**架构评审文档**

**软工1803-U201817044-王粟鹏**

**软工1803-U201817033-李佳欣**

**软工1803-U201817030-刘文文**

**软工1801-U201816968-刘欣媛**

**软工1803-U201817039-周澍**

**评审人：全组人员**

|  |  |  |
| --- | --- | --- |
| 项目名称  Project Name | | 密级  Confidentiality Level |
| 60321火车购票系统 | | 仅供收件方查阅 |
| 项目编号  Project ID | 版本  Version | 文档编号  Document Code |
| 1 | 1.0 | 1\_AR |

60321火车购票系统

架构评审文档

评审人：全组人员

评审日期：2020.10.14

Revision Record

修订记录

| 日期 | 修订版本 | CR/ Defect号 | 修改章节 | 修改描述 | 作者 |
| --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

目录

[一、 ATAM概述 4](#_Toc54614673)

[二、 商业动机表述 4](#_Toc54614674)

[三、 软件架构表示 5](#_Toc54614675)

[四、 质量属性提炼及其战术 7](#_Toc54614676)

[五、 质量属性效用树 9](#_Toc54614677)

[六、 质量场景架构分析 12](#_Toc54614678)

[七、 系统架构再分析 13](#_Toc54614679)

[八、 评审结果 15](#_Toc54614680)

# ATAM概述

选用ATAM方法对《火车购票系统架构设计文档》进行综合评估。

**架构涉众：**

* 经理
* 开发人员
* 测试人员
* 管理员
* 客户

**架构评估方式：**

从客户和开发组织的角度对系统的商业目的进行阐述，根据阐述分析出质量属性的优先级。

描述系统整体结构以及与架构商业周期的关系。

对软件的质量属性需求进行提炼，并提炼出架构设计决策的精确表述，分析他们的战术。将架构放在一定的使用场景中，评估这些设计决策。判定战术是否起到应有的作用、是否令人满意的实现了这些质量属性需求。

对于一些特定的架构设计决策，要进一步进行敏感点、权衡点和风险决策评估。

通过分析找出系统总在的重要问题和漏洞，然后再针对评估结果进行架构设计的改进。

得出评估的总结。

# 商业动机表述

**从开发组织的角度：**

开发一个火车购票系统，具有比较高的实用性，可以与其他的外部系统进行良好的通信，具有良好的界面，因此能够将整个系统卖给某个客户。因为系统比较好的性能，能够获得客户的好评，占据比较高的市场份额，赚取一定的利润。

**从客户角度 ：**

系统易操作，稳定而且安全，功能全面，能够准确地处理用户的订单查询以及购买操作。

**划分质量属性需求优先级：**

**高优先级质量属性需求：**

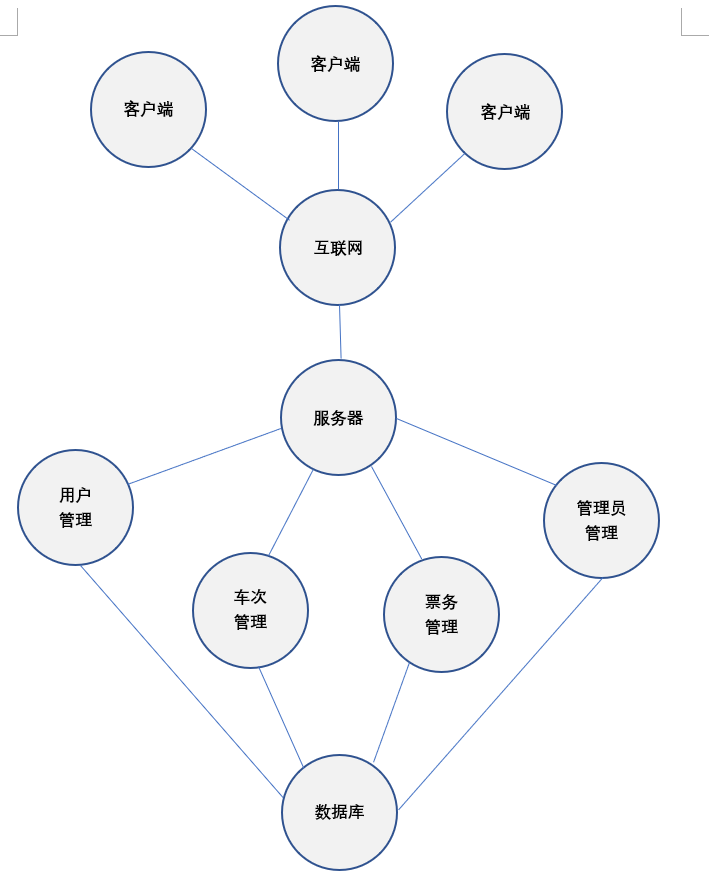
* 可用性
* 性能
* 安全性
* 易用性

**低优先级质量属性需求：**

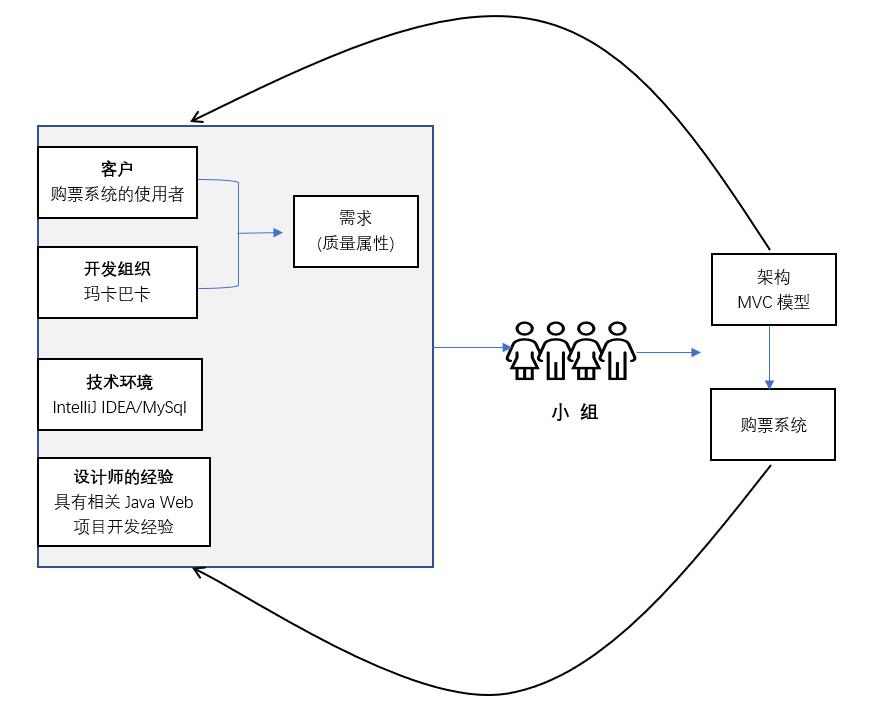
* 可移植性
* 可修改性
* 可测试性
* 可维护性

# 软件架构表示

**系统的整体结构：**



**与架构商业周期的关系：**

****

# 质量属性提炼及其战术

|  |  |  |
| --- | --- | --- |
| 目标 | 实现方式 | 所采用的战术 |
| 性能 | 系统应在规定时间范围内对请求做出响应，若因其他原因未能达成，则要弹出警告并写明原因，避免用户无故长时间等待。 | 限制访问队列大小  缓冲 |
| 可用性 | 在用户购票时进行一致性检查，也就是在并发时需要对数据加锁，保证数据一致性 | 锁机制 |
| 系统数据库出现错误时自动更换备份数据库，保证系统不停机。 | 数据备份  内置故障恢复机制 |
| 系统出现错误或异常但不影响操作结果时，系统给出警告信息并保持运行 | 异常检测  内置故障恢复机制 |

|  |
| --- |
|  |

|  |  |  |
| --- | --- | --- |
| 安全性 | 用户信息、票务信息加密存放在数据库当中。同时对数据库信息进行冗余备份，当原始数据库遭到破坏时，能够短时间内调用备份，恢复正常使用。 | 数据机密性  冗余副本 |
| 管理员进行登录时，采用多种方式验证身份。用户进行不可逆操作时采用多方验证，若非用户本人进行的不可逆操作，提供密码修改、强制下线服务。 | 用户验证  操作源检测攻击 |
| 系统使用限流阀机制和过期机制，以及辅助检测手段防止Dos攻击。 | 限流阀机制  过期机制 |
| 易用性 | 为用户提供易于使用的、能根据用户需求来改变系统状态的指令以提高用户对系统的控制以及操作的简便性。同时使得用户的操作能获得系统及时的反馈。 | 支持用户主动 |
| 将用户接口与应用的其余部分分离开来。可以根据每个版本中用户的使用情况来及时的修改用户接口所定义的行为，同时使修改能够在较短时间内完成。 | 分离用户接口 |
| 构建与用户行为、系统提供的功能、系统的整体行为有关的模型，并封装这些模型。使得系统能够预测某些用户的行为意图以及自身的行为，这使得用户在使用系统时，系统对用户的行为能够做出更精确且及时的反应。同时设计师也能更轻松地通过裁剪与修改模型来使模型更适应用户的行为。 | 构建用户模型 |
| 可修改性 | 确保每个模块在职责上具有语义一致性，不需要过多依赖其他模块来工作。  对预想的变更集合提供了一个评估特定的责任分配方法。  是一个模块更通用能够使它根据输入计算更广泛的功能。  对系统中可能出现的变化点限制可能的选择。 | 维持语义一致性  预期期望变更  泛化模块  限制可能的选择 |
| 把一个系统或系统的一个分解的职责分解为更小的部分。并选择性的将一部分信息私有化，另一部分信息公有化。将未来可能的变更有效的隔离在一个模块内。  通过添加接口、添加适配器、提供占位程序的方法来维持现有的接口不变。这使得某个模块发生变化时，调用了这个变化了的模块的接口的其他模块受到的影响最小化。  减少与特定的模块共享数据的模块。即减少使用该特定模块产生的数据的模块数量，以及产生该模块所使用数据的模块数量。  如果B模块对A模块具有非语义的任何类型的依赖，可以在这两个模块间插入一个仲裁者来管理与该依赖相关的活动。 | 信息隐藏  维持现有的接口  限制通信路径  使用仲裁者 |
| 支持即插即用操作。  可以在启动时设置参数。  允许载入时间绑定。  允许方法调用的后期绑定。  允许独立进程的运行时绑定。 | 运行时注册  配置文件  组件更换  多态  遵守已定义的协议 |
| 可移植性 | 采用一个可移植层把与平台相关问题封装起来，使上层应用软件与其环境具有抽象接口 | 信息隐藏、添加接口 |
| 可测试性 | 在组件完成时执行单元测试，系统内的各组件有控制行为的接口并且显示其输出 | 输入/输出 |
| 对完整的应用进行测试，系统可以对出现的错误或故障进行详细提示 |
| 可维护性 | 将不同模块分离，后续可较容易地对单独的模块进行修改替换等 | 模块划分 |
| 系统运行进行日志记录，撰写详细的设计和帮助文档 | 日志记录工具、文档 |

# 质量属性效用树

|  |  |  |  |
| --- | --- | --- | --- |
| 质量属性 | 属性求精 | 场景编号 | 场景 |
| 性能 | 响应时间 | XN01 | 在系统除于高峰期时，系统保证购票的平均响应时间低于7秒，票务查询的平均响应时间低于2秒，若超出15秒未做出响应时系统对用户弹出友好提示。（H，H） |
| 吞吐量 | XN02 | 系统可以保证1000万用户同时使用系统、500万人同时进行票务操作。（H，M） |
| 可用性 | 数据一致性 | KY01 | 在用户购票时进行一致性检查，也就是在并发时需要对数据加锁，保证数据一致性，从而避免出现用户下单后却没票的情况。（M，M） |
| 停机时间 | KY02 | 系统正常工作，年度停机时间小于等于8.8小时，达到可用性级别99.9%。（H，M） |
| 系统稳定 | KY03 | 系统数据库出现错误时在10分钟内完成备份数据库的自动更换，保证系统不停机。（H，M） |
| 容错性 | KY04 | 用户操作出现错误或异常但不影响操作结果时，系统给出警告信息并保持运行。（H，M） |
| 安全性 | 多重验证 | AQ01 | 用户进行不可逆操作时采用多方验证，若非用户本人进行的不可逆操作，提供密码修改、强制下线服务；管理员进行登录时，采用多种方式验证身份。  （H，H） |
| 防止恶意攻击 | AQ02 | 杜绝非法用户试图绕过服务器直接连接到数据库服务器端口；用户进行在线支付时，系统可以阻止100%的病毒攻击，防止用户交易信息泄露和资金被非法转移；屏蔽某IP短时间内大量无意义访问。  （H，M） |
| 封闭性 | AQ03 | 对局域网外的用户来说，不能直接访问数据库，更不能随意更改其中的信息。（H，M） |
| 数据恢复 | AQ04 | 对数据库中的信息进行冗余备份，在原始数据库崩溃或被破坏时，能够调用备份数据库进行替换。（H，M） |
| 机密性 | AQ05 | 用户只能查询自己的信息，不能查询其他用户的信息，不能对其他用户的票进行操作。管理员不能查询用户的私密信息。（H，M） |
| 数据完整性 | AQ06 | 在并发用户数目较多时，系统能够保证数据的完整性。（H，L） |
| 可移植性 |  | YZ01 | 系统在新的操作系统或新的数据库上能够正常运行。(H,L) |
| 可测试性 | 系统功能模块测试 | CS01 | 对每一个功能模块进行单元测试，保证每一个模块能正确运行。(M,M) |
| 系统整体测试 | CS02 | 对整个系统进行测试，以确保系统能正常运行且满足所有的需求(M,M) |
| 系统界面测试 | CS03 | 进行对界面的测试，体验界面是否对用户友好，是否易用。(H,L) |
| 可维护性 | 功能扩展 | XG02 | 如果增加某一功能，要能在短时间内完成，且不会影响其他模块。(M,M) |
| 可升级性 | XG03 | 系统在修复问题、增加功能等后可以自动更新，且不会影响整个系统的正常运行。(H,L) |
| 文档完整 | XG04 | 系统需要提供详细可读的文档，以便于维护人员维护。(H,L) |
| 易用性 | 高效使用 | YY01 | 新用户在使用系统进行购票时，可以在10分钟内完成所有流程。老用户在使用系统购票时，可以在5分钟内完成所有流程。（H，M） |
| YY02 | 老用户在使用系统进行购票时，可以熟练使用简单的快捷键操作系统，使用快捷键可以提高40%的系统使用效率。（H，M） |
|  | YY03 | 用户在用在使用系统查询车票的时候，系统会显示默认的起点站和终点站，其分别为上一次购票的终点站和起点站，默认的出行时间为第二天。且会将用户在历史订单中涉及的站点已可选的形式显示在起点站与终点站一栏的下方供用户选择。（M，M） |
| 易于学习 | YY04 | 所有具有基本电脑操作常识的用户，在学习使用该售票系统时，根据系统在界面上的一步步的使用指导，在3分钟内可以学会系统基本的使用方法。（H，M） |
| YY05 | 用户在查看系统使用帮助界面学习系统操作时，该界面采用舒适的结构，图文结合，标注重点的方式，让用户能够迅速熟悉系统提供的功能，使用时的流程以及、些操作步骤的涉及的按键位置。用户可以在5分钟内有效学到系统的操作。（M，M） |
| 提高用户自信和满意度 | YY06 | 用户在长时间不使用该系统后重新用该系统进行买票，不会因为遗忘该系统的操作导致购票过程变得困难。所有购票流程依旧可以在10分钟内完成。  （M，M） |
| 及时反馈 | YY07 | 用户使用系统时的所有操作，系统都会在2秒内提供文字或者声音反馈，并实现合适的界面跳转。（H，M） |
| YY08 | 用户进行账号注册和账号登录等输入操作时，若出现输入错误或输入内容不当情况，系统利用界面文字提醒用户及时更正，使用户每次出错平均更正时间低于2秒。(M，M） |
| 错误处理 | YY09 | 当用户进行错误操作后，可取消这一操作，系统对取消操作应在1秒内给出反馈。（M，M） |
| 可修改性 | 功能扩展 | XG01 | 开发人员为满足用户的需求，新增功能模块，只要在设计时实现该功能模块的功能，并将其与系统原有模块结合即可。对系统的原有模块不会产生大的改动。且每个新增功能模块的迭代在5天内完成。具有新功能的系统未发布前，不影响用户对原有系统的使用。（H，M） |
| 界面修改 | XG02 | 开发人员根据用户使用反馈修改系统的界面，在设计时只修改界面及相关配置文件，不影响用户对其他部分的使用，能在5小时内完成修改并发布更新。（M，M） |
| 功能删除 | XG03 | 开发人员删除没用的功能或选项，在设计时只需要删除该功能或选项的代码以及修改与其有联系但耦联度不高的代码模块，修改不影响其他功能的使用，修改可以在两小时内完成。（M，M） |
| 质量提升 | XG04 | 开发人员为避免某一段时间内人们对售票系统使用激增而导致的系统崩溃，在维护时对系统的数据库进行更新迭代，迭代过程不影响原数据库的使用，且每次迭代在3天内完成。（M，M） |
| 系统扩展 | XG05 | 开发人员对系统进行整体性扩展，只要符合系统规范，就可以简单加入模块。修改时不影响系统原有模块的使用，且每次模块的增加可在3天内完成。（M，M） |
| 高效维护 | XG06 | 系统的维护人员在系统出现问题进行要进行维护时，可以迅速定位要修改的代码块，在相关代码块中完成修改，不影响其他功能模块，且修改可以在2小时内完成。（M，M） |

# 质量场景架构分析

在质量属性效用树中，对场景的优先级进行了划分，在此对最重要也最难实现的场景进行进一步架构分析，即对性能给出架构分析表格

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 场景号：AQ01 | 场景：管理员准备登录账号，管理后台 | | | |
| 属性： | 安全性 | | | |
| 环境： | 管理员准备登录 | | | |
| 刺激： | 管理员发起登录请求 | | | |
| 响应： | 系统对身份进行多重验证 | | | |
| 架构决策 | 敏感点 | 权衡点 | 有风险决策 | 无风险决策 |
| 采用多重验证的方式，验证账号使用者身份 | S1 |  | R1 |  |
| 提供单独的验证通道 | S2 | T1 | R2 |  |
| 推理 | 1. 因为管理员的操作影响着整个购票系统，所以要采取多方面验证方式，一方面确定是否为管理员本人操作，另一方面帮助系统确认该次登录是否经过更高权限的管理员允许。 2. 为该方式提供单独的验证通道，保证在系统繁忙的时候，管理员也可以及时登录系统进行管理。 | | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 场景号：XN01 | 场景：系统访问量达到高峰 | | | |
| 属性： | 性能 | | | |
| 环境： | 系统使用处于高峰时期 | | | |
| 刺激： | 用户提出访问请求 | | | |
| 响应： | 系统对用户刺激的良好响应 | | | |
| 架构决策 | 敏感点 | 权衡点 | 有风险决策 | 无风险决策 |
| 超出限制访问量的请求放在等待队列中 | S1 |  | R1 |  |
| 缓存 | S2 |  | R2 |  |
| 每个IP每次只允许同时发出一个请求 | S3 | T1 | R3 |  |
| 数据库连接池 | S4 |  |  | N1 |
| 推理 | 1. 因为使用的服务器所能容许的最大访问数量有限，所以当某一时刻的数据访问数量超过最大访问权限时，采用队列的方式，可以暂时接受用户的访问请求，但暂缓系统对用户发出请求的响应，这会降低系统的响应速度，影响用户体验。 2. 缓存的容量也是有限的 3. 可以预防恶意攻击，可以提高系统的安全性，但在一定程度上会降低系统的可用性。 4. 减轻数据访问负担，提高了系统性能。 | | | |

# 系统架构再分析

1. **风险决策和敏感点**

|  |  |  |
| --- | --- | --- |
| 采用战术 | 敏感点（S） | 有风险决策（R） |
| 采用多重验证的方式，验证账号使用者身份 | 保证了系统管理的安全性，减少了恶意修改系统信息的可能 | 管理员设备被盗会导致验证方式被盗 |
| 提供单独的验证通道 | 在系统繁忙时，管理员仍可以及时登录系统，进行管理操作，保证了管理员对系统数据的掌控 | 增加管理员通道的优先级会相对的降低用户优先级，可能会造成用户响应迟钝。 |
| 超出限制访问量的请求放入等待队列 | 提高了系统的稳定性和可用性，减少崩溃的可能 | 会降低最大并发数目，使用户等待时间过长，造成用户不满 |
| 缓存 | 提高系统的访问速度和性能 | 提供的缓存数目有限，在并发用户过多的情况下，系统的处理缓慢 |
| 每个IP每次只允许同时发出一个请求 | 避免了非法用户的恶意攻击，提高了其他用户的访问速度 | 用易用性的降低来换取安全性的提高 |
| 数据库连接池 | 在该技术的支持下，应用程序会重复使用一个现有的数据库连接，而不是新建一个数据库连接，有助于提高系统的可用性 |  |

1. **问题分析**

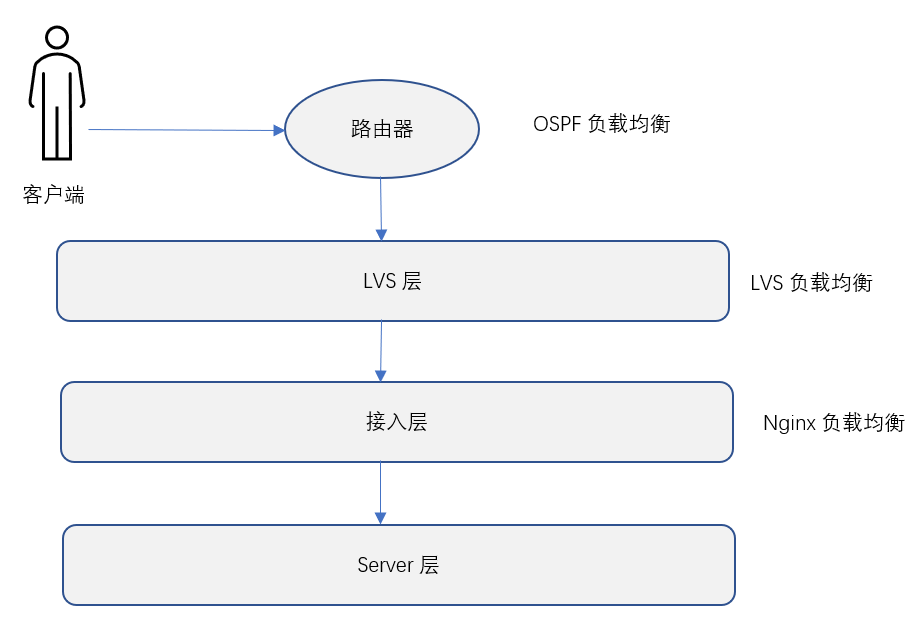
该系统采用基于B/S的分层部署方式，系统部署在单一服务器上，这种方式具有一定的优点。在用ATAM方法对系统架构进行分析之后发现，特别是对系统优先级高以及比较重要的属性分析之后，发现系统在上述场景之下还是有以下的问题：

**安全性：**在管理员验证多层次进行时，因为使用优先通道，所以可能会降低用户响应的速度，特别是在用户峰值的情况下，会使用户等待时间加长，影响用户体验。

**性能：**在非常多用户并发操作下，因为使用了队列，所以系统对用户的响应时间会有延迟，用户会有一定的等待时间，影响用户体验。

**(3)改进系统的架构**

因为系统的使用并发人数很庞大，这样导致用户对系统发出的请求和操作指令也数目庞大，改进后的系统采用分布式集群部署的方式，并提供各种容灾手段(双火机房、节点容错、服务器灾备等)保证系统的高可用，流量也会根据不同的负载能力和配置策略均衡到不同的服务器上。在这里主要采用三层分布式方法，分别是OSPF、LVS、Nginx。



OSPF(开放式最短链路优先)是一个内部网关协议，通过路由器之间通告网络接口的状态来建立链路状态数据库，生成最短路径树。OSPF会自动计算路由接口上的Cost值，但也可以通过手工指定该接口的Cost值，手工指定的优先于自动计算的值。到达目标相同Cost值的路径，可以执行负载均衡，最多6条链路同时执行负载均衡。

LVS (Linux VirtualServer)，它是一种集群(Cluster)技术，采用IP负载均衡技术和基于内容请求分发技术。调度器具有很好的吞吐率，将请求均衡地转移到不同的服务器上执行，且调度器自动屏蔽掉服务器的故障，从而将一组服务器构成一个高性能的、高可用的虚拟服务器。

Nginx是一款非常高性能的http代理/反向代理服务器，Nginx实现负载均衡的方式主要有三种:轮询、加权轮询、ip hash轮询。

# 评审结果

总体来说，通过对原有架构设计的评审，我们发现了其中的不足，然后针对性的对系统架构进行了改进，提出了一种更加高效的架构设计。改进后的架构设计具有更高的安全性、易用性、性能、可用性等等，达到了设计目的符合质量属性需求分析的要求。