12306NgWeb 概要设计

V0.2.1

Radium

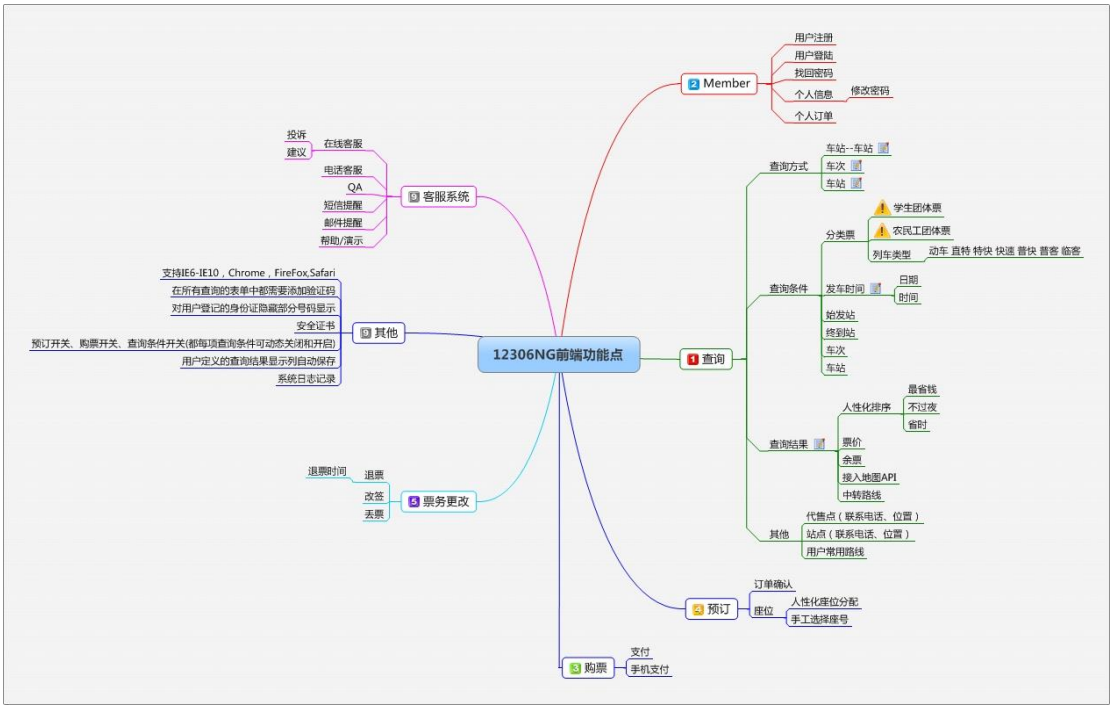
目录

NgWeb 总览

NgWeb 是第一层接收到客户请求的地方，面对客户的海量请求，web 层不可避免的冲在最前面。web 层的主要目的就是要能够将海量请求分流，稳定的将请求转移给后端程序进行处理然后返回给客户。

NgWeb 将使用多种负载均衡技术来分流数据；采用全站页面静态化来减少 WebServer 负载；压缩数据，减小流量负载；依靠硬件资源和正确的负载均衡方案来解决效率；而在稳定性方面，选择 java 作为开发语言，使用 spring 框架进行开发，虽然并不是最高效的 web 解决方案，但却是最稳定的方案。

NgWeb 功能纵览



NgWeb 主要模块

一、静态资源处理

由有 nginx 负载静态资源的处理。

二、动态交易处理

处理用户所有动态交易请求，如车次查询，购票等。我们只负责转发用户的请求，真实的业务逻辑将有后端其他服务器处理。此模块要求具有高扩展能力，以便解决需求变更的问题。

三、内容分发

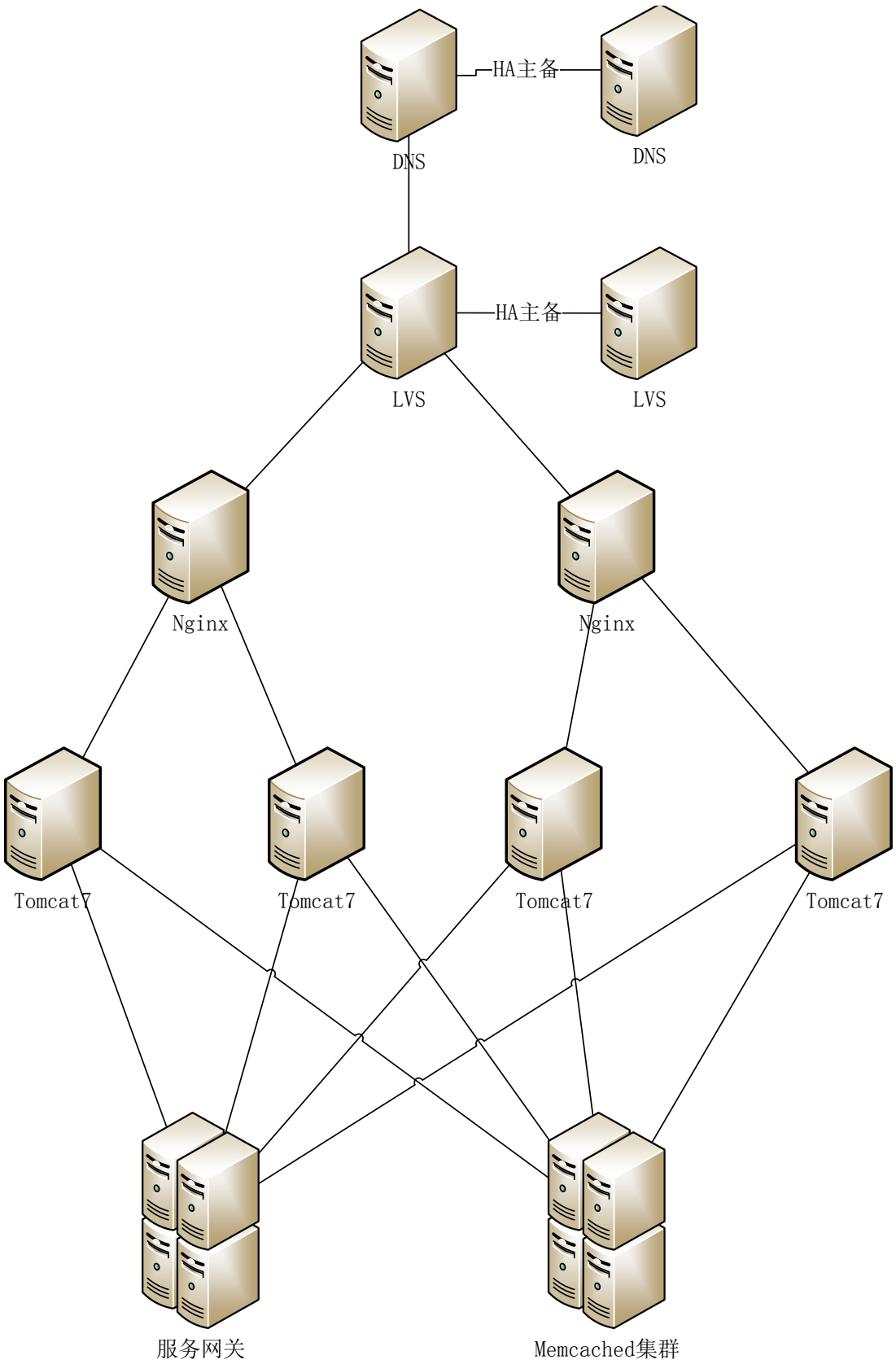
基于模板的静态页生成功能，同时可以将生成的静态页面分发到各个静态资源服务器中。

四、系统监控

监控平台将负责监控整个 NgWeb 的硬件和软件运行情况，随时了解系统的健康度，具有一定的自动调节能力。同时提供日志收集、分析功能。

概要设计

一、 前端负载均衡


HA主备
DNS
DNS
HA主备
LVS
LVS
Nginx
Nginx
Tomcat7
Tomcat7
Tomcat7
Tomcat7
服务网关
Memcached集群

NgWeb 网络结构图

我们由《NgWeb 网络结构图》中可以清晰的看到 NgWeb 的负载解决方案。

第一层我们将使用 DNS 做负载均衡服务器，因为 DNS 拥有很强的负载能力，很难遇到性能问题，而且支持横线扩展。DNS 服务器将采用智能 IP 地址分析的方式负载，将用户解析到离他最近的服务器集群中。

第二层我们使用 LVS 做负载均衡，LVS 有着接近硬件级的负载能力，拥有很强的横线扩展的能力，而且可以很好的解决 DNS 服务器缓存的问题，真实的 WebServer 都是在 LVS 后边，DNS 只要解析到我们的 LVS 就可以访问到后端的 WebServer，在增减 WebServer 时，无需调整 DNS。

第一层和第二层都将使用 HA 主备模式以防止单点故障。

第三层就是我们的静态资源 WebServer，这里我们选择了 Nginx（Tengine 的性能未测试）作为我们的 WebServer，无论是在反向代理还是在对静态资源处理上 Nginx 都有着很强的能力，单机可以做到 1w 的负载。Nginx 将处理全部的静态资源，如 js,css,img,html 资源，客户的动态请求 Nginx 将转发给第四层处理。做到动静分离，进一步提高负载能力。

第四层是我们的动态资源服务器，这里我们选择了 Tomcat7，Tomcat7 已经发布了快 2 年的时间，经过了市场的考验，拥有着比 Tomcat6 更好的性能。在这里并没有测试 Jboss 和 tomcat 的性能，

理论上讲 Tomcat 和 Jboss 应该不存在太多的性能差异，因为 JBoss 也只是封装了 Tomcat，而且 JBoss 对我们来讲还是太重了。

第五层是我们的 Memcached 缓存服务器集群，它将为我们提供数据缓存服务，如登录用户的信息，常用查询的结果信息。在关于查询结果缓存的地方和虫子讨论过一次，参考虫子的意见将不再缓存全部车次数据，只缓存用户查询过的数据，提供定时更新缓存的服务。

以上就是网络层的负载均衡解决方案，由于 DNS 和 LVS 的存在，让我们的方案没有横向负载均衡的瓶颈问题，nginx 和 tomcat7 配合 memcached 也能给我们提供很好的性能。

二、 HTML 页面处理

原则：减少请求数，减小数据量。

我们将使用全站静态化处理，全站将只使用一个 css 文件，只在部分需要的页面使用 3 个以内的 js 文件，css 和 js 文件全部做压缩处理，删除空格，缩短变量，减小流量。每个页面的图片不得超过 10 张，使用 CSS Sprites 合并图标文件，使用 gzip 技术压缩静态资源。

将在支持 HTML5 技术的浏览器上使用 HTML5 Local Storage 技术缓存静态资源。

在部分需要显示动态信息的位置将使用 HTML5 Local Storage + cookie + ajax 的方式来显示。如显示当前登录用户的位置，我们将在用户登录后将用户信息保存到客户的 Local Storage，同时在 cookie 保存一份，在需要显示时，如果支持 Local Storage 将优先使用

LocalStorage 显示，否则由 Cookie 中获取，如果 Cookie 也意外无法获取将使用 ajax 的方式来查询当前登录信息。如果是存在保密性的内容，将直接使用 ajax 获取。

对于用户查询的车次信息，我们也将使用 HTML5 Local Storage。根据更新时间和服务网关判断有效性。在每次提取了 Local Storage 中的数据库，都写入一个新的读取时间，如果 Local Storage 容量满了以后，自动删除读取时间最早的数据，写入新数据。

三、 动态资源处理

原则：稳定第一

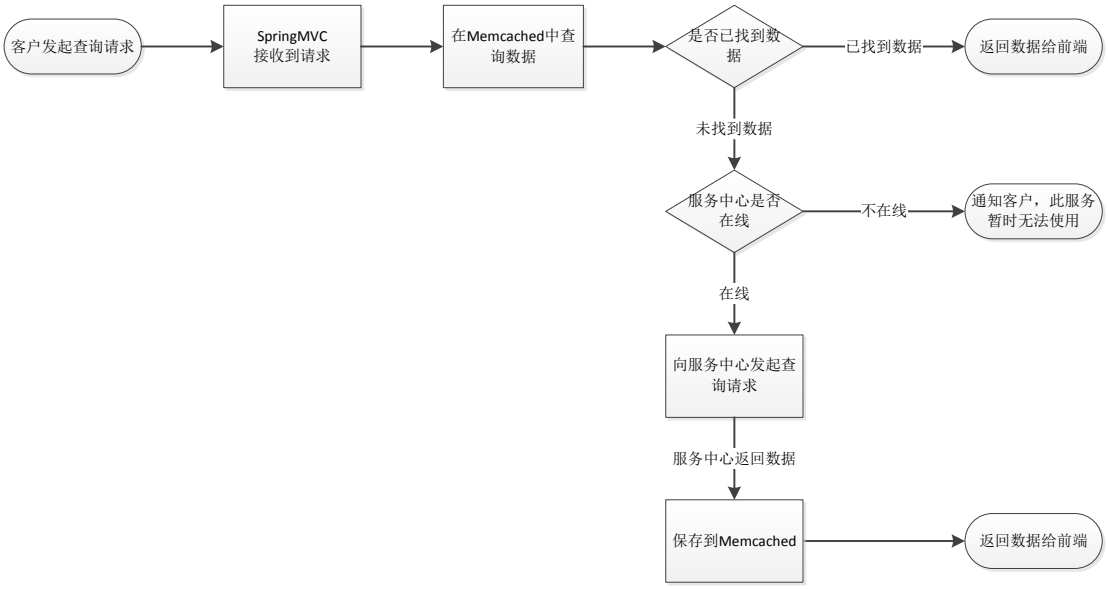
对于动态资源的处理我们遵循稳定第一的原则，在不影响稳定的前提下提高性能。动态资源处理服务器，将会是由 java 编写运行在 tomcat7 上。考虑到安全性以及性能的影响，它将独立与其他所有的模块，单独部署。

我们选用 SpringMVC 来处理客户请求。SpringMVC 在性能方面仅次于 Struts2，但是鉴于 SpringMVC 与 Spring 组合的扩展性和可维护性，放弃了 Struts2 选择 SpringMVC。而且两者的性能差距也并不是很大。

由于集群环境的配置，使得 Session 没有在单机时那么有效，所以将禁止使用 Session 保存数据，所有需要放在 Session 中的数据，全部放入 Memcached 中。如用户的登录信息、订单信息等。

Memcached 作为 web 与服务网关的中间缓存层存在，当有客户发起有关车次的查询信息是，首先我们查询 Memcached 中是否已经有

此查询的结果，如果有则直接返回给前端，如果没有我们则向服务网关发起查询请求，得到结果后，保存到 Memcached 中，然后返回给客户。与服务中心对接的流程图如下：



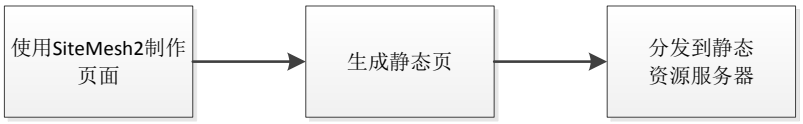
与服务网关的具体对接细节还需要同服务网关商定。

由于前端并没有数据库服务器的存在，所以我们将日志记录在文件中，为避免 IO 操作带来的磁盘开销，日志需记录在单独的磁盘中，不得与应用服务器在同一块磁盘中。日志的格式和记录细度也需要进一步确认。

四、 内容分发

内容分发模块包括两部分。一个是静态页面的生成，一个才是内容分发。

我们将使用 SiteMesh2 作为外层页面装饰器，并且遵循 **HTML 页面处理**中的原则来设计制作页面。当页面制作完成后，将页面进行静态化，然后由内容分发系统发布到各台服务器之间。流程如下：



由于我们可能存在多台静态资源服务器，如果只一台服务器做推送服务器可能会影响效率，我们将在每个静态资源服务器驻留一个分发系统的客户端，用于做二次分发，加快分发速度。

由于我们发布的都是静态资源，所以我们采用直接覆盖的方法替换正在运行的方式，分发客户端也将负责这个工作。在接收到新内容后暂存在磁盘中暂不替换，内容分发系统会告知客户端在什么时间进行替换。如在凌晨 1:00 替换，那么所以的静态资源服务器，就都将在凌晨 1：00 替换新的静态资源，完成版本更新。

五、 系统监控

通过系统监控能够随时了解到每台服务器的运行状况，包括 CPU、内容、磁盘 IO、网络流量、已运行的服务。我们还将对我们每台服务器做内置的监控客户端，随时返回当前系统运行的情况，如已处理的请求数量，正在处理的请求量等信息。

系统监控还将负责负载均衡系统的健康度检查，伸缩性调整等。

nagios 也许会是我们的一个选择，不过还未做了解，需要进一步确认。