分层架构，应用上好看点了，团队的开发效率有了一定的提升。此时业务量进一步增大，并且有了一定的用户规模，逐渐发现一台主机上应用和数据资源争夺的非常厉害。因为每种服对硬件资源的要求是不同的，应用服务器需要更快的 CPU，文件

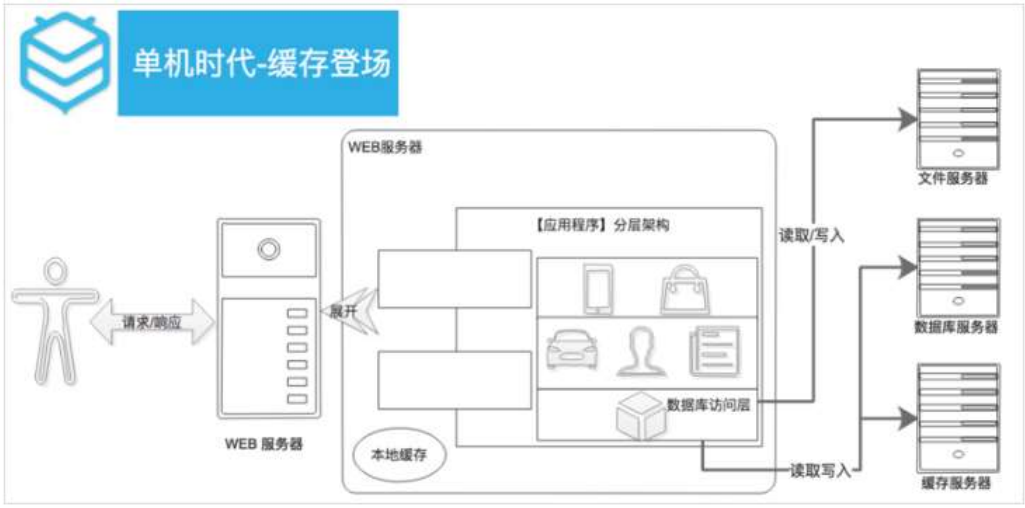
服务器需要更大的硬盘，数据库服务器需要更大的内存和硬盘，于是决定把应用和数据服务分离，形成了如下架构：



优点：资源分散，提高不同服务对硬件的利用率，方便维护
缺点：增加了资源消耗和网络开销，同时还存在单点

1.4 加入缓存

产品有了一定的口碑，用户量持续增长，访问开始频繁，想提升访问速度，缓存必不可少，闪亮登场。

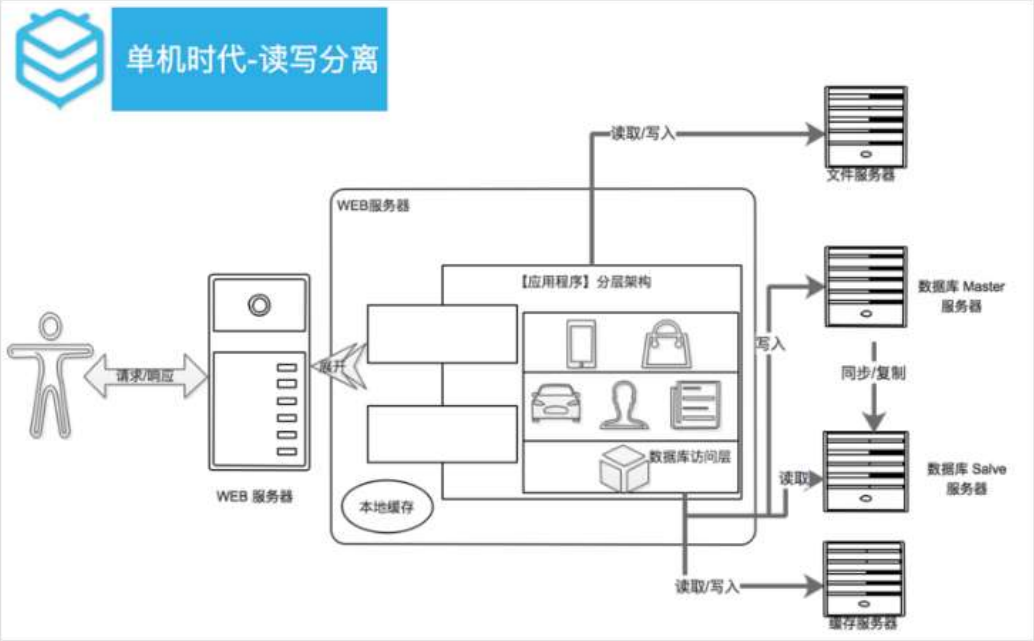


服务端缓存又可以分为本地缓存和远程缓存，各有优劣，本地缓存访问速度快，但数据量有限，而且后续集群化不方便共享；远程缓存可以共享，可以集群，容量不受限制，但要注意缓存更新的问题。

优点：简单有效，减少对 DB 的查询
缺点：增加逻辑判断，不适合存储大对象，此架构同样有单点

1.5 读写分离

市场反响不错，业务也在持续增长，但性能又有所下降，分析整个架构，发现数据库读写非常频繁，甚至有些业务，读大于写，单台数据库服务器又成了瓶颈，此时就可以尝试做读写分离和主从复制了。



优点：降低数据库单台压力，从机的数量可以灵活变更
缺点：架构开始变得复杂，维护难度加大

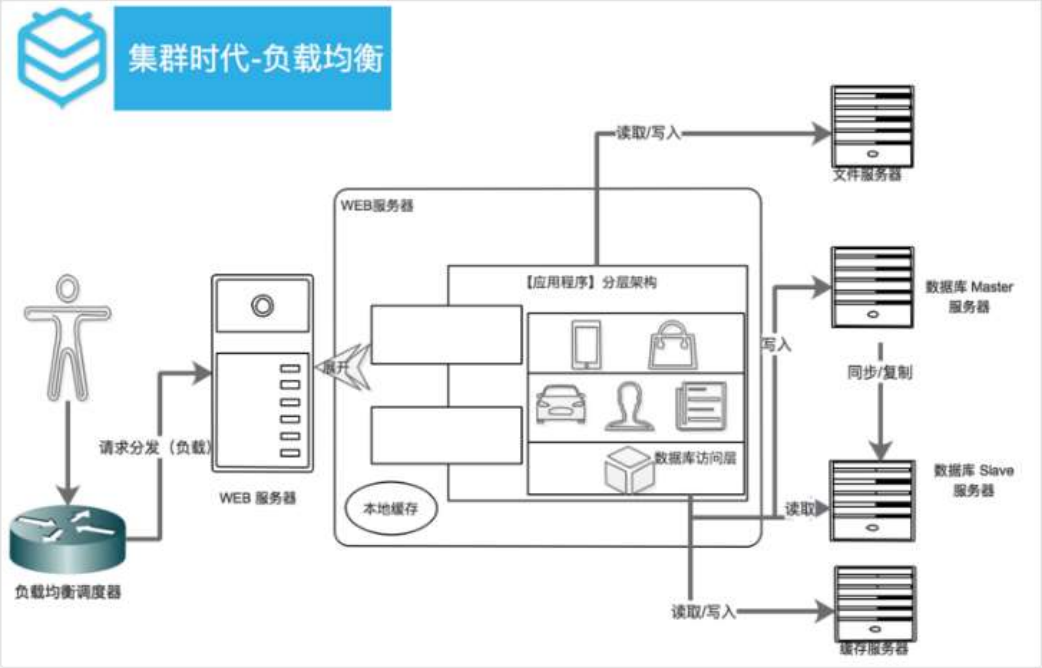
自此，单机时代的架构已然成型，“麻雀虽小五脏俱全”，初期已经能很好的支撑业务的运转。但随着业务的增长，各个模块还是可能出现瓶颈。而单机时代最大的问题，就是整个架构都存在单点，这个问题将在集群时代一一解决。

2 集群时代

单机时代，做了不少措施来缓解数据库层的压力，包括服务器分离、引入缓存、数据分离等，但随着访问量的猛增，对高可用的要求越来越高，减轻应用层压力、解决单点问题是当务之急，这就是集群时代需要做的事情。

2.1负载均衡

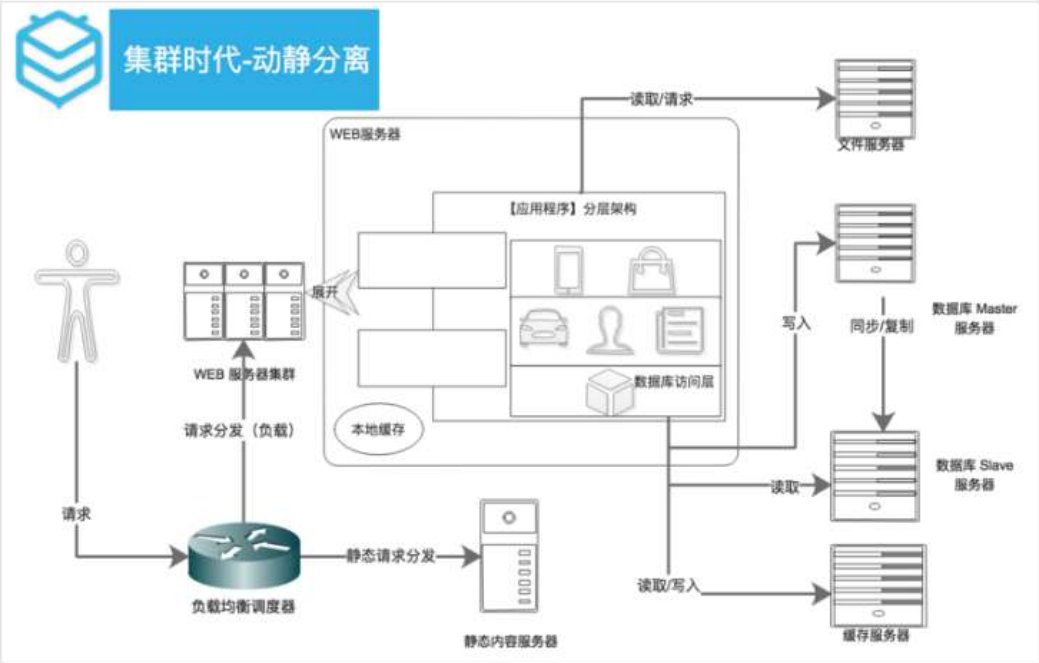
代码是架构的基础，但前期改造代码的工作量较大，如果人员变动频繁，那风险就更高了，所以提高服务器性能，常用的手段还是先将应用集群化，做负载均衡。



优点：去除应用层单点，可用性得到保证，性能有所提高
缺点：这时要注意应用之间的一致性问题，比如对缓存的访问，对 Session 的存储

2.2动静分离

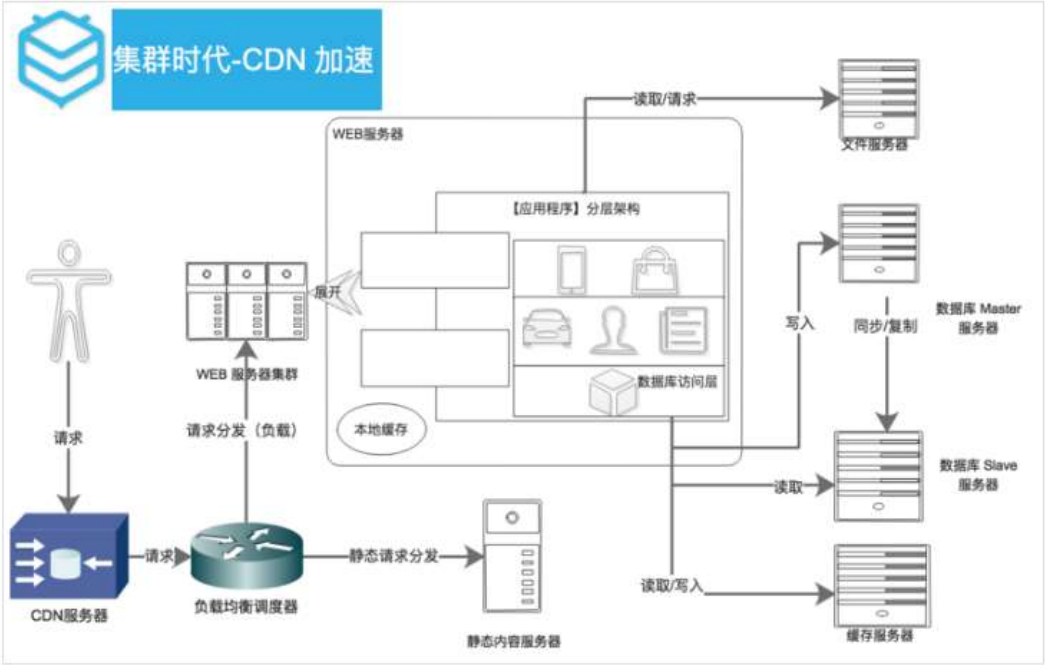
希望进一步降低应用服务器的压力，可以采用动静分离技术。动静分离是让动态网站里的动态网页，根据一定规则把不变的资源和经常变的资源区分开来，动静资源做好了拆分以后，我们还可以根据静态资源的特点将其做缓存操作，以加快响应速度。开发中常用做法还会将前后端分离，后端应用提供 API，根据前端的请求进行处理，并将处理结果通过 JSON 格式返回至前端。



优点：减轻应用服务器压力，缓存静态文件，加快响应速度，前后端分离，开发可以并行。
缺点：静态文件缓存更新失效问题，前后端沟通成本提高

2.3CDN 加速

内容分发网络（Content Delivery Network），简称 CDN），可以进一步加快网站相应，其原理是将源内容同步到全国各边缘节点，配合精准的调度系统，将用户的请求分配至最适合他的节点，使用户可以以最快的速度取得他所需的内容。



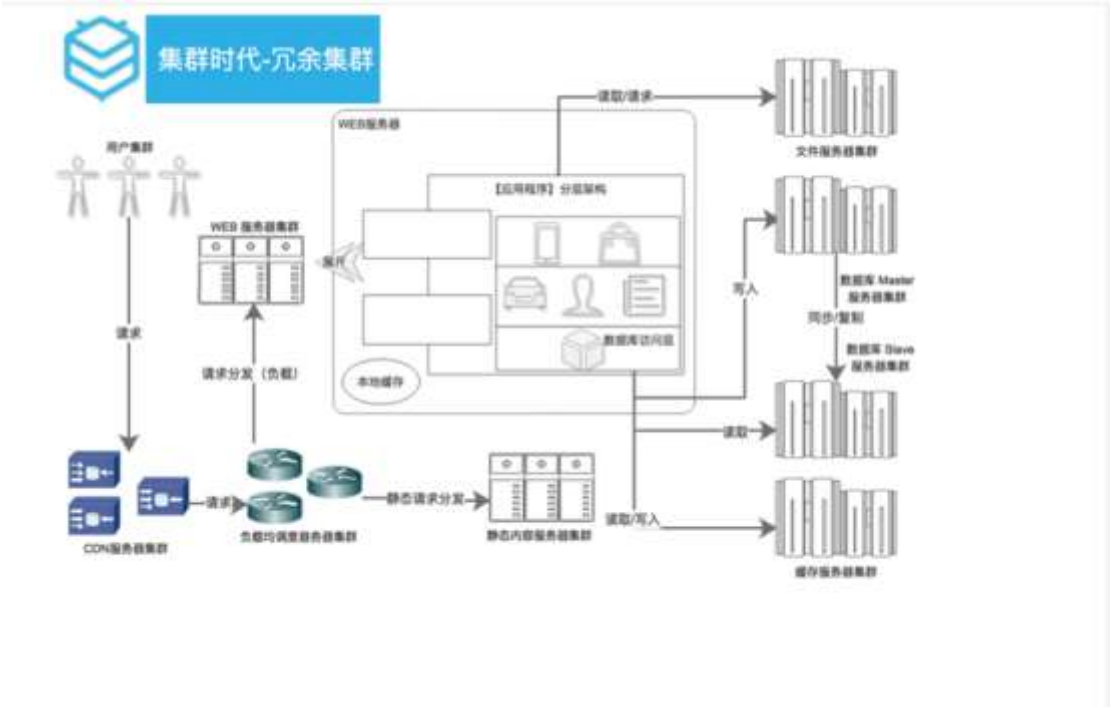
优点：解决网络带宽小、用户访问量大、网点分布不均等问题，提高用户访问的响应速度，减轻应用负载压力。
缺点：显然成本上去了，CDN 服务一般是按流量计费，同时也存在静态文件缓存更新失效问题。

2.4冗余集群

以上一个中型网站架构基本成型。当中型网站继续向大型网站演进，最终的目标是要保证“三高”：高并发、高性能、高可用。以上架构基本可以满足性能需求，接下来更多的是关注“高可用”，确保“无单点”。此时，就要对关键的服务，做冗余集群负载。理想情况下，我们将以下服务/应用都集群化：

- 数据库服务集群
- 文件服务集群
- 缓存服务集群
- 应用服务集群
- 负载均衡调度器集群
- 静态内容服务集群
- CDN 服务器集群
- 优点：去单点，高可用
- 缺点：数据有状态问题、数据一致性问题，资源成本、人力维护成本都上去了

到此为止，一个大型网站的架构也基本成型了，能“加机器”的地方都加完了，是不是就终结？当然不是！伴随着 DT/分布式 时代的到来，大流量和大数据的场景的出现，对应用提出了更高的要求，接下来就需要对应用程序开刀了。



优点：去单点，高可用
缺点：数据有状态问题、数据一致性问题，资源成本、人力维护成本都上去了

到此为止，一个大型网站的架构也基本成型了，能“加机器”的地方都加完了，是不是就终结？当然不是！伴随着 DT/分布式 时代的到来，大流量和大数据的场景的出现，对应用提出了更高的要求，接下来就需要对应用程序开刀了。

3 分布式时代

3.1 应用拆分

在前面，我们只是把应用程序做了分层架构，在创业初期或产品前期还是一个不错的选择。虽然应用也做了集群和负载均衡，但应用架构层面还是“集中式”的。随着业务越来越复杂，网站的功能越来越多，应用拆分势在必行了。

优点：应用解耦，分拆团队负责，分而治之
缺点：架构变复杂

应用拆分之后，还伴随着一个相互依赖、公共模块的问题，特别是依赖于相同的逻辑或功能代码。这时就可以考虑将这些共用的服务提取出来，独立部署，统一治理，提高重用度，这就是面向服务的架构（service-oriented architecture，缩写 SOA）了。

3.2 消息队列

应用拆分、服务独立部署之后，还是会出现一些通信或依赖问题，这时就可以引入消息队列，提高吞吐量。

优点：异步、解耦，提高吞吐量
缺点：消息消费延迟等问题

3.3 数据分库

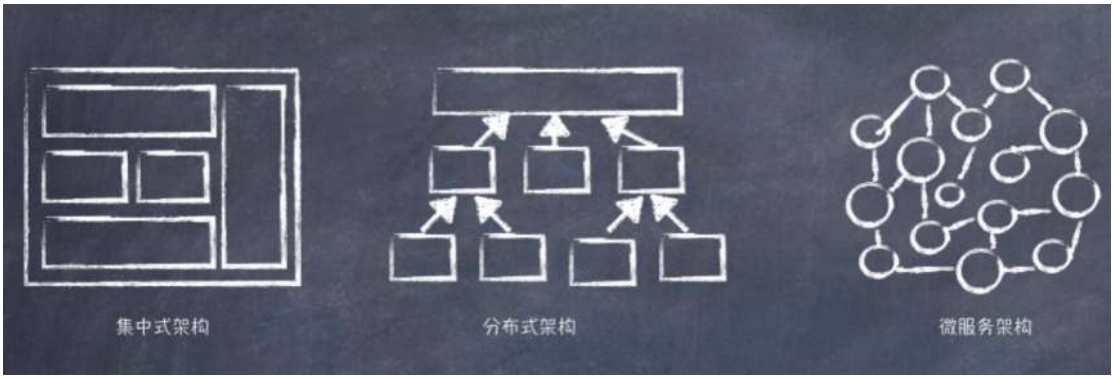
应用拆分之后，DB 分库理所当然，否则多个应用连接在单个数据库上，连接数、QPS、TPS、I/O 处理能力都非常有限。

优点：DB 分压，降低耦合**缺点：**数据访问模块冗余、复杂

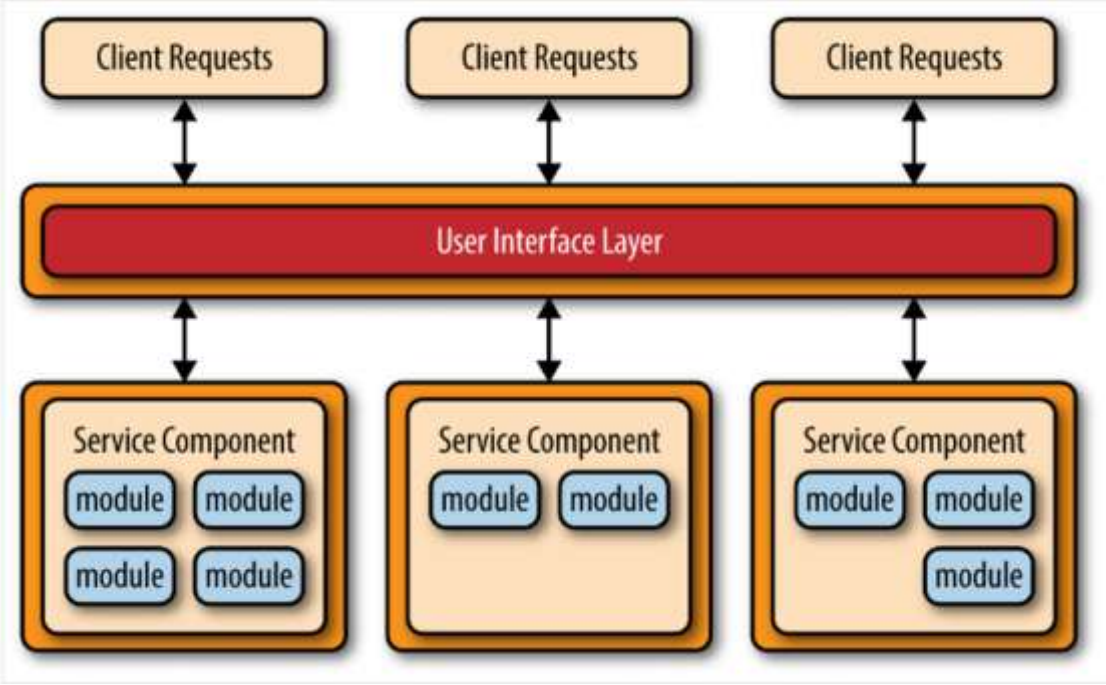
提到分库，不少人会想到分表，这一块我并未实践过，不好下笔。但想来会引入更复杂的数据架构和数据一致性问题，而且市面身上成熟开源的分库分表方案并没有，保不准又是一个深坑。拆或不拆，也是一个值得思考的问题。

3.4 微服务架构

微服务架构（microservices architecture）一度成为热点，在文章、博客、大会演讲上经常被提及。微服务并不是凭空出现，有人说，它是面向服务的架构（SOA）的升级，在此之前，还有诸如集中式架构、分布式的架构等。这里借用一副抽象的图来描述下常见的几种架构：



微服务架构由多个微小服务构成，每个服务就是一个独立的可部署单元或组件，它们是分布式的，相互解耦的，通过轻量级远程通信协议（比如 REST）来交互，每个服务可以使用不同的数据库，而且是语言无关性的。它的特征是彼此独立、微小、轻量、松耦合，又能方便的组合和重构，犹如《超能陆战队》中的微型机器人，个体简单，但组合起来威力强大。



优点：扩展性好，服务之间耦合性低，服务间相互独立，容易部署，易于开发，方便测试每一个服务

缺点：容易过度关注服务的大小，可能拆分的很细，导致系统依赖于大量的微服务，而服务之间的相互通信也会变得复杂，系统集成复杂度增加，很难实现原子性操作。

微服务之所以这么火，另一个原因是因为 Docker 的出现，它让微服务有一个非常完美的运行环境，Docker 的独立性和细粒度非常匹配微服务的理念，Docker 的优秀性能和丰富的管理工具，让大家对微服务有了一定的信息，概括来说 Docker 有如下四点适合微服务：

- 独立性：一个容器就是一个完整的执行环境，不依赖外部任何的东西。
- 细粒度：一台物理机器可以同时运行成百上千个容器。其计算粒度足够的小。
- 快速创建和销毁：容器可以在秒级进行创建和销毁，非常适合服务的快速构建和重组。
- 完善的管理工具：数量众多的容器编排管理工具，能够快速的实现服务的组合和调度。

当然，好的架构和技术，要应用于实践、让用户认可才行，这就需要在微服务架构和 Docker 技术之上有丰富的场景化应用。至此，架构变迁的三个时代介绍完成。总的来说架构不是一成不变的，时间不停，进步不止，人如此，架构依然。

Emil：rockyzhao_coder@163.com