一、大规模分布式应用之海量数据和高并发解决方案

开发一个网站的应用程序，当用户规模比较小的时候，使用简单的：一台应用服务器+一台数据库服务器+一台文件服务器，这样的话完全可以解决一部分问题，也可以通过堆硬件的方式来提高网站应用的访问性能，当然，也要考虑成本的问题。

当问题的规模在经济条件下通过堆硬件的方式解决不了的时候，我们应该通过其他的思路去解决问题，互联网发展至今，已经提供了很多成熟的解决方案，但并不是都具有适用性，你把淘宝的技术全部都搬过来也不一定达到现在淘宝的水平，道理很简单。

当然，很多文章都在强调，一个网站的发展水平，是逐渐的演变过来的，并不是一朝一夕的事情。虽然目前的情况互联网的泡沫越来越大，但是整个互联网技术的发展确实为我们提供了方便快捷的上网体验。下边是一张早期的淘宝官网的界面：

下边整理的是一些针对海量数据和高并发情况下的解决方案，技术水平有限，欢迎留言指导。

二、针对海量数据和高并发的主要解决方案

海量数据的解决方案：

1. 使用缓存；
2. 页面静态化技术；
3. 数据库优化；
4. 分离数据库中活跃的数据；
5. 批量读取和延迟修改；
6. 读写分离；
7. 使用NoSQL和Hadoop等技术；
8. 分布式部署数据库；
9. 应用服务和数据服务分离；
10. 使用搜索引擎搜索数据库中的数据；
11. 进行业务的拆分；

高并发情况下的解决方案：

1. 应用程序和静态资源文件进行分离；
2. 页面缓存；
3. 集群与分布式；
4. 反向代理；
5. CDN；

三、海量数据的解决方案

（1）使用缓存

网站访问数据的特点大多数呈现为“二八定律”：80%的业务访问集中在20%的数据上。

例如：在某一段时间内百度的搜索热词可能集中在少部分的热门词汇上；新浪微博某一时期也可能大家广泛关注的主题也是少部分事件。

总的来说就是用户只用到了总数据条目的一小部分，当网站发展到一定规模，数据库IO操作成为性能瓶颈的时候，使用缓存将这一小部分的热门数据缓存在内存中是一个很不错的选择，不但可以减轻数据库的压力，还可以提高整体网站的数据访问速度。

使用缓存的方式可以通过程序代码将数据直接保存到内存中，例如通过使用Map或者ConcurrentHashMap；另一种，就是使用缓存框架：Redis、Ehcache、Memcache等。

使用缓存框架的时候，我们需要关心的就是什么时候创建缓存和缓存失效策略。

缓存的创建可以通过很多的方式进行创建，具体也需要根据自己的业务进行选择。例如，新闻首页的新闻应该在第一次读取数据的时候就进行缓存；对于点击率比较高的文章，可以将其文章内容进行缓存等。

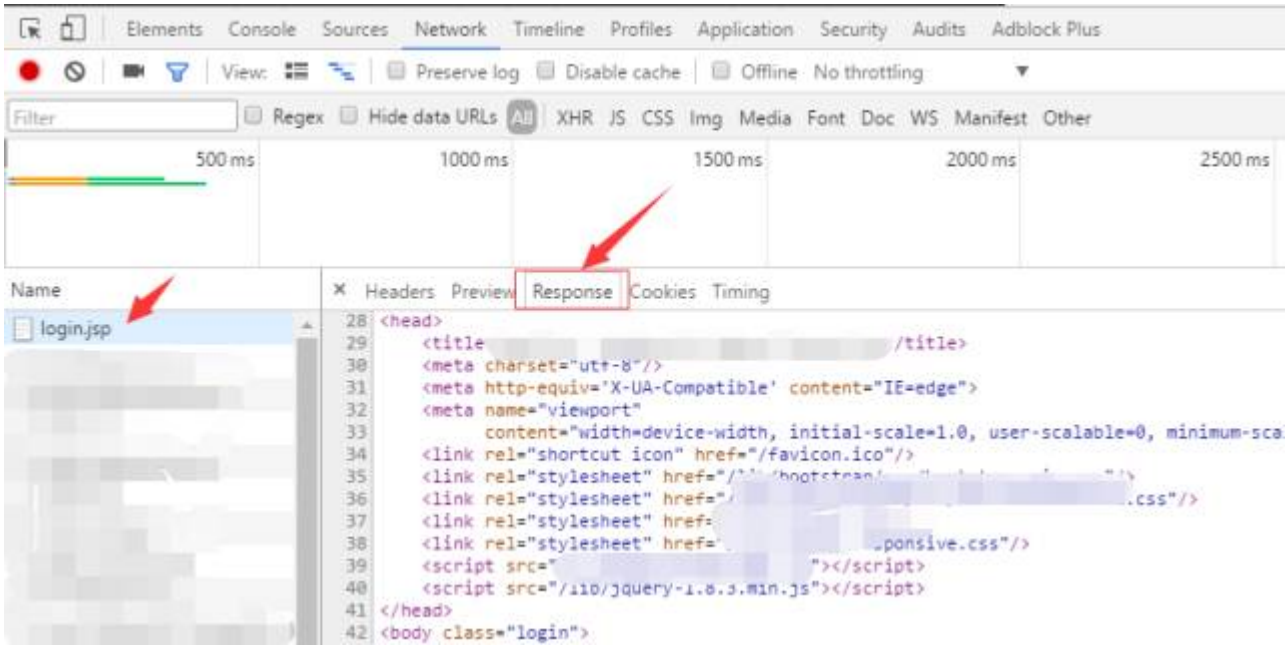
内存资源有限，选择如何创建缓存是一个值得思考的问题。另外，对于缓存的失效机制也是需要好好研究的，可以通过设置失效时间的方式进行设置；也可以通过对热门数据设置优先级，根据不同的优先级设置不同的失效时间等；

需要注意的是，当我们删除一条数据的时候，我们要考虑到删除该条缓存，还要考虑在删除该条缓存之前该条数据是否已经到达缓存失效时间等各种情况！

使用缓存的时候还要考虑到缓存服务器发生故障时候如何进行容错处理，是使用N多台服务器缓存相同的数据，通过分布式部署的方式对缓存数据进行控制，当一台发生故障的时候自动切换到其他的机器上去；还是通过Hash一致性的方式，等待缓存服务器恢复正常使用的时候重新指定到该缓存服务器。Hash一致性的另一个作用就是在分布式缓存服务器下对数据进行定位，将数据分布在不用缓存服务器上。关于数据缓存的Hash一致性也是一个比较打的问题，这里只能大致描述一下，关于Hash一致性的了解，推荐一篇文章：
http://blog.csdn.net/liu765023051/article/details/49408099

（2）页面静态化技术

使用传统的JSP界面，前端界面的显示是通过后台服务器进行渲染后返回给前端游览器进行解析执行，如下图：



当然，现在提倡前后端分离，前端界面基本都是HTML网页代码，通过Angular JS或者NodeJS提供的路由向后端服务器发出请求获取数据，然后在游览器对数据进行渲染，这样在很大程度上降低了后端服务器的压力。

还可以将这些静态的HTML、CSS、JS、图片资源等放置在缓存服务器上或者CDN服务器上，一般使用最多的应该是CDN服务器或者Nginx服务器提供的静态资源功能。

另外，在《高性能网站建设进阶指南-Web开发者性能优化最佳实践（口碑网前端团队 翻译）》这本书中，对网站性能的前端界面提供了一些很宝贵的经验，如下：

- 规则 1：尽量减少 HTTP 请求。
- 规则 2：使用 CDN。
- 规则 3：添加 Expires 头。
- 规则 4：采用 Gzip 压缩组件。
- 规则 5：将样式表放在顶部。
- 规则 6：将脚本放在底部。
- 规则 7：避免 CSS 表达式。
- 规则 8：使用外部的 JavaScript 和 CSS。
- 规则 9：减少 DNS 查询。
- 规则 10：精简 JavaScript。
- 规则 11：避免重定向。
- 规则 12：删除重复的脚本。
- 规则 13：配置 ETag。
- 规则 14：使 Ajax 可缓存。

因此，在这些静态资源的处理上，选择正确的处理方式还是对整体网站性能还是有很大帮助的！

（3）数据库优化

数据库优化是整个网站性能优化的最基础的一个环节，因为，大多数网站性能的瓶颈都是开在数据库IO操作上，虽然提供了缓存技术，但是对数据库的优化还是一个需要认真的对待。一般公司都有自己的DBA团队，负责数据库的创建，数据模型的确立等问题，不像我们现在几个不懂数据库优化的人只能在网上找一篇篇数据库优化的文章，自己去摸索，并没有形成一个系统的数据库优化思路。

对于数据库的优化来说，是一种用技术换金钱的方式。数据库优化的方式很多，常见的可以分为：数据库表结构优化、SQL语句优化、分区、分表、索引优化、使用存储过程代替直接操作等 。

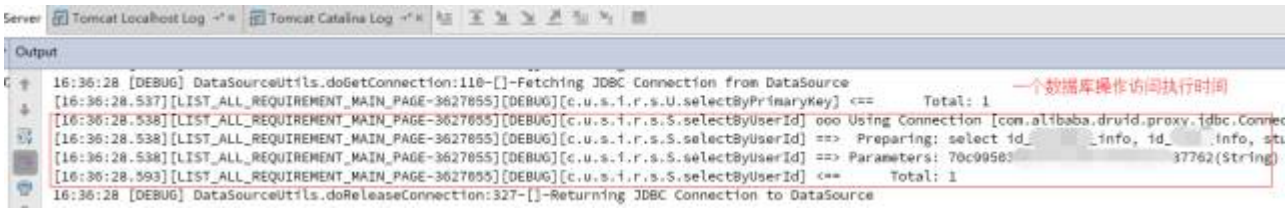
1、表结构优化

对于数据库的 开发规范与使用技巧以及设计和优化，前边的时候总结了一些文章，这里偷个懒直接放地址，有需要的可以移步看一下： a) MySQL开发规范与使用技巧总结：
http://blog.csdn.net/xlgen157387/article/details/48086607 b) 在一个千万级的数据库查寻中，如何提高查询效率？：http://blog.csdn.net/xlgen157387/article/details/44156679

另外，再设计数据库表的时候需不需要创建外键，使用外键的好处之一可以方便的进行级联删除操作，但是现在在进行数据业务操作的时候，我们都通过事物的方式来保证数据读取操作的一致性，我感觉相比于使用外键关联MySQL自动帮我们完成级联删除的操作来说，还是自己使用事物进行删除操作来的更放心一些。当然可能也是有适用的场景，大家如有很好的建议，欢迎留言！

2、SQL优化

对于SQL的优化，主要是针对SQL语句处理逻辑的优化，而且还要根据索引进行配合使用。另外，对于SQL语句的优化我们可以针对具体的业务方法进行优化，我们可以将执行业务逻辑操作的数据库执行时间记录下来，来进行有针对性的优化，这样的话效果还是很不错的！例如下图，展示了一条数据库操作执行调用的时间：



关于SQL优化的一些建议，以前整理了一些，还请移步查看：

a) 19个MySQL性能优化要点解析：http://blog.csdn.net/xlgen157387/article/details/50735269

b) MySQL批量SQL插入各种性能优化：http://blog.csdn.net/xlgen157387/article/details/50949930

分表

分表是将一个大表按照一定的规则分解成多张具有独立存储空间的实体表，我们可以称为子表，每个表都对应三个文件，MYD数据文件，.MYI索引文件，.frm表结构文件。这些子表可以分布在同一块磁盘上，也可以在不同的机器上。数据库读写操作的时候根据事先定义好的规则得到对应的子表名，然后去操作它。

例如：用户表 用户的角色有很多种，可以通过枚举类型的方式将用户分为不同类别category：学生、教师、企业等 ，这样的话，我们就可以根据类别category来对数据库进行分表，这样的话每次查询的时候现根据用户的类型锁定一个较小的范围。

不过分表之后，如果需要查询完整的顺序就需要使用多表操作了。

分区

数据库分区是一种物理数据库设计技术，DBA和数据库建模人员对其相当熟悉。虽然分区技术可以实现很多效果，但其主要目的是为了在特定的SQL操作中减少数据读写的总量以缩减响应时间。

分区和分表相似，都是按照规则分解表。不同在于分表将大表分解为若干个独立的实体表，而分区是将数据分段划分在多个位置存放，可以是同一块磁盘也可以在不同的机器。分区后，表面上还是一张表，但数据散列到多个位置了。数据库读写操作的时候操作的还是大表名字，DMS自动去组织分区的数据。

当一张表中的数据变得很大的时候，读取数据，查询数据的效率非常低下，很容易的就是讲数据分到不同的数据表中进行保存，但是这样分表之后会使得操作起来比较麻烦，因为，将同类的数据分别放在不同的表中的话，在搜索数据的时候需要便利查询这些表中的数据。想进行CRUD操作还需要先找到对应的所有表，如果涉及到不同的表的话还要进行跨表操作，这样操作起来还是很麻烦的。

使用分区的方式可以解决这个问题，分区是将一张表中的数据按照一定的规则分到不同的区中进行保存，这样进行数据查询的时候如果数据的范围在同一个区域内那么就可以支队一个区中的数据进行操作，这样的话操作起来数据量更少，操作速度更快，而且该方法是对程序透明的，程序不需要进行任何的修改。

索引优化

索引的大致原理是在数据发生变化的时候就预先按指定字段的顺序排列后保存到一个类似表的结构中，这样在查找索引字段为条件记录时就可以很快地从索引中找到对应记录的指针并从表中获取到相应的数据，这样速度是很快地。

不过，虽然查询的效率大大提高了，但是在进行增删改的时候，因为数据的变化都需要更新相应的索引，也是一种资源的浪费。

关于使用索引的问题，对待不同的问题，还是需要进行不同的讨论，根据具体的业务需求选择合适的索引对性能的提高效果是很明显的一个举措！

推荐文章阅读：

a) 数据库索引的作用和优点缺点以及索引的11中用法：
http://blog.csdn.net/xlgen157387/article/details/45030829

b) 数据库索引原理：http://blog.csdn.net/kennyrose/article/details/7532032

使用存储过程代替直接操作

存储过程（Stored Procedure）是在大型数据库系统中，一组为了完成特定功能的SQL 语句集，存储在数据库中，经过第一次编译后再次调用不需要再次编译，用户通过指定存储过程的名字并给出参数（如果该存储过程带有参数）来执行它。存储过程是数据库中的一个重要对象，任何一个设计良好的数据库应用程序都应该用到存储过程。

在操作过程比较复杂并且调用频率比较高的业务中，可以将编写好的sql语句用存储过程的方式来代替，使用存储过程只需要进行一次变异，而且可以在一个存储过程里做一些复杂的操作。

（4）分离数据库中活跃的数据

正如前边提到的“二八定律”一样，网站的数据虽然很多，但是经常被访问的数据还是有限的，因此可以讲这些相对活跃的数据进行分离出来单独进行保存来提高处理效率。

其实前边使用缓存的思想就是一个很明显的分离数据库中活跃的数据的使用案例，将热门数据缓存在内存中。

还有一种场景就是，例如一个网站的所用注册用户量很大千万级别，但是经常登录的用户只有百万级别，剩下的基本都是很长时间都没有进行登录操作，如果不把这些“僵尸用户”单独分离出去，那么我们每次查询其他登录用户的时候，就白白浪费了这些僵尸用户的查询操作。

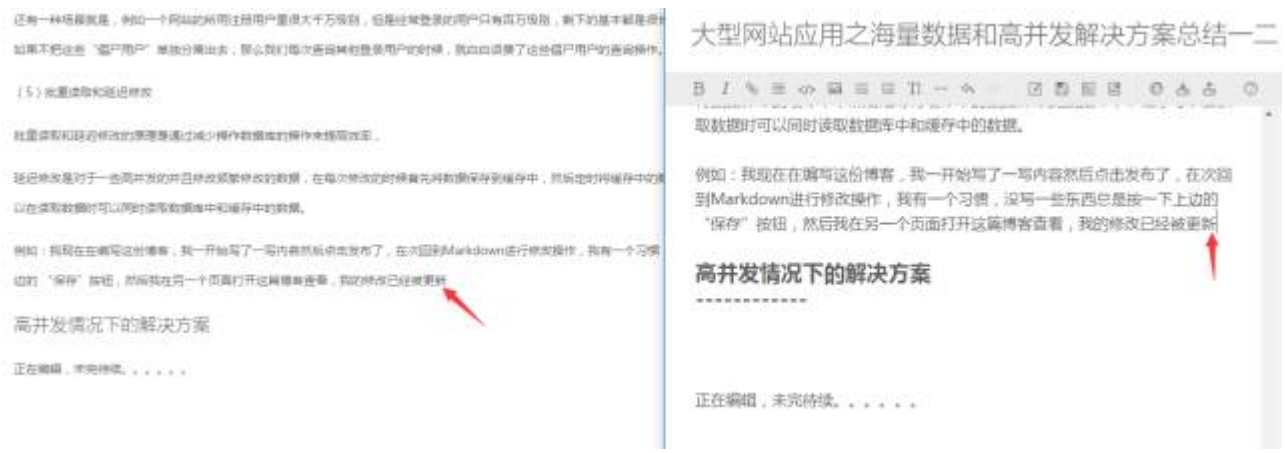
（5）批量读取和延迟修改

批量读取和延迟修改的原理是通过减少操作数据库的操作来提高效率。

批量读取是将多次查询合并到一次中进行读取，因为每一个数据库的请求操作都需要链接的建立和链接的释放，还是占用一部分资源的，批量读取可以通过异步的方式进行读取。

延迟修改是对于一些高并发的并且修改频繁修改的数据，在每次修改的时候首先将数据保存到缓存中，然后定时将缓存中的数据保存到数据库中，程序可以在读取数据时可以同时读取数据库中和缓存中的数据。

例如：我现在在编写这份博客，我一开始写了一写内容然后点击发布了，在次回到Markdown进行修改操作，我有一个习惯，没写一些东西总是按一下上边的 “保存”按钮，然后我在另一个页面打开这篇博客查看，我的修改已经被更新，但是我还在 编辑！



不知道CSDN的技术是不是在我没有点击发布之前，这些数据都是先放到缓存里边的。

（6） 读写分离

读写分离的实质是将应用程序对数据库的读写操作分配到多个数据库服务器上，从而降低单台数据库的访问压力。

读写分离一般通过配置主从数据库的方式，数据的读取来自从库，对数据库增加修改删除操作主库。

相关文章请移步观看： a) MySQL5.6 数据库主从（Master/Slave）同步安装与配置详解：http://blog.csdn.net/xlgen157387/article/details/51331244 b) MySQL主从复制的常见拓扑、原理分析以及如何提高主从复制的效率总结:http://blog.csdn.net/xlgen157387/article/details/52451613

（7）使用NoSQL和Hadoop等技术

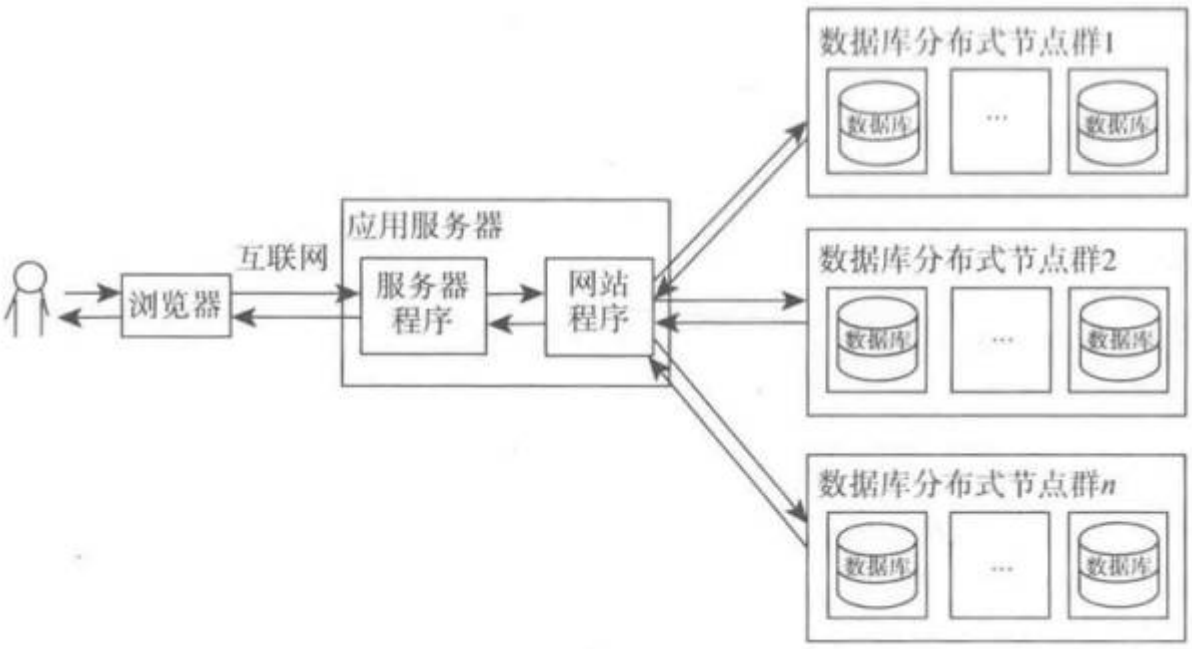
NoSQL是一种非结构化的非关系型数据库，由于其灵活性，突破了关系型数据库的条条框框，可以灵活的进行操作，另外，因为NoSQL通过多个块存储数据的特点，其操作大数据的速度也是相当快的。

（8）分布式部署数据库

任何强大的单一服务器都满足不了大型网站持续增长的业务需求。数据库通过读写分离之后将一台数据库服务器拆分为两台或者多台数据库服务器，但是仍然满足不了持续增长的业务需求。分布式部署数据库是将网站数据库拆分的最后手段，只有在单表数据规模非常庞大的时候才使用。

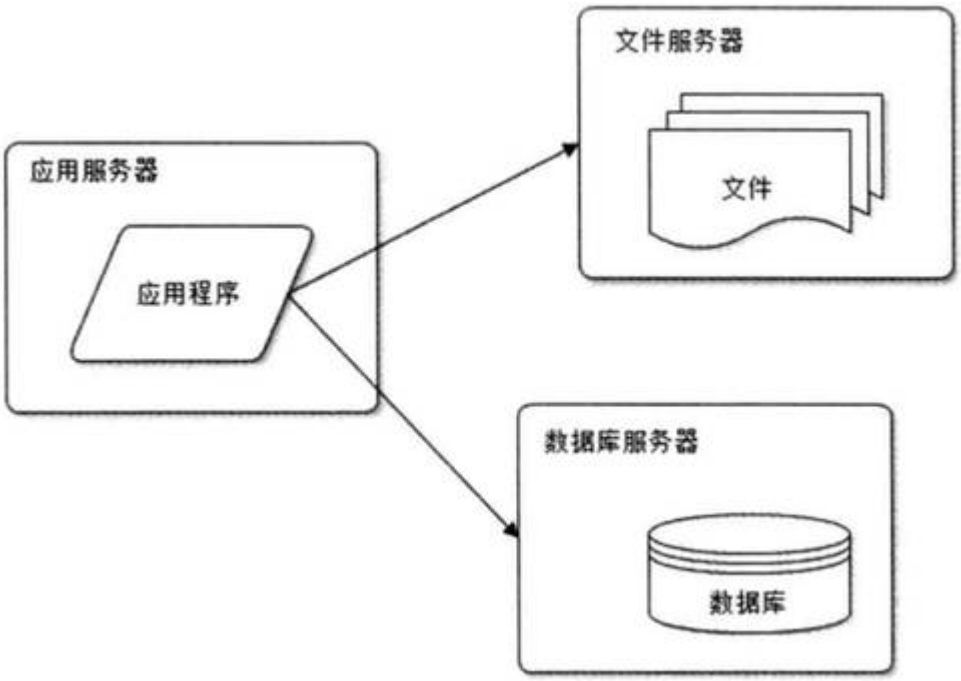
分布式部署数据库是一种很理想的情况，分布式数据库是将表存放在不同的数据库中然后再放到不同的数据库中，这样在处理请求的时候，如果需要调用多个表，则可以让多台服务器同时处理，从而提高处理效率。

分布式数据库简单的架构图如下：



（9）应用服务和数据服务分离

应用服务器和数据库服务器进行分离的目的是为了根据应用服务器的特点和数据库服务器的特点进行底层的优化，这样的话能够更好的发挥每一台服务器的特性，数据库服务器当然是有一定的磁盘空间，而应用服务器相对不需要太大的磁盘空间，这样的话进行分离是有好处的，也能防止一台服务器出现问题连带的其他服务也不可以使用。



（10）使用搜索引擎搜索数据库中的数据

使用搜索引擎这种非数据库查询技术对网站应用的可伸缩分布式特性具有更好的支持。

常见的搜索引擎如Solr通过一种反向索引的方式，维护关键字到文档的映射关系，类似于我们使用《新华字典》进行搜索一个关键字，首先应该是看字典的目录进行查找然后定位到具体的位置。

搜索引擎通过维护一定的关键字到文档的映射关系，能够快速的定位到需要查找的数据，相比于传统的数据库搜索的方式，效率还是很高的。

目前一种比较火的ELK stack技术，还是值得学习的。

一篇关于Solr与MySQL查询性能对比文章： Solr与MySQL查询性能对比：
http://www.cnblogs.com/luxiaoxun/p/4696477.html?utm_source=tuicool&utm_medium=referral

（11） 进行业务的拆分

为什么进行业务的拆分，归根结底上还是使用的还是讲不通的业务数据表部署到不用的服务器上，分别查找对应的数据以满足网站的需求。各个应用之间用过指定的URL连接获取不同的服务，

例如一个大型的购物网站就会将首页、商铺、订单、买家、卖家等拆分为不通的子业务，一方面将业务模块分归为不同的团队进行开发，另外一方面不同的业务使用的数据库表部署到不通的服务器上，体现到拆分的思想，当一个业务模块使用的数据库服务器发生故障也不会影响其他业务模块的数据库正常使用。另外，当其中一个模块的访问量激增的时候还可以动态的扩展这个模块使用到的数据库的数量从而满足业务的需求。

四、高并发情况下的解决方案

（1）应用程序和静态资源文件进行分离

所谓的静态资源就是我们网站中用到的Html、Css、Js、Image、Video、Gif等静态资源。应用程序和静态资源文件进行分离也是常见的前后端分离的解决方案，应用服务只提供相应的数据服务，静态资源部署在指定的服务器上（Nginx服务器或者是CDN服务器上），前端界面通过Angular JS或者Node JS提供的路由技术访问应用服务器的具体服务获取相应的数据在前端游览器上进行渲染。这样可以在很大程度上减轻后端服务器的压力。

例如，百度主页使用的图片就是单独的一个域名服务器上进行部署的

（2）页面缓存

页面缓存是将应用生成的很少发生数据变化的页面缓存起来，这样就不需要每次都重新生成页面了，从而节省大量CPU资源，如果将缓存的页面放到内存中速度就更快。

可以使用Nginx提供的缓存功能，或者可以使用专门的页面缓存服务器Squid。

（3）集群与分布式

（4）反向代理

参考文章请移步至： http://blog.csdn.net/xlgen157387/article/details/49781487

（5）CDN

CDN服务器其实是一种集群页面缓存服务器，其目的就是尽早的返回用户所需要的数据，一方面加速用户访问速度，另一方面也减轻后端服务器的负载压力。

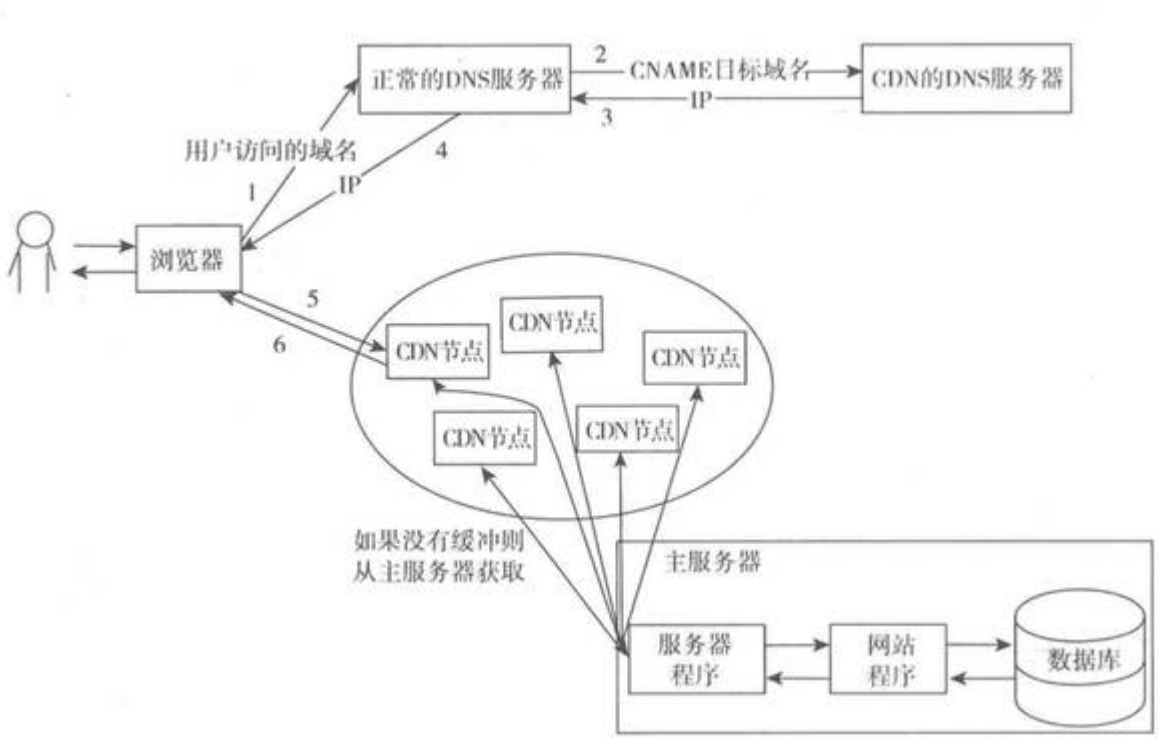
CDN的全称是Content Delivery Network，即内容分发网络。其基本思路是尽可能避开互联网上有可能影响数据传输速度和稳定性的瓶颈和环节，使内容传输的更快、更稳定。

CDN通过在网络各处放置节点服务器所构成的在现有的互联网基础之上的一层智能虚拟网络，CDN系统能够实时地根据网络流量和各节点的连接、负载状况以及到用户的距离和响应时间等综合信息将用户的请求重新导向离用户最近的服务节点上。其目的是使用户可就近取得所需内容，解决 Internet网络拥挤的状况，提高用户访问网站的响应速度。

也就是说CDN服务器是部署在网络运行商的机房，提供的离用户最近的一层数据访问服务，用户在请求网站服务的时候，可以从距离用户最近的网络提供商机房获取数据。电信的用户会分配电信的节点，联通的会分配联通的节点。

CDN分配请求的方式是特殊的，不是普通的负载均衡服务器来分配的那种，而是用专门的CDN域名解析服务器在解析与名的时候就分配好的。

CDN结构图如下所示：



五、总结

文章提到的几点并没有详细的介绍，如需要对其中的一种方式进行研究，可以自行去找资源学习研究，这里只起到抛砖引玉的作用。当然对于大型网站应用之海量数据和高并发解决方案不局限于这些技巧或技术，还有很多成熟的解决方案，有需要的可以自行了解。

特此说明：本文的配图来自《网站架构及其演变过程-韩路彪》，多谢原作者提供的精美配图，并且文章的结构大致也参考了作者的思路，只不过内容是自己的一些经验和学习到的东西进行整理的