**大型网站的特点**

**大型网站一般有如下特点：**

* **用户多，分布广泛**
* **大流量，高并发**
* **海量数据，服务高可用**
* **安全环境恶劣，易受网络攻击**
* **功能多，变更快，频繁发布**
* **从小到大，渐进发展**
* **以用户为中心**
* **免费服务，付费体验**

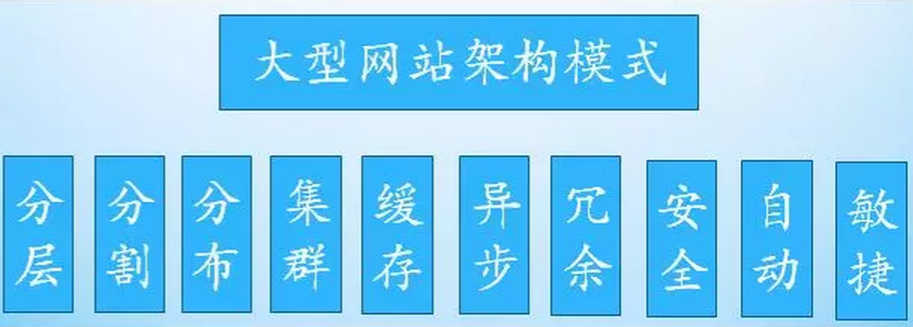
**大型网站架构目标**

****

**大型网站的架构目标有如下几个：**

* **高性能：提供快速的访问体验。**
* **高可用：网站服务一直可以正常访问。**
* **可伸缩：通过硬件增加/减少，提高/降低处理能力。**
* **扩展性：方便地通过新增/移除方式，增加/减少新的功能/模块。**
* **安全性：提供网站安全访问和数据加密、安全存储等策略。**
* **敏捷性：随需应变，快速响应。**

**大型网站架构模式**

****

**如上图是大型网站的架构模式：**

* **分层：一般可分为应用层、服务层、数据层、管理层与分析层。**
* **分割：一般按照业务/模块/功能特点进行划分，比如应用层分为首页、用户中心。**
* **分布式：将应用分开部署（比如多台物理机），通过远程调用协同工作。**
* **集群：一个应用/模块/功能部署多份（如：多台物理机），通过负载均衡共同提供对外访问。**
* **缓存：将数据放在距离应用或用户最近的位置，加快访问速度。**
* **异步：将同步的操作异步化。客户端发出请求，不等待服务端响应，等服务端处理完毕后，使用通知或轮询的方式告知请求方。一般指：请求——响应——通知模式。**
* **冗余：增加副本，提高可用性、安全性与性能。**
* **安全：对已知问题有有效的解决方案，对未知/潜在问题建立发现和防御机制。**
* **自动化：将重复的、不需要人工参与的事情，通过工具的方式，使用机器完成。**
* **敏捷性：积极接受需求变更，快速响应业务发展需求。**

**高性能架构**

**高性能的架构是以用户为中心，提供快速的网页访问体验，主要参数有较短的响应时间、较大的并发处理能力、较高的吞吐量与稳定的性能参数。**

**大型网站应该在任何时候都可以正常访问，正常提供对外服务。因为大型网站的复杂性，分布式，廉价服务器，开源数据库，操作系统等特点，要保证高可用是很困难的，也就是说网站的故障是不可避免的。**

**如何提高可用性，就是需要迫切解决的问题。首先，需要从架构级别考虑，在规划的时候，就考虑可用性。**

**行业内一般用几个 9 表示可用性指标，比如四个 9（99.99），一年内允许的不可用时间是 53 分钟。**

**可伸缩架构**

**伸缩性是指在不改变原有架构设计的基础上，通过添加/减少硬件（服务器）的方式，提高/降低系统的处理能力：**

* **应用层：对应用进行垂直或水平切分。然后针对单一功能进行负载均衡（DNS、HTTP[反向代理]、IP、链路层）。**
* **服务层：与应用层类似。**
* **数据层：分库、分表、NoSQL 等；常用算法 Hash，一致性 Hash。**

**安全架构**

**对已知问题有有效的解决方案，对未知/潜在问题建立发现和防御机制。对于安全问题，首先要提高安全意识，建立一个安全的有效机制，从政策层面，组织层面进行保障。**

**比如服务器密码不能泄露，密码每月更新，并且三次内不能重复；每周安全扫描等。**

**以制度化的方式，加强安全体系的建设。同时，需要注意与安全有关的各个环节。**

**安全问题不容忽视，包括基础设施安全，应用系统安全，数据保密安全等：**

* **基础设施安全：硬件采购，操作系统，网络环境方面的安全。一般采用正规渠道购买高质量的产品，选择安全的操作系统，及时修补漏洞，安装杀毒软件防火墙。**

**防范病毒，后门。设置防火墙策略，建立 DDOS 防御系统，使用攻击检测系统，进行子网隔离等手段。**

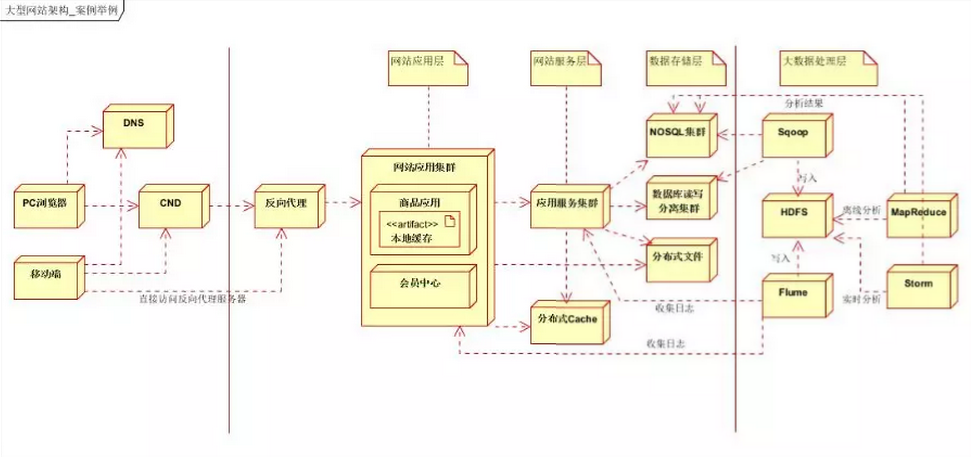
* **数据保密安全：存储安全（存储在可靠的设备，实时，定时备份），保存安全（重要的信息加密保存，选择合适的人员复杂保存和检测等），传输安全（防止数据窃取和数据篡改）。**

**常用的加解密算法（单项散列加密[MD5、SHA]，对称加密[DES、3DES、RC]），非对称加密[RSA]等。**

**敏捷性**

**网站的架构设计，运维管理要适应变化，提供高伸缩性，高扩展性。方便的应对快速的业务发展，突增高流量访问等要求。**

**大型架构举例**

****

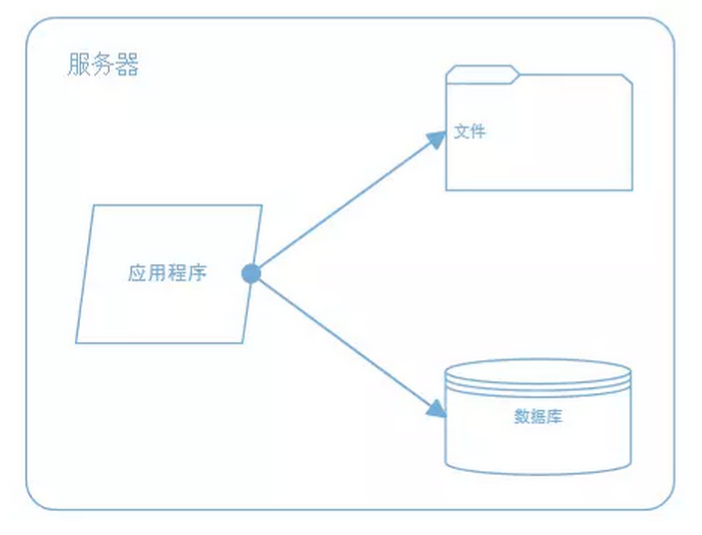
**以上采用七层逻辑架构：**

* **第一层客户层：支持 PC 浏览器和手机 App。差别是手机 App 可以直接通过IP访问，反向代理服务器。**
* **第二层前端层：使用 DNS 负载均衡，CDN 本地加速以及反向代理服务。**
* **第三层应用层：网站应用集群；按照业务进行垂直拆分，比如商品应用，会员中心等。**
* **第四层服务层：提供公用服务，比如用户服务，订单服务，支付服务等。**
* **第五层数据层：支持关系型数据库集群（支持读写分离），NOSQL 集群，分布式文件系统集群；以及分布式 Cache。**
* **第六层大数据存储层：支持应用层和服务层的日志数据收集，关系数据库和 NOSQL 数据库的结构化和半结构化数据收集。**
* **第七层大数据处理层：通过 Mapreduce 进行离线数据分析或 Storm 实时数据分析，并将处理后的数据存入关系型数据库。Mapreduce是一种分布式计算模型，主要用于搜索领域，解决海量数据的计算问题**

**通常一个组织中有价值的数据都要存储在关系型数据库系统中。但是为了进一步进行处理，有些数据需要抽取出来，通过MapReduce程序进行再次加工。Sqoop就是一个开源的工具，它允许用户将数据从关系型数据库抽取到hadoop中；也可以把MapReduce处理完的数据导回到数据库中。**

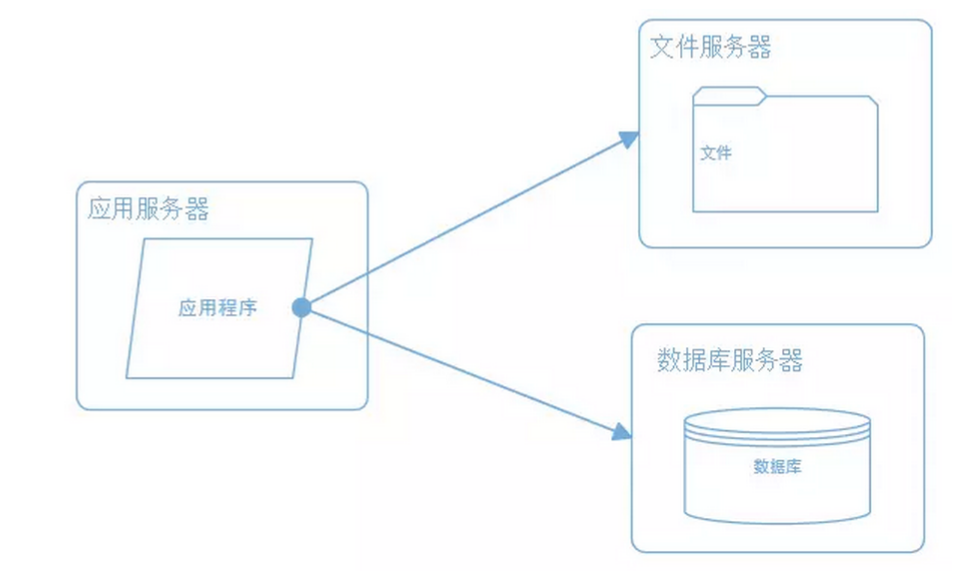
**最开始的网站架构**

**最初的架构，应用程序、数据库、文件都部署在一台服务器上，如下图：**

****

**应用、数据、文件分离**

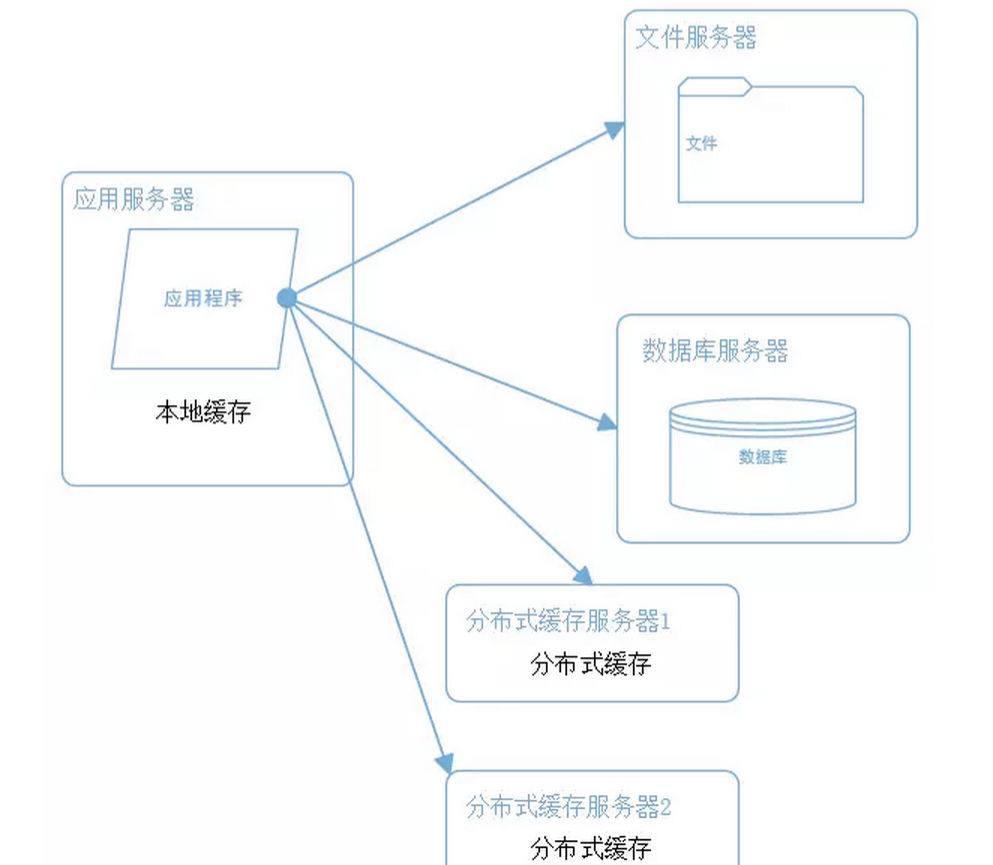
**随着业务的扩展，一台服务器已经不能满足性能需求，所以将应用程序、数据库、文件各自部署在独立的服务器上，并且根据服务器的用途配置不同的硬件，达到最佳的性能效果。**

****

**利用缓存改善网站性能**

**在硬件优化性能的同时，也通过软件进行性能优化，在大部分的网站系统中，都会利用缓存技术改善系统的性能。**

**使用缓存主要源于热点数据的存在，大部分网站访问都遵循 28 原则（即 80% 的访问请求，最终落在 20% 的数据上），所以我们可以对热点数据进行缓存，减少这些数据的访问路径，提高用户体验。**

****

**缓存实现常见的方式是本地缓存、分布式缓存。当然还有 CDN、反向代理等。**

**本地缓存，顾名思义是将数据缓存在应用服务器本地，可以存在内存中，也可以存在文件**

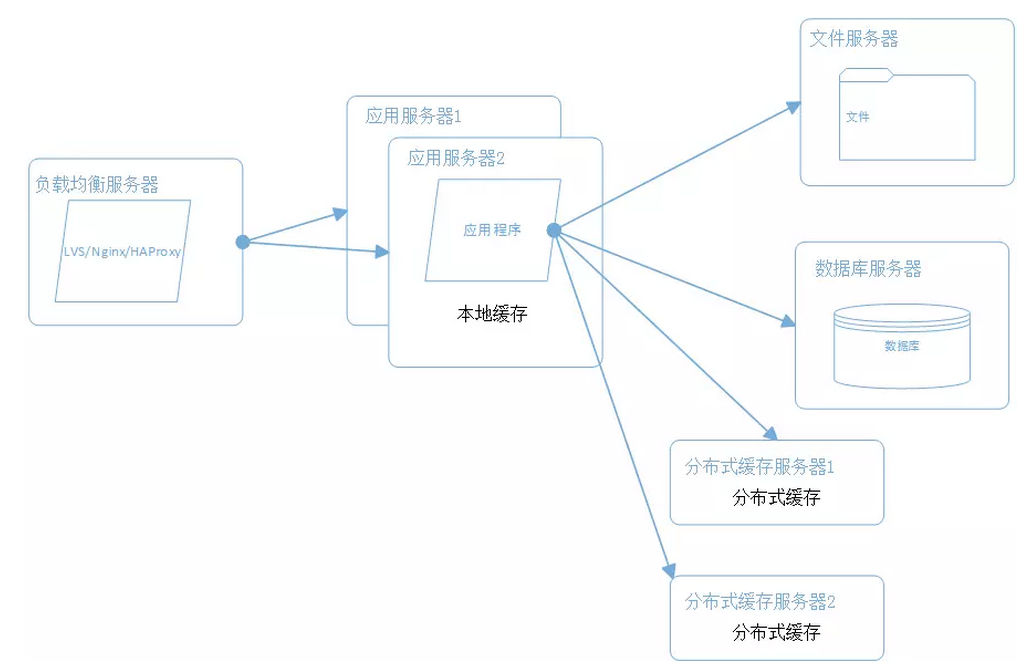
**本地缓存的特点是速度快，但因为本地空间有限所以缓存数据量也有限。**

**分布式缓存的特点是，可以缓存海量的数据，并且扩展非常容易，在门户类网站中常常被使用，速度按道理没有本地缓存快，常用的分布式缓存是 Memcached、Redis。**

**使用集群改善应用服务器性能**

**应用服务器作为网站的入口，会承担大量的请求，我们往往通过应用服务器集群来分担请求数。**

**应用服务器前面部署负载均衡服务器调度用户请求，根据分发策略将请求分发到多个应用服务器节点。**

****

**常用的负载均衡技术硬件的有 F5，价格比较贵，软件的有 LVS、Nginx、HAProxy。**

**LVS 是四层负载均衡，根据目标地址和端口选择内部服务器，Nginx 和 HAProxy 是七层负载均衡，可以根据报文内容选择内部服务器。**

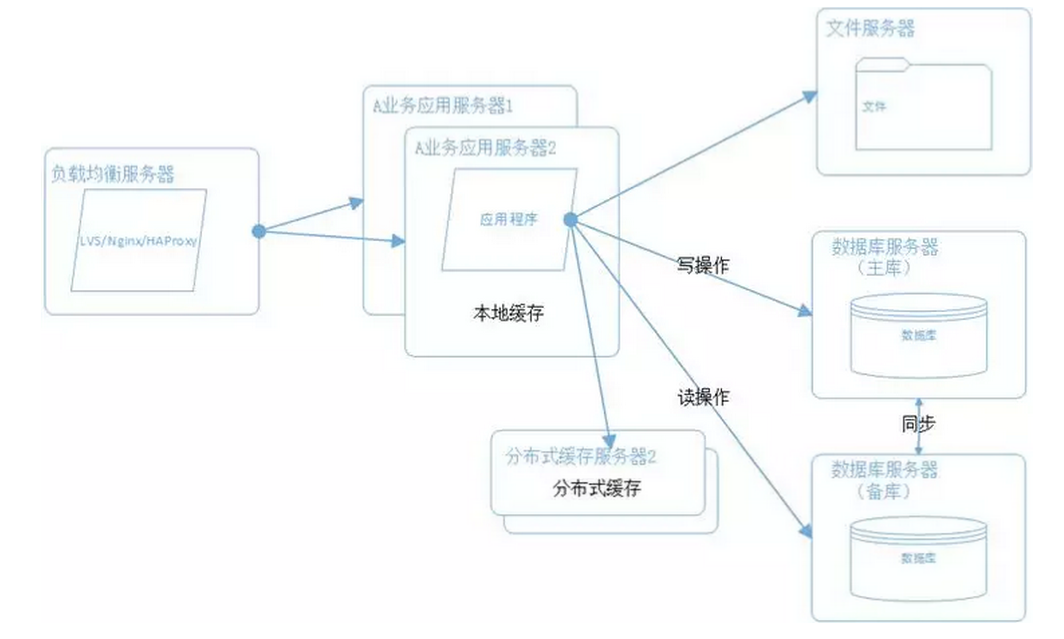
**因此 LVS 分发路径优于 Nginx 和 HAProxy，性能要高些；而 Nginx 和 HAProxy 则更具配置性，如可以用来做动静分离（根据请求报文特征，选择静态资源服务器还是应用服务器）。**

**数据库读写分离和分库分表**

**随着用户量的增加，数据库成为最大的瓶颈。改善数据库性能常用的手段是进行读写分离以及分库分表，读写分离顾名思义就是将数据库分为读库和写库，通过主备功能实现数据同步。**

**分库分表则分为水平切分和垂直切分，水平切分是对一个数据库特大的表进行拆分，例如用户表。**

**垂直切分则是根据业务的不同来切分，如用户业务、商品业务相关的表放在不同的数据库中。**

****

**使用 CDN 和反向代理提高网站性能**

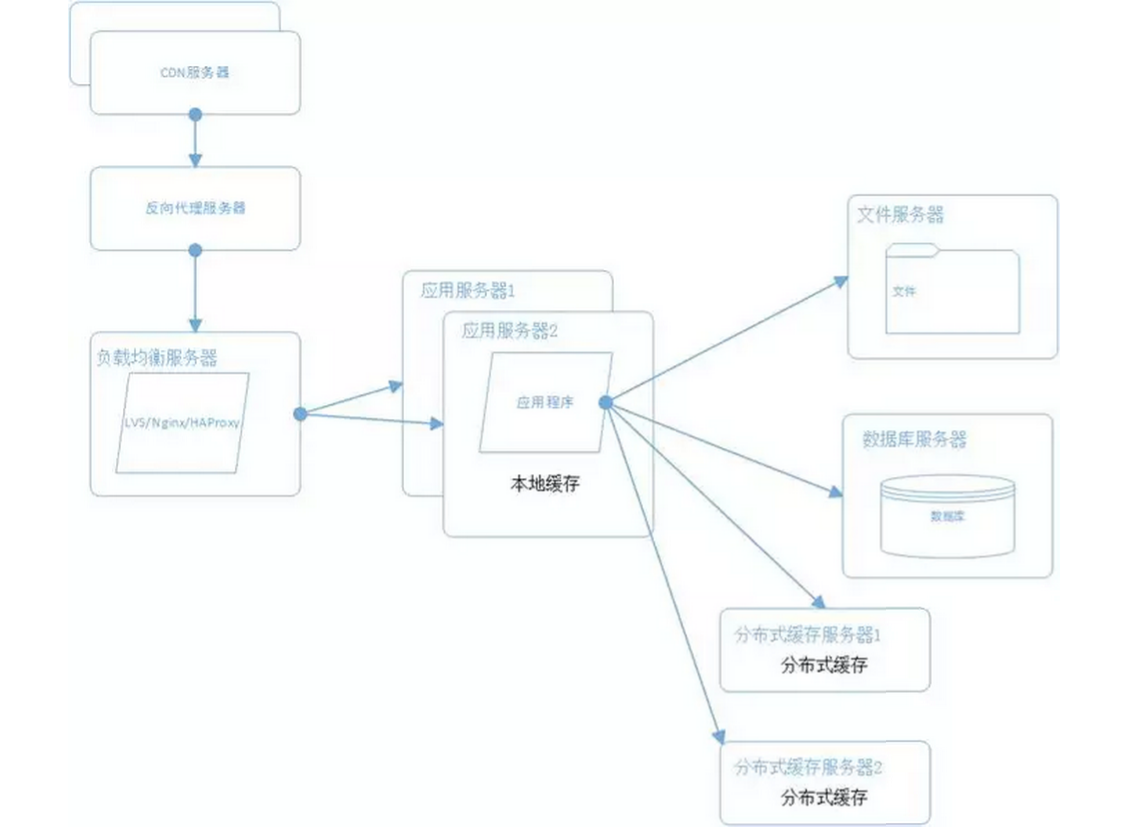
**假如我们的服务器都部署在成都的机房，对于四川的用户来说访问是较快的，而对于北京的用户访问是较慢的。**

**这是由于四川和北京分别属于电信和联通的不同发达地区，北京用户访问需要通过互联路由器经过较长的路径才能访问到成都的服务器，返回路径也一样，所以数据传输时间比较长。**

**对于这种情况，常常使用 CDN 解决，CDN 将数据内容缓存到运营商的机房，用户访问时先从最近的运营商获取数据，这样大大减少了网络访问的路径。比较专业的 CDN 运营商有蓝汛、网宿。**

**而反向代理，则是部署在网站的机房，当用户请求达到时首先访问反向代理服务器，反向代理服务器将缓存的数据返回给用户。**

**如果没有缓存数据才会继续访问应用服务器获取，这样做减少了获取数据的成本。反向代理有 Squid、Nginx。**

****

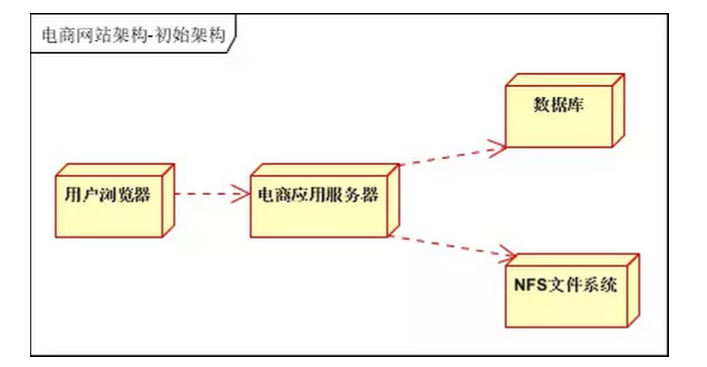
**使用分布式文件系统**

**用户一天天增加，业务量越来越大，产生的文件越来越多，单台的文件服务器已经不能满足需求，这时就需要分布式文件系统的支撑。常用的分布式文件系统有 GFS、HDFS、TFS。**

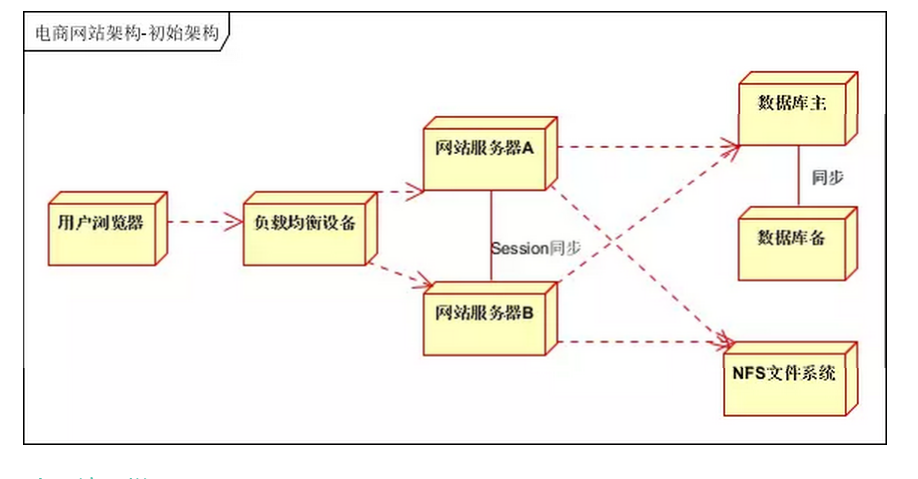
**网站初级架构**

**一般网站，刚开始的做法，是三台服务器，一台部署应用，一台部署数据库，一台部署 NFS 文件系统。**

**使用了一台服务器部署了应用，数据库以及图片存储。出现了很多性能问题，如下图：**

****

**但是，目前主流的网站架构已经发生了翻天覆地的变化。一般都会采用集群的方式，进行高可用设计。**

****

**至少是上面这个样子：**

* **使用集群对应用服务器进行冗余，实现高可用。（负载均衡设备可与应用一块部署）**
* **使用数据库主备模式，实现数据备份和高可用。**

**网站架构分析**

**根据以上预估，有几个问题：**

* **需要部署大量的服务器，高峰期计算，可能要部署 30 台 Web 服务器。并且这三十台服务器，只有秒杀，活动时才会用到，存在大量的浪费。**
* **大量应用存在冗余代码。**
* **数据需要频繁访问数据库，数据库访问压力巨大。**

**多级缓存**

**缓存按照存放的位置一般可分为两类：本地缓存和分布式缓存。采用二级缓存的方式，进行缓存的设计。**

**一级缓存为本地缓存，二级缓存为分布式缓存。（还有页面缓存，片段缓存等）**

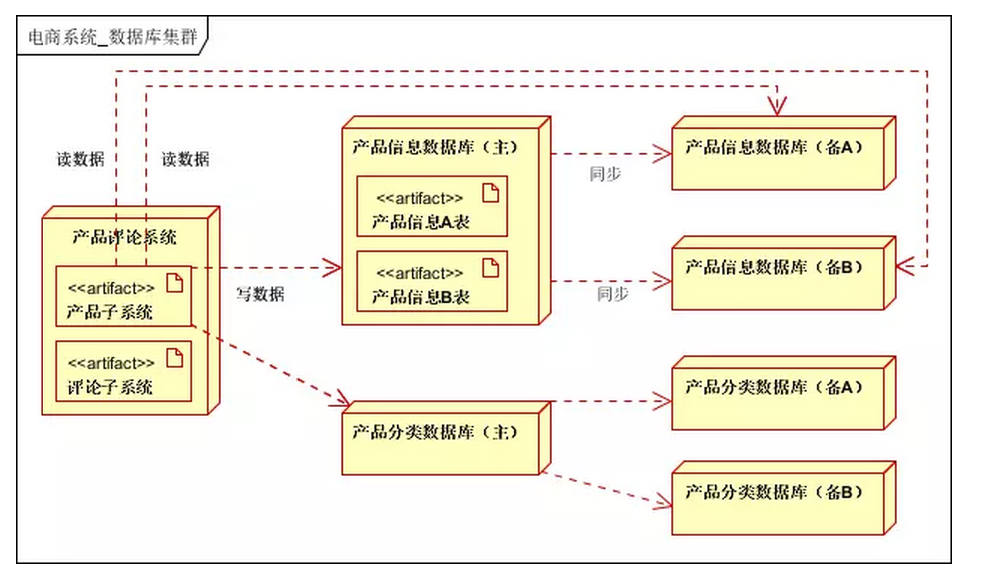
**一级缓存，缓存数据字典，和常用热点数据等基本不可变/有规则变化的信息，二级缓存缓存需要的所有缓存。**

**当一级缓存过期或不可用时，访问二级缓存的数据。如果二级缓存也没有，则访问数据库。**

**数据库集群（读写分离，分库分表）**

**大型网站需要存储海量的数据，为达到海量数据存储，高可用，高性能一般采用冗余的方式进行系统设计。一般有两种方式读写分离和分库分表。**

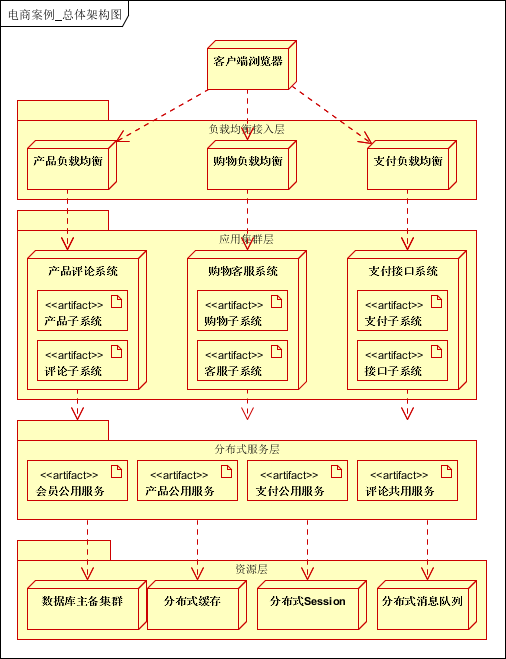
**读写分离：一般解决读比例远大于写比例的场景，可采用一主一备，一主多备或多主多备方式。**

****

**本案例在业务拆分的基础上，结合分库分表和读写分离，如上图：**

* **业务拆分后：每个子系统需要单独的库。**
* **如果单独的库太大，可以根据业务特性，进行再次分库，比如商品分类库，产品库。**
* **分库后，如果表中有数据量很大的，则进行分表，一般可以按照 ID，时间等进行分表；（高级的用法是一致性 Hash）**
* **在分库、分表的基础上，进行读写分离。**

**架构汇总**

****