计算机系统应用 ISSN 1003-3254, CODEN CSAOBN
Computer Systems & Applications,2020,29(10):103−108 [doi: 10.15888/j.cnki.csa.007631]

E-mail: csa@iscas.ac.cn
http://www.c-s-a.org.cn
Tel: +86-10-62661041

大型遗留系统的性能与安全优化①

林平荣, 施晓权, 杨俊钦, 杨少冬, 林哲源

(广州大学 华软软件学院 软件研究所, 广州 510990)
通讯作者: 林平荣, E-mail: lin_pingrong@163.com

摘　要: 为了解决大型遗留系统日益凸显的性能与安全问题, 保证系统的可靠平稳运行, 分析和研究了遗留系统的性能及安全现状, 并在此基础上提出了优化策略. 分别从前端、数据库、服务器、系统架构、系统安全五个方面阐述了优化方案, 最后结合实例进行了优化策略验证. 验证结果表明, 该方案可以有效提升遗留系统的性能和安全等级.

关键词: 遗留系统; 性能优化; 安全优化; 系统架构; 微服务

引用格式: 林平荣,施晓权,杨俊钦,杨少冬,林哲源.大型遗留系统的性能与安全优化.计算机系统应用,2020,29(10):103–108. http://www.c-s-a.org.cn/1003-3254/7631.html

Performance and Safety Optimization of Large Legacy Systems

LIN Ping-Rong, SHI Xiao-Quan, YANG Jun-Qin, YANG Shao-Dong, LIN Zhe-Yuan

(Software Institute, South China Institute of Software Engineering, Guangzhou University, Guangzhou 510990, China)

Abstract: In order to solve the increasingly prominent performance and safety problems of large-scale legacy systems, and to ensure the reliable and stable operation of the system, the performance and safety status of legacy systems are analyzed and studied, and an optimization strategy is proposed based on this. The optimization scheme is described from five aspects: front-end, database, server, system architecture, and system security. Finally, the optimization strategy is verified with examples. The verification results show that the scheme can effectively improve the performance and security level of the legacy system.

Key words: legacy system; performance optimization; security optimization; system architecture; micro-services

软件开发技术的不断发展以及业务需求的不断变更, 使越来越多的软件系统成为遗留系统. 遗留系统指对新技术或业务需求不再适应, 但由于替换和修改成本过高而仍在被使用的计算机系统或应用程序[1]. 遗留系统大部分为早期开发并投入使用[2], 具有如下特征: (1) 业务逻辑复杂, 维护难; (2) 架构陈旧且高度耦合, 牵一发而动全身, 效率低; (3) 开发人员流动性大; (4) 系统文档缺失; (5) 业务逻辑与代码逻辑较为混乱. 对于企业或其他组织而言, 处理遗留系统面临选择, 要么抛弃遗留系统开发更适应新软硬件技术和新业务需求的系统, 要么保留继续运行使用. 鉴于新系统开发的人力物力成本较高, 且有一定风险, 同时遗留系统包含大量数据和业务逻辑信息, 是企业相当重要的企业资源[3], 完全抛弃会造成资源的丢失, 所以尽管遗留系统存在诸多问题, 仍处于运行状态[4].

随着时间的推移, 仍在运行的遗留系统的用户数以及数据量不断增加, 暴露出来的问题越来越多, 尤其是性能和安全问题. 经过文献检索, 发现有大量的关于 Web 应用系统性能和安全方面的优化方案被提出. 这些方法虽然在某种程度上有一定的优化效果, 但大部

① 基金项目: 广东省高校省级重点平台和重大科研项目 (2018KTSCX340)
Foundation item: Provincial Level Key Platform and Major Scientific Research Program of Guangdong Higher Educations (2018KTSCX340)
收稿时间: 2020-03-17; 采用时间: 2020-04-14; csa 在线出版时间: 2020-09-30

分并不是在遗留系统背景下所提出, 许多方法并不完全适用, 即使是针对遗留系统在工程方面的文献[4–7] 也较少涉及安全方面的优化. 基于此, 本文将在大型遗留系统特定背景下, 多角度挖掘遗留系统性能和安全方面的痛点, 综合考虑各方面优化技术, 但不赘述穷举大部分文献提及的通用方法, 最后给出具体优化方案.

1 性能优化

1.1 前端

遗留系统当时的开发时期并没有像现在各种开发技术的多样与规范. 前端开发技术主要以表格布局进行排版, 以切片进行内容展示, 以微软操作系统自带的 IE 浏览器作为浏览网页的载体. 随着人们的需求与技术的发展, 优雅的界面、响应的速度、数据的安全愈加获得人们的重视.

(1) 随着各个新的基于不同引擎的浏览器的加入, 遗留系统的 Web 前端出现了浏览器兼容性问题, 访问页面有排版错乱、按钮点击无反应等现象. 究其原因主要在各个厂商的浏览器内核对同一段代码解析不一, 导致页面呈现效果不统一, 操作效果无法被解析. 为了解决系统在各个浏览器的兼容问题, 可以选择优雅降级和渐进增强. 尝试从低版本浏览器到高版本浏览器的逐步优化的过程, 结合渐进增强的优化方式, 即能保证遗留系统存在的功能不被破坏又能在新一代的主流浏览器中进行操作.

(2) 遗留系统的页面布局皆为表格布局, 而现在编写代码更多的是讲究规范化和语义化, 且在非表格化数据显示的模块下并不提倡用表格标签做其他处理, 在使用 table 标签布局会使网页浏览的速度变慢, 这与 table 标签的渲染有关[8]. 可以用 div 加载方式进行渲染, 即读即加载, 同时引入异步加载局部更新技术. 局部更新只针对需要重新加载的局部信息, 可以有效提升响应速度, 提高用户体验, 服务器压力也会减少.

(3) 遗留系统中的 CSS 文件和 JavaScript 文件代码编写大多以覆盖为主. 无论是 HTML 文件还是 CSS 文件, 代码里都存在着一些被废弃的标签和样式写法, 文件编写没有进行一个整理和归类, 导致代码冗余. 采用现代前端开发技术的工程化能对遗留系统的脚本进行优化, 有利于代码的更好维护与管理. 前端工程化是以宏观的角度、更远的思维看待和开发项目的一种思想, 而模块化和组件化则是工程化的表现形式, 如图 1 所示. 模块化是将一个功能当做一个模块进行开发, 实现高效复用和分开管理, 不仅 JavaScript 代码能实现模块化, CSS 样式也可以实现模块化. 常见的 JavaScript 模块化方案有 AMD、CommonJS、ES6 Module 等, 而 CSS 模块化则有 less、sass、stylus 等预处理器进行实现. 组件化则是将页面上的可视区域和交互区域分成各个组件, 每个组件都是独立的, 每个组件包含模板、样式和逻辑. 不同的页面根据内容不同, 对各个组件进行自由组合. 模块化和组件化能对代码实现高效复用和降耦管理, 这也是现代前端开发技术倡导前端工程化的原因之一.

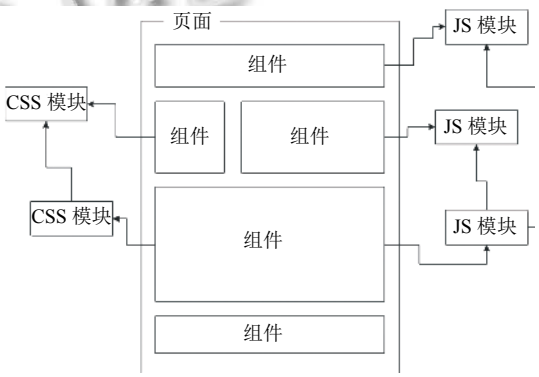

图 1 模块化和组件化示意图

1.2 数据库

数据库优化的主要目的就是最大限度地降低数据响应时间和提高数据库的吞吐量[9]. 一个应用程序的数据库性能, 通常是指数据的增、删、改、查耗时性能, 而耗时主要是 SQL 语句从执行开始到完成所消耗的时间, 这个过程主要指标有: 数据和索引等读写的 I/O 消耗时间、CPU 消耗时间、总的执行消耗时间. 以微软的 SQL Server 数据库为例, 可以通过 3 种工具 (SQL Server Profiler、查询窗口、SQL 活动监视器) 进行相应的性能查询分析.

一般情况下可以通过优化 SQL 语句、创建索引等手段来进行数据库优化, 然而在排除硬件资源充足、SQL 语句和索引创建合理的情况下, 有可能发现性能提升仍然不够理想, 分析深层次原因是数据库碎片积累过多且没有定期进行整理.

遗留系统运行年限较长, 信息数据、索引数据等都在不断增加, 相应存储页数量也不断增长, 数据页上也容易形成碎片 (即部分数据的大小超出了当前页的剩余空间, 导致无法在当前继续增加数据, 只能新开一

个页). 在运维的过程中只注重索引创建却忽视了索引碎片的维护, 从而导致性能的直线下降. 可以修复表的数据及索引碎片, 把相关数据文件重新整理, 但也不能过于频繁, 可根据表的读写频率按月、季、年来进行修复.

碎片整理前, 如果监测到某条 SQL 语句执行过慢, 通过查询其中某个表索引碎片, 如图 2 所示, 发现其中逻辑扫描碎片和区扫描碎片都超过 70%, 扫描密度只有 18%, 并且平均页密度也只有 58.94%, 这意味着“ROOM”表的数据页存在很多碎片, 其性能会损耗一部分在碎片当中. 这个时候单独查询表“ROOM”, 其逻辑读取 (已经查询过并且没有数据变更, 数据已被放到内存中) 就达到 72 次.

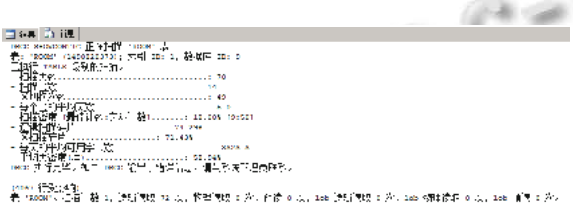

图 2 整理前逻辑扫描碎片

在扫描碎片整理后, 发现扫描密度变为 85.71%, 扫描逻辑碎片降低为 2.17%, 平均页密度提升到 99.25%, 并且其逻辑读取次数降低为 48 次, 如图 3 所示. 也就是在碎片较低的时候, 其逻辑读取次数相应的降低, 性能损耗降低了, 查询性能就提高了.

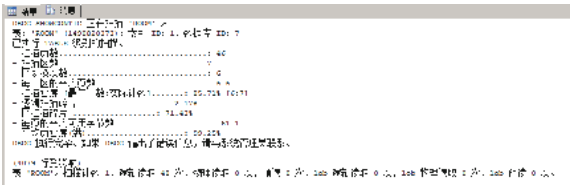

图 3 整理后逻辑扫描碎片

1.3 服务器

早期开发的遗留系统基本以单体形态部署在服务器上. 随着用户量及数据量的增加, 系统性能瓶颈逐渐暴露, 用户体验越来越差. 常规做法就是加大硬件投入, 使用性能更好的服务器, 但受限于系统采用的陈旧技术, 硬件资源使用率不是很高, 并不能很好地发挥服务器全部性能. 比如一些使用 JDK1.4 开发的 Web 系统, 它并不支持使用泛型, 所有的数据对象只能使用 Object 接收, 在对象转换之间花费了大量的不必要计算, 而在多线程只能使用传统的线程池方式实现. 使用 JDK1.5 以上开发的 Web 系统在数据对象转换的开销要比前者少得多, 性能也就优于前者, 但因此升级 JDK 版本会导致许多代码需要重写, 代价太大.

为了应对传统的单体应用模式无法承担大量并发业务请求的问题, 服务器集群技术应运而生. 为了能在服务器集群中合理分配业务, 使各个服务器都发挥应有的性能, 负载均衡机制及均衡算法成为了关键[10]. Nginx 作为中间服务器, 可以友好的切入到大型遗留系统, 实现动态拓展而不须改变系统本身, 从而改进系统的性能, 增加用户体验. 遗留系统可以采用 Nginx 负载均衡技术, 使用 ip_hash 算法, 将特定用户的请求分发给特定的服务器, 通过动态拓展服务器, 增加系统的承载能力, 动态拓展示意图如图 4 所示.

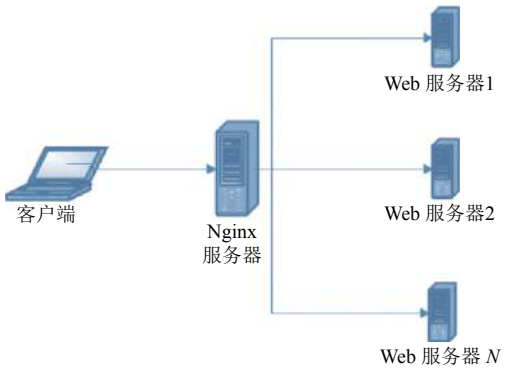

图 4 Nginx 动态拓展示意图

使用此方法可以在很大程度上缓解并发压力, 但是对数据库服务器的性能也有了更高要求, 因此对数据库也要进行优化, 同样是采用数据库动态拓展技术, 根据使用的数据库不同可以选择不同的集群捆绑中间件, 比较常用的有阿里 B2B 团队的 Cobar、360 团队的 Atlas 等等.

1.4 架构重构与演进

遗留系统陈旧的技术架构很难支撑日益复杂的业务线, 无法满足系统新的性能要求, 另外如 JDK、Weblogic 等未及时升级换代, 导致技术革新寸步难行, 只能在原架构基础上继续堆砌新业务, 逐渐演化成一个大型遗留系统. 为了提升性能及日后嵌入更多的新需求, 选择合适的技术架构进行重构, 保证架构平滑演进是必须考虑的. 此外, 还有一种方法就是对架构进行重写, 但很难确保系统的所有业务逻辑都被正确认识和实现, 而且成本和风险很高. 为了稳定现有业务的正

常开展, 选择在维护现有系统的基础上, 同时进行新技术的重构工作是比较好的方式, 这样既保障了业务的迭代需求, 又能进行新架构的重构.

针对目前大型遗留系统, 架构升级不能一蹴而就, 要依据不同功能以及业务逻辑, 完成系统级别的拆分, 同时对于第三方服务进行解耦, 拆分出来独立部署, 并且实现数据库的主从分离. 独立拥有拆分的 DB,避免业务引发连锁故障问题, 系统间通过 Hessian 实现 RPC 通信.

第 1 步. 将遗留系统拆分出多个业务逻辑相对独立的子系统, DB 暂不拆分, 多套系统继续共用一个 DB, 只是根据业务逻辑划分依赖的表, 不同业务逻辑系统不能相互访问, 只能访问归属自己的表, 进而保证原系统业务不受影响, 同时也保障新拆分的业务系统工作得以继续进行. 第 2 步. 通过系统层面的拆分后, 接下来就是 DB 层面的拆分, 将各个系统依赖的表独立拆分, 分别放到不同的 RDS 数据库, 做到物理层隔离. 这两个步骤充分将大型遗留系统解耦, 独立拆出服务化组件, 供其他服务调用.

有了合理的服务化拆分, 可轻易演化为微服务架构. 微服务可以将单体应用细化为可相互协作、配合的一组小服务, 使得服务间开发自由、独立部署、易于维护[11]. 微服务架构形式在满足系统功能之外, 对系统的非功能特性也有显著提升[12]. 将那些容易引起性能问题的业务拆分独立微服务组件, 微服务都围着具体业务进行构建, 由专人研发和维护, 结合架构拆分, 实施微服务架构. 实施过程中, 会产生很多微服务以及子系统, 各个系统的配置信息都以明文形式在配置文件中, 固定化了各种定时任务的执行规则, 难以集中管理, 可以选择 Disconf 和 Elastic-Job 分别作为分布式配置管理、任务调度.

通过以上步骤实施, 系统与微服务组件已非常容易扩展. 以服务为中心, 全面构建微服务组件, 大型遗留系统可以具备高可用、高性能等特性. 随着服务越来越多越复杂, 服务链路调用必须加以跟踪, 以便快速发现调用过程中需要优化的地方, 可以引入美团点评的 APM 工具 Cat 实现实时监控, 与 Dubbo 快速整合, 通过全局链路 ID 实现链路跟踪功能. 虽然对大型遗留系统进行拆分, 但还未做到快速弹性扩展. 接下来, 拆好的微服务与 Docker 容器结合, 抵御业务高峰期冲击, 快速自动扩展服务器, 低峰时期, 可以自动回收服务器. 整个微服务架构如图 5 所示.

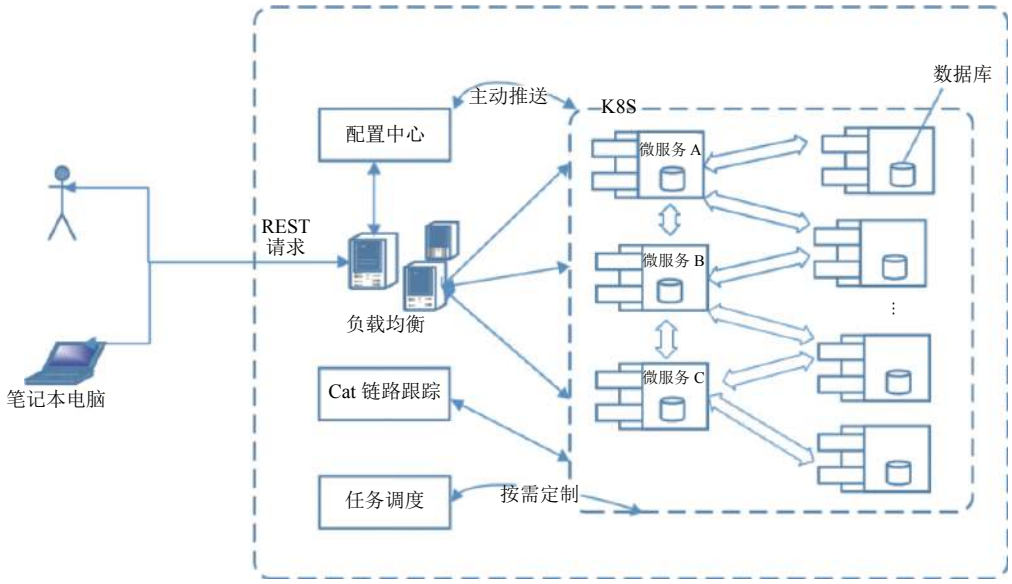

图 5 微服务架构图

2 安全优化

2017 年 6 月 1 号《中华人民共和国网络安全法》正式颁布实施, 各大企业、高校对各自现有运作的系统安全问题更加重视. 遗留系统使用的技术过于陈旧导致一些安全问题修复难度增加.

以基于 JDK1.4 开发的遗留系统为例, 系统采用

Cookie 技术来维持会话, 通过使用 js 进行 XSS 脚本攻击, 可以轻易地获取 Cookie 信息. 如果 Cookie 中设置了 HttpOnly 属性, 那么通过 js 脚本将无法读取到 Cookie 信息, 这样能有效地防止 XSS 攻击, 增加 Cookie 的安全性. 但是由于 JDK1.4 无法设置 HttpOnly 这一属性, 因此在原系统层面无法解决该问题, 只能引入中间件拦截用户请求, 进行相关处理后再请求系统, 即给系统加多一层防火墙来保护系统的安全. 采用 Nginx 来进行攻击防护, 通过在 Nginx 的 proxy 模式将客户端的请求拦截, 利用 HttpOnly 防止 XSS 攻击获取 Cookie 信息, SameSite 防止 CSRF 攻击和用户追踪, 再结合 ngx_lua_waf 实现安全 web 应用防火墙. 通过该模块可以防止 ApacheBench 之类压力测试工具的攻击、防止 SQL 注入、fuzzing 测试、XSS、SSRF 等 Web 攻击, 亦可屏蔽常见的扫描黑客工具, 屏蔽异常的网络请求、屏蔽图片附件类目录 php 执行权限、防止 webshell 上传等.

遗留系统采用 HTTP 协议开发, HTTP 协议以明文方式发送内容, 不提供任何方式的数据加密. 如果攻击者截取了 Web 浏览器和网站服务器之间的传输报文, 就可以直接读取其中的信息, 特别是在传输账号密码这一过程, 使得用户信息存在泄露的可能. 采用 SSL 加密可以在数据传输过程进行加密, 让用户信息的安全得到保障. 通过在 Nginx 配置 SSL 证书还可以将 HTTP 请求强制定向为更安全的 HTTPS 请求, 从而提高系统的安全性.

增加 Nginx 中间件可以进一步提高系统安全, 但是由于系统本身存在的一些漏洞, Nginx 中间件无法全面保护系统. 如遗留系统中到处可见的明文 GET、POST 请求, 每一个参数都暴露在客户端, 用户只需要拼接特定的 URL 就可以进行越权操作. 拼接 URL 越权访问有两种方式, 分别为横向越权和纵向越权. 为了防止攻击者尝试访问相同权限用户资源的横向越权现象发生, 建立用户和目标资源的绑定关系. 只有通过绑定关系的用户才可以访问或者在请求参数进行关键参数间接映射, 避免直接访问; 对于低权限越级访问高权限资源的纵向越权, 采用 RBAC 访问控制机制, 定义不同角色的访问权限, 每个用户都有特定的权限, 当用户在执行某个动作或者进行某些行为时, 通过角色判定是否允许进行操作. 此外, 将请求路径和参数用 MD5 算法按一定的规则进行加密, 再返回给客户端处理, 防止通过拼接参数进行越权操作.

3 实例验证

广州大学华软软件学院信息管理系统于 2004 年开发建设, 现有招生、教务、科研、财务、学工、人事、后勤七大管理模块, 至今已运行 16 年, 累计源代码将近 60 万行, 涉及数据表 400 多张, 服务全校 15 000 多名师生. 系统开发时间早, 架构技术较为陈旧, 功能模块之间耦合度高, 属于典型的大型遗留系统. 随着用户量和数据量不断攀升, 系统的性能和安全面临很大的考验. 根据系统暴露的问题, 结合本文提出的优化方案对系统进行优化验证.

前端: 从兼容、渲染、规范、响应数据等进行不同层次的优化. 优化后系统不仅兼容 IE9 及以上版本的浏览器, 在谷歌浏览器、火狐浏览器等主流浏览器下布局也得以改正且可添加主流浏览器支持的样式效果, 同时将 JavaScript 代码更改为通用写法与短路写法, 在主流浏览器下按钮失效现象也得以解决. 在表格布局方面, 将 table 标签的代码进行重构, 将语义化的标签代替用于布局的 table 标签, 保留用于展示数据表格的 table 标签, 代码清晰, 方便阅读, 而且网页从渲染到展示内容的速度也得到一定程度的改善. 在工程化的思想下进行重构, 对代码整体进行一个规范和梳理, 文件不仅体积减小而且后期代码管理也更加便利, 同时对需要频繁加载数据的页面采用异步加载进行局部更新, 服务器传输压力得以减小且响应数据的效率也得到相应的提高.

数据库: 针对排课、教室资源查询、考勤登记等响应速度较慢的功能模块进行分析与优化, 找出耗时较高的 SQL 语句, 然后重建数据库中涉及相关表的索引并不定期对数据库进行索引碎片整理, 代码层面则对业务数据的存储逻辑进行优化. 采用 JMeter 工具分别对优化前后各模块的响应时间进行测试与采集, 具体如表 1 所示, 可以看出优化之后响应时间明显缩短.

表 1 优化前后响应时间对比

模块	并发数	优化前平均响应时间(s)	优化后平均响应时间(s)
排课	200	6.3	2.7
教室资源查询	1000	17.6	1.7
考勤登记	1000	11.2	2.5
教师课表查询	1000	7.2	2.3

服务器与架构: 主要针对系统的选课模块进行了优化. 选课模块优化前存在较大的性能问题, 在并发数

3000 左右就会出现异常或导致系统宕机. 现通过服务器动态扩展方式对选课模块进行优化, 改变传统单体应用模式, 同时进行微服务化演进. 优化之后, 用户并发数 3000 时没有出现异常, 逐渐增加用户数量后系统表现也较为稳定, 可以满足 10 000 人同时选课的需求, 并且服务器内存、磁盘 IO、CPU 得到了充分的利用, 达到了系统运行预期效果.

安全: 优化之前我们按照广东省教育厅公布的《关于开展信息系统安全等级保护工作的通知》的工作要求[13], 采用安全扫描工具 AppScan 对系统进行扫描, 同时开展了人工渗透测试, 发现系统存在 SSRF、反射型 XSS 等诸多中、高危漏洞. 结合本文提出的优化方案, 通过引入 Nginx 中间件, 进行了 HTTPS 的部署升级, 解决了通信加密的问题; 通过 Nginx 过滤, 解决了 XSS 攻击、SSRF 漏洞、SQL 注入等问题; 通过小跨越的升级框架版本, 解决 Apache Struts 历史漏洞问题. 图 6 和图 7 分别是 AppScan 在优化之前和之后对系统进行安全扫描的风险图.

图 6 优化前的风险图

图 7 优化后的风险图

由图 7 可以看到优化之后已经没有了中、高危风险, 有效防止黑客模仿用户身份执行不合法行为, 证明了安全优化方案的有效性. 不足之处在于在修补漏洞的同时衍生了一些低危漏洞, 但对于整体系统而言安全方面提升到了更高一个等级.

4 结束语

本文先阐述了大型遗留系统在运行过程中可能存在的性能与安全问题, 然后在大型遗留系统特定背景下, 多角度挖掘遗留系统性能和安全方面的痛点, 分别从前端、数据库、服务器、系统架构、系统安全五个方面给出具体优化方案, 最后通过广州大学华软软件学院信息管理系统进行了优化策略验证. 本文优化方案具有一定的代表性和典型性, 可以有效提高遗留系统的性能与安全等级, 对于大型遗留系统的优化改造具有一定参考意义.

参考文献

1 Yang HJ, Ward M. Successful Evolution of Software Systems. Boston: Artech House, 2003. 2–6.
2 Warren I. The Renaissance of Legacy Systems: Method Support for Software-System Evolution. London: Springer, 1999. 1–7.
3 罗伯特·C. 塞克德, 丹尼尔·普拉考士, 格雷斯·A. 刘易. 遗留系统的现代化改造——软件技术、工程过程和业务实践. 梁海华, 译. 北京: 清华大学出版社, 2004.
4 李寒. 遗留系统再工程的若干问题研究 [博士学位论文]. 大连: 大连理工大学, 2013.
5 吴博. 面向分布式架构的遗留系统再工程 [硕士学位论文]. 杭州: 浙江大学, 2010.
6 汪志成. 从单机至分布式架构的遗留系统再工程 [硕士学位论文]. 杭州: 浙江大学, 2011.
7 纪鹏. 基于 REST 对遗留系统再工程研究与实现 [硕士学位论文]. 北京: 北京邮电大学, 2010.
8 佚名. Table 布局的优缺点介绍 Table 布局的优缺点介绍. https://www.jb51.net/web/175937.html. [2014-06-12].
9 左渭斌. 基于 ORACLE 数据库性能优化及监控研究. 才智, 2011, (26): 65.
10 戴伟, 马明栋, 王得玉. 基于 Nginx 的负载均衡技术研究与优化. 计算机技术与发展, 2019, 29(3): 77–80. [doi: 10.3969/j.issn.1673-629X.2019.03.016]
11 辛园园, 钮俊, 谢志军, 等. 微服务体系结构实现框架综述. 计算机工程与应用, 2018, 54(19): 10–17. [doi: 10.3778/j.issn.1002-8331.1808-0014]
12 刘旺森. 基于微服务架构的遗留系统重构研究与实践 [硕士学位论文]. 呼和浩特: 内蒙古大学, 2019.
13 佚名. 关于开展信息系统安全等级保护工作的通知. http://edu.gd.gov.cn/zxzx/tzgg/content/post_1604645.html. [2010-01-27].