➢ **客户端（浏览器）缓存**

能够让浏览器缓存的数据一定要缓存；浏览器能够处理的运算，决不放在服务器端来处理。

根据 HTTP 协议特性，修改 Header 参数（Cache-Control、Expires、Pragma、Last-Modified、Etag），让浏览器来缓存页面（一些优秀开发框架会对此做透明的封装，例如：Beetle）http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html。

使用 HTTP1.1 协议，由于 http pipelining 技术特性，能够使用 get 请求的决不采取 post 请求。

为了节约带宽，压缩页面（Content-Encoding: gzip）；页面各个元素能小即小，例如，js 包压缩、js 合并、图片压缩等。

会话状态信息采取 Cookie 代替传统使用服务器 Sessions 对象存储习惯做法；使用 Ajax 实现页面局部刷新。

如果可能，可采取浏览器插件技术突破浏览器功能限制，将原本在服务器端运算，尽量迁到浏览器端。ActiveX/Applet/Flash/….HTML5（最值得期待，它的出现也许会改变整个 Web 世界）。

➢ **前端页面缓存**

访客向网站发出访问请求，由前端页面缓存器负担原服务器的处理进程做出响应，获取原服务器的相应网页内容，将其储存在自身的内存中，与此同时，传送给访客这一缓存的内容；如有另一访客也请求访问之前的相同内容，前端页面缓存器毋须再次获取原服务器上的相应内容，而直接从自身的内存中获取，将这一内容传送给访客。反之，前端页面缓存器也可缓存访客的 GET 和 POST 请求。客户实际面对的是前端页面缓存器，与网站之间的通讯完全由前端页面缓存器反向代理，而非原服务器直接响应客户，这将大大加快访客上网流畅度，有效提升访问量，显著降低带宽占用，减轻原始服务器的繁忙度，加快响应速度，毋须不停地购置大内存，大硬盘，扩容电力设施为服务器端节省成本。可以采用具备缓存功能的 http 反向代理服务器作前端页面缓存器。

➢ **页面片段缓存 ESI(edge side includes)**

ESI 是一个基于 XML 的标记语言，目的是在 HTTP 中组装各种资源。在实际环境中，一个动态生成的页面，当中可能只有少量的内容是频繁变化的或是个性化的，对于传统的 Cache 服务器来说，为了能够保证页面的时效性，却由于页面中这些少量的动态内容而无法将整个页面进行缓存。ESI 通过使用简单的标记语言来对那些可以缓存和不能缓存的网页中的内容片断进行描述，每个网页都被划分成不同的小部分分别赋予不同的缓存控制策略，使 Cache 服务器可以根据这些策略在将完整的网页发送给用户之前将不同的小部分动态地组合在一起。通过这种控制，可以有效地减少从服务器抓取整个页面的次数，而只用从原服务器中提取少量的不能缓存的片断，因此可以有效降低原服务器的负载，同时提高用户访问

的响应时间。

现在解决动态内容缓存的最新技术就是通过 CSI、SSI、ESI 技术来设计网站的内容。

CSI (client side includes)通过 iframe、javascript、ajax 等方式将另外一个页面的内容动态包含进来。它能够利用浏览器客户端并行处理及装载的机制，这种技术基本不需要服务器支持和修改，计算和操作放在客户端，能够降低服务器端压力。

SSI (server side includes)可以通过 HTML 文件中注释行调用的命令或指针。实现整个网站的内容更新。SSI 需要特殊的文件后缀(shtml,inc)。例如：

```
<!--#include virtual="../date.jsp" -->
```

SSI 不受具体语言限制，只需要 Web 服务器或应用服务器支持即可，Ngnix、Apache、IIS、Tomcat、Jboss 等对此都有较好的支持。

ESI 通过使用简单的标记语言来对那些可以加速和不能加速的网页中的内容片断进行描述，每个网页都被划分成不同的小部分分别赋予不同的缓存控制策略，使 Cache 服务器可以根据这些策略在将完整的网页发送给用户之前将不同的小部分动态地组合在一起。通过这种控制，可以有效地减少从服务器抓取整个页面的次数，而只用从原服务器中提取少量的不能缓存的片断，因此可以有效降低原服务器的负载，同时提高用户访问的响应时间。更适合用于缓存服务器上，缓存整个页面或页面片段，因此 ESI 特别适合用于缓存。一般用于缓存服务器如 varnish、squid、Akamai。

ESI 通过使用简单的标记语言来对那些可以加速和不能加速的网页中的内容片断进行描述，每个网页都被划分成不同的小部分分别赋予不同的缓存控制策略，使 Cache 服务器可以根据这些策略在将完整的网页发送给用户之前将不同的小部分动态地组合在一起。通过这种控制，可以有效地减少从服务器抓取整个页面的次数，而只用从原服务器中提取少量的不能缓存的片断，因此可以有效降低原服务器的负载，同时提高用户访问的响应时间。CDN 网络也可以利用在分布全国各地的节点中安装支持 ESI 的 Cache 服务器来提供对网站动态内容提供 CDN 服务。

ESI 需要服务器端支持，常见的有 Apache(mod_esi)、WebLogic、JSP 标签库(JESI)等。

- **本地数据缓存**

关系数据库系统（如 Oracle\MySql）Query Cache 策略：一般以 sql 为 key 来缓存查询结果，尽量不要拼 sql，使用 PreparedStatement 的“？”模式 sql;Query Cache 大小要根据数据库系统具体情况合理设置，过大只会浪费内存，参考值：128 M。

关系数据库系统 Data Buffer 策略：就是数据库数据内存缓存器，其访问命中率决定数据库性能，可根据实际物理内存大小适量增大，如 MySql 建议 buffer 值为物理内存 60%-80%

应用服务器 Cache 包括：对象缓存（例如：对象线程安全，做成单例），更新频率不大

数据考虑缓存（如基表数据、配置文件信息），考虑使用线程池、对象池、连接池等。

- **分布式缓存**

网站业务发展迅速，数据量大幅增大是当前最大的挑战，用户分散各地区，某些地方用户访问响应很慢，影响体验和业务发展；同时，由于数据量过大，数据缓存在本地内存已经不现实，分布式缓存是必然选择了。异地缓存有效解决不同地方用户访问过慢的问题；分库策略带来网站性能整体提升。不过成本也会大幅增加，架构进一步复杂化，也维护难度进一步增大，架构开始臃肿了。主要技术点包括 CDN、分布式缓存、Shard 分库等。

采用分布式缓存方案突破容量限制，具备良好伸缩性；但分布式涉及远程网络通信消耗其性能本地缓存来得优秀，并可涉及节点状态维护及数据复制问题，其稳定性和可靠性是个挑战。目前流行分布式缓存方案有 memcached、membase、redis 等，基本上当前的 NoSQL 方案都可以用来做分布式缓存方案。

分库包括垂直分区和水平分区两种。垂直分库前提是良好的松耦合的模块化设计。水平分区中，Shard 是分布式解决方案，与数据库集中式的表空间分区是两个不同方案。