软件复用讨论课 02

——复用产品方案

王思尧 1352896

－负载均衡
－分布式
－数据存储
－垃圾消息过滤
－消息队列

负载均衡

程序扩展

对本应用程序采用负载均衡策略的时候，需要考虑到应用程序自身的功能。本应用程序最初的需求是当一个客户端发送消息时其他客户端均可以接收到这个客户端发送的消息，后来程序扩展的功能需要将用户分组，同一组的用户相互可以收发消息。在目前的程序实现中是在内存中存储登录用户，这种实现的基础是用户量极少的情况。

使用负载均衡策略的一个前提是大量用户并发访问，这种情况下将登录用户存储于内存显然是不合理的，一来考虑的内存大小的限制，二来考虑到负载均衡后不同服务器之间的同步问题，一个合理的解决方法是将用户的登录信息存储于数据库，这种方法将导致大规模的数据库访问操作，这里暂且不考虑大规模数据库访问带来的问题。

在选取负载均衡技术实现时，ip层负载均衡于链路层负载均衡均可以满足要求，且各有优劣。需要用心选择的是负载均衡的算法，由于目前应用程序需要维持长链接，而且上次讨论课讨论到的维持长链接的心跳机制在未来程序的实现时也可能需要考虑，因此轮询、随机、最少链接等方法均不适用于这个应用程序，因为这些方法均不能保证一个客户端在登录过程中始终被一台服务器处理从而保持连接，Hash源地址散列的方法可以用于这个应用程序，但是需要作出一定的改动以适应于客户端在登录状态下IP地址改变导致的影响。

Background

————————————————————————————————

负载均衡详解

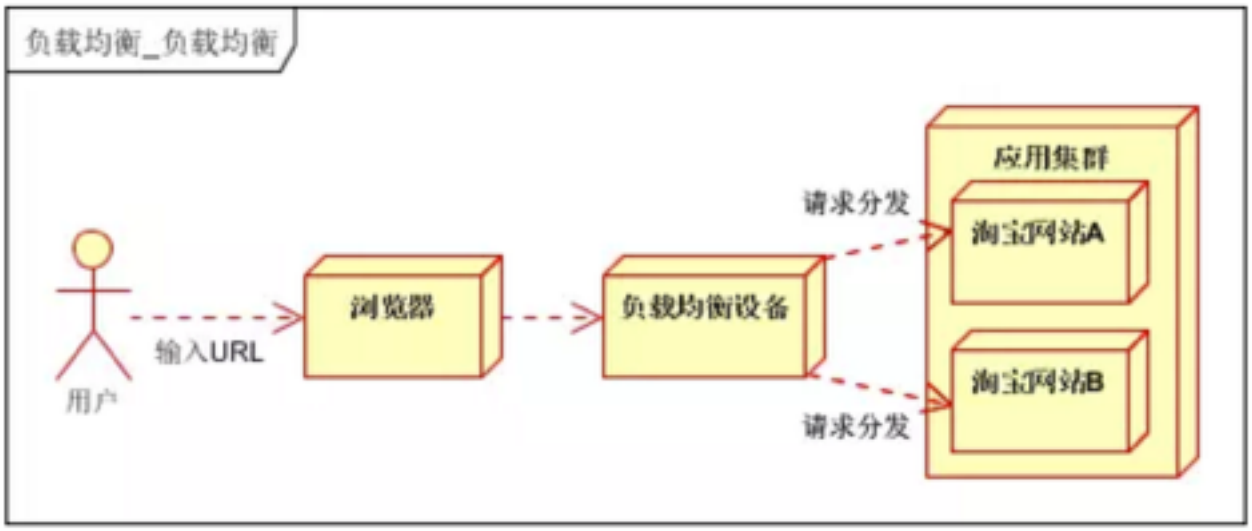
面对大量用户访问、高并发请求，海量数据，可以使用高性能的服务器、大型数据库，存储设备，高性能Web服务器，采用高效率的编程语言比如(Go,Scala)等，当单机容量达到极限时，我们需要考虑业务拆分和分布式部署，来解决大型网站访问量大，并发量高，海量数据的问题。

从单机网站到分布式网站，很重要的区别是业务拆分和分布式部署，将应用拆分后，部署到不同的机器上，实现大规模分布式系统。分布式和业务拆分解决了，从集中到分布的问题，但是每个部署的独立业务还存在单点的问题和访问统一入口问题，为解决单点故障，我们可以采取冗余的方式。将相同的应用部署到多台机器上。解决访问统一入口问题，我们可以在集群前面增加负载均衡设备，实现流量分发。

负载均衡（Load Balance），意思是将负载（工作任务，访问请求）进行平衡、分摊到多个操作单元（服务器，组件）上进行执行。是解决高性能，单点故障（高可用），扩展性（水平伸缩）的终极解决方案。

负载均衡原理

系统的扩展可分为纵向（垂直）扩展和横向（水平）扩展。纵向扩展，是从单机的角度通过增加硬件处理能力，比如CPU处理能力，内存容量，磁盘等方面，实现服务器处理能力的提升，不能满足大型分布式系统（网站），大流量，高并发，海量数据的问题。因此需要采用横向扩展的方式，通过添加机器来满足大型网站服务的处理能力。比如：一台机器不能满足，则增加两台或者多台机器，共同承担访问压力。这就是典型的集群和负载均衡架构：如下图：



应用集群：将同一应用部署到多台机器上，组成处理集群，接收负载均衡设备分发的请求，进行处理，并返回相应数据。

负载均衡设备：将用户访问的请求，根据负载均衡算法，分发到集群中的一台处理服务器。（一种把网络请求分散到一个服务器集群中的可用服务器上去的设备）

负载均衡的作用（解决的问题）：

1.解决并发压力，提高应用处理性能（增加吞吐量，加强网络处理能力）；

2.提供故障转移，实现高可用；

3.通过添加或减少服务器数量，提供网站伸缩性（扩展性）；

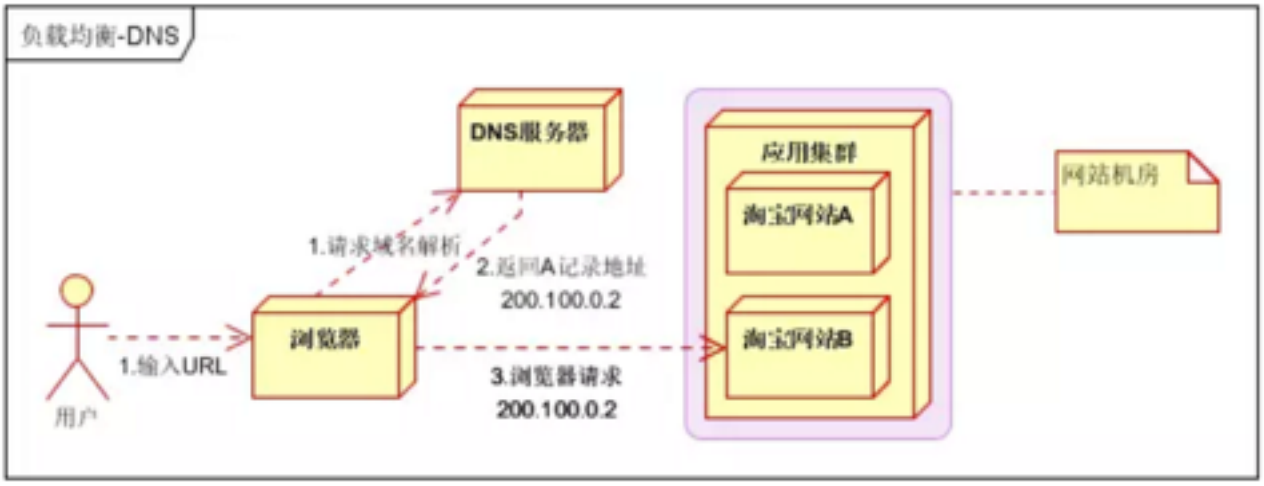
4.安全防护；（负载均衡设备上做一些过滤，黑白名单等处理）

负载均衡分类

根据实现技术不同，可分为DNS负载均衡，HTTP负载均衡，IP负载均衡，链路层负载均衡等。

1.DNS负载均衡

最早的负载均衡技术，利用域名解析实现负载均衡，在DNS服务器，配置多个A记录，这些A记录对应的服务器构成集群。大型网站总是部分使用DNS解析，作为第一级负载均衡。如下图：



优点

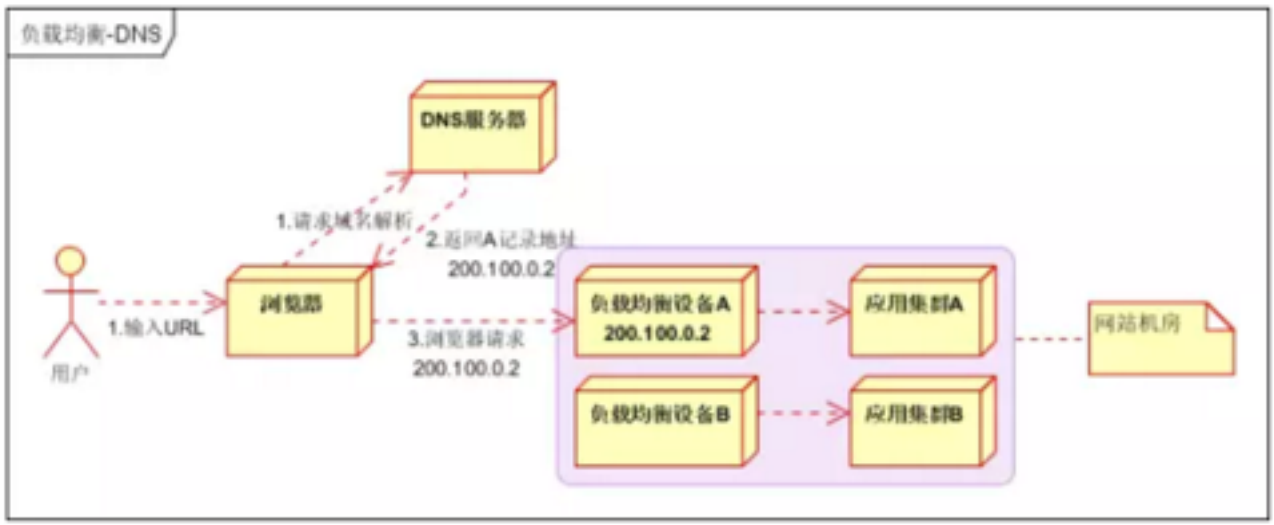
1. 使用简单：负载均衡工作，交给DNS服务器处理，省掉了负载均衡服务器维护的麻烦
2. 提高性能：可以支持基于地址的域名解析，解析成距离用户最近的服务器地址，可以加快访问速度，改善性能；

缺点

1. 可用性差：DNS解析是多级解析，新增/修改DNS后，解析时间较长；解析过程中，用户访问网站将失败；
2. 扩展性低：DNS负载均衡的控制权在域名商那里，无法对其做更多的改善和扩展；
3. 维护性差：也不能反映服务器的当前运行状态；支持的算法少；不能区分服务器的差异（不能根据系统与服务的状态来判断负载）

实践建议

将DNS作为第一级负载均衡，A记录对应着内部负载均衡的IP地址，通过内部负载均衡将请求分发到真实的Web服务器上。一般用于互联网公司，复杂的业务系统不合适使用。如下图

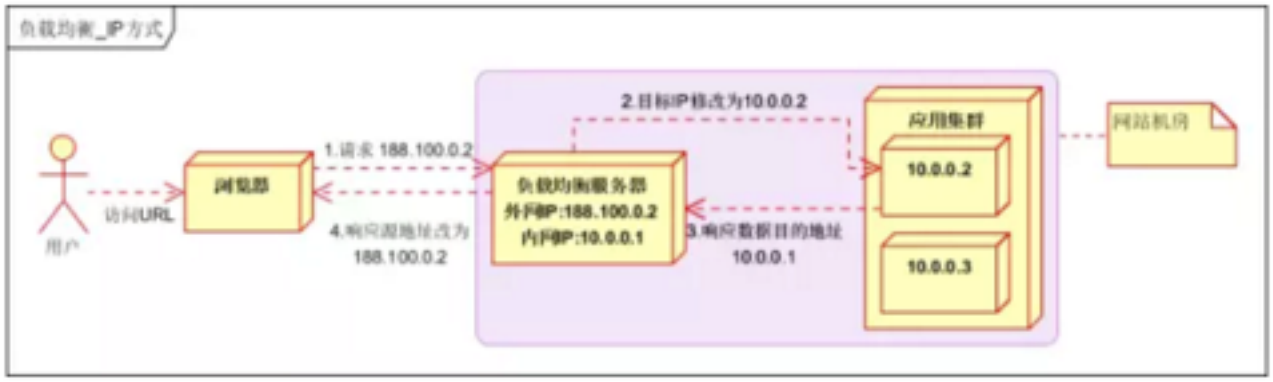


2.IP负载均衡

在网络层通过修改请求目标地址进行负载均衡。

用户请求数据包，到达负载均衡服务器后，负载均衡服务器在操作系统内核进程获取网络数据包，根据负载均衡算法得到一台真实服务器地址，然后将请求目的地址修改为，获得的真实ip地址，不需要经过用户进程处理。

真实服务器处理完成后，响应数据包回到负载均衡服务器，负载均衡服务器，再将数据包源地址修改为自身的ip地址，发送给用户浏览器。如下图：



IP负载均衡，真实物理服务器返回给负载均衡服务器，存在两种方式：（1）负载均衡服务器在修改目的ip地址的同时修改源地址。将数据包源地址设为自身盘，即源地址转换（snat）。（2）将负载均衡服务器同时作为真实物理服务器集群的网关服务器。

优点：

（1）在内核进程完成数据分发，比在应用层分发性能更好；

缺点：

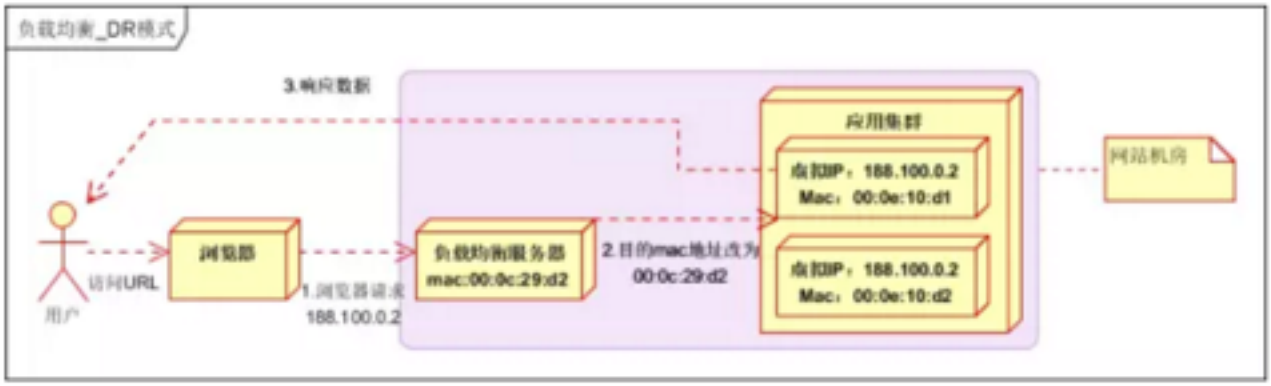
（2）所有请求响应都需要经过负载均衡服务器，集群最大吞吐量受限于负载均衡服务器网卡带宽；

3.链路层负载均衡

在通信协议的数据链路层修改mac地址，进行负载均衡。

数据分发时，不修改ip地址，指修改目标mac地址，配置真实物理服务器集群所有机器虚拟ip和负载均衡服务器ip地址一致，达到不修改数据包的源地址和目标地址，进行数据分发的目的。

实际处理服务器ip和数据请求目的ip一致，不需要经过负载均衡服务器进行地址转换，可将响应数据包直接返回给用户浏览器，避免负载均衡服务器网卡带宽成为瓶颈。也称为直接路由模式（DR模式）。如下图：



优点：性能好；

缺点：配置复杂；

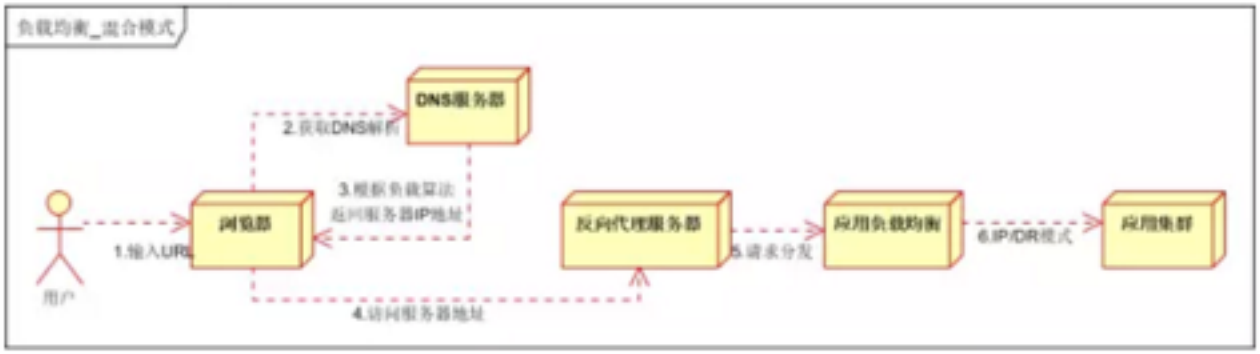
实践建议：DR模式是目前使用最广泛的一种负载均衡方式。

4.混合型负载均衡

由于多个服务器群内硬件设备、各自的规模、提供的服务等的差异，可以考虑给每个服务器群采用最合适的负载均衡方式，然后又在这多个服务器群间再一次负载均衡或群集起来以一个整体向外界提供服务（即把这多个服务器群当做一个新的服务器群），从而达到最佳的性能。将这种方式称之为混合型负载均衡。

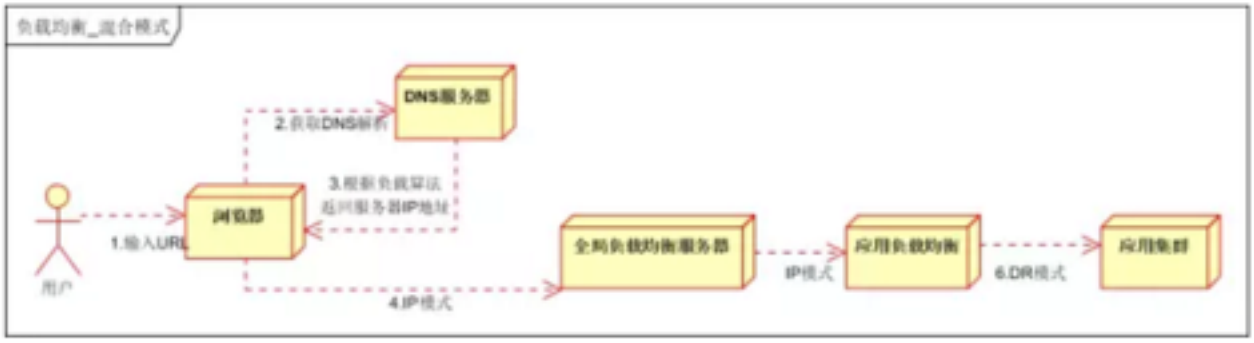
此种方式有时也用于单台均衡设备的性能不能满足大量连接请求的情况下。是目前大型互联网公司，普遍使用的方式。

方式一，如下图：



以上模式适合有动静分离的场景，反向代理服务器（集群）可以起到缓存和动态请求分发的作用，当时静态资源缓存在代理服务器时，则直接返回到浏览器。如果动态页面则请求后面的应用负载均衡（应用集群）。

方式二，如下图：



以上模式，适合动态请求场景。
因混合模式，可以根据具体场景，灵活搭配各种方式，以上两种方式仅供参考。

负载均衡算法
常用的负载均衡算法有，轮询，随机，最少链接，源地址散列，加权等方式；

3.1 轮询
将所有请求，依次分发到每台服务器上，适合服务器硬件同相同的场景。
优点：服务器请求数目相同；
缺点：服务器压力不一样，不适合服务器配置不同的情况；

3.2 随机
请求随机分配到各个服务器。
优点：使用简单；
缺点：不适合机器配置不同的场景；

3.3 最少链接
将请求分配到连接数最少的服务器（目前处理请求最少的服务器）。
优点：根据服务器当前的请求处理情况，动态分配；
缺点：算法实现相对复杂，需要监控服务器请求连接数；

3.4 Hash（源地址散列）
根据IP地址进行Hash计算，得到IP地址。
优点：将来自同一IP地址的请求，同一会话期内，转发到相同的服务器；实现会话粘滞。
缺点：目标服务器宕机后，会话会丢失；

3.5 加权
在轮询，随机，最少链接，Hash等算法的基础上，通过加权的方式，进行负载服务器分配。
优点：根据权重，调节转发服务器的请求数目；
缺点：使用相对复杂；

参考资料
http://mp.weixin.qq.com/s__biz=MjM5NzMyMjAwMA==&mid=403285425&idx=1&sn=46ddf244e3bcb193831c98dc81bb436c&scene=21#wechat_redirect

分布式

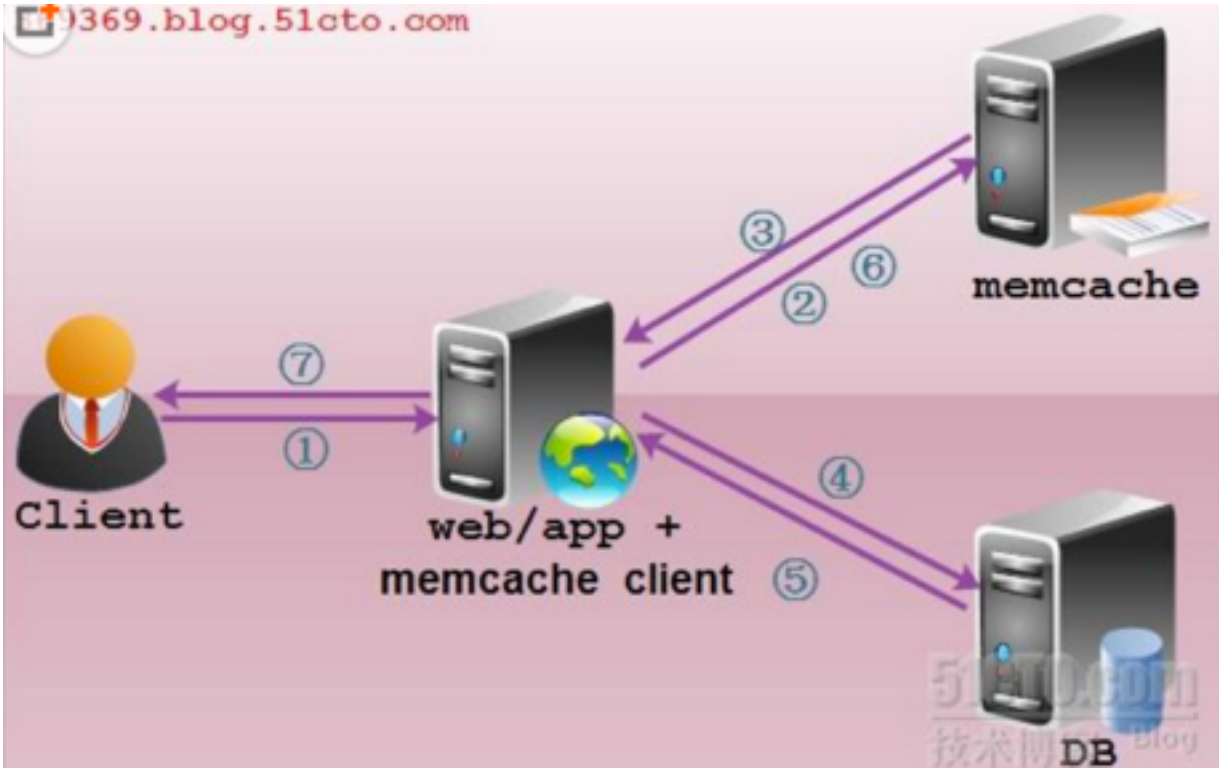
对于一个用户访问量比较大的互联网系统，当用户数达到一定数量时，系统总会存在瓶颈或处理极限，即很难做出快速响应，处理效率逐步低下。对于如何应对用户量不断增长的情形，较直观的方案就是采用分布式系统架构。所谓分布式系统架构，简单的理解就是原来由一台服务器处理的业务，现在分摊给多台服务器处理；原来由一组服务器组处理的业务，现在由多个服务器组处理。

在上述负载均衡的应用中已经基本实现了分布式服务器的框架，但是所带来的大规模数据库并发访问的问题仍未解决。

数据存储

程序扩展
上面提到的负载均衡以及分布式所带来的大规模数据库并发访问的问题需要在数据存储这边进行解决。考虑到以上讨论的情况，即分布式的服务器集群以及数据库的大规模访问，由于每个服务器都需要得到一定的用户的登录情况，而登录情况需要在数据库中存储，在这些过程中一定会涉及到数据的同步问题，而大规模的数据并发访问也会涉及到数据的缓存问题。

————————————————————————————————————
以Memcached为例
Memcached是一个免费开源的，高性能的，具有分布式对象的缓存系统，它可以用来保存一些经常存取的对象或数据，保存的数据像一张巨大的HASH表，该表以Key-value对的方式存在内存中。



操作流程：

1、检查客户端的请求数据是否在memcached中，如有，直接把请求数据返回，不再对数据库进行任何操作，路径操作为①②③⑦。

2、如果请求的数据不在memcached中，就去查数据库，把从数据库中获取的数据返回给客户端，同时把数据缓存一份到memcached中（memcached客户端不负责，需要程序明确实现），路径操作为①②④⑤⑦⑥。

3、每次更新数据库的同时更新memcached中的数据，保证一致性。

4、当分配给memcached内存空间用完之后，会使用LRU（Least Recently Used，最近最少使用）策略加上到期失效策略，失效数据首先被替换，然后再替换掉最近未使用的数据。

Memcached特征：

1.协议简单

它是基于文本行的协议，直接通过telnet在memcached服务器上可进行存取数据操作

2.基于libevent事件处理

Libevent是一套利用C开发的程序库，它将BSD系统的kqueue,Linux系统的epoll等事件处理功能封装成一个接口，与传统的select相比，提高了性能。

3.内置的内存管理方式

所有数据都保存在内存中，存取数据比硬盘快，当内存满后，通过LRU算法自动删除不使用的缓存，但没有考虑数据的容灾问题，重启服务，所有数据会丢失。

4.分布式

各个memcached服务器之间互不通信，各自独立存取数据，不共享任何信息。服务器并不具有分布式功能，分布式部署取决于memcache客户端。

淘宝分布式数据库参考：

相对于银行交易系统的数据强一致性和数据零丢失的要求，互联网企业在传统业务范围（如门户，社交，娱乐，教育等方面），他们更强调的是系统高可用性和系统高容错能力，在业务上能够接受数据不一致，甚至接受数据部分丢失的状况。但是淘宝网做的并不是传统的互联网业务，而正是传

统银行的支付结算类业务，数据不一致也是同样不能容忍的。那么在双十一购物节这一疯狂的交易压力下，他们的交易系统是如何做到数据的一致性和数据零丢失的呢?
淘宝网整个交易系统是个复杂的系统，由各种不同功能的模块组成，比如说其中有负责存储大量小文件的淘宝文件系统（TFS），负责对高频使用的数据进行内存缓存功能的淘宝KV缓存系统（Tair），对图像进行分发以降低网络流量的内容分发网络（CDN），负责内部通讯调度的高性能服务框架（HSF），负责存储海量数据的分布式数据库（OceanBase），负责负载均衡的LVS服务器，负责可靠信息传递的消息中间件系统，负责网站内容展示的网站应用框架（WEBX）。另外还有高性能低消耗的基于X86架构的服务器。在这些技术中，OceanBase分布式数据库是技术集大成者，最关键和最成功的系统之一。

扩展数据库性能

为了提高系统性能，我们往往采取将数据分拆的方式处理，常见的处理办法有以下三种选择:
1、根据业务范围垂直拆分数据库
根据业务范围垂直拆分数据库，支持同一数据库下的跨行跨表事务。根据业务特征，比如信用卡业务、储蓄卡业务、彩票业务、黄金业务等范围，将数据集中分布到不同的数据库上，在一个数据库内实现事务控制功能，跨数据库的事务需求不与支持，需上层应用系统自行保证交易原子的完整性。但我们业务间耦合度比较高，根据业务范围拆分的能力有限，扩展性受限制。因为耦合度高，跨库事务一致性需求强烈存在，如果不支持跨库事务，则程序复杂度将大大提高，从而影响到系统的稳定性和开发的效率。
2、根据业务范围水平拆分数据库
通常的做法是根据某个业务字段，通常取用户编号，哈希后取模，根据取模的结果将数据分布到不同的数据库服务器上，客户端请求通过数据库中间层路由到不同的分区。这种方式目前也存在一定的弊端，如服务器扩展操作复杂，有些范围查询需要访问所有服务器，性能低下。
3、建立一个分布式数据系统
参考分布式表格系统的做法，例如Google Bigtable系统，将大表划分为几万、几十万甚至几百万个子表，子表之间按照主键有序，如果某台服务器发生故障，它上面服务的数据能够在很短的时间内自动迁移到集群中所有的其它服务器。这种方式解决了可扩展性的问题，少量突发的服务器故障或者增加服务器对使用者基本是透明的，能够轻松应对促销或者热点事件等突发流量增长。另外，由于子表是按照主键有序分布的，很好地解决了范围查询的问题。但万事有其利必有一弊，分布式表格系统虽然解决了可扩展性问题，但往往无法支持事务，例如Bigtable只支持单行事务，针对同一个user_id下的多条记录的操作都无法保证原子性。

解决单点性能问题

OceanBase架构的优势在于既支持跨行跨表事务，又支持存储服务器线性扩展。当然，这个架构也有一个明显的缺陷：更新服务器单点。这个问题限制了OceanBase集群的整体读写性能。为了避免单台更新服务器的处理能力形成性能瓶颈，淘宝技术团队在内存、网络、磁盘等方面进行优化。
1、内存操作的优化
根据业务特征，数据库每天更新的次数相对读取次数来说，数量是有限的，同时每次更新的数据量是比较小的，因此一天的修改量大概20GB左右，如果内存数据结构膨胀2倍，占用内存只有40GB。而当前主流的服务器都可以配置96GB内存，一些高档的服务器甚至可以配置192GB，384GB乃至更多内存。因此内存大小方面一般不存在问题，但考虑极端情况，如双十一促销或批量更新大量数据，更新服务器设计成支持当内存表达到一定大小时，可自动或者手工冻结并转储到SSD中，另外支持通过定期合并或者数据分发的方式将更新服务器上的数据分散到集群中所有基准数据服务器中，这样不仅避免了更新服务单机数据容量问题，还能够使得读取操作往往只需要访问更新服务器内存中的数据，避免访问SSD磁盘，提高了读取性能。
2、网络框架的优化

给更新服务器配置四块或更多块的千兆网卡或者万兆网卡，并根据更新服务器全内存操作、收发的网络包一般比较小的特点，对网络框架做了专门的优化，大大提高了每秒收发网络包个数，使得网络不会成为瓶颈。
３、硬盘操作的优化
将数据的写入操作分为两类，一类是数据库事务操作日志写入SAS磁盘，另外一类是其他数据写入SSD硬盘。更新服务器上配置了一块带有缓存模块的RAID卡，操作日志只需要写入到RAID卡的缓存模块即可，该RAID卡再写入到SAS磁盘上。该RAID卡自带电池，在服务器断电时能够确保将缓存中的数据刷入SAS磁盘，不会出现丢数据的情况。另外，更新服务器实现了将多个用户的操作日志组成一组，汇总提交写入，以减少磁盘I/O次数。对于其他数据，则采取批量的顺序写方式写入到SSD硬盘中，通过SSD硬盘高效的读取能力提高读取能力。

参考资料：
http://369369.blog.51cto.com/319630/833234/
http://blog.csdn.net/pengkunstone/article/details/46825829

垃圾消息过滤

Mollom
Mollom是一项Web服务，用于将各种类型的垃圾信息从用户生成的内容中过滤掉，这些内容包括：评论留言、论坛帖子、博客帖子、民意调查、联系表单、登记表单和密码请求表单。确定垃圾信息的依据不仅仅有所发的内容，还有发帖者过去的行为和信誉。Mollom的机器学习算法为你充当24 x 7不间断的数字版主，所以你不必操心。
Mollom使用：

比如说，Drupal等应用程序使用模块（Module）来整合Mollom；模块可以将自己安装到内容编辑集成点上，那样可以在内容写到数据库之前，先检查内容里面有没有垃圾信息。这个过程就像这样：

用户将评论提交到网站上后，对后端服务器进行API调用。

内容经过分析；如果是垃圾信息，就会告知网站阻止内容；或者如果后端服务器不确信，它会建议网站显示验证码，后端服务器同时会提供验证码。

验证码正确填写后，内容将得到接受。在大多数情况下，人是看不到验证码的，内容将直接作为非垃圾信息的正常信息（ham）而被接受。ham是好的内容，而垃圾信息是不好的内容。

只有机器学习算法不是百分之百确信的情况下，才会显示验证码；所以通常来说，不会给人带来不便。

http://os.51cto.com/art/201102/244509.htm

消息队列

消息队列概述
消息队列中间件是分布式系统中重要的组件，主要解决应用耦合，异步消息，流量削锋等问题。实现高性能，高可用，可伸缩和最终一致性架构。是大型分布式系统不可缺少的中间件。

目前在生产环境，使用较多的消息队列有ActiveMQ，RabbitMQ，ZeroMQ，Kafka，MetaMQ，RocketMQ等。

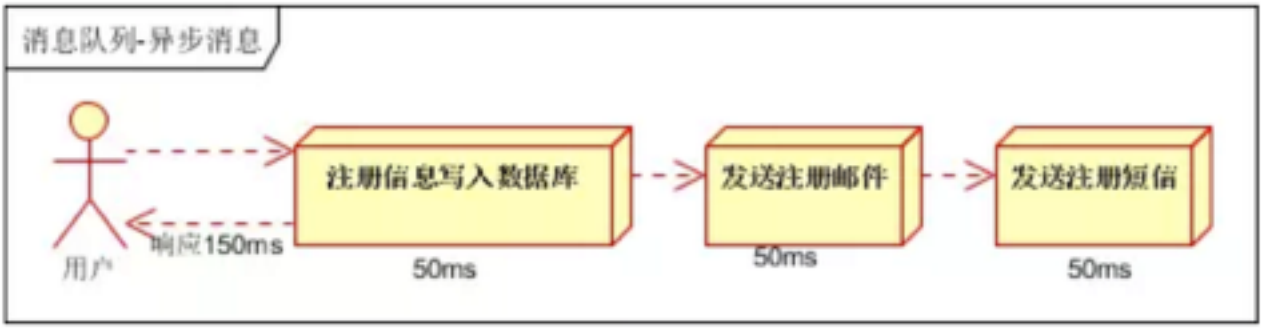
消息队列应用场景

以下介绍消息队列在实际应用中常用的使用场景。异步处理，应用解耦，流量削锋和消息通讯四个场景。

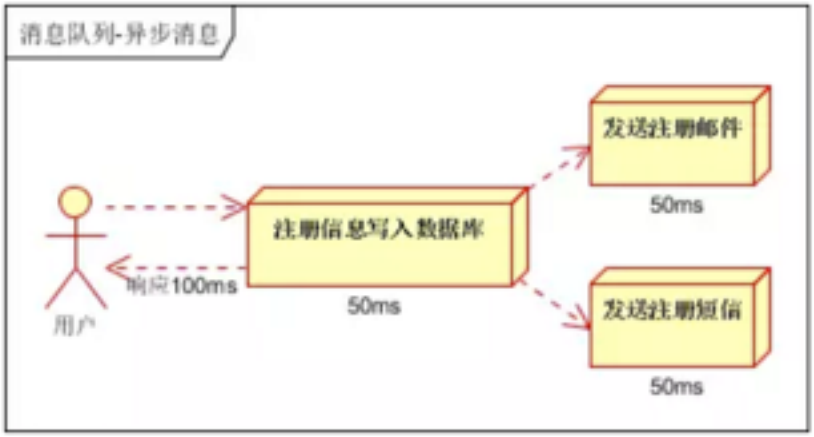
1.异步处理

场景说明：用户注册后，需要发注册邮件和注册短信。传统的做法有两种1.串行的方式；2.并行方式。

（1）串行方式：将注册信息写入数据库成功后，发送注册邮件，再发送注册短信。以上三个任务全部完成后，返回给客户端。



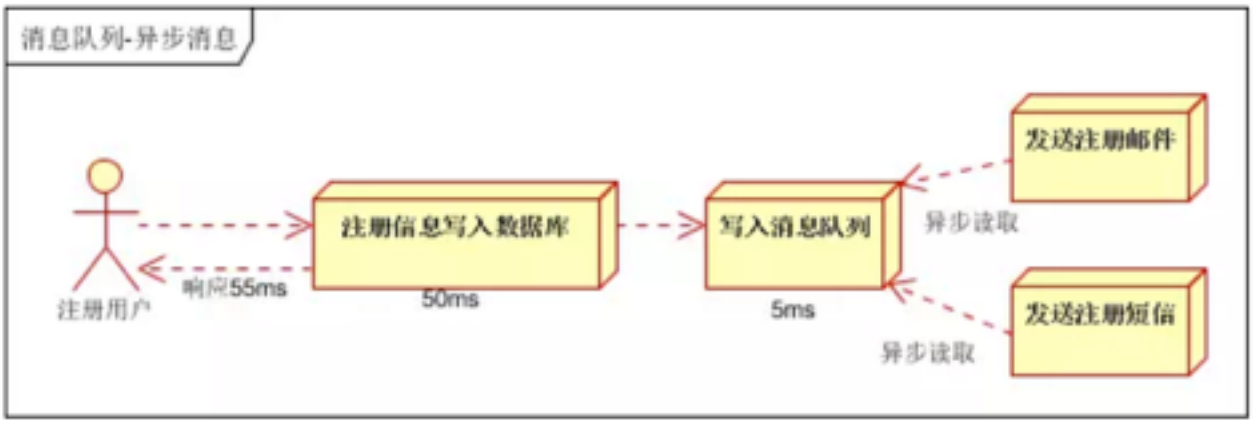
（2）并行方式：将注册信息写入数据库成功后，发送注册邮件的同时，发送注册短信。以上三个任务完成后，返回给客户端。与串行的差别是，并行的方式可以提高处理的时间。



假设三个业务节点每个使用50毫秒钟，不考虑网络等其他开销，则串行方式的时间是150毫秒，并行的时间可能是100毫秒。

因为CPU在单位时间内处理的请求数是一定的，假设CPU1秒内吞吐量是100次。则串行方式1秒内CPU可处理的请求量是7次（1000/150）。并行方式处理的请求量是10次（1000/100）。

引入消息队列，将不是必须的业务逻辑，异步处理。改造后的架构如下：



按照以上约定，用户的响应时间相当于是注册信息写入数据库的时间，也就是50毫秒。注册邮件，发送短信写入消息队列后，直接返回，因此写入消息队列的速度很快，基本可以忽略，因此用户的响应时间可能是50毫秒。因此架构改变后，系统的吞吐量提高到每秒20 QPS。比串行提高了3倍，比并行提高了两倍。

2.应用解耦

场景说明：用户下单后，订单系统需要通知库存系统。传统的做法是，订单系统调用库存系统的接口。

传统模式的缺点：

1） 假如库存系统无法访问，则订单减库存将失败，从而导致订单失败；

2） 订单系统与库存系统耦合；

引入应用消息队列后的方案：

订单系统：用户下单后，订单系统完成持久化处理，将消息写入消息队列，返回用户订单下单成功

库存系统：订阅下单的消息，采用拉/推的方式，获取下单信息，库存系统根据下单信息，进行库存操作。

假如：在下单时库存系统不能正常使用。也不影响正常下单，因为下单后，订单系统写入消息队列就不再关心其他的后续操作了。实现订单系统与库存系统的应用解耦。

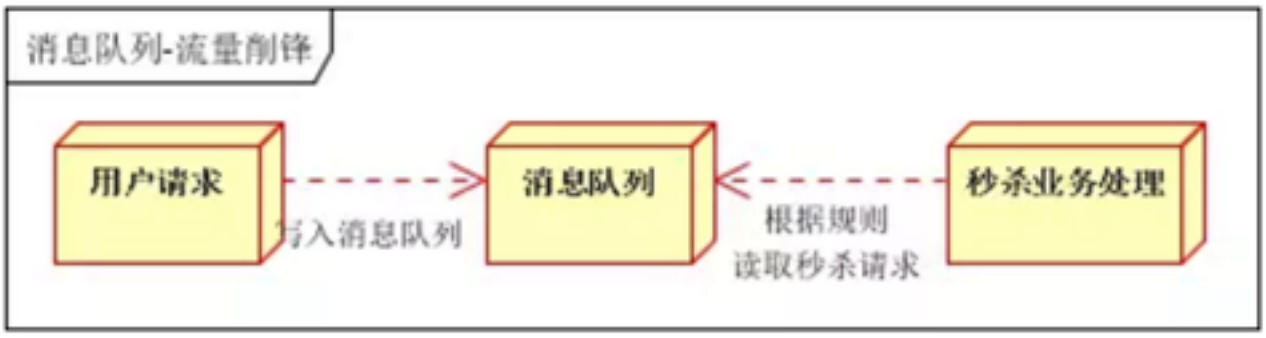
3.流量削锋

流量削锋也是消息队列中的常用场景，一般在秒杀或团抢活动中使用广泛。

应用场景：秒杀活动，一般会因为流量过大，导致流量暴增，应用挂掉。为解决这个问题，一般需要在应用前端加入消息队列。

可以控制活动的人数；

可以缓解短时间内高流量压垮应用；



用户的请求，服务器接收后，首先写入消息队列。假如消息队列长度超过最大数量，则直接抛弃用户请求或跳转到错误页面；

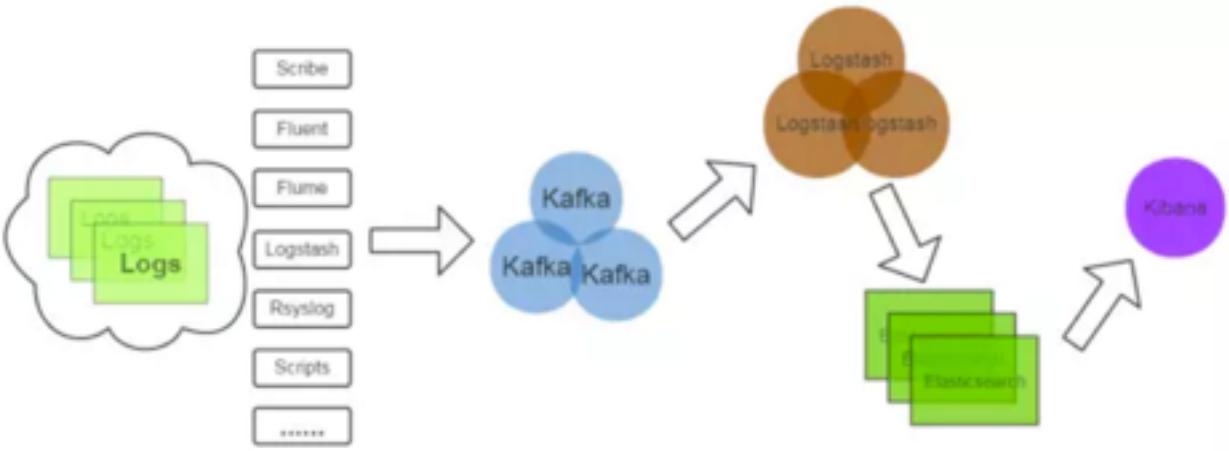
秒杀业务根据消息队列中的请求信息，再做后续处理。

4.日志处理

日志处理是指将消息队列用在日志处理中，比如Kafka的应用，解决大量日志传输的问题。架构简化如下：

日志采集客户端，负责日志数据采集，定时写受写入Kafka队列；
Kafka消息队列，负责日志数据的接收，存储和转发；
日志处理应用：订阅并消费kafka队列中的日志数据；

以下是新浪kafka日志处理应用案例：

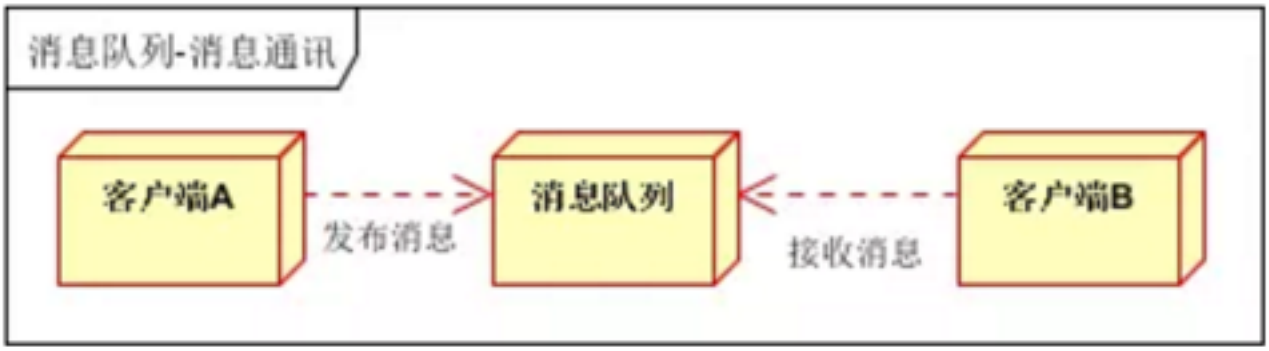


(1)Kafka：接收用户日志的消息队列。
(2)Logstash：做日志解析，统一成JSON输出给Elasticsearch。
(3)Elasticsearch：实时日志分析服务的核心技术，一个schemaless，实时的数据存储服务，通过index组织数据，兼具强大的搜索和统计功能。
(4)Kibana：基于Elasticsearch的数据可视化组件，超强的数据可视化能力是众多公司选择ELK stack的重要原因。

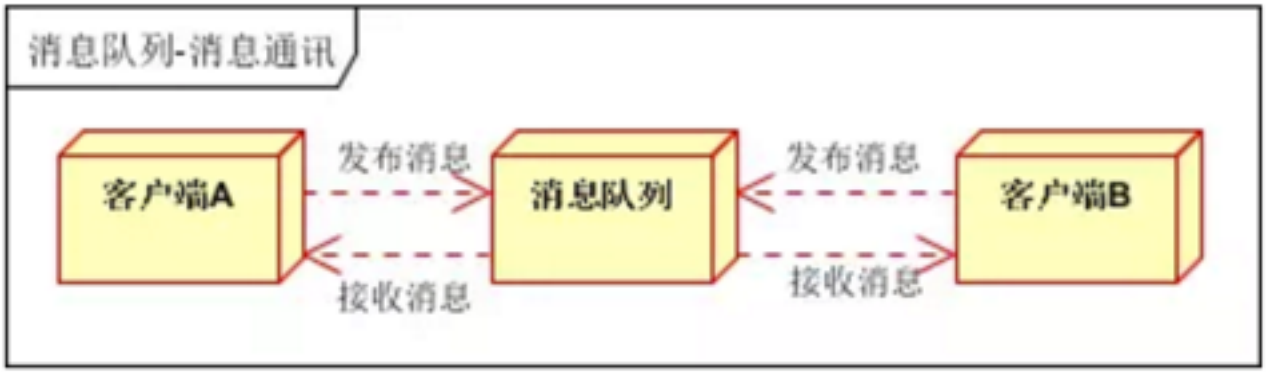
5.消息通讯

消息通讯是指，消息队列一般都内置了高效的通信机制，因此也可以用在纯的消息通讯。比如实现点对点消息队列，或者聊天室等。

点对点通讯：



客户端A和客户端B使用同一队列，进行消息通讯。
聊天室通讯：



客户端A，客户端B，客户端N订阅同一主题，进行消息发布和接收。实现类似聊天室效果。
以上实际是消息队列的两种消息模式，点对点或发布订阅模式。

常用消息队列
以小组使用的**RabbitMQ**为例
RabbitMQ是流行的开源消息队列系统，用erlang语言开发。RabbitMQ是AMQP（高级消息队列协议）的标准实现。支持多种客户端，如：Python、Ruby、.NET、Java、JMS、C、PHP、ActionScript、XMPP、STOMP等，支持AJAX，持久化。用于在分布式系统中存储转发消息，在易用性、扩展性、高可用性等方面表现不俗。
结构图如下：