

**毕业论文（设计）外文翻译**



**题目：基于SPA的购物网站设计**

**学 院：管理工程与电子商务学院**

**专 业：电子商务**

**班 级：商务16A**

**学 　 号：1612200132**

**学生姓名：王勇**

**指导教师：赵文敏**

**二○一七年 十二 月**

Performance Driven Development Framework for Web Applications

作者：K. S. Shailesh\* and P. V. Suresh

国籍：India

出处：Global Journal of Enterprise Information System

原文正文：

**Abstract**

The performance of web applications is of paramount importance as it can impact end-user experience and the business revenue. Web Performance Optimization (WPO) deals with front-end performance engineering. Web performance would impact customer loyalty, SEO, web search ranking, SEO, site traffic, repeat visitors and overall online revenue. In this paper we have conducted the survey of state of the art tools, techniques, methodologies of various aspects of web performance optimization. We have identified key web performance patterns and proposed novel web performance driven development framework. We have elaborated on various techniques related to different phases of web performance driven development framework.

**Keywords**: Application, Framework, Images, Traffic, Video, Web

1. **Introduction to Web Performance Optimization (WPO)**

Pages with good performance increase revenue2,3 and improve the search engine ranking1 . Page performance also has positive impact on user traffic56,57 and the download rate impacts the perceived success by end users71.

Web performance optimization (WPO) involves all methods to improve the performance of web page87. The key components used in WPO are web content, images, videos, CSS/JS files, XML/JSON files and such presentation components. WPO also involves various performance rules, techniques and processes to improve end to end performance optimization of the web page.

* 1. **Organization of the Paper**

The paper is organized as follows. We will start with introduction concepts of WPO and we will then examine various aspects of WPO such as impact and dimensions of WPO in the introduction section. In the next section we elaborate the performance driven development framework. We explain each step in the performance driven development framework including performance based design, performance based development, performance based testing and performance monitoring. In each of these phases we discuss relevant performance rules, design principles and performance best practices. Wherever applicable, we will provide the Java-based web application examples

* 1. **Impact of WPO**

WPO has impact in following aspects:

• Customer churn: Research indicates that customers would abandon the slower web pages88,91,92

• User impact: User experience is drastically impacted due to page performance. The performance of landing/ gateway pages and key processes is directly co-related to overall user experience

Site Traffic: Site traffic is impacted if the page takes more than 3 seconds89 and most users expect the page to load within 2 seconds89.

• Conversion rate increases 74% when page loads within 2 seconds

• Page abandonment rate increases to 40% if page takes > 3 seconds

• Nearly half the population expects the page to load within 2 seconds97

• Revenue: Online revenue is directly correlated to the performance of key pages and transactions for ecommerce sites.

• Multi-device optimization: An optimized web page also impacts the performance on various devices.

• Search engine ranking: Google includes site speed in search ranking algorithm.

• Omni-Channel advantage: A good performing page can also be easily access by mobile devices.

* 1. Dimensions of WPO The process of Web optimization can be analyzed from several dimensions:

• Optimization of Request pipeline processing Systems: In this category we will examine all the systems involved in the web request processing pipeline. This involves browser software, CDN, proxy server, network, load balancer, web server, application server, integration middleware, backend services, database server and such

• Client-side and server side optimization: Client-side performance optimization includes all optimizations performed on client-side presentation components such as HTML pages, images, assets and such. Server side optimization includes performance tuning of server –side components such as fine-tuning business components, setting optimal server configuration, right infrastructure sizing and such

• Design time and run time optimization: Design time optimization includes the static and offline performance optimizations activities such as performance code reviews, performance testing, and offline performance tuning and such. Run time and dynamic performance optimization activities include real-time performance monitoring and notification, run time performance optimization and such.

• Web component optimization: Another aspect of web optimization is to optimize each of the constituent’s web components such as HTML, images, JavaScript, CSS, Rich media files and such.**Table 2. Categorized Performance Rules**

|  |  |  |
| --- | --- | --- |
| **Category** | **Performance Rule** | **Impact on web performance** |
| Request Optimization  Web object size optimization  HTTP Header Optimization  Asset placement  Image optimization  Network optimization  External Dependency optimization  Web Application design optimization | Reduce the number of HTTP Requests  Merge the static assets such as JS and CSS files  Remove all duplicate file includes  Remove all invalid URLs which result in HTTP 404  Load the JavaScript sascynhronously  Minimize usage of iframes  Minimize redirects  Cache DNS records  Remove any unused CSS, JS file includes  Minify JS and CSS files  Compress images  Leverage gzip for HTTP compression  Remove white space in the HTML document  Leverage cache headers for static assets (images, JS, CSS, JSON and other binary files) using Cache-Control header with maxage directive  Use expires header for the assets  Place CSS files at the top  Place JS files at the bottom  Externalize inline JS or CSS  Asynchronous image load  Optimize image size  Convert JPEG image formats to progressive format  Optimize image dimensions  Use image maps  Use CSS sprites  Usage of CDN  Use multiple asset hosting servers (for hosting images, videos and other multimedia content)  Identify all external scripts and HTTP requests which impact the page performance and which block the page load and optimize them  • Perform regular and iterative performance testing to identify performance bottlenecks and fix them. Use automated and manual performance code reviews at regular intervals.  • Use light-weight service based integration model and load the data asynchronously ondemand | Reduces the consumed bandwidth and data transferred  Merging would reduce the number of HTTP requests and would improve the page response times by about 38%  Avoids unnecessary and invalid HTTP requests  This would reduce the blocked loading of JS files  iFrames block the loading of parent window till iframe source is loaded and hence affects load times  Minimize additional requests  We can reduce the DNS lookup time through DNS cache maintained at browser level  Minimizes HTTP requests  Minification would reduce the size of JS and CSS file  Compresed images would reduce overall page size  Compression would reduce the overall page size by about 70%  Removal of white space would reduce the overall page size  Allows browsers to optimally cache the assets  CSS elements in the head tag  JS files at the bottom would improve the perceived page load time. I would avoid the blocked loading of other assets  Enables browser caching and parallel downloads  Load the images on-demand and in asynchronous when they are visible in the user’s view port  Use the right size image based on the requesting device  This would reduce the overall image size  Specify the exact width and height for all images  Reduces multiple image requests  All images are combined into a single one and the required image is displayed using style rules. This reduces number of image requests  CDN would optimize the resource request by serving the resource from nearest location to the requestor  Allows browsers to download the content in parallel.  Iterative performance testing uncovers performance bottleneck during early stages |

1. **Performance Driven Development Framework**

Performance driven development approach can be adapted in following phases for ground-up development project:

• Defining sound performance based design guidelines

• Implementing the design guidelines during development

• Thorough testing strategy to cover all performance scenarios

• Continuous real-time monitoring

The various phases are depicted in figure 3.

* 1. **Web performance Design**

In this section, we will look at the design guidelines and best practices for achieving optimal web performance. The books provide excellent performance guidelines from web perfor mance stand point. Web developers and architects can use this as reference while developing web applications. Some of these optimizations are also available as filters for Apache’s mode\_pagespeed module.

As 80%66 of load time is spent in making HTTP requests for non-HTML content, we could look at ways to optimize these web components in table 2. The key page performance design principles are listed below:

• Light weight design

• Include only core functionality on landing/gateway JSPs

• Highly optimized/compressed marquee image and other media

• SLA-based 3rd party integrations on frequently used JSPs

• Search centered experience

• Position highly optimized search as key tool for information discovery

• Provide intuitive information architecture

Think Asynchronous alternatives

• Use AJAX tags in page to optimize perceived page load time

• Lazy loading data model for page components

• Omni-channels optimized

• Page components for mobile devices

• Layered architecture

• Separation of concerns

Given below are performance anti patterns in a typical Java web application:

• JSP Page size contributors

• Media (Marquee image, flash, video)

• JavaScript files

• CSS

• Uncompressed/un-optimized images

• JSP Includes/calls

• Numerous JS/CSS includes

• Duplicate calls

• Broken links

• Unnecessary calls

• Other common causes

• Placement of JS/CSS calls

• Bloated size of JSP

• Frequent resource requests with huge payload

• Inline styles and JS logic

**Table 1. Performance Bottleneck and anti-patterns**

|  |  |
| --- | --- |
| **Bottleneck Area** | **Performance anti-patterns** |
| Web page Design  Third-party components  Network bandwidth  Server configuration  Infrastructure capacity  Performance testing  Page code  Service calls  Integration design  Process validation  Omni channel strategy | • Heavy landing/gateway pages  • UI design with many components and functionality  • Pages designed with huge images/flash files  Third-party Scripts and widgets would block page load and impact overall page performance  Usage of sub-optimal network bandwidth across internal systems  Not adopting optimal server settings for parameters such as heap memory, thread pool size, connection pool size etc.  • Usage of sub-optimal memory, CPU, disk capacity for servers  • Not conducting load testing, stress testing, endurance testing and related performance tests  • Not conducting all necessary performance tests (such as load test, stress test, endurance test) for web application  • Conducting performance testing at the end of the application development  • Not conducting Omni-channel testing to test performance on mobile platforms  • Not conducting performance code review  • Not performing iterative performance testing  • Non validated frequent service calls  • Heavy usage of synchronous service calls  • 3rd Party component integration without proper SLA framework  • Lack of performance testing of overall steps and/or for process/transaction  • Absence of mobility enabled sites or lack of multi-device testing. |

**2.2 Performance based Development**

Performance based development is the key step in the performance driven development framework. Firstly, in this phase we will identify the performance bottlenecks and performance anti patterns and then we will apply all the performance optimization rules and best practices. We will also look at content optimization and the impact of security on WPO. In this section we will also look at optimizing performance for existing web applications.

2.2.1 **Web Performance bottlenecks and Web Performance anti-patterns**

Let us look at common performance bottlenecks and anti-patterns which impact the web performance. Table 1 provides a list of commonly occurring performance bottleneck

**2.2.2 Performance Optimization Practices**

Given below are the performance optimization best practices and thumb rules which can be used during development stages:

Given below are the performance optimization rules for a Java-based web application and these rules can be used for performance code review and as a performance checklist:

• Maximum usage of AJAX

• Cache list items in web page

• Properly scoped managed beans for JSF

• Precompiled JSP

• Optimize intervals for checking JSP and servlet modification

• Usage of JSP cache

• Tune session timeouts (optimal 30 mins) and call session. invalidate() for logout

• Disable JSP/Servlet auto-reload

• Heavy objects in HttpSession needs to be avoided

• Usage of gzip compression wherever supported

• Include the static JSP fragments like header, footer using include directive instead of include action

• Wherever required pre-load servlet and cache the application data using “load-on-startup” feature

• Perform client side validation to avoid un-necessary server round trip

• One time creation of cached data in init() method

• Disable auto-reload feature unless required.

性能驱动的Web应用程序开发框架

作者：K. S. Shailesh\* and P. V. Suresh

国籍：印度

出处：全球企业信息系统杂志

# 概要：

Web应用程序的性能至关重要，因为它可能会影响最终用户体验和业务收入。Web性能优化（WPO）需要处理前端表现的性能。 Web所表现的性能会影响客户忠诚度，搜索引擎优化，网络搜索排名，搜索引擎优化，网站流量，回头访客量和整体在线收入。在这篇文章中，我们对Web性能优化各个方面的工具，技术和方法进行了研究。我们已经确定了关键的Web性能模式，并提出了新的Web性能驱动开发框架并详细阐述了与Web性能驱动开发框架在不同阶段的相关的各种技术。

关键词：应用，框架，图片，流量，音频，互联网

# Web性能优化概要

性能良好的页面可以增加网站的在线收入并提高搜索引擎排名。 页面性能对用户流量也有正面的影响，页面的加载速度会影响最终用户的感受。

Web性能优化（WPO）涉及提高网页性能的所有方法。 WPO中的关键要素是Web内容，图像，视频，CSS、JS文件和XML/JSON文件。WPO还涉及各种性能规则，技术和流程，以改进网页的端到端性能优化。

## 本文的结构 本文的结构如下。 我们开始会介绍WPO的概念，然后我们将在引言部分中讨论WPO的各个方面，比如WPO的影响和维度。 在下一节中，我们将详细介绍性能驱动的开发框架。 我们解释性能驱动的开发框架中的每一步，包括基于性能的设计，基于性能的开发，基于性能的测试和性能监控。 在这些阶段中，我们将讨论相关的性能表现规则，设计原则和页面性能表现的最佳实践。 我们将提供基于Java的Web应用程序示例。

## 网页性能优化的影响

网页性能优化会涉及到以下几个方面:

* 客户流失：研究表明，客户会放弃加载较慢的网页
* 用户影响：用户体验会受到页面性能的严重影响。网站的着陆页的表现会直接影响到网站所有用户的体验
* 网站流量：如果页面加载的时间超过3秒，则网站流量就会受到影响，大多数用户希望页面在2秒内加载
  + 在2秒内页面加载完成时用户留存的转换率提高了74％
  + 如果页面加载的时间大于3秒，当前页面被用户放弃的几率会增加到40％
  + 近一半的用户希望页面在2秒内加载完成
* 收入：在线收入与页面性能直接相关,特别是电子商务网站的关键交易页面的性能
* 多设备优化：页面的性能优化表现能作用于不同的设备。
* 搜索引擎排名：Google搜索排名算法中就会考虑网站页面加载的速度
* 全方位渠道优势：性能良好的网页可以也很容易被移动设备访问。

## 网页性能优化涉及的方面

Web页面优化的过程可以从以下几个方面进行分析：

* 请求信道系统的优化：在这个环节中，我们将检查所有涉及Web请求处理。这涉及浏览器软件，内容分发网络(CDN)，代理服务器，网络，负载均衡器，Web服务器，应用服务器，集成的中间件，后端服务，数据库服务器等。
* 客户端和服务器端优化：客户端性能优化包括在客户端的表现内容（如HTML页面，图像，资产等）。服务器端优化包括服务器端组件的性能调整，如微调业务组件，设置最佳服务器配置，考虑合适的基础架构等。
* 设计时间和运行时间优化：设计时间优化包括静态和离线性能优化，如代码性能评估，性能测试和离线性能调优等。 运行时间和动态性能优化包括实时的性能监控，运行时的性能优化等。
* Web组件优化：web 另一个优化的方式就是优化web页面所需要加载的每一个文件比如HTML文件，图片文件，JavaScript, CSS，富文本等。

**表2. 分类的性能规则**

|  |  |  |
| --- | --- | --- |
| 分类 | 表现规则 | 网页性能影响 |
| 请求优化 | 减少http请求 | 减少消耗的带宽和数据传输 |
| 文件大小优化  HTTP 头部优化  资源安置  图片优化  网络优化  外部依赖优化 | 合并静态资源比如JS和CSS文件  删除重复的文件  将所有不合法的404路径移除  异步加载js文件  最少程度使用iframe  缓存DNS记录  移除任何不需要的CSS JS文件  压缩CSS,JS文件  利用gzip进行HTTP压缩  删除HTML文档中的空格和换行  利用带有maxage指令的Cache-Control标头，为静态资产（图像，JS，CSS，JSON和其他二进制文件）  将css文件放到头部，js文件放到尾部  异步加载图片  优化图片的大小  将JPEG图像格式转换为渐进格式  优化图像尺寸  使用CSS精灵图  使用CDN  识别影响页面性能的所有外部脚本和HTTP请求，并阻止页面加载并 | 合并会减少HTTP请求的数量，并将页面响应时间提高约38％  避免不必要的和不正确的HTTP请求  这样不会导致页面加载阻塞  iframe会阻塞主窗口的加载，直到iframe源文件加载完毕，才会继续加载主窗口，从而影响加载时间  我们可以通过在浏览器端维护的DNS缓存来减少DNS查找时间  减少HTTP的请求  压缩会减少JS和CSS文件的大小  压缩能使得整个页面大小减少约70％  去除空格将减少整体页面大小  允许浏览器最佳地缓存资源  CSS元素放到头部  JS文件放到文档底部加载不会阻塞页面  在用户视窗中可见图片时，按需异步加载图片  根据请求设备来加载正确尺寸的图片  这会减少图片的整体大小  指定所有图像的确切宽度和高度  将所有需要的图标合并到一张图片。可以减少http请求  CDN将通过从最近位置向请求者提供资源来优化资源请求 |

# 性能驱动的开发框架

性能驱动型开发方法可以应用在底层项目开发的几个阶段中：

•定义健全的性能设计指南

•在开发过程中实施设计指南

•彻底的测试策略覆盖所有性能方案

•持续的实时监控

在本节中，我们将介绍实现最佳网络性能的设计指南和最佳实践。 该书66,67从网站性能角度出发提供了出色的性能指南。 Web开发人员和架构师可以在开发Web应用程序时将其作为参考。 其中一些优化也可用作Apache的mode\_pagespeed模块的过滤器

由于80%加载时间用于对非HTML内容进行HTTP请求，因此我们可以参考表2的方法来优化这些资源。

关键页面性能设计原则如下：

* 减重设计
  + 网站的着陆页仅包括核心功能
  + 对图片和多媒体内容进行压缩
* 以搜索为中心的体验
  + 将高度优化的搜索定位为获得网站信息的关键工具
  + 提供直观的信息架构
* 首选异步的方式进行数据通讯
  + 使用页面中的AJAX去后台请求数据，加强页面的加载速度。
  + 采用懒加载的模块数据的方式来优化页面
* 全设备优化
  + 移动设备的页面
* 分层架构
  + 关注点分离

下面给出的是典型Java Web应用程序中的性能反模式：

* JSP页面大小因素
  + 媒体（字幕图像，闪光灯，视频）
  + JavaScript文件
  + CSS
  + 未压缩/未优化的图像
* JSP包含/调用
  + 许多JS / CSS文件需要多次请求
  + 重复请求
  + 中断链接
  + 不必要的请求
* 其他常见原因
  + 放置JS / CSS调用
  + JSP的膨胀大小
  + 频繁的资源请求会有巨大的负载
  + 内联样式和JS逻辑

表1. 性能瓶颈和反性能模式

|  |  |
| --- | --- |
| 瓶颈项 | 反性能模式 |
| 网页设计 | 着陆页面需要加载的资源太多 |
| 第三方组件  带宽  服务器设置  基础设施容量  性能测试  页面代码  服务呼叫  整合设计  过程验证  全方位渠道战略 | 用户界面的设计需要过多的组件  网页设计包含了大量的图片和flash文件  第三方脚本和小部件的加载将阻止主页面加载并影响整体页面性能。  内部系统使用次优网络带宽  没有对服务器的堆内存，线程池大小，连接池大小等参数进行最佳设置。  使用次优内存，CPU，服务器的磁盘容量  不进行负荷测试，压力测试，耐力测试和相关的性能测试  不进行所有必要的性能测试（如负载测试，压力测试，耐力测试）的Web应用程序  在应用程序开发结束时进行性能测试  不进行全渠道测试来测试移动平台上的性能  不进行性能代码审查  不执行迭代性能测试  未经验证的频繁服务请求  大量使用同步服务调用  第三方组件集成没有用适当的SLA（服务等级协议）框架  缺乏整体步骤的性能测试  没有对移动端进行网站的测试或缺乏多设备测试 |

2.2基于性能的开发  
基于性能的开发是性能驱动开发框架中的关键步骤。 首先，在这个阶段我们将识别性能瓶颈和反性能模式，然后我们将应用所有的性能优化规则和最佳实践。 我们也将看看内容优化和安全对WPO的影响。在本节中，我们还将着眼于优化现有Web应用程序的性能。  
2.2.1 Web性能瓶颈和Web反性能模式

让我们看看影响网络性能的常见性能瓶颈和反模式。 表1列出了常见的性能瓶颈

2.2.2性能优化实践  
以下是在开发阶段可以使用的性能优化最佳实践和规则：

下面给出的是性能优化规则  
基于Java的Web应用程序和这些规则可用于性能代码审查和作为一个性能清单：

* 最大限度的使用AJAX
* 在网页中缓存列表项目
* 适用于JSF的托管Bean预编译的JSP
* 优化检查JSP和Servlet修改的时间间隔
* 使用JSP缓存
* 调整会话超时（最佳30分钟）和呼叫会话。
* 禁用JSP直接加载
* HttpSession中的不要存储太大的对象
* 在任何支持的地方使用gzip压缩
* 包括静态JSP片段，如页眉，页脚使用
* 包括指令而不是包含动作
* 无论何处需要预加载servlet并缓存应用程序
* 数据使用“加载启动”功能
* 执行客户端验证，以避免不必要的服务器往返
* 在init（）方法中一次创建缓存数据除非需要，否则禁用自动重新加载功能。