# [架构设计：负载均衡层设计方案（1）——负载场景和解决方式](http://blog.csdn.net/yinwenjie/article/details/46605451)

在上一篇[《标准Web系统的架构分层》](http://blog.csdn.net/yinwenjie/article/details/46480485%20%E6%A0%87%E5%87%86Web%E7%B3%BB%E7%BB%9F%E7%9A%84%E6%9E%B6%E6%9E%84%E5%88%86%E5%B1%82" \t "_blank)文章中，我们概述了WEB系统**[架构](http://lib.csdn.net/base/architecture" \o "大型网站架构知识库" \t "_blank)**中的分层架设体系，介绍了包括负载均衡层、业务层、业务通信层、数据存储层的作用和存在意义。从本篇文章开始，我们将首先详细讲解负载均衡层的架构原理和选型场景。

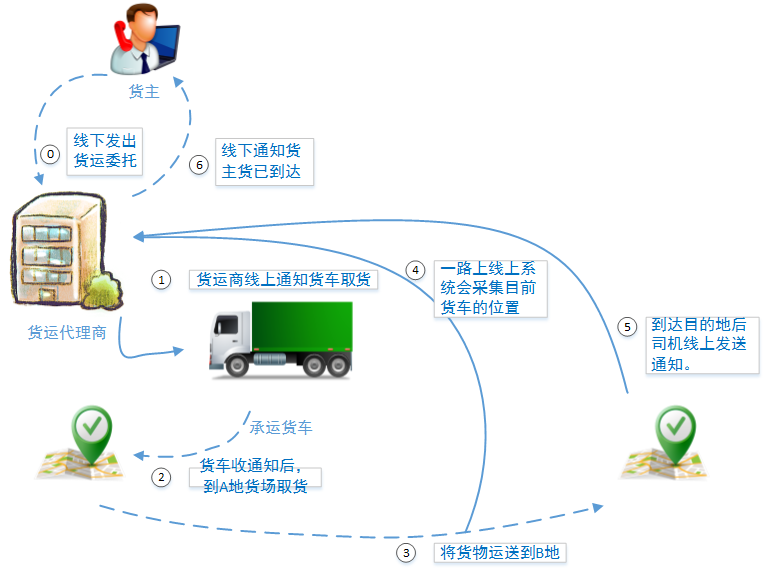
1、不同的负载场景

我们知道负载均衡层的作用是“将来源于外部的处理压力通过某种规律/手段分摊到内部各个处理节点上”，那么不同的业务场景需要的负载均衡方式又是不一样的，架构师还要考虑架构方案的成本、可扩展性、运维难易度等问题。下面我们先介绍几种典型的不同业务场景，大家也可以先想一下如果是您，会怎么架设这些场景的负载均衡层。

需要注意的是，这个系统的文章，我们都将使用这几个典型的业务场景来讲解系统架构的设计递归设计。在后续几篇介绍负载层架构的文章中，我们也将通过这几个典型的业务场景讲解负载层的架构方案。

1.1、负载场景一

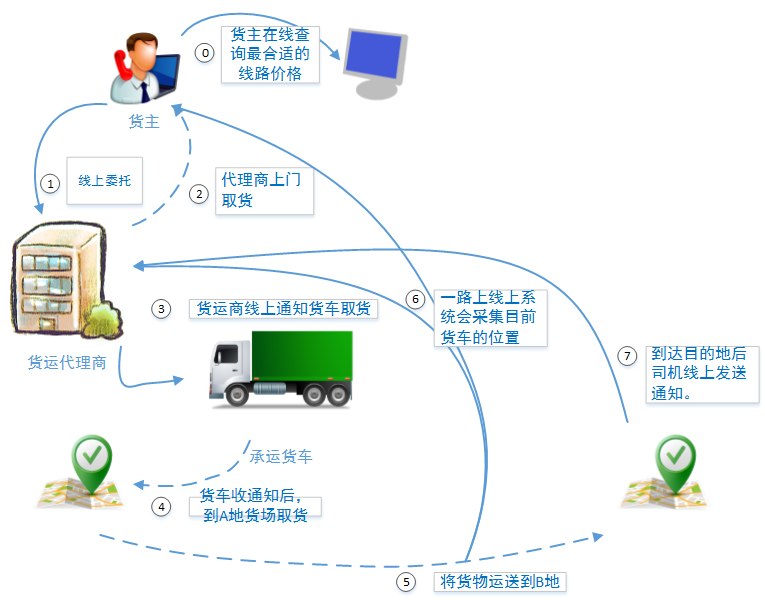
这是一个国家级物流园区的货运订单和物流管理系统。在物流园区内的货运代理商、合作司机（货运车辆）、园区管理员和客服人员都要使用这套系统。每日RUV在1万人次，日PV在10万左右。甲方总经理使用这套系统的缘由是抱着“试一下移动互联网对物流产品是否能起到提高效率的作用”。可以看出整个系统基本上没有访问压力，甲方对您设计的系统只有一个要求：能够保证系统以后的功能和性能扩展性。



1.2、负载场景二

效果不错！在第一版系统架设后的6个月，货场丢货的情况大大减少，并且由于货车在途情况的监控，按时到达率也显著提升，货车司机也反映由于整个货场货车信息都是共享的，货车的待货时间也明显缩短。在这期间物流园中越来越多的货运代理商、货车司机都开始使用这套系统了，整个系统的访问量成线性增长。

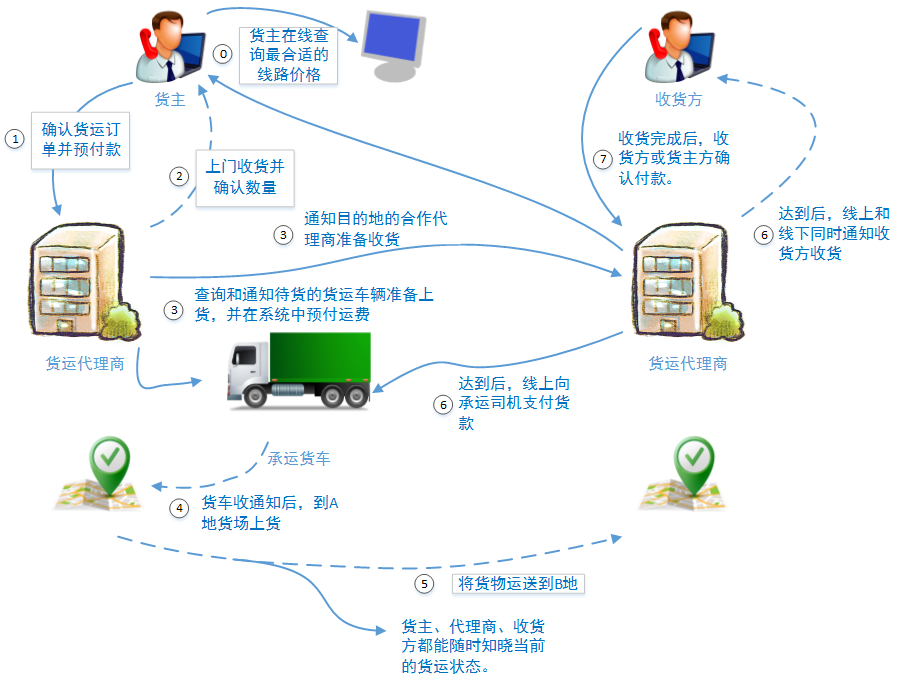
物流园的总经理对整个系统的作用感到满意，决定扩大系统的使用范围，并增加新的功能。经过讨论甲方最终决定把整套系统开放给货主：或者可以在系统上查看货运代理商的线路报价、线上通知代理商上门取货、监控目前自己货品的运输状态、了解第三方签收情况。初步估计系统的日RUV将达到10万，日PV将突破50万。



1.3、负载场景三

一年后，赞不绝口的大宗货品运输服务质量终于传到了政府领导的耳朵里。省里分管运输的领导亲自领队到物流园区参观考察，最终决定由省政府牵头，各地方政府参与，将这套管理办法在整个省级范围进行推广使用。全省10家大型物流园和50家二级物流园中的上万货运代理商、散落省内的零散代理商、10万个人/企业货主、40万优等资质车源共同接入系统。

新的功能上，增加了费用结算和运费保障功能，从货主预付款开始到第三方确认收货的整个环节都进行费用管理。为了保证线上收货环节的顺利，新版本中还增加了代理商之间的合作收货功能。新系统的日RUV将超过50万，日PV将突破250万。



1.4、负载场景四

服务效应、经济效应、口碑效应不断发酵，经过近两年多的发展，目前这套系统已经是省内知名的物流配送平台，专门服务大宗货运物流。联合政府向全国推广服务的时机终于到来。预计全国1000多个物流园区，50万左右物流代理商，500万货运车辆、数不清的个人和企业货主都将使用该系统。预估的RUV和PV是多少呢？无法预估，如果按照全国32省来进行一个简单的乘法，是可以得到一个大概的值（50万 \* 32 = 1500万+；500万 \* 32 = 1.5亿+，已经超过了JD.com的平峰流量），但是各省的物流业规模是不一样的，从业者数量也不一样，所以这样的预估并不科学。而且再这样的系统规模下我们应该更多的考虑系统的峰值冗余。

业务功能的情况：为了保证注册货车的有效性，您所在的公司被政府允许访问政府的车辆信息库，在车辆注册的过程中进行车辆信息有效性的验证（第三方系统接口调用，我们并不知道第三方系统是否能够接收一个较高水平的并发量，所以这个问题留给我我们的架构师，我们将在业务层讲解时进行详细的描述）。

1.5、沉思片刻

看到这里，我们已经将几个递进的业务场景进行了详细的说明（甚至在后文中我们讨论业务层、业务通信层、数据存储层时所涉及的业务场景也不会有什么大的变化了）。看客们看到这里，可以稍作休息，先想想如果是您，您会如何搭建负载层，甚至整个系统的顶层架构。

由于整个系统的性能除了和硬件有关外，业务层的拆分规则，代码质量，缓存技术的使用方式，[**数据库**](http://lib.csdn.net/base/mysql)的优化水平都可能对其产生影响。所以：

**我们在讨论负载层的几篇文章中，我们要假设系统架构中各层的设计都没有对系统性能产生瓶颈**

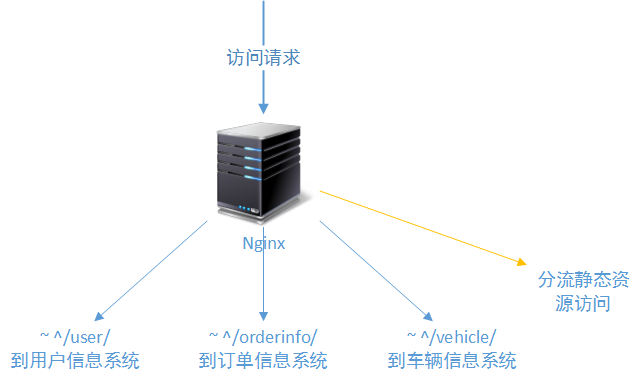
如果您已经思考好了，那么可以继续看以下的内容。

2、负载方案构想

2.1、解决方案一：独立的Nginx/Haproxy方案

很显然，第一个业务场景下，系统并没有多大的压力就是一套简单业务系统，日访问量也完全没有“有访问压力”这样的说法。但是客户有一个要求值得我们关注：要保证系统以后的功能和性能扩展性。为了保证功能和性能扩展性，在系统建立之初就要有一个很好的业务拆分规划，例如我们首先会把用户信息权限子系统和订单系统进行拆分，独立的车辆信息和定位系统可能也需要拆分出来（me：按照微服务架构，将每个子系统做成一个个的服务）。

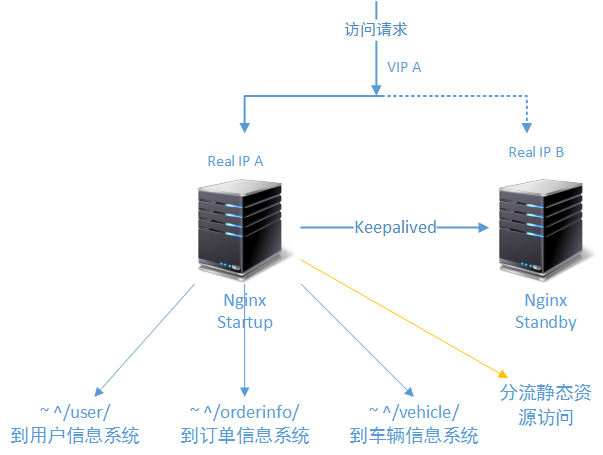
这也是我们在系统建立时就要引入负载均衡层的一个重要原因。也是负载均衡层的重要作用之一。如下图所示：



可以看出，这时负载均衡层只有一个作用，就是按照设定的访问规则，将访问不同系统的请求转发给对应的系统，并且在出现错误访问的情况下转发到错误提示页面。

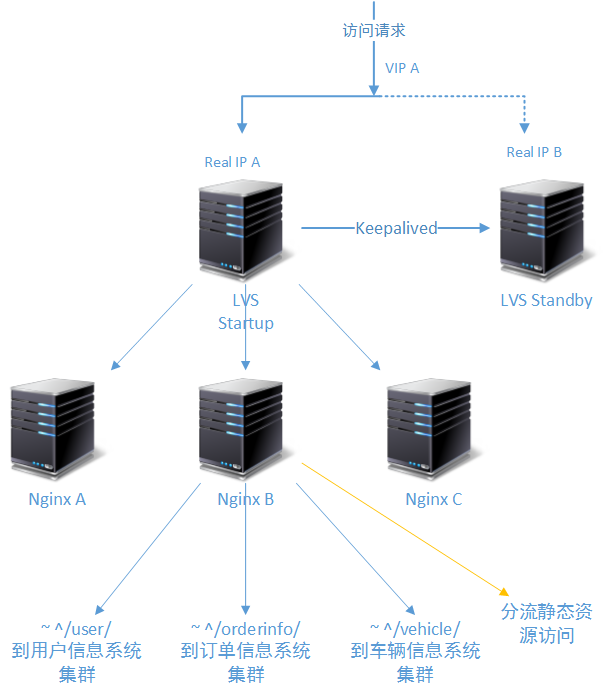
2.2、解决方案二：Nginx/Haproxy + Keepalived方案

此后，系统的访问压力进一步加大，系统的稳定性越来越受到我们的关注。所以在单节点处理还能满足业务要求的情况下，我们为负载层（还有各层）引入热备方案，以保证一个节点在崩溃的情况下，另一个节点能够自动接替其工作（me：即同时部署两个或多个节点），为工程师解决问题赢得时间。如下图所示：



2.3、解决方案三：LVS（DR）+ Keepalived+ Nginx方案

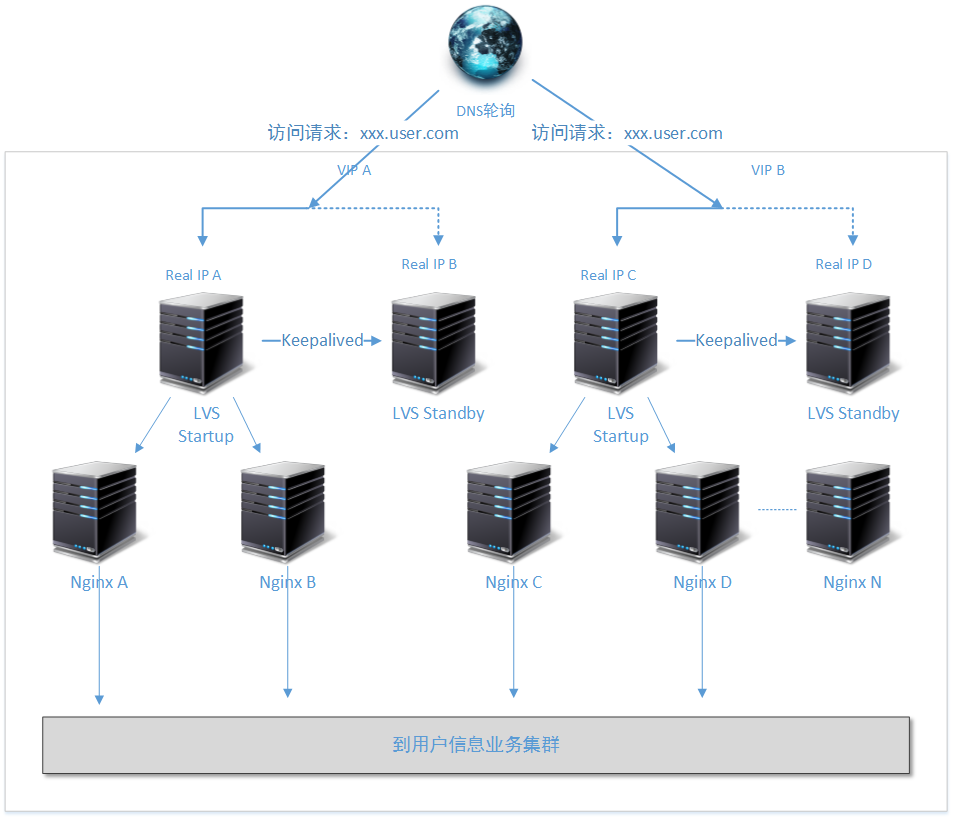
在第三版本架构方案中，为了保证负载层足够稳定的状态下，适应更大的访问吞吐量还要应付可能的访问洪峰，我们加入了LVS技术。LVS负责第一层负载，然后再将访问请求转发到后端的若干台Nginx上（me：即在解决方案一的基础上增加多个nginx服务器，再结合方案二的keepalived，其实这时的nginx是一个集群，就没有必要使用keepalived策略了）。LVS的DR工作模式，只是将请求转到后端，后端的Nginx服务器必须有一个外网IP，在收到请求并处理完成后，Nginx将直接发送结果到请求方，不会再经LVS回发（具体的LVS工作原理介绍将在后文中详细介绍）。



这里要注意的是：

* 有了上层的LVS的支撑Nginx就不再需要使用Keepalived作为热备方案。因为首先Nginx不再是单个节点进行负载处理，而是一个集群多台Nginx节点；另外LVS对于下后端的服务器自带基于端口的健康检查功能；
* LVS是单节点处理的（me：即LVS不能做成集群模式），虽然LVS是非常稳定的，但是为了保证LVS更稳定的工作，我们还是需要使用Keepalived为 LVS做一个热备节点，以防不时之需。

2.4、解决方案四：DNS轮询 + LVS（DR）+ Keepalived + Nginx方案



场景四中，为了满足平均上亿的日PV访问，在对业务进行外网暴露的基础上，我们在互联网的最前端做了一个DNS轮询。然后将（对用户信息系统）访问压力首先分摊到两个对称LVS组上（me：两组LVS同时启动，分摊访问压力），再由每个组向下继续分拆访问压力。

注意上图的负载层方案的不同：

* 首先我们不在像前面的方案中，使用目录名分割业务系统了，而是直接将业务系统的访问使用不同的二级域名进行拆分。这样的变化有利于每个业务系统都拥有自己独立的负载均衡层。
* 请注意上图中的细节，这个负载均衡层是专门为“用户信息子系统”提供负载均衡支撑的，而可能还存在的“订单子系统”、“车辆信息子系统”都会有他们独立的负载均衡层。
* 在LVS下方的Nginx服务可以实现无限制的扩展，同样的就像场景三种所给出的解决方案一样，Nginx本身不在需要Keepalived保持热备，而是全部交由上层的LVS进行健康情况检查。而即使有一两台Nginx服务器出现故障，对整个负载集群来说问题也不大。

方案扩展到了这一步，LVS层就没有必要再进行扩展新的节点了。为什么呢？根据您的业务选择的合适的LVS工作模式，两个LVS节点的性能足以支撑地球上的所有核心WEB站点。如果您对LVS的性能有疑惑，请自行谷歌百度。这里我们提供了一份参考资料：《LVS性能，转发数据的理论极限》[http://www.zhihu.com/question/21237968](http://www.zhihu.com/question/21237968" \t "_blank)

3、为什么没有独立的LVS方案

在下一篇文章《架构设计：负载均衡层设计方案（2）——LVS、keepalived（me:热备）、Nginx安装和核心原理解析》中我们将提到这个问题。实际上通过本篇文章的架构演进分析，一些读者都可以看出端倪。如果用一句话说明其中的原因，那就是LVS为了保证其性能对配置性有所牺牲，单独使用的话往往无法满足业务层对负载层灵活分配请求的要求。

4、说明

4.1、术语说明

* TPS: 衡量业务层处理性能的重要指标（每秒钟request/事务的处理数量，total per second）。业务服务处理一个完整的业务过程，并向上层返回处理结果的过程就是一个 request/事务。那么在一秒钟内整个业务系统能够完成多少个这样的过程，其衡量单位就是TPS。TPS不但和系统架构有很大的关系（特别是业务层和业务通信层的架构祥泰），和物理环境、代码质量的关系也非常密切。
* PV: 网页浏览数是评价网站流量最常用的指标之一，简称为PV。Page Views中的Page一般是指普通的html网页，也包含**[PHP](http://lib.csdn.net/base/php" \o "PHP知识库" \t "_blank)**、jsp等动态产生的html内容。注意是完整的显示一个Page成为一个PV。但是一个PV，一般需要多次HTTP请求，以便获取多个静态资源，这是需要注意的。
* UV: Unique Visitor 一个独立IP，在一个单位时间内（例如一日/一小时）对系统的一个PV请求，成为一个UV（重复的PV不在进行计数，而是采用RUV来记录一个独立IP的重复访问）。
* RUV: Repeat User Visitor 一个独立用户，在一个单位时间内（例如一日/一小时）对系统的一个PV请求，并且重复的访问要进行计数。

4.2、后文介绍

下篇文章《架构设计：负载均衡层设计方案（2）——LVS、keepalived、Nginx安装和核心原理解析》中，我们将提到LVS的工作原理和Nginx中核心负载策略的工作原理（Hash、轮询、权重），以及独立的LVS、Nginx、keepalived的安装方式（虽然安装方式在网络上一个搜索就能搜出来一大把，但是很多都是抄袭，且没有经过详细的正误检查，所以我觉得还是需要重新进行一下说明）