



票务系统架构评审案例分析

- 1 ATAM方法表述
- 2 商业动机的表述
- 3 构架的表述
- 4 质量属性效用树
- 5 质量场景的构架分析
- 6 对系统构架的再分析
- 7 评审结论

1 ATAM方法表述

(1) 概述

ATAM (Architecture Tradeoff Analysis Method) :

SEI提出的一种软件构架评估方法。ATAM评估方法的主要目的:

- 1) 提炼出软件质量属性需求的精确描述;
- 2) 提炼出构架设计决策的精确描述;
- 3) 评估这些构架设计决策, 并判定其是否令人满意的实现了这些质量需求。

ATAM评估方法:

并非把每个可以量化的质量属性都进行详尽的分析, 而是使众多的风险承担者 (包括经理、开发人员、测试人员、用户、客户等等) 都参与进来, 由此而达到上述目标的。

ATAM是一种挖掘潜在风险, 降低或者缓和现有风险的软件构架评估方法。因此, 以下三点是评估中要特别注重的: 风险、敏感点和权衡点。

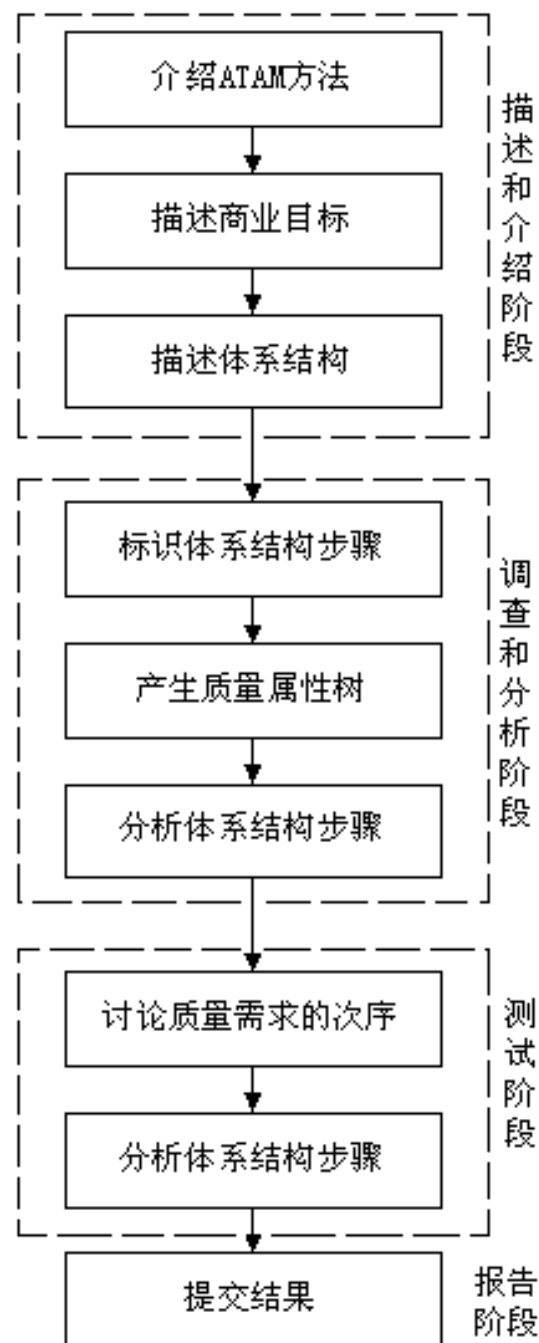
(2) 构架涉众

- 普通用户
- 用户管理员
- 票务管理员
- 开发人员
- 测试人员

(3) 评估步骤

ATAM主要分以下几个步骤：

- 1) ATAM描述；
- 2) 商业动机表述；
- 3) 软件构架表述；
- 4) 确定构架方式；
- 5) 生成效用树；
- 6) 分析构架方式；
- 7) 确定场景及其优先级；
- 8) 进一步分析构架方式；
- 9) 得出结论。



2 商业动机的描述

项目经理从开发组织和客户角度，来表述票务系统的商业目标，综合如下：

- 从开发组织角度：开发一个模块性强、实时高效、界面良好、与外部其他系统兼容良好的系统，这使得开发组织能够把整个产品或某个模块卖给其他客户，同时由于良好的界面和业务处理效率而受市场欢迎。
- 从客户角度：系统容易操作，可维护性好、系统稳定、可以及时准确的处理用户的在线订票或查询业务。

根据上述目标，质量属性可以划分为两类：

高优先级质量属性：

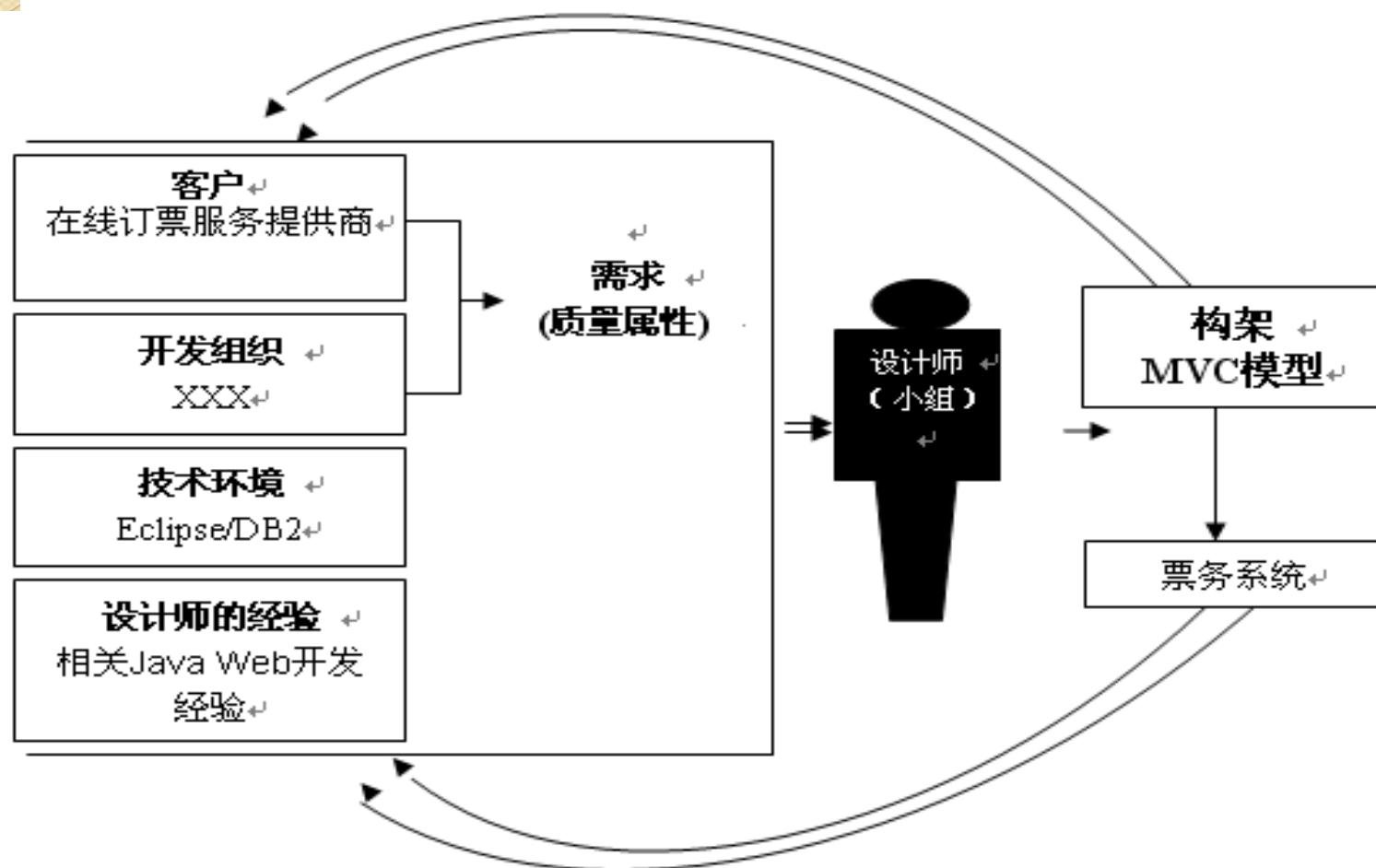
- 1) 性能
- 2) 安全性
- 3) 易用性
- 4) 可用性

重要但优先级较低的属性：

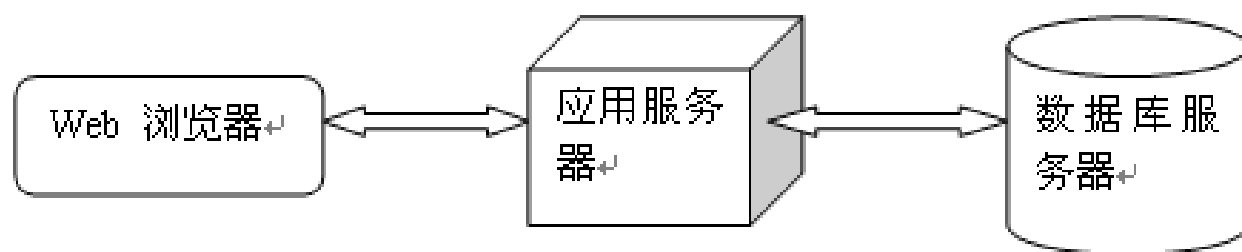
- 1) 模块性
- 2) 可维护性
- 3) 可修改性
- 4) 可测试性

3 架构表述

(I) 与构架商业周期的关系



(2) 系统的整体结构



(3) 质量属性及采用的战术

目标↵	实现方式↵	所采用的战术↵
性能↵	用户访问的系统应该能在规定的时间内做出响应，如果系统由于网络或者数据库原因不能在规定时间内做出反应，那么系统应该提出警告，不能出现用户无故长时间等待的情况。↵	限制访问队列大小↵ 缓冲池技术↵
	当应用程序需要在关联关系间进行导航的时候，由 Hibernate 获取关联对象。同时 Hibernate 的 Session 在事务级别进行持久化数据的缓存操作。↵	抓取策略↵ 二级缓存↵
易用性↵	遵从 J2EE 的系统提供了诸如 JSP 和 servlet 这样的 Java 技术，它们支持内容的渲染，以满足不同用户的需要↵	单独的用户接口↵
	用户对系统的操作能得到正确及时的反馈。↵	支持用户主动↵
安全性↵	遵从 J2EE 的系统提供了由容器进行授权校验的基于角色的安全性机制，以及已经为使用做好准备在程序中进行授权检查的安全性机制↵	身份验证↵ 授权↵ 数据机密性↵ 验证码↵
	并发操作时，保证数据的排他性↵	锁机制↵
	Spring Framework 利用 AOP 来实现权限拦截，还提供了成熟的，简洁清晰的安全框架，通过对 spring bean 的封装机制来实现↵	AOP↵ Acegi 安全框架↵

可用性↵	当系统试图超出限制范围来进行票务查询或者订购票时必须进行错误检测并且抛出异常，中止进一步的错误操作。↵	异常检测↵ ↵
	遵从 J2EE 的系统提供了可以使用的事务服务，通过提供内建的故障恢复机制，提高了应用的可用性和可靠性↵	内建故障恢复机制↵ ↵
模块性↵	根据功能将系统划分为几个模块，系统满足“松耦合高内聚”的设计原则。↵	模块划分↵
可维护性↵	系统运行有日志记录。↵	日志记录工具↵
	系统可以扩展到新的系统的。↵	XML 配置↵
可修改性↵	在变更到达时，系统在时间和预算内所完成，测试和部署的变更。↵	局部化修改↵ 防止连锁反应↵ 推迟绑定时间↵
可测试性↵	在完成系统开发的一个增量后，较轻松的对软件进行测试。↵	输入/输出↵

4 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	在系统处于高峰时期，保证登陆的每个用户发出的买票或查询的响应时间在 5s 以内，如果需要等待则给出友好的提示。(H, H)
	吞吐量	XN02	系统可以保证同时相应 5000 个用户的操作。(H, M)
易用性	界面友好，操作简单	YY01	要求具有基本电脑操作常识的人，可以根据良好的界面设计迅速学会使用方法。让熟手用户使用快捷键。(M, M)
	及时反馈性	YY02	当系统发生错误或程序运行时间较长时，用户界面应该为用户提供有意义的反馈信息，并有良好的上下文感知功能。(H, M)
	界面一致性	YY03	用户界面应该遵循一定的标准和常规，尽可能的在一个界面上完成，不要时常出现弹出框。(M, M)

安全性↕	机密性↕	AQ01↕	允许用户查看本人的订票信息，但不能查询他人的订票信息，更不能退订他人已选定的票。系统管理员不能随意查看用户隐私。(H, M)↕
	封闭性↕	AQ02↕	对局域网外的用户来说，不能直接访问数据库，更不能对其随意进行修改。(H, M)↕
	防止恶意攻击↕	AQ03↕	杜绝非法用户试图的绕过应用服务器直接连接到数据库服务器的端口上；屏蔽某IP 短时间内的大量无意义的访问，以防被挤爆，使正常用户无法使用。(H, M)↕
	客户端功能无关性↕	AQ04↕	客户端只包含人机交互界面功能，不包含业务功能描述，即客户端发送给服务器的是用户请求，而不是业务所代表的 SQL 语句，以防止非法用户修改客户端的 SQL 语句以实现越权功能的非法行为。(M, L)↕
	数据的完整性↕	AQ05↕	在并发用户多的情况下，系统能保证数据的完整性。(H, L)↕

可用性↕ ↕	容错性↕	KY01↕	应该容忍用户在使用过程中发生的各种操作错误，并且能够方便的地从错误中恢复过来，保证系统不受或尽可能少的受到用户错误操作的影响。(M, M)↕
	备份与恢复↕	KY02↕	备份时间应尽可能的短且在用户访问极少时进行。系统崩溃时能在1小时内恢复。(H, M)↕
	硬件更换↕	KY03↕	硬件（网线、路由器、硬盘等）发生故障时，可以方便更换。(H, L) ↕
模块性↕ ↕	模块职责划分明确↕	MK01↕	系统自上而下划分为：系统-子系统-模块-子模块。(H, M) ↕
	接口清晰↕	MK02↕	模块之间通过接口通信，只要是遵循同一接口完成同一功能的模块即可相应替换，由此实现了平台无关性。(H, L) ↕

可测试性↵	类的测试↵	CS01↵	每个类及其函数都应该单独测试，以验证其正确性。(M, L)↵
	系统功能模块测试↵	CS02↵	对与系统功能相对应的模块进行测试，以保证业务的完备性。(M, M)↵
	系统的性能测试↵	CS03↵	对整个系统进行压力测试，看能否达到设计时的访问量。(H, M)↵
	Beta 测试↵	CS04↵	邀请用户代表进行 beta 测试，体验界面的友好性和相应速度。(H, L)↵
可修改性↵	功能扩展↵	XG01↵	如增加票务预定功能，能在一天内完成，并且不影响系统其他部分。(M, M)↵
	界面修改↵	XG02↵	易于修改。(M, M)↵
	文档完备↵	XG03↵	各个模块、系统必须提供详细可读的文档，以便于维护人员维护。(H, L)↵
	可配置性↵	XG04↵	维护人员可以方便的配置系统参数、业务参数。(H, L)↵
↵	可升级性↵	XG05↵	客户端发现缺陷后，可以自动更新，以解决。新功能产生或界面更换时同样。(H, L)↵
可移植性↵	↵	YZ01↵	系统在新的操作系统或者新的数据库上能够正常运行。(H, L)↵

5 质量场景的构架分析

在质量属性效用树中，我们对场景的优先级进行了划分，而同时由于分析时间宝贵，所以我们应该把宝贵的分析时间最先用于最重要且最难实现的场景上，即标注为(H,H)的场景。在质量属性效用树的表格中，仅在性能和可用性这2个质量属性下发现标注有(H,H)的场景，下面根据系统的体系结构和实现质量属性所采用的战术分别给出这些重要场景的构架方法分析表格。

- 性能

场景号: XN01↵	场景: 系统访问量达到高峰↵			
属性↵	性能↵			
环 境↵	系统处于高峰访问时期↵			
刺激↵	用户请求访问↵			
响应↵	良好响应请求↵			
构架决策↵	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↵	↵	T1↵	R3↵	↵
数据库连接池↵	S4↵	↵	↵	N1↵
推 理↵	1、 由于在系统部署的时候用的是单应用服务器,而一台应用服务器可同时支持的并发用户数是很少的,在几千甚至上万的用户访问系统时,由于限制最大访问量,不能保证每个用户都能随时登陆,降低了用户的满意度。↵ 2、 单服务器提供的缓存数有限。↵ 3、 避免了非法用户的恶意攻击,但有可能降低系统的可用性↵ 4、 减轻数据库的负担,提高系统的性能。↵			

- 可用性

场景号: KY02↵	场景: 系统备份与恢复↵			
属性↵	可用性↵			
环 境↵	系统发生错误↵			
刺激↵	用户进行恢复↵			
响应↵	尽快恢复并为用户提供有意义的反馈信息↵			
构架决策↵	敏感点↵	权衡点↵	有风险决策↵	无风险决策↵
系统备份↵	S5↵	↵	R3↵	↵
故障恢复↵	S6↵	↵	R4↵	↵
推 理↵	1. 系统备份是进行故障恢复的前提,但频繁的备份会影响正常的业务处理,存在一定的风险。↵ 2. 如果是因为系统掉电或其它的误操作,利用备份数据可以很快恢复。但由于使用的是单服务器机制,一旦这台服务器出现故障或者崩溃,硬件发生故障,系统的恢复时间会很长,存在风险。↵			

6 对系统构架的再分析

(1) 风险决策和敏感点

采用战术↵	敏感点 (S) ↵	有风险决策 (R) ↵
超出限制访问量的请求放在等待队列中↵	提高了系统的稳定性和可用性，减少了崩溃的可能↵	会降低最大并发数目，使得用户等待时间过长，可能造成用户不满↵
缓存↵	提高系统的访问速度和性能↵	单服务器提供的缓存数目有限，并发用户数多的情况下，系统处理缓慢。↵
每个 IP 每次只允许发出一个请求↵	合理的要求，避免了非法用户的恶意攻击↵	可能将低了易用性，但系统的安全性提高了。↵
数据库连接池↵	数据库连接池允许应用程序重复使用一个现有的数据库连接，而再不是重新建立一个，提高应用系统的性能↵	↵
容错性↵	能够对用户出现的误操作进行检测和处理，并给出相应的处理信息，可以提高系统的可用性。↵	↵
系统备份与恢复↵	增强系统的容错能力↵	操作系统和数据库软件发生崩溃时，恢复时间较长。↵

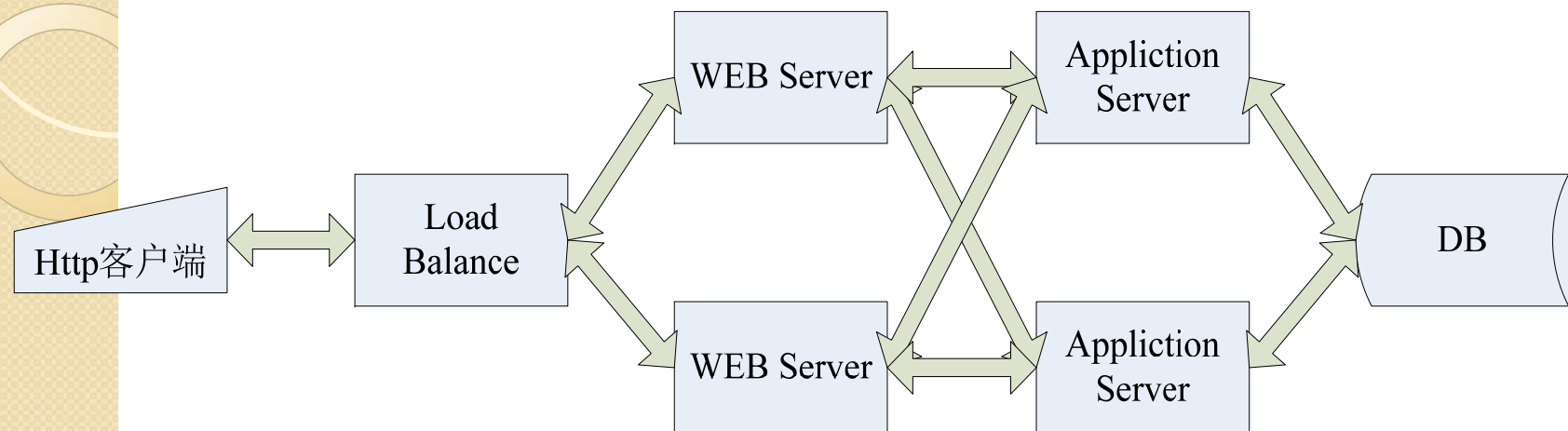
(2) 问题分析

在前面对系统结构的描述中，系统采用基于B/S的分层结构，系统部署在一台应用服务器上，这种结构有它独特的优点。但经过构架方法的分析，特别是对系统的关键质量属性和优先级最高的质量属性场景的分析，发现系统在上述场景下会出现如下的问题：


(1) 性能方面：在非常多的用户并发操作的情况下，单服务器系统将不能对用户的请求做出及时的响应，严重情况下服务器还会崩溃。

(2) 可用性方面：在仅有的一台应用服务器出现故障或者崩溃的情况下，用户将不能访问系统，故障恢复需要花费较长时间。

(3)改进系统的构架

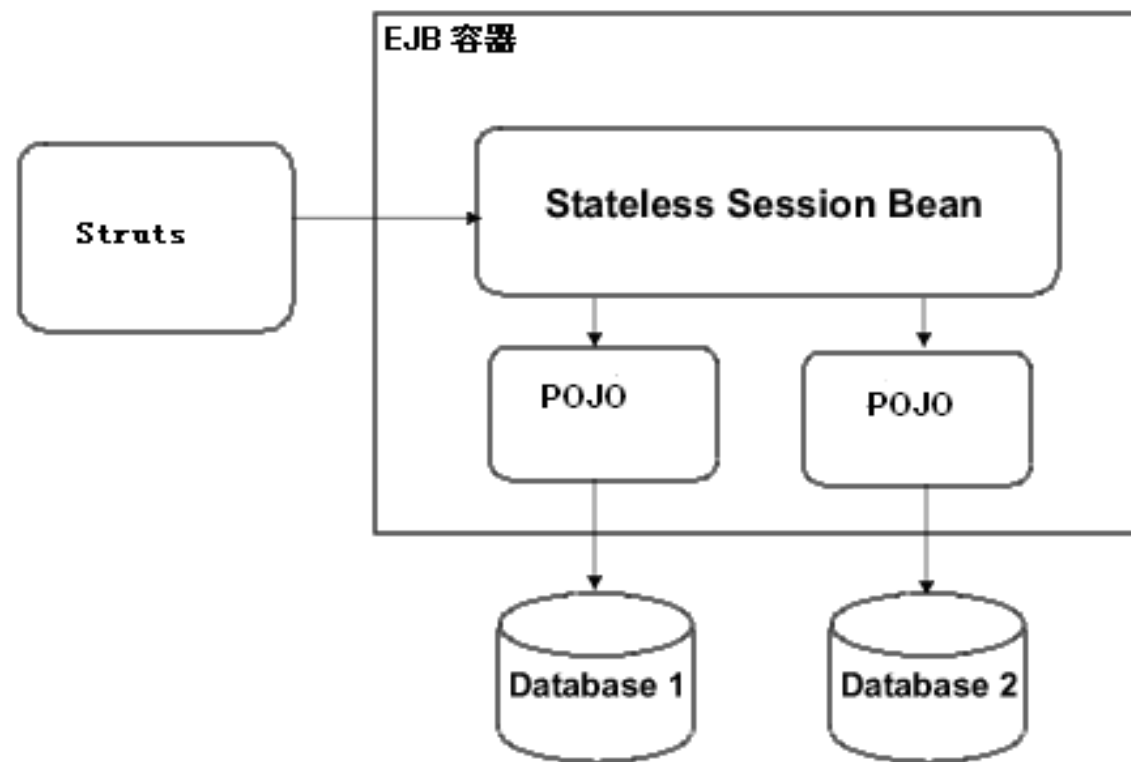


考虑到使用票务系统的用户数目非常庞大，这样造成用户对系统的访问请求数目和对系统进行业务操作的请求数目也非常庞大，改进后的系统采用多层分布式结构，使用Web服务器集群和应用服务器集群来实现，这种集群机制支持动态负载平衡（Load Balance）和容错机制，可以将用户的请求以及对用户请求的处理分发到负载低的服务器中，非常适合具有并发用户数多，服务地点分散等这些特点，有较高的稳定性，能有效避免访问流量过多导致服务器瘫痪以及整个系统因为某台服务器崩溃而彻底瘫痪。



为了使系统达到集群分布式的目的，在第一套方案的基础上，我们采用Spring介入EJB容器的方式，使用EJB的无状态会话Bean来封装业务逻辑，即调用POJO中的业务逻辑操作(POJO中包含了业务逻辑处理，在原来的SSH框架中它是指业务层的JavaBean，通过持久层与数据库交互，这些POJO通过IOC容器来管理)。这相当于在Struts和业务逻辑层之间增加了EJB，重用原SSH框架的业务逻辑，即系统框架变Struts+EJB+Hibernate+Spring，这种组合可以将视图和业务逻辑以及对数据库的操作很好的分离。

(4) 新的框架如下：



7 评审结论

总体而言，通过对质量属性场景的分析，我们发现了最先提出的构架方案的不足，由此得出改进后的构架方案。采用改进后的构架方案可以获得了良好的性能、易用性、安全性、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！