PaperDay标准版检测报告简明打印版

相似度：30.8%
编　号：OMZ5PDMXEIEX1Q6N
标　题：毕业论文_计科12002丁斌斌_初稿
作　者：丁斌斌
长　度：37920字符
时　间：2024-05-15 18:13:30
比对库：本地库（学术期刊、学位论文、会议论文）；互联网资源

本地库相似资源（学术期刊、学位论文、会议论文）
1. 相似度：0.38%　篇名：《通用高性能密码服务系统模型》
来源：《微电子学与计算机》　年份：2016　作者：寇文龙;陈莉君;
2. 相似度：0.34%　篇名：《如何提高板框压滤机的使用效率》
来源：《机械管理开发》　年份：2019　作者：张淑萍;
3. 相似度：0.18%　篇名：《通过压力测试改善Web应用程序的性能》
来源：《数字技术与应用》　年份：2017　作者：杭佳祺;王川;罗西;
4. 相似度：0.13%　篇名：《本科毕业论文致谢》
来源：《应用写作》　年份：2016　作者：杨墨;
5. 相似度：0.11%　篇名：《煤质化验中如何确保数据的可靠性》
来源：《当代化工研究》　年份：2019　作者：许艳琴;
6. 相似度：0.10%　篇名：《论数字档案资源数据化发展》
来源：《档案学研究》　年份：2021　作者：倪代川;金波;
7. 相似度：0.10%　篇名：《语音搜索是一个必然的趋势 相应的SEO会有什么变化》
来源：《计算机与网络》　年份：2018　作者：王鹤铮;
8. 相似度：0.09%　篇名：《嵌入式Web服务器下光谱成像技术研究》
来源：《激光杂志》　年份：2021　作者：江建;
9. 相似度：0.08%　篇名：《应用型本科院校计算机专业毕业设计(论文)改革探析》
来源：《池州学院学报》　年份：2018　作者：夏启寿;殷晓玲;
10. 相似度：0.05%　篇名：《WEB服务器集群系统的设计与实现研究》
来源：《信息与电脑(理论版)》　年份：2018　作者：任晨曦;
11. 相似度：0.05%　篇名：《多时滞计算机网络拥塞控制仿真研究》
来源：《计算机仿真》　年份：2019　作者：韩存武;李梦奇;童薇;刘蕾;
12. 相似度：0.05%　篇名：《基于Web应用系统的性能测试的实现》
来源：《通讯世界》　年份：2016　作者：孔德华;廖春琼;
13. 相似度：0.05%　篇名：《煤质化验中如何确保数据的可靠性》
来源：《当代化工研究》　年份：2019　作者：许艳琴;
14. 相似度：0.05%　篇名：《试飞实时监控WEB服务器安全及性能优化》
来源：《中国科技信息》　年份：2019　作者：聂睿;吴海东;

互联网相似资源(博客，百科，论坛，新闻等）
1. 相似度：3.23%　标题：《Linux高并发服务器开发—项目实战-CSDN博客》
来源：https://blog.csdn.net/m0_46152793/article/details/125666651
2. 相似度：1.34%　标题：《操作系统复习总结_讨论帖_牛客网》
来源：https://ac.nowcoder.com/discuss/353147925620531200?sourceSSR=users
3. 相似度：1.26%　标题：《webserver之日志系统 c++11以后,使用局部变量懒汉不用加锁 CSDN...》
来源：https://blog.csdn.net/qq_44859952/article/details/120550066
4. 相似度：1.14%　标题：《解析linux同步与异步、阻塞与非阻塞概念以及五种IO模型_li...》
来源：https://blog.csdn.net/m0_74282605/article/details/131994978
5. 相似度：1.13%　标题：《Linux下C++轻量级Web服务器_基于linux的轻量级web服务器-CSDN...》
来源：https://blog.csdn.net/qq_40808423/article/details/123981618
6. 相似度：1.03%　标题：《Linux高并发服务器开发_linux高并发服务器开发资料-CSDN博客》

来源：https://blog.csdn.net/ambition_zhou/article/details/118577759
7. 相似度：0.97% 标题：《【Linux】Linux/Unix五种I/O模型-CFANZ编程社区》
来源：https://www.cfanz.cn/resource/detail/jKzpjzKBrvWMg
8. 相似度：0.90% 标题：《nginx 进程worker CSDN文库》
来源：https://wenku.csdn.net/answer/c95691d3ada24f29802ad665ff5ae33b
9. 相似度：0.82% 标题：《轻量级Linux服务器_ziggy7的博客-CSDN博客》
来源：https://blog.csdn.net/qq_36459662/category_10083134.html
10. 相似度：0.81% 标题：《epoll_wait详解-CSDN博客》
来源：https://blog.csdn.net/zhoumuyu_yu/article/details/112476477
11. 相似度：0.77% 标题：《【TinyWebServer】09日志系统(上) - -zx- - 博客园》
来源：https://www.cnblogs.com/Wangzx000/p/17716077.html
12. 相似度：0.75% 标题：《初识epoll与Reactor设计模式_epoll reactor-CSDN博客》
来源：https://blog.csdn.net/Jacksqh/article/details/127479130
13. 相似度：0.75% 标题：《项目04定时器处理非活动连接_牛客网》
来源：https://www.nowcoder.com/discuss/353150683325079552
14. 相似度：0.75% 标题：《定时器槽函数没执行_Web服务器项目详解07 定时器处理非活动连接...》
来源：https://blog.csdn.net/weixin_39883079/article/details/112720261
15. 相似度：0.64% 标题：《IO多路复用 ioffd-CSDN博客》
来源：https://blog.csdn.net/qq_46309680/article/details/124496295
16. 相似度：0.62% 标题：《HTTP 协议详解_http协议-CSDN博客》
来源：https://blog.csdn.net/u012581020/article/details/130585462
17. 相似度：0.59% 标题：《Webbench 原理及其优缺点 CSDN博客》
来源：https://blog.csdn.net/Sansipi/article/details/121914708
18. 相似度：0.53% 标题：《我国互联网普及率达77.5% 中国经济网 国家经济门户》
来源：http://www.ce.cn/xwzx/gnsz/gdxw/202403/22/t20240322_38944819.shtml
19. 相似度：0.50% 标题：《监听套接字与已连接套接字 监听套接字和连接套接字的区别 CSDN...》
来源：https://blog.csdn.net/lihao21/article/details/64951446
20. 相似度：0.50% 标题：《Linux通信总结 非阻塞、同步/异步、IO模型 萌新的学习之路》
来源：https://www.cnblogs.com/tristatl/p/15120498.html
21. 相似度：0.48% 标题：《(4.5)监听端口实战、epoll介绍及原理详析_c++ epoll监听多个对象-...》
来源：https://blog.csdn.net/qq_34863439/article/details/88378902
22. 相似度：0.45% 标题：《c++事件量封装,带等待超时功能_c++的wait_for条件等待-CSDN博客》
来源：https://blog.csdn.net/non_hercules/article/details/122688105
23. 相似度：0.43% 标题：《HTTP(超文本传输协议)-HyperText Transfer Protocol_超文本传输协议...》
来源：https://blog.csdn.net/fjxcsdn/article/details/85754488
24. 相似度：0.41% 标题：《epoll与reactor模式_reactor模型和epoll关系-CSDN博客》
来源：https://blog.csdn.net/lswdcyy/article/details/123774523
25. 相似度：0.41% 标题：《【Socket通信】关于Socket通信原理解析及python实现- virtualman - ...》
来源：https://www.cnblogs.com/virtualman/p/9519959.html
26. 相似度：0.39% 标题：《php多进程• Worktile社区》
来源：https://worktile.com/kb/ask/86073.html
27. 相似度：0.37% 标题：《我国互联网普及率达77.5% 中国经济网 国家经济门户》
来源：http://www.ce.cn/cysc/tech/gd2012/202404/01/t20240401_38954393.shtml
28. 相似度：0.37% 标题：《性能测试工具之WebBench 破解孤独》
来源：https://www.cnblogs.com/L-Test/p/9484080.html
29. 相似度：0.36% 标题：《线程和进程的区别是什么 线程和进程有什么区别 常见问题 PHP中文网》
来源：https://m.php.cn/faq/590378.html
30. 相似度：0.33% 标题：《WebServer:C++LinuxWeb服务器服务器_webserver里面buffer_,read...》
来源：https://download.csdn.net/download/weixin_42105570/15914391
31. 相似度：0.33% 标题：《java 线程池的作用_线程池管理器的作用-CSDN博客》
来源：https://blog.csdn.net/zj972535075/article/details/46762287
32. 相似度：0.32% 标题：《epoll I/O 多路复用_int epoll_create(int size) 流程图-CSDN博客》
来源：https://blog.csdn.net/liuchaoswu/article/details/45249353
33. 相似度：0.32% 标题：《Nginx服务-CSDN博客》
来源：https://blog.csdn.net/weixin_64447699/article/details/137080153

34. 相似度：0.27% 标题：《...线程池的概念、优点、缺点,如何使用线程池,最大线程池...》
来源：https://blog.csdn.net/Li2992973708/article/details/132611670
35. 相似度：0.27% 标题：《王先生的个人简历 杭州云聘网》
来源：https://www.hzyhzp.com/resume/92833.html
36. 相似度：0.26% 标题：《作为Web开发人员,必须知道的网络协议有哪些?-CSDN博客》
来源：https://blog.csdn.net/weixin_30699443/article/details/96489717
37. 相似度：0.25% 标题：《Socket 是什么? 总结+详解_scoket即时通讯-CSDN博客》
来源：https://blog.csdn.net/weixin_57672347/article/details/133964448
38. 相似度：0.25% 标题：《Linux IO模型 linux 用套接口进行信号驱动io,安装一个信号处理函数,...》
来源：https://blog.csdn.net/a777122/article/details/124505565
39. 相似度：0.25% 标题：《HTTP请求方法:GET、HEAD、POST、PUT、DELETE、CONNECT...》
来源：https://www.cnblogs.com/yangwenli/articles/12377475.html
40. 相似度：0.25% 标题：《线程池:优化多线程管理的利器_线程池管理-CSDN博客》
来源：https://blog.csdn.net/YXXXYX/article/details/135856032
41. 相似度：0.24% 标题：《轻量级web并发服务器——TinyWebServer的学习了解_专注云...》
来源：https://pyhost.cn/tutorial/3710.html
42. 相似度：0.24% 标题：《什么是线程池以及它的作用是什么?_线程池的作用和意义-CSDN博客》
来源：https://blog.csdn.net/W_Y_L_/article/details/83312791
43. 相似度：0.22% 标题：《...多线程(九)服务器处理并发客户端请求的方式_客户端多个请求同时...》
来源：https://blog.csdn.net/chongbin007/article/details/96603366
44. 相似度：0.21% 标题：《八、Nginx工作线程_最新版的nginx启动后会有两个worker进程吗-...》
来源：https://blog.csdn.net/u010285974/article/details/80869364
45. 相似度：0.21% 标题：《1.多线程学习笔记之进程与线程的概念 “进程”(process)与“线程”...》
来源：https://blog.csdn.net/m0_46188681/article/details/121891043
46. 相似度：0.20% 标题：《...线程池、模板的使用实例:1.向队列中放待执行函数,2.取出队列中待...》
来源：https://www.cnblogs.com/codingbigdog/p/17541086.html
47. 相似度：0.20% 标题：《I/O多路转接复用机制---select,poll,epoll_i/o多路复用机制实验报告-...》
来源：https://blog.csdn.net/woaimeinuo/article/details/51133570
48. 相似度：0.20% 标题：《web服务器性能/压力测试工具webbench安装使用_用webbench测试...》
来源：https://blog.csdn.net/cat715/article/details/11889363
49. 相似度：0.19% 标题：《服务器端启动监听-listen函数-SOCKET完美教程_socket监听多个...》
来源：https://download.csdn.net/download/weixin_42189611/17781322
50. 相似度：0.19% 标题：《Django、Nginx、uWSGI详解及配置示例 nginx反向代理...》
来源：https://asanosaki.blog.csdn.net/article/details/133513368
51. 相似度：0.18% 标题：《自己动手模拟开发一个简单的Web服务器_简易web 服务器开发一个...》
来源：https://blog.csdn.net/sunday_qh/article/details/54925832
52. 相似度：0.18% 标题：《IO模型:同步、异步、阻塞、非阻塞 麦克煎蛋》
来源：https://www.cnblogs.com/mazhiyong/articles/13502009.html
53. 相似度：0.18% 标题：《readv()和write()sendfile() - 药剂学徒- 博客园》
来源：https://www.cnblogs.com/yjds/p/8994048.html
54. 相似度：0.18% 标题：《og4j的日志级别(OFF、FATAL、ERROR、WARN、INFO、DEBUG...》
来源：https://blog.csdn.net/dhj199181/article/details/84230748
55. 相似度：0.17% 标题：《封装线程池管理模型,支持多连接的并发服务器_下载资源_代码源码-...》
来源：https://download.csdn.net/tagalbum/1237326
56. 相似度：0.17% 标题：《Tomcat对Http请求解析与处理流程 tomcat处理请求 CSDN博客》
来源：https://blog.csdn.net/kankan_s/article/details/132116526
57. 相似度：0.17% 标题：《Linux下的网络编程学习(十):高级I/O函数(2):readv和writev_linux readv...》
来源：https://blog.csdn.net/weixin_44581722/article/details/102381554
58. 相似度：0.16% 标题：《Web压力测试工具webbench-腾讯云开发者社区-腾讯云》
来源：https://cloud.tencent.com/developer/article/1478832
59. 相似度：0.16% 标题：《奇安信C++后端面经,问的很奇怪!》
来源：https://www.360doc.cn/article/81013658_1102024176.html
60. 相似度：0.16% 标题：《Qt 工具类(05):在Qt里使用Lamda表达式_qt要使用lambda表达式需要...》
来源：https://blog.csdn.net/hitzsf/article/details/109315318
61. 相似度：0.16% 标题：《ejet: eJet 是一个轻量级、高性能、嵌入式Web服务器,实现H...》

来源：https://gitee.com/kehengzhong/ejet
62. 相似度：0.16% 标题：《swoole开发功能的高性能HTTP服务器实现原理 Swoole-PHP中文网》
来源：https://m.php.cn/faq/588261.html
63. 相似度：0.16% 标题：《进程和线程相关知识总结 进程可以并发执行,一个进程的每个线程均...》
来源：https://blog.csdn.net/JIEJIEMENG/article/details/97399195
64. 相似度：0.15% 标题：《TCP可靠传输的实现[流量控制、拥塞控制] - CSDN博客》
来源：https://blog.csdn.net/u012441545/article/details/52231417
65. 相似度：0.15% 标题：《通过完整示例来理解如何使用epoll wait reported that client ...》
来源：https://blog.csdn.net/shixin_0125/article/details/78944405
66. 相似度：0.15% 标题：《为什么使用std::make_shared-CSDN博客》
来源：https://blog.csdn.net/suwanworld/article/details/103940013
67. 相似度：0.15% 标题：《一文看懂web服务器、应用服务器、web容器、反向代理服务器区别...》
来源：https://www.cnblogs.com/dbei/p/13908255.html
68. 相似度：0.15% 标题：《Nginx入门学习 I _:: I \-CSDN博客》
来源：https://blog.csdn.net/qq_29951983/article/details/87383599
69. 相似度：0.15% 标题：《Nginx 入门学习教程- 极客星云- 博客园》
来源：https://www.cnblogs.com/xingyunblog/p/9066865.html
70. 相似度：0.15% 标题：《web服务器是什么意思 - 千锋教育》
来源：http://wap.mobiletrain.org/about/BBS/121729.html
71. 相似度：0.15% 标题：《HTTP协议:持久连接、非持久连接-CSDN博客》
来源：https://blog.csdn.net/neninee/article/details/79634187
72. 相似度：0.14% 标题：《nginx 多进程+ io多路复用实现高并发- xiaobaiskill - 博客园》
来源：https://www.cnblogs.com/xiaobaiskill/p/10969180.html
73. 相似度：0.14% 标题：《什么是Http协议无状态?怎么解决Http无状态_http的无状态性...》
来源：https://blog.csdn.net/songxiao1124/article/details/120119388
74. 相似度：0.13% 标题：《muduo网络库——实现Boost.Asio聊天服务器:完整代码+注释-CSDN...》
来源：https://blog.csdn.net/amoscykl/article/details/83278567
75. 相似度：0.13% 标题：《HTTP请求过程(http是一种无状态协议,即不建立持久的连接)_http无...》
来源：https://blog.csdn.net/gaos2310/article/details/78399525
76. 相似度：0.13% 标题：《MySql免安装版配置以及修改root密码_免安装版mysql修改root账户的...》
来源：https://blog.csdn.net/xiaoran797/article/details/105978407/
77. 相似度：0.13% 标题：《什么是http2.0?-腾讯云开发者社区-腾讯云》
来源：https://cloud.tencent.com/developer/article/2040982
78. 相似度：0.12% 标题：《Android nativePollOnce函数解析_Android_脚本之家》
来源：https://www.jb51.net/article/208493.htm
79. 相似度：0.12% 标题：《Python Socket通信的实现-CSDN博客》
来源：https://blog.csdn.net/Running_free/article/details/86509305
80. 相似度：0.12% 标题：《深入探索 OpenSSL:概念、原理、开发步骤、使用方法、使用...》
来源：https://blog.csdn.net/qq_37037348/article/details/131489812
81. 相似度：0.12% 标题：《如何建设日志服务的监控分析平台及其区别 阿里云帮助中心》
来源：https://help.aliyun.com/zh/sls/product-overview/comparison-of-monitoring-and-analysis-platforms
82. 相似度：0.11% 标题：《HTTP请求和响应_http请求头用于描述客户端信息-CSDN博客》
来源：https://blog.csdn.net/qq_54054566/article/details/125094428
83. 相似度：0.11% 标题：《HTTP协议概述_http是专门为web服务器和web浏览器进行交换数据...》
来源：https://blog.csdn.net/yangjian1156/article/details/71455707
84. 相似度：0.11% 标题：《HTTP请求报文和响应响应结构 http消息响应结构 logevey的博客 ...》
来源：https://blog.csdn.net/lwxdjk/article/details/51223651
85. 相似度：0.11% 标题：《HTTP协议——HyperText Transfer Protocol(超文本传输协议)-CSDN...》
来源：https://blog.csdn.net/huangwwu11/article/details/23531111
86. 相似度：0.11% 标题：《socket网络编程(一)——初识socket - 一点sir - 博客园》
来源：https://www.cnblogs.com/kiwiblog/p/13870853.html
87. 相似度：0.11% 标题：《【并发编程】谈谈你对线程池的理解【建议收藏】_说说你对...》
来源：https://blog.csdn.net/FMC_WBL/article/details/120248154
88. 相似度：0.11% 标题：《Nginx Web服务器简介_nginx 能做web服务器?-CSDN博客》

来源：https://blog.csdn.net/qq_44865556/article/details/90406529
89. 相似度：0.10% 标题：《[计科]多进程和多线程的程序在使用上有何区别? - SkyBiuBiu - 博客园》
来源：https://www.cnblogs.com/Skybiubiu/p/17319214.html
90. 相似度：0.10% 标题：《C++/C++11中std::queue的使用_c++11 queue-CSDN博客》
来源：https://blog.csdn.net/fengbingchun/article/details/70495791
91. 相似度：0.10% 标题：《Linux IO模型介绍以及同步异步阻塞非阻塞的区别 linux的io模型,同步...》
来源：https://blog.csdn.net/a834595603/article/details/94510661
92. 相似度：0.10% 标题：《epoll reactor模式 · C++技术 · 看云》
来源：https://www.kancloud.cn/evenleo/evenloe/2161469
93. 相似度：0.10% 标题：《HTTP状态码_http请求状态代码有三位数字组成,第一个数字定义了...》
来源：https://blog.csdn.net/qq_34360094/article/details/87539973
94. 相似度：0.09% 标题：《一文聊聊Redis中的epoll和文件事件 Redis-PHP中文网》
来源：https://m.php.cn/faq/483764.html
95. 相似度：0.09% 标题：《什么是socket套接字?_什么是socket的套接字-CSDN博客》
来源：https://blog.csdn.net/zhanyd/article/details/103425359
96. 相似度：0.09% 标题：《硕士论文谢词》
来源：https://www.unjs.com/lunwen/f/20190514220714_2041444.html
97. 相似度：0.09% 标题：《并发与多线程的基本概念_线程并发是什么意思-CSDN博客》
来源：https://blog.csdn.net/whhcs/article/details/122005729
98. 相似度：0.09% 标题：《基于小根堆实现的定时器,关闭超时的非活动连接 - guanyubo...》
来源：https://www.cnblogs.com/yubo-guan/p/18048379
99. 相似度：0.09% 标题：《服务器端启动监听-listen函数-Socket网络编程-C#文档类资源-CSDN...》
来源：https://download.csdn.net/download/weixin_42191359/17655416

全文简明报告

附录
毕业论文(设计)
题目名称：基于Linux的WebServer服务器的设计与开发
题目类型： 毕业设计
学生姓名： 丁斌斌
院 (系)： 计算机科学学院
专业班级： 计科12002班
指导教师： 朱朝霞
辅导教师：
时 间： 2023-11-18 至 2024-05-30
Ⅱ
目 录

毕业论文(设计)任务书

院（系） 计算机科学学院 专业 计算机科学与技术 班级 计科12002

学生姓名 丁斌斌 指导教师/职称 朱朝霞/讲师

1．毕业论文(设计)题目

基于Linux的Webserver服务器的设计与开发

2．毕业论文(设计)起止时间

2023年10月28日～2024年5月30日

3．毕业论文(设计)所需资料及原始数据（指导教师选定部分）

[1]中国互联网络信息中心，第53次中国互联网络发展状况统计报告，2024
[2]聂松松，赵禹，施洪宝，景罗，Nginx底层设计与源码分析，机械工业出版社，2021
[3]金华, 胡书敏，基于Docker的Redis入门与实战，机械工业出版社，2021
[4]小林coding，图解网络，https://www.xiaolincoding.com
[5]马常霞，张占强，TCP/IP网络协议分析及应用，南京大学出版社，2020
[6]杨传栋, 张焕远, 范昊, 徐洪丽，Windows网络编程基础教程，清华大学出版社，2020

4．毕业论文(设计)应完成的主要内容

服务器初始化模块

启动服务器并进行初始化，包括服务器参数的配置与Socket套接字的初始化等等工作；

HTTP协议模块

{ 58%：根据客户端的请求，处理并返回相应的HTTP响应报文； }

IO多路复用模块

以异步非阻塞IO模式保证多用户对服务器的并发访问；

线程池模块

使用线程池调度管理线程，{ 58%：避免过多的新建与销毁线程带来的额外开销； }

日志模块

{ 55%：用于记录和存储系统的运行状态、事件和错误信息； }

定时器模块

{98%：服务器主循环为每一个连接创建一个定时器。 }

5．毕业论文(设计)的目标及具体要求

规范化开发流程，最终开发出一个符合教学要求，具有高度可行性的Webserver服务器，该系统应该保证的非功能性需求有：{ 67%：系统处理的准确性和及时性、系统的标准性、系统的响应速度。 }

6．完成毕业论文(设计)所需的条件及上机时数要求

（1）电脑一台：WINDOWS系统

（2）开发工具：VSCode、MYSQL

（3）开发环境：C++、ubuntu18.04LTS

（4）上机时间数：200小时左右

任务书批准日期 2023 年 11 月 3 日 教研室(系)主任(签字)

任务书下达日期 2023 年 11 月 4 日 指导教师(签字)

完成任务日期 2024 年 5 月 30 日 学生(签字)

长江大学

毕业设计开题报告

题 目 名 称 基于Linux的Webserver服务器的设计与开发

院 (系) 计算机科学学院

专 业 班 级 计科12002

学 生 姓 名 丁斌斌

指 导 教 师 朱朝霞

辅 导 教 师

开题报告日期 2023年12月8日

Ⅲ

基于Linux的Webserver服务器的设计与开发

学 生：丁斌斌，计算机科学学院

指导教师：朱朝霞，计算机科学学院

一、题目来源

社会实践

二、研究目的和意义

Web服务器在现代社会中有着广泛的应用，几乎所有的网站和在线服务都需要依赖Web服务器。因此，研究Web服务器不仅可以帮助我们更好地理解互联网的工作原理，还可以为我们提供丰富的实践机会。

随着互联网的发展，Web服务器也在不断地进化和升级。例如，近年来出现的云服务器、边缘计算等新技术，都对Web服务器的设计和实现提出了新的要求。因此，研究Web服务器可以帮助我们跟上技术的发展趋势。

掌握Web服务器的设计和实现技术，可以为我们在IT行业的职业发展提供有力的支持。无论是在互联网公司工作，还是在研究机构从事研究，这些技能都是非常有价值的。

三、阅读的主要参考文献及资料名称

[1]中国互联网络信息中心，第53次中国互联网络发展状况统计报告，2024

[2]聂松松，赵禹，施洪宝，景罗，Nginx底层设计与源码分析，机械工业出版社，2021

[3]金华, 胡书敏，基于Docker的Redis入门与实战，机械工业出版社，2021

[4]小林coding，图解网络，https://www.xiaolincoding.com

[5]马常霞，张占强，TCP/IP网络协议分析及应用，南京大学出版社，2020

[6]杨传栋, 张焕远, 范昊, 徐洪丽，Windows网络编程基础教程，清华大学出版社，2020

四、国内外现状和发展趋势与研究的主攻方向

Web服务器是互联网的核心组件，{ 63%：它负责处理HTTP请求并返回HTTP响应。 }{ 71%：近年来，随着互联网技术的快速发展， }Web服务器也经历了一系列的变革和创新。

国内研究现状

{ 58%：国内的Web服务器研究起步较晚， }但近年来有了较为快速的发展。与国外相比，虽然在某些领域还存在差距，但国内研究者正努力迎头赶上，推动技术创新和应用。

国外研究现状

{ 59%：国外的Web服务器研究更加成熟， }对Web开发的各个方面都进行了深入的探究，{ 63%：这为整个行业的发展提供了强大的动力。 }

发展趋势与研究的主攻方向

（1）后端发展趋势：{ 75%：随着微服务、容器化和云原生技术的发展， }后端开发正朝着更高效、灵活和可扩展的方向发展。

（2）前端发展趋势：前端技术也在不断进步，包括新的框架、工具和方法，以提高用户体验和交互性。

（3）新兴数据概览与技术趋势：{ 56%：随着大数据、人工智能和机器学习的崛起， }Web开发也在积极探索如何更好地利用这些技术来提供更加智能和个性化的服务。

（4）语音搜索和导航：{ 56%：随着语音识别技术的进步，语音搜索和导航预计将成为未来Web发展的重要趋势。 }

总的来说，Web服务器的发展趋势是多元化、智能化和个性化的，而研究的主攻方向则是为了满足用户日益增长的需求和提高服务质量。

五、主要研究内容

（1）服务器初始化模块

启动服务器并进行初始化，包括服务器参数的配置与Socket套接字的初始化等等工作；

（2）HTTP协议模块

{ 58%：根据客户端的请求，处理并返回相应的HTTP响应报文； }

（3）IO多路复用模块

以异步非阻塞IO模式保证多用户对服务器的并发访问；

（4）线程池模块

使用线程池调度管理线程，{ 58%：避免过多的新建与销毁线程带来的额外开销； }

（5）日志模块

{ 55%：用于记录和存储系统的运行状态、事件和错误信息； }

（6）定时器模块

{98%：服务器主循环为每一个连接创建一个定时器。 }

六、完成毕业设计所必须具备的工作条件（如工具书、计算机辅助设计、某类市场调研、实验设备和实验环境条件等）及解决的办法

1）电脑一台：{ 64% : Windows 10及以上系统， }2GHZ以上的CPU、1G以上的内存、足够的硬盘空间；

2）阿里云服务器ubuntu18.04LTS

3）VSCode软件；

4）MySQL数据库；

七、工作的主要阶段、进度与时间安排

拟定课题，获取相关资料：2023年10月18日到11月20日

确定课题，撰写开题报告：2023年11月21日到12月7日

查找资料确定框架：2023年12月7日到12月底

软件设计：2024年1月初到2024年3月31日

撰写论文：2024年4月初到2024年5月上旬

修改完善系统及论文：2024年五月中旬完成

八、指导教师审核意见

指导老师（签字）：

年 月 日

长江大学毕业论文(设计)指导教师评审意见

学生姓名 专业班级

毕业论文 (设计)题目

指导教师 职 称 评审日期

评审参考内容：毕业论文(设计)的研究内容、研究方法及研究结果，难度及工作量，质量和水平，存在的主要问题与不足。学生的学习态度和组织纪律，{ 57% : 学生掌握基础和专业知识的情况，解决实际问题的能力， }毕业论文(设计)是否完成规定任务，达到了学士学位论文的水平，是否同意参加答辩。

评审意见：

指导教师签名： 评定成绩(百分制)：_____分

（注：此页不够，请转反面）

长江大学毕业论文(设计)评阅教师评语

学生姓名 专业班级

毕业论文 (设计)题目

评阅教师 职 称 评阅日期

评阅参考内容：毕业论文(设计)的研究内容、研究方法及研究结果，难度及工作量，质量和水平，存在的主要问题与不足。{ 57% : 学生掌握基础和专业知识的情况，解决实际问题的能力， }毕业论文(设计)是否完成规定任务，达到了学士学位论文的水平，是否同意参加答辩。

评语：

评阅教师签名： 评定成绩(百分制)：_____分

（注：此页不够，请转反面）

长江大学毕业论文(设计)答辩记录及成绩评定

学生姓名 专业班级

毕业论文 (设计)题目

答辩时间 年 月 日 ~ 时 答辩地点

一、答辩小组组成

答辩小组组长：

成 员：

二、答辩记录摘要

答辩小组提问(分条摘要列举) 学生回答情况评判

三、答辩小组对学生答辩成绩的评定（百分制）：_____分

毕业论文(设计)最终成绩评定(依据指导教师评分、评阅教师评分、答辩小组评分和学校关于毕业论文(设计)评分的相关规定) 等级(五级制)：_____

答辩小组组长(签名) ： 秘书(签名)： 年 月 日

院(系)答辩委员会主任(签名)： 院(系)(盖章)

基于Linux的WebServer服务器的设计与开发

学 生：丁斌斌，计算机科学学院

指导教师：朱朝霞，计算机科学学院

{ 58%：【摘要】研究了用C++实现的基于 Linux的WebServer服务器的关键技术，本文采用了IO复用技术Epoll以及线程池实现多线程的Reactor高并发模型，此模型经过webbenchh压力测试可以实现上万的QPS， }利用正则与状态机解析HTTP请求报文，实现处理静态资源的请求，通过对理论知识的研究，在Linux系统下设计了一个WebServer服务器，在实现的过程中利用了研究的关键技术，并通过压力测试证明构建的服务器是可以承受一定的高并发访问的。

【关键词】Web服务器；高并发模型；线程池； IO复用技术；压力测试

Design and Development of WebServer Server Based on Linux

Student：Ding binbin，The College of Computer Science

Teacher：Zhu zhaoxia，The College of Computer Science

【Abstract】We have studied the key technologies of a Linux based WebServer server implemented in C++. This paper uses IO reuse technology Epoll and thread pool to implement a multi-threaded Reactor high concurrency model. This model has undergone webbench stress testing and can achieve tens of thousands of QPS. By using regularization and state machine to parse HTTP request packets, we can handle static resource requests. Through theoretical knowledge research, we have designed a WebServer server in a Linux system. During the implementation process, we have utilized the key technologies studied and demonstrated through stress testing that the constructed server can withstand certain high concurrency access.

【Keywords】 Web server；High concurrency model；thread pool；IO multiplexing technology；Stress testing

附录

第 29 页 (共 41 页)

基于Linux的WebServer服务器的设计与开发

1 绪论

1.1 课题背景及研究意义

近些年来，{ 68%：随着互联网技术不断的发展和成熟， }Web技术应用在各个领域，具有良好的发展趋势，包括电子商务、社交网络、在线教育、金融科技等。通过访问浏览器，用户就能获取到丰富的信息，了解天下大事，而无需离开家门。{ 70%：如图1，中国互联网络信息中心2024年3月29日在京发布第53次《中国互联网络发展状况统计报告》。 }{91%：报告显示，截至2023年12月，我国网民规模达10.92亿人[1]，较2022年12月增长2480万人，互联网普及率达77.5%， }{85%：较2022年12月提升1.9个百分点。 }

图1 近年中国网民规模与互联网普及率

{ 59%：Web服务器是Web应用的核心组件之一， }扮演着至关重要的角色。在互联网发展的初期，{ 56%：Web服务器通常采用基于进程的模型来处理请求。 }在这种模型中，对于每个传入的请求，服务器会为其分配一个独立的进程来处理。{ 63%：处理完结果之后再返回给客户端， }这种处理模式效率是非常低下的，在互联网高速发展的时代，许多Web系统的访问量已经大幅增加，从百万次到千万次甚至更多。这种情况下，传统的基于进程的Web服务器模型可能面临一些挑战，因为它可能无法有效地处理如此大量的并发请求。因此，研究一些更高效的并发处理模型可能更为合适。

1.2 国内外研究现状

绪论

第 29 页 (共 45 页)

自1989年发明了万维网以来，经历了从最初的静态页面服务到高性能、高可靠性Web服务器的发展，Web服务器不仅能够提供稳定的服务以及数据安全的保障，还能够抵御网络攻击、优化性能，符合法规和标准等，{ 61% : 为互联网企业的发展提供了有力的支持。 }

图2 早期多进程服务器

如图2，早期的Web服务器中广泛采用带主进程的多进程服务器，{ 56% : 这种模型也比较简单直观，易于实现和理解。 }其中，NCSA开发的httpd就是一个典型的带有主进程的多进程服务器。在这种服务器中，{80% : 主进程负责监听和接收新的连接请求， }{ 63% : 而子进程则负责实际的请求处理， }{ 57% : 从主进程获取请求并进行相应的事务处理，然后将处理结果返回给客户端。 }

基于Linux的WebServer服务器的设计与开发

图3 不带子进程的多进程服第 29 页 (共 45 页)

现如今多进程服务器不再区分是否带有子进程，{ 58% : 服务器预先创建了多个进程，并在每个进程中都开始监听来自客户端的连接请求。 }当有新的请求到达时，操作系统会

将请求分配给进程中空闲的进程，这个进程将会调用accept()方法，{ 79% : 然后进行处理和响应HTTP请求。 }{ 55% : 图3显示了这种模型，每个进程都是相对独立的， }{ 55% : 可以并发处理多个请求，提高了服务器的并发能力和响应速度。 }{ 68% : Apache作为市场上应用最广泛的Web服务器之一， }采用了这种预先派生多个进程的多进程模型。它具有稳定性高、性能优异、功能丰富等特点，被广泛应用于各种Web应用场景中。

图4 单进程服务器

绪论

{ 58% : 多进程服务器在处理大量并发请求的情况下存在一些缺点， }单进程事件驱动（Single Process Event-Driven）服务器被设计出来以克服这些问题。在SPED服务器中，{ 72% : 单个进程负责监听所有的HTTP请求， }{ 68% : 并采用事件驱动的方式进行处理。 }{ 79% : 当有新的请求到达时，服务器会触发相应的事件， }{ 60% : 然后调用相应的处理程序进行处理。 }{ 59% : 由于只有一个进程，避免了进程间的切换开销， }同时利用事件驱动机制能够更有效地处理并发请求。尽管SPED服务器在处理大量并发请求时表现不如多进程服务器，但在请求量较小的情况下，由于其简单、轻量的设计，可以获得满意的性能表现。这种服务器模型在某些场景下具有一定的优势，尤其是对于资源受限、请求量不高的情况。第 29 页 (共 45 页)

图5 多线程服务器

在面对高访问量的情况下，单进程服务器无法满足需求，{ 72% : 需要采用更高效的并发处理模型。 }为了尽可能地解决单进程服务器的性能瓶颈问题，多线程模型被提了出来。如图5所示，{ 55% : 典型的多线程服务器包括俄罗斯研究员Igor Sysoev开发的Nginx[2]， }以及德国领导开发的Lighttpd等。这些服务器采用了多线程的高并发模型，能够更好地适应大规模并发的请求场景，{ 75% : 同时充分利用多核处理器的性能优势， }{ 66% : 提高了服务器的响应速度和并发处理能力。 }

{ 67% : 通过对比多进程服务器和多线程服务器， }可以发现它们各有优缺点，适用于不同的应用场景。对于需要稳定性和可靠性较高、能够充分利用多核CPU的场景，多进程服务器更合适；而对于对并发处理能力和资源利用率要求较高的场景，多线程服务器可能更适合。

多线程服务器在应对高并发[3]请求、提高服务器性能和资源利用率等方面的优势在市场上不断地扩大，许多国内外知名站点如淘宝、网易、优酷、Facebook、GitHub等都搭建了基于Nginx的服务器框架。{ 75% : Nginx作为一个轻量级、高性能的多线程服务器， }{ 56% : 在处理大规模并发请求时表现出色， }因此受到了越来越多网站的青睐，成为了当今互联网行业中的主流服务器之一。

1.3 论文主要内容

基于Linux的WebServer服务器的设计与开发

本论文主要研究了一种基于Linux的高性能WebServer服务器的原理，设计以及实现。课题主要内容包括以下三点：第 29 页 (共 45 页)

（1）研究了重要的网络通信协议，{ 58% : 包括HTTP、TCP等重要协议， }{ 65% : 这是建立高性能Web服务器的基础。 }通过研究这些内容，可以建立起对Web通信协议和服务器架构的深入理解。最后研究出一

种基于Linux的高性能多线程WebServer服务器，{98%：利用IO复用技术Epoll与线程池实现多线程的Reactor高并发模型，}此模型经过webbenchh压力测试可以实现上万的QPS。

{ 65%：（2）了解如何有效地管理和复用线程资源，}实现线程池功能。设计了一种智能调整规模的线程池：它会根据系统的负载情况灵活地增减线程数量。当系统负载上升或者下降时，线程池会相应扩大或减小规模以应对更多的任务需求，保证系统性能；

（3）编写了服务器的源代码并进行了测试，以确保其在不同条件下的正常运行和性能表现。经过webbenchh工具的压力测试，{ 63%：WebServer服务器表现出良好的性能，}{ 68%：在面对高并发情况下仍能正常运行。 }

2 服务器的原理

2.1 计算机网络协议的深入分析和探究

2.1.1 HTTP协议

{ 77%：HTTP[4]（Hypertext Transfer Protocol，超文本传输协议）是一种用于传输超文本数据（例如HTML、图片、视频等）的应用层协议。 }{ 71%：它是互联网上应用最为广泛的协议之一，}{ 77%：主要用于Web浏览器和Web服务器之间的通信。 }{80%：HTTP协议是无连接的，即每次请求和响应之间都是独立的，}{ 79%：服务器在响应后会立即关闭连接。 }{80%：同时HTTP协议是无状态的，即服务器不会保存任何关于客户端的状态信息。 }{90%：每个请求都是独立的，服务器无法识别两个请求是否来自同一个客户端。 }

图6 HTTP请求报文格式

服务器的原理

{ 66%：HTTP协议主要负责客户端向服务器发送请求，并接收服务器返回的响应。 }{ 58%：如图6展示了HTTP请求报文的一般格式，包括请求行、请求头部、空行和正文4个部分。 }第 29 页 (共 45 页)

图7 HTTP响应报文格式

如图7，{ 61%：HTTP响应报文是服务器发送给客户端的消息，}它由三个主要部分组成：状态行、响应头部和消息体。其中，状态代码只有三位，表1展示了HTTP响应状态码的5中取值。

表1 HTTP响应状态码

{93%：1xx 指示信息：表示请求已接收，继续处理。 }

2xx 成功：表示请求已被成功处理

3xx 重定向：请求需要进一步的操作

4xx 客户端错误：请求错误或无法实现

5xx 服务器端错误：服务器未能响应请求

2.1.2 TCP协议

{82%：TCP（Transmission Control Protocol，传输控制协议）是一种面向连接的、可靠的、基于字节流的传输层协议。 }{ 61%：它是互联网协议套件中的一个核心协议，}负责在网络上可靠地传输数据。{ 55%：TCP协议在互联网中的应用非常广泛，}例如在Web浏览器和服务器之间的HTTP通信、电子邮件传输协议SMTP、文件传输协议FTP等都是基于TCP协议的。由于TCP提供了可靠性、流量控制和拥塞控制等功能，因此它适用于各种需要可靠传输的应用场景。

基于Linux的WebServer服务器的设计与开发

TCP连接的建立与销毁是TCP通信的重要过程，包括连接的建立和连接的关闭。下面是TCP连接的建立与销毁过程的详细描述[5]。第 29 页 (共 45 页)

图8 TCP的连接与断开

流量控制是TCP协议中的一项重要功能，用于控制数据的传输速率，以避免发送方发送数据过快导致接收方无法处理的情况，或者网络拥塞引起的数据丢失。{ 74%：TCP通过使用滑动窗口机制来实现流量控制。 }

{ 57%：拥塞控制是TCP协议中的一项重要功能，}{ 62%：用于调节数据在网络中的传输速率，}以避免网络拥塞导致的数据丢失、延迟和性能下降。{ 62%：TCP通过使用拥塞窗口和拥塞避免算法来实现拥塞控制，}{ 69%：保证数据的可靠传输和网络的稳定性。 }

2.2 网络程序设计基础

2.2.1 Socket编程

服务器的原理

{ 73% : Socket（套接字）是一种用于网络通信的编程接口， }{ 61% : 它允许程序员创建网络连接、发送和接收数据。 }Socket在网络编程中起着至关重要的作用，它提供了一种通用的方法来实现不同机器间的通信。第 29 页 (共 45 页)

Socket 地址主要由地址协议族 sa_family_t，端口号 in_port_t 和网际地址 in_addr几部分组成。

struct sockaddr_in {

short int sin_family; // 地址族，通常设置为AF_INET表示IPv4

sin_port; // 端口号，使用网络字节序（大端序）表示

struct in_addr sin_addr; // IP地址

};

{ 72% : Socket的本质是一种编程接口，用于实现网络通信。 }它在操作系统内核中提供了一组函数或系统调用，{ 69% : 使程序能够创建、连接、发送和接收数据。 }{ 61% : 通过Socket接口，程序可以与其他计算机上的进程进行通信， }实现数据交换和协作。{ 61% : 图9展示了客户端与服务器通过Socket建立连接的过程[6]。 }{ 65% : 首先，客户端和服务器分别创建一个未定义的Socket套接字。 }接着，它们分别使用bind()函数将Socket返回的文件描述符fd与特定的网络地址和端口进行绑定，从而完成套接字的初始化。然后，{ 56% : 服务器端使用listen()函数开始监听Socket套接字，等待客户端的连接请求。 }{ 55% : 当服务器监听到客户端发送的连接请求时， }{84% : 它调用accept()函数接受客户端的请求， }并开始处理客户端的连接。

基于Linux的WebServer服务器的设计与开发

图9 Socket连接过程第 29 页 (共 45 页)

2.2.2 进程与线程

{ 67% : 进程是操作系统中的一个基本概念，它是程序执行的一个实例。 }换句话说，{ 62% : 进程是正在运行的程序的抽象。 }{ 59% : 每个进程都有自己的地址空间、内存、代码和数据，以及操作系统所需的其他资源。 }

{ 76% : 线程是程序执行流的最小单元， }{90% : 也是操作系统能够进行运算调度的最小单位。 }{90% : 在一个进程中，可以包含多个线程， }{ 57% : 这些线程共享进程的资源，如内存空间、文件句柄等。 }{ 57% : 线程之间可以并发执行，从而提高了程序的性能和效率。 }

相比于进程，线程具有以下几个优势：

{ 58% : 资源开销较小：线程是进程的子集， }{ 56% : 线程之间共享进程的资源，如内存空间、文件句柄等。 }{ 63% : 因此，创建和销毁线程的开销通常比创建和销毁进程要小得多，线程的切换也更加快速高效。 }

并发执行：{84% : 由于线程共享进程的地址空间和其他资源， }{ 57% : 线程之间可以并发执行，从而提高了程序的性能和效率。 }{ 76% : 多个线程可以同时执行不同的任务， }实现并行处理，加快了程序的执行速度。

更好的响应性：由于线程可以并发执行，{80% : 可以将耗时的任务放在后台线程中执行， }从而提高了程序的响应性。例如，可以使用一个线程处理用户界面的交互操作，另一个线程处理后台的数据处理任务，以保持界面的流畅和响应。

更方便的数据共享：{ 78% : 线程之间共享进程的地址空间， }{ 60% : 因此它们可以更方便地共享数据和通信。 }这使得线程之间的数据共享更加简单直接，不需要使用额外的机制来实现进程间通信。

{ 59% : 更好的资源利用率：由于线程共享进程的资源，可以更有效地利用系统资源。 }例如，多个线程可以共享相同的代码段和数据段，{ 66% : 减少了内存的开销，提高了系统的资源利用率。 }

2.2.3 服务器IO模型

服务器的原理

高性能WebServer服务器的设计需要高性能的IO模型[7]，常见的Unix/Linux上的五种IO模型有以下几种：第 29 页 (共 45 页)

阻塞（blocking）：{97%：调用者调用了某个函数，等待这个函数返回，在这期间什么也不做，不停的去检查这个函数有没有返回，必须等这个函数返回才能进行下一步动作。 }

图10 阻塞模型

{97%：非阻塞（non-blocking）： }{100%：非阻塞等待，每隔一段时间就去检测IO事件是否就绪。 }{97%：没有就绪就可以做其他事。非阻塞I/O执行系统调用总是立即返回，不管事件是否已经发生，若事件没有发生，则返回-1，此时可以根据 errno 区分这两种情况，对于accept，recv 和 send，事件未发生时，errno 通常被设置成 EAGAIN。 }

基于Linux的WebServer服务器的设计与开发

图11 非阻塞模型第 29 页 (共 45 页)

{94%：IO复用（IO multiplexing）： }{99%：Linux 用 select/poll/epoll 函数实现 IO 复用模型，这些函数也会使进程阻塞，但是和阻塞IO所不同的是这些函数可以同时阻塞多个IO操作。 }{100%：而且可以同时对多个读操作、写操作的IO函数进行检测。 }{100%：直到有数据可读或可写时，才真正调用IO操作函数。 }

图12 IO复用模型

{86%：信号驱动（signal-driven）： }{96%：Linux 用套接口进行信号驱动 IO，安装一个信号处理函数，进程继续运行并不阻塞，当IO事件就绪，进程收到SIGIO 信号，然后处理 IO 事件。 }{97%：内核在第一个阶段是异步，在第二个阶段是同步；与非阻塞IO的区别在于它提供了消息通知机制，不需要用户进程不断的轮询检查，减少了系统API的调用次数，提高了效率。 }

服务器的原理

图13 信号驱动模型第 29 页 (共 45 页)

{95%：（5）异步（asynchronous）：Linux中，可以调用 aio_read 函数告诉内核描述字缓冲区指针和缓冲区的大小、文件偏移及通知的方式，然后立即返回，当内核将数据拷贝到缓冲区后，再通知应用程序。 }

图14 异步模型

2.3 Nginx架构研究

{ 73%：Nginx是一个高性能的开源Web服务器，也是一个反向代理服务器和负载均衡器。 }{98%：它最初由Igor Sysoev创建，并于2004年首次公开发布。 }Nginx专注于高并发性能和低内存消耗，因此在处理静态文件、反向代理和负载均衡等方面表现出色。它已经成为许多网站和应用程序的首选Web服务器之一，因为它的性能和可靠性。所以对Nginx的研究有助于学习多线程高性能Web服务器的设计。

基于Linux的WebServer服务器的设计与开发

{ 69%：Nginx支持多进程与多线程工作方式[8]， }{ 73%：在Nginx的架构中，Master进程负责管理和监控多个Worker进程，而每个Worker进程都是独立的进程，负责处理客户端的请求。 }这种多进程的设计有助于提高Nginx的并发处理能力和稳定性，{ 67%：因为每个Worker进程都是相互独立的，它们之间不会共享内存空间， }从而减少了因共享状态而可能引起的竞争和死锁等问题。第 29 页 (共 45 页)

图15 Nginx架构模型

{ 62%：Nginx默认采用的是pre-fork方式工作， }这是一种常见的多进程模型。在pre-fork模型中，{ 68%：Master进程在启动时会创建多个Worker进程， }{86%：每个Worker进程独立处理客户端请求。 }这些Worker进程在处理请求时是相互独立的，它们之间不会共享内存空间，因此不会出现因共享状态而引起的竞争和冲突问题。

多路IO复用的核心思想是通过一个系统调用，{ 70%：让一个进程能同时监控多个IO事件， }而不是每个事件都单独地等待。在Nginx的场景中，它利用了Linux系统提供的Epoll机制来实现多路IO复用。{83%：当一个客户端连接到服务器时， }{ 69%：服务器会将这个连接添加到Epoll的监听列表中。 }而后，Epoll会监视这个连接上的各种IO事件，比如可读事件（表示有数据可以从连接中读取）、可写事件（表示可以向连接中写入数据）等。当有任何一个事件发生时，Epoll会通知Nginx主进程，{ 59%：然后Nginx主进程会将相应的任务分配给可用的Worker进程来处理。 }{ 68%：通过利用Epoll的多路IO复用机制， }{ 55%：Nginx可以同时监听大量的连接， }并且能够高效地处理这些连接上的IO事件，{ 64%：从而实现高性能和高并发处理能力。 }{ 60%：这种设计使得Nginx能够在保持低资源消耗的同时， }处理大规模的并发请求。

2.4 本章小结

服务器的原理

本章首先介绍了服务器中的关键网络协议，包括HTTP和TCP，这是理解网络通信的基础。接着对网络程序设计的基础进行了分析，{ 61% : 涉及了Socket编程通信、进程、线程和线程池的概念， }以及各种服务器IO模式的原理和应用。最后，通过学习当今第 29 页 (共 45 页)

最流行的轻量级服务器Nginx的架构设计，深入了解了Nginx如何实现IO多路复用。综合而言，本章系统介绍了Web服务器的关键原理，{ 55% : 为进一步设计轻量级高性能的Web服务器打下了基础。 }

3 需求分析与设计

3.1 高性能Web服务器需求分析

1. 参数配置

为了确保服务器在启动之前完成必要的配置，我们需要在程序内进行设置，以便用户可以指定一个或多个参数来进行动态配置。如果没有提供特定的配置参数，服务器将会以默认方式启动。

参数配置的格式如下：该命令表示服务器的端口号为1316，开启ET模式，超时时间为60000毫秒，关闭优雅退出，{ 55% : Mysql端口3306，账号，密码， }数据库，sql连接池数量为12，线程池线程数量为6，开启日志，日志系统等级为1，日志异步队列容量为1024。

```
WebServer server(
1316, 3, 60000, false,
3306, "root", "root", "webserver",
12, 6, true, 1, 1024);
```

2. 支持并发连接

支持并发连接是Web服务器的基本要求之一。这意味着服务器能够同时处理多个客户端的连接请求，而不会因此而降低性能或出现响应延迟。通过支持并发连接，{ 66% : 服务器可以更高效地处理大量的请求， }提高用户的访问体验，{ 72% : 并确保网站或应用程序的稳定性和可靠性。 }

3. HTTP协议支持

基于Linux的WebServer服务器的设计与开发

{ 64% : HTTP/0.9是HTTP协议的最早版本之一， }于1991年推出。它非常简单，只支持GET方法，没有头部信息，也没有状态码。每个请求和响应只由一个HTML页面组成，使用空行来分隔请求和响应，但已经较少使用。目前，{ 64% : 主流的HTTP协议版本包括HTTP/1.0和HTTP/1.1。 }{ 66% : HTTP/1.0 默认情况下不支持持久连接， }每个请求/响应都第 29 页 (共 45 页)

需要建立一个新的连接。{ 73% : 而 HTTP/1.1 默认情况下支持持久连接， }{ 62% : 一个连接可以被多个请求复用， }{ 67% : 从而减少了连接建立的开销，提高了性能。 }在我们设计的服务器中，{ 60% : 只需支持最为普遍的HTTP/1.1版本。 }{ 62% : 如果客户端请求的协议版本与服务器不兼容， }{ 59% : 则服务器应通过适当的状态码向用户返回相应的错误信息。 }

4. 请求方法支持

{ 69% : HTTP/1.1共定义了八种请求方法，它们分别是：GET、PUT、POST、DELETE、HEAD、TRACE、OPTIONS、CONNECT， }{ 56% : 这些请求方法定义了客户端与服务器之间的交互方式， }允许客户端对服务器上的资源进行各种不同类型的操作。对于本文设计的基于Linux的WebServer服务器来说，其主要职责是为用户提供静态或动态的网络页面服务，{ 57% : 因此，GET和POST请求方法是必需的， }而其他方法是可选择支持的。

5. 支持线程池

{89% : 通过使用线程池，可以避免频繁地创建和销毁线程，减少了系统资源的消耗和性能开销。 }同时，{ 64% : 线程池可以根据实际需求动态调整线程数量， }{ 66% : 以适应不同负载下的需求，提高系统的性能和稳定性。 }

6. 日志功能

服务器日志功能是服务器软件记录和存储关于其运行状态和用户请求的信息的功能。这些日志对于监控服务器的性能、诊断问题以及跟踪用户活动都非常重要。服务器日志功能通常包括：访问日志、错误日志、性能日志、安全日志、自定义日志。通过分析和解释服务器日志，管理员可以了解服务器的运行情况和用户行为，及时发现和解决问题，{ 64% : 提高服务器的性能、稳定性和安全性。 }

7. 安全性保证

OpenSSL提供了一套安全套接字编程接口（SSL/TLS API），可以方便地在应用程序中实现安全的网络通信。开发人员可以使用这些API来构建安全的客户端和服务器程序，保护敏感数据的传输和存储。{ 65%：OpenSSL是一个功能强大的加密库，为开发人员提供了丰富的安全功能和工具， }帮助他们构建安全可靠的应用程序和网络系统。

8. 平台兼容性支持

需求分析与设计

服务器平台兼容性支持指的是服务器软件或应用程序能够在不同操作系统和硬件平台上正常运行和提供相同的功能。在开发服务器软件时，通常会考虑到跨平台兼容性，以确保尽可能多的用户能够使用该软件。第 29 页 (共 45 页)

3.2 Web服务器总体设计

本文的主要任务是设计一个基于Linux的WebServer服务器并编写代码实现，在此基础之上研究了一种线程池[9]，以及用于线程池的任务调度算法，以满足在硬件资源有限的前提下尽量满足更多的用户请求。如图16，根据3.1节对高性能轻量级服务器的需求分析，本文设计的的Web服务器将划分为6 大模块：

（1）服务器初始化模块：启动服务器并进行初始化，包括服务器参数的配置与

Socket套接字的初始化等等工作；

（2）HTTP协议模块：{ 58%：根据客户端的请求，处理并返回相应的HTTP响应报文； }

（3）IO多路复用模块：以异步非阻塞IO模式保证多用户对服务器的并发访问；

{ 59%：（4）线程池模块：使用线程池调度管理线程， }{ 58%：避免过多的新建与销毁线程带来的额外开销； }

（5）日志模块：{ 55%：用于记录和存储系统的运行状态、事件和错误信息； }

（6）定时器模块：{98%：服务器主循环为每一个连接创建一个定时器，并对每个连接进行定时， }超时的连接将会被关闭。

图16 服务器总体设计图

基于Linux的WebServer服务器的设计与开发

第 29 页 (共 45 页)

3.3 Web 服务器主要模块设计

3.3.1 服务器初始化模块

在启动服务器时执行的一段代码或一组代码，用于初始化服务器的各种配置、资源和环境。这个模块具有以下功能：

{ 57%：（1）加载配置文件：读取服务器配置文件， }包括网络配置、日志配置、数据库连接配置等。

（2）初始化日志系统：配置日志系统，{ 58%：设置日志级别、日志输出位置等， }{ 60%：以便后续记录服务器运行状态和错误信息。 }

（3）初始化网络环境：创建服务器监听套接字，{ 62%：绑定端口，开始监听客户端连接请求。 }

（4）初始化数据库连接池：如果服务器需要与数据库交互，那么在初始化阶段应该建立数据库连接池，以便后续的数据库操作能够高效地复用连接。

（5）加载静态资源：{ 57%：如果服务器提供静态文件服务， }如网页、图片、视频等，那么在初始化阶段需要加载这些静态资源到内存中，以提高访问速度。

（6）启动定时任务：{ 70%：如果服务器需要定时执行一些任务， }如数据备份、日志清理等，那么在初始化阶段应该启动这些定时任务。

（7）加载应用程序模块：加载业务逻辑模块、路由配置、中间件等，准备处理客户端请求。

3.3.2 HTTP协议模块

需求分析与设计

HTTP协议模块是负责处理和解析HTTP请求和响应的组件。{ 55%：它通常包括请求解析、响应生成、请

求处理、响应处理、错误处理、会话管理的功能。 }它是服务器中的一个核心组件，{ 64% : 负责处理和解析HTTP请求和响应， }以及执行相应的请求处理逻辑[10]。它通过与其他组件协同工作，实现了服务器对HTTP协议的完整支持，{ 71% : 从而提供了高效、可靠的Web服务。 }图17展示了HTTP协议模块处理的一般流程。第 29 页 (共 45 页)

图17 HTTP请求处理流程图

3.3.3 高性能IO模块

本文研究了一种基于Reactor的高并发模型。通过利用IO复用技术Epoll[11]，{ 67% : 在单个线程中处理多个并发连接。 }{ 73% : 线程池中的线程负责具体的事件处理。 }例如，当有新连接到达时，{ 56% : 线程池中的某个线程接受连接并将新的连接套接字添加到Epoll监听器中； }当套接字上有数据可读时，线程池中的某个线程读取数据并进行处理；当套接字上可写时，{ 55% : 线程池中的某个线程发送数据给客户端。 }通过这种方式，服务器能够利用Epoll实现事件驱动的高效IO操作，{ 61% : 同时通过线程池实现多线程并发处理， }{ 70% : 提高了系统的并发性能和吞吐量。 }具体示意图如下：

基于Linux的WebServer服务器的设计与开发

图18 Reactor模式的工作流程第 18 页 (共 45 页)

3.3.4 线程池模块

{ 58% : 线程池模块是服务器系统中的重要组件，用于管理和调度线程， }实现任务的并发处理。{ 68% : 它能够提高系统的并发性能和吞吐量， }同时有效地利用系统资源，{ 56% : 是实现高性能服务器的关键技术之一。 }

本文研究了一种可高效运行的线程池模型，该线程池模型采用一个任务队列来存储待执行的任务。任务队列采用先进先出（FIFO）的原则，确保任务按照到达顺序执行。{ 56% : 任务队列的大小可以根据系统的负载情况和性能需求进行动态调整， }以确保线程池的高效运行。{ 55% : 线程池模型通过一个线程池管理器来创建和管理线程。 }线程池管理器根据配置参数创建一定数量的线程，并在需要时动态地调整线程数量。{ 55% : 线程池管理器还负责监控线程的状态和活跃度， }及时回收空闲线程，以减少资源的浪费。当任务到达时，{ 57% : 线程池将任务分配给空闲线程进行处理。 }每个线程在处理完任务后会从任务队列中取出下一个任务，{ 63% : 直到任务队列为空或线程池关闭为止。 }{ 59% : 线程池采用预先创建线程的方式来避免线程创建和销毁的开销，提高了任务的处理效率。 }具体示意图如下：

图19 线程池模型

3.3.5 日志模块

服务器中的日志模块是一个非常重要的组件，{ 57% : 它负责记录和管理服务器运行时产生的各种日志信息， }为系统的监控、故障排查和性能优化提供了重要的支持。{ 55% : 一个高效可靠的日志模块可以帮助管理员及时发现和解决问题，提高系统的稳定性和可靠性。 }

本文设计的日志模块实现以下基础功能：

{ 61% : （1）设置了四种日志级别，包括"ERROR"，"WARN"，"INFO"，"DEBUG"； }

（2）日志模块可以将记录的日志信息输出到多种目标，包括控制台、日志文件、远程服务器、数据库等。

需求分析与设计

（3）可以对记录的日志信息进行格式化处理。第 19 页 (共 45 页)

（4）日志模块不阻塞正常的执行流程；

（5）在记录大量日志的同时对系统性能的影响小，{ 77% : 不影响系统的稳定性和可靠性。 }

3.3.6 定时器模块

每个连接对象需要包含一个定时器结构，用于记录连接的超时时间点。使用小根堆（最小堆）来管理定时器。小根堆可以保证堆顶元素是最小的，也就是最先要超时的连接。当新的连接建立时，根据连接的超时时间，在小根堆中插入一个定时器节点。定时器模块会定期检查小根堆的堆顶元素，如果堆顶元素的超时时间小于当前时间，则表示该连接已经超时，需要关闭。定时器模块将超时的连接对象从小根堆中删除，并执行关闭连接的操作。通过以上步骤，{ 63% : 基于小根堆实现的定时器可以高效地管理连接的超时，实现自动关闭超时的非活动连接， }{ 72% : 从而提高服务器的性能和稳定性。 }

3.4 本章小结

本章描述了一个基于Linux的WebServer服务器的研究过程和设计方案。首先，对需求进行了分析，确定了

高性能和轻量级的要求。然后，在此基础上进行了整体设计，包括模块化设计和创新的服务器编程模型。{ 57% : 该模型结合了线程池、非阻塞IO和IO多路复用技术， }以确保在数据计算和网络IO方面都具有高效率。另外，还设计了一种灵活动态的线程池，并结合负载均衡算法，以提升服务器性能。接下来的步骤是将这个设计实现为具体的代码。

4 Web服务器的具体实现

4.1 模块的实现

4.1.1 服务器初始化模块的实现

基于Linux的WebServer服务器的设计与开发

服务器初始化时，首先需要加载配置文件，读取服务器的配置信息，包括网络端口、IP地址、日志路径、线程池大小、数据库连接信息等。这些配置信息可以影响服务器的运行方式和性能参数，因此配置加载是初始化的第一步。在服务器启动时，需要初始化日志模块，配置日志的输出方式、日志级别、日志格式等参数。服务器通常是基于网络进行通信的，因此需要初始化网络模块，包括创建监听套接字、绑定IP第 20 页 (共 45 页)

地址和端口、设置套接字选项、启动监听等操作。网络模块的初始化是服务器初始化的重要组成部分，{ 59% : 它负责建立服务器与客户端之间的通信通道。 }

{ 71% : 服务器通常使用线程池来处理客户端请求， }因此需要初始化线程池。线程池的初始化包括创建一定数量的线程、设置线程属性、初始化任务队列等操作。线程池的初始化是服务器初始化的关键步骤，它决定了服务器能够处理的并发请求数量。如果服务器需要与数据库进行交互，就需要初始化数据库连接池。数据库连接池的初始化包括创建一定数量的数据库连接、设置连接属性、初始化连接池管理器等操作。数据库连接池的初始化是服务器初始化的重要组成部分，它提高了服务器与数据库之间的交互效率和性能。

服务器初始化模块可以完成对服务器各个组件的初始化工作，使服务器能够在启动时正常运行，并为后续的请求处理提供必要的基础环境。以下对服务器初始化模块中的主要函数及其作用作简单介绍：

WebServer(

int port, int trigMode, int timeoutMS, bool OptLinger,

int sqlPort, const char *sqlUser, const char *sqlPwd,

const char *dbName, int connPoolNum, int threadNum,

bool openLog, int logLevel, int logQueSize);

{ 57% : 这段代码是一个Web服务器的构造函数， }负责对服务器进行初始化。在构造函数中，首先使用参数列表对成员变量进行了初始化，然后调用SqlConnPool:{ 55% : :Instance()-> }Init()方法，传入数据库连接相关参数，对数据库连接池进行初始化，调用InitEventMode_()方法，根据传入的触发模式参数初始化事件模式为ET模式，调用InitSocket_()方法初始化套接字，{ 58% : 调用Log::Instance()-> }{ 59% : init()方法初始化日志记录器， }并输出初始化信息。总之，该构造函数完成了对服务器各个组件的初始化工作，包括网络模块、线程池、定时器、日志记录等，为服务器的正常运行奠定了基础。

void WebServer::Start();

Web服务器的具体实现

通过epoll实现了事件驱动的服务器主循环，{ 72% : 当有事件发生时，根据事件类型调用相应的处理函数进行处理。 }首先，设置epoll等待超时时间，如果timeoutMS_大于0，表示设置了超时时间，调用timer_->{ 72% : GetNextTick()获取下一个计时器超时时间。 }将第 21 页 (共 45 页)

该超时时间作为参数传入epoller_->Wait(timeMS)，以等待事件发生。调用epoller_->Wait(timeMS)等待事件发生，返回事件数量eventCnt。在循环中处理事件，通过epoller_->GetEventFd(i)获取事件对应的文件描述符fd，通过epoller_->GetEvents(i)获取事件类型events。如果事件是监听套接字实例listenFd_，则调用DealListen_()处理监听事件。如果事件是错误事件或连接关闭事件，则调用CloseConn_(&users_[fd])关闭用户连接。如果事件是可读事件，则调用DealRead_(&users_[fd])处理读取事件。如果事件是可写事件，则调用DealWrite_(&users_[fd])处理可写事件。如果出现了意外的事件类型，则输出错误日志。循环直到isClose_标志为true，即服务器关闭。

bool WebServer::InitSocket_();

Web服务器的套接字初始化函数InitSocket_()，负责创建、设置、绑定和监听套接字，{ 61% : 以及将套接字添加到epoll实例中进行事件监听。 }

（4）void WebServer::DealWrite_(HttpConn *client);

将写事件处理任务添加到线程池中，实现了异步处理客户端写事件的功能，避免了在主线程中阻塞等待写事件的完成。

(5)void WebServer::DealRead_(HttpConn *client);

将读事件处理任务添加到线程池中，实现了异步处理客户端读事件的功能，避免了在主线程中阻塞等待读事件的完成。

4.1.2 HTTP模块的实现

HTTP模块是负责处理客户端发来的HTTP请求和向客户端发送HTTP响应的部分。本文设计了httpconn、httprequest、httpresponse这三种结构体来实现HTTP模块相应功能的实现。其中，httpconn类是对一个客户端连接的描述，{ 57%：而httprequest类和httpresponse类则分别是对请求报文和响应报文的描述。 }

以下是对HTTP模块中的主要函数及其作用作简单介绍：

void HttpConn::init(int fd, const sockaddr_in& addr);

基于Linux的WebServer服务器的设计与开发

HTTP连接对象(HttpConn)的初始化函数init()，用于初始化一个HTTP连接对象。在函数中，使用assert()函数确保传入的文件描述符fd大于0，以确保文件描述符有效。每次初始化一个新的HTTP连接对象，连接计数器userCount会增加。这个计数器通常用于记录当前活跃的连接数。将传入的远程地址信息addr复制给HTTP连接对象的成员变量addr_，用于记录客户端的地址信息。将传入的文件描述符fd赋值给第 22 页 (共 45 页)

HTTP连接对象的成员变量fd_，用于标识该连接。清空HTTP连接对象的读写缓冲区，以确保在使用之前没有遗留的数据。将连接状态isClose_设置为false，表示连接处于打开状态。最后，使用日志记录函数LOG_INFO()记录连接建立的日志信息。

ssize_t HttpConn::write(int* saveErrno);

这段代码实现了HTTP连接对象(HttpConn)的写操作函数write()，{ 62%：用于将数据写入到套接字文件描述符中。 }使用do-while循环进行数据写入操作，直到所有数据都被写入或写入过程中出现错误为止。{ 75%：使用writev()函数将分散的数据一并写入到套接字文件描述符中， }并返回写入的字节数。{ 65%：iov_是一个iovec结构体数组， }用于存储待发送数据的分散信息，iovCnt_表示数组中元素的数量。如果写入的字节数小于等于0，表示写入失败，将错误码保存到指定的变量中，并跳出循环。如果待发送的数据量为0，表示传输结束，跳出循环。如果写入的字节数大于第一个缓冲区中的数据量，表示第一个缓冲区中的数据已经全部发送完毕，更新iov_数组中的数据信息，并将写缓冲区中已发送的数据部分删除。否则，更新iov_数组中的数据信息，并将写缓冲区中已发送的数据删除。如果是ET模式（边缘触发模式），或者待写入的数据量超过一定阈值（10KB），则继续循环写入数据。{ 55%：最后，返回实际写入的字节数len。 }

（3）bool HttpRequest::parse(Buffer &buff);

实现了HTTP请求对象(HttpRequest)的解析函数parse()，用于解析从客户端接收到的HTTP请求报文。

（4）bool HttpRequest::ParseRequestLine_(const string &line);

实现了解析HTTP请求行的函数ParseRequestLine_()，使用了正则表达式来匹配HTTP请求行中的方法、路径和HTTP版本。

（5）void HttpResponse::MakeResponse(Buffer& buff);

HTTP响应对象(HttpResponse)中的MakeResponse()函数，用于生成HTTP响应报文。

（6）void HttpResponse::AddStateLine_(Buffer& buff)

HTTP响应对象(HttpResponse)中的AddStateLine_()函数，用于向缓冲区中添加HTTP状态行。

4.1.3 IO模块的实现

Web服务器的具体实现

IO模块通常负责处理与网络IO相关的操作，包括接收请求、发送响应等。这个第 23 页 (共 45 页)

模块通常是整个服务器的核心部分之一。本文使用了多路IO复用技术Epoll来实现高效处理大量并发连接的功能。设计了epoller类来保证Reactor 模式的实现。 在服务器启动后，{ 56%：通过epoll实现事件驱动的服务器主循环， }{ 72%：当有事件发生时，根据事件类型调用相应的处理函数进行处理。 }如果事件是监听套

接字实例listenFd_，则调用DealListen_()处理监听事件。如果事件是可读事件，则调用DealRead_(&users_[fd])处理读取事件。如果事件是可写事件，则调用DealWrite_(&users_[fd])处理可写事件。

本文设计的高性能 IO 模块基于 Epoll实现，主要调用了三个函数：

{85% : int epfd = epoll_create(int size)； }

{83% : 创建一个新的 epoll 实例，并返回一个文件描述符， }该描述符可以用于后续的 epoll 系统调用。

{99% : int epoll_ctl(int epfd, int op, int fd, struct epoll_event *event)； }

{ 65% : 用于控制 epoll 实例上的事件。 }它可以用来注册新的文件描述符、修改已注册的文件描述符上的事件或者删除已注册的文件描述符。{90% : epfd 参数是 epoll 实例的文件描述符， }{ 69% : 通过 epoll_create() 创建得到。 }op 参数表示要执行的操作，{ 62% : fd 参数是要添加、修改或删除的文件描述符。 }{ 67% : event 参数是一个指向 epoll_event 结构体的指针， }其中包含了要注册或修改的事件信息。{ 55% : 函数成功执行返回 0，失败返回 -1，并设置相应的错误码。 }

（3） int epoll_wait(int epfd, struct epoll_event * events, int maxevents, int timeout)；

{ 65% : epoll_wait() 函数用于等待 epoll 实例上的事件发生， }并将就绪的事件填充到指定的事件数组中。{90% : epfd 参数是 epoll 实例的文件描述符， }{ 69% : 通过 epoll_create() 创建得到。 }{ 75% : events 参数是一个指向 epoll_event 结构体数组的指针， }用于存储就绪的事件信息。{81% : maxevents 参数表示 events 数组的大小，即最多可以存储多少个就绪事件。 }{ 72% : timeout 参数指定等待事件发生的超时时间，单位为毫秒。 }可以有以下几种情况，{ 70% : 如果 timeout 为正数， }则表示等待指定的毫秒数后返回，即超时等待。{ 75% : 如果 timeout 为 0，则表示立即返回，即非阻塞模式。 }{80% : 如果 timeout 为 -1，则表示无限等待，直到有事件发生。 }{ 58% : 函数成功执行返回就绪事件的数量，失败返回 -1， }并设置相应的错误码。就绪事件的具体信息将存储在 events 数组中。

基于Linux的WebServer服务器的设计与开发

4.1.4 线程池模块的实现第 24 页 (共 45 页)

{100% : 线程池是一种管理线程的机制， }用于提高并发任务处理的效率。{ 69% : 其主要组成部分包括线程池管理器、工作队列和线程池工作线程。 }{ 58% : 线程池管理器负责管理线程池的生命周期，包括创建、销毁线程池。 }{ 78% : 工作队列用于存储需要执行的任务。 }通常采用队列结构，支持任务的入队和出队操作。当有任务需要执行时，{ 74% : 会被放入工作队列中等待执行。 }{ 68% : 线程池中的工作线程负责从工作队列中获取任务并执行。 }{ 57% : 同时，使用互斥锁和条件变量等机制实现线程之间的同步和互斥，确保多个线程之间对共享资源的安全访问。 }

以下简单介绍下线程池模块中的主要函数以及结构体：

```
struct Pool {
std::mutex mtx;
std::condition_variable cond;
bool isClosed;
std::queue<std::function<void()>> tasks;
};
```

这段代码定义了一个简单的线程池结构体 Pool，{ 60% : std::mutex mtx表示互斥量，用于实现对线程池的互斥访问， }确保在多线程环境下对任务队列的安全访问。{ 77% : std::condition_variable cond表示条件变量， }用于实现线程的同步等待，当任务队列为空时，工作线程将会等待条件变量的通知。bool isClosed表示标志位，指示线程池是否已关闭。std::queue<std::function<void()>> tasks表示任务队列，存储待执行的任务。使用 std::function<void()> 表示任务的类型，支持任意可调用对象作为任务。

Web服务器的具体实现

（2）explicit ThreadPool(size_t threadCount = 8): pool_(std::make_shared<Pool>()) {

```
assert(threadCount > 0);
for(size_t i = 0; i < threadCount; i++) {
```

```
std::thread([pool = pool_] {
std::unique_lock<std::mutex> locker(pool->mtx);
while(true) {
if(!pool->tasks.empty()) {
auto task = std::move(pool->tasks.front());
pool->tasks.pop();第 25 页 (共 45 页)
locker.unlock();
task();
locker.lock();
}
else if(pool->isClosed) break;
else pool->cond.wait(locker);
}
}).detach();
}
}
```

这段代码定义了线程池的构造函数，{ 63% : 使用了C++11中的lambda表达式来创建工作线程， }并将其逻辑包装在一个while循环中。在构造函数中，接受一个默认参数 threadCount，{ 60% : 表示线程池中的线程数量，默认值为 8。 }利用std:{ 67% : :make_shared创建了一个Pool对象，并通过std::shared_ptr进行管理， }pool_是线程池的成员变量，用于共享线程池对象。根据传入的线程数量，循环创建工作线程。{ 68% : 使用lambda表达式创建了一个新的线程， }并立即分离（detach）它，以允许该线程独立执行。{ 63% : 通lambda表达式的捕获列表捕获了 pool_， }以确保在lambda函数体中能够访问到线程池对象。在每个工作线程中创建了一个独占锁locker，以锁定线程池的互斥量，确保多个线程同时访问任务队列时的安全性。然后循环执行工作线程的主体逻辑。{ 58% : 当任务队列不为空时，线程会从队列中取出任务并执行。 }当线程池已关闭时，退出循环。当任务队列为空且线程池未关闭时，{ 55% : 线程进入等待状态，等待条件变量的通知。 }这样，{ 59% : 每个工作线程在一个循环中不断地从任务队列中取出任务执行， }如果任务队列为空且线程池未关闭，则进入等待状态，直到有新任务加入或线程池被关闭。

4.1.5 日志模块的实现

基于Linux的WebServer服务器的设计与开发

{97% : 本项目中，使用单例模式创建日志系统[12]，对服务器运行状态、错误信息和访问数据进行记录，该系统可以实现按天分类，超行分类功能，可以根据实际情况分别使用同步和异步写入两种方式。 }{100% : 其中异步写入方式，将生产者-消费者模型封装为阻塞队列，创建一个写线程，工作线程将要写的内容push进队列，写线程从队列中取出内容，写入日志文件。 }第 26 页 (共 45 页)

日志模块的实现较为简单，本文在服务器初始化的同时可以设置日志记录的等级，{ 59% : 同时本文设计包括"ERROR"，"WARN"，"INFO"，"DEBUG"的四种日志等级， }{ 55% : 同时为了保证日志系统的功能正常运转， }实现了log类来保证，在log类中可以设置log文件存放的位置。在服务器运行的同时，日志默认会被记录并保存在以时间命名的日志文件中，以便后期开发人员查阅。

4.1.6 定时器模块的实现

{98% : 在本项目中，服务器主循环为每一个连接创建一个定时器，并对每个连接进行定时。 }{100% : 另外，利用升序时间链表容器将所有定时器串联起来，若主循环接收到定时通知，则在链表中依次执行定时任务。 }

以下简单介绍下定时器模块中的主要函数：

（1）void HeapTimer::siftup_(size_t i);

实现了堆的向上调整操作。

void HeapTimer::add(int id, int timeout, const TimeoutCallBack& cb);

{ 63% : 当服务器有新的客户端添加进来时， }创建新的定时器节点。

void HeapTimer::tick();

处理堆中已经超时的定时器节点。

4.2 本章小结

本章主要内容是在前文的需求分析与设计的基础上，对基于Linux的WebServer服务器的主要模块进行了具体实现与详细介绍，包括服务器初始化模块、HTTP协议模块、高性能IO模块、线程池模块、日志模块以及定时器模块。至此完成了本文对于基于Linux的WebServer服务器的研究及实现，接下来的章节是对于所设计Web服务器的性能测试以及优化。

5 性能测试与结果分析

5.1 测试

5.1.1 测试环境

Web服务器的具体实现

对于服务端而言，本文采用UCloud云主机，操作系统为ubuntu18.04LTS,在此操作系统上自动配置了数据库mysql5.7.41。如图20，选择服务器硬件配置。对于客户第 27 页 (共 45 页)

端而言，使用了压测工具Webbench对服务器的性能进行测试[13]，{95% : Webbench是Linux上一款知名的、优秀的web性能压力测试工具。 }{100% : 它是由Lionbridge公司开发。 }{94% : 它的基本原理：Webbench首先fork出多个子进程，每个子进程都循环做web访问测试。 }{100% : 子进程把访问的结果通过pipe告诉父进程，父进程做最终的统计结果。 }

图20 服务器硬件配置

所以，对本文设计的Web服务器进行测试的硬件环境如表2。

CPU 16 Core

内存 64G

硬盘 80G SATA

操作系统 ubuntu18.04LTS

表2 HTTP响应状态码

基于Linux的WebServer服务器的设计与开发

第 28 页 (共 45 页)

5.1.2 Web功能测试

首先启动UCloud云主机，进入服务器实例，进行服务器的初始化以及启动，等待HTTP请求的到来，核心代码如下，测试的结果我们通过网页抓包来显示，图21为GET方法测试结果。

创建server实例，设置初始化参数，端口为1316，事件模式为ET模式，超时时间为60000毫秒，优雅退出设置为否，{ 57% : mysql端口3306，账号为root，密码为root， }{ 65% : 数据库为webserver， }连接池数量为12，线程池线程数量为6，日志开关为开，日志等级为1，日志异步队列容量为1024，最后启动服务器等待客户端的连接。

```
WebServer server(
1316, 3, 60000, false,
3306, "root", "root", "webserver",
12, 6, true, 1, 1024);
void WebServer::Start() {
int timeMS = -1;
if (!isClose_) { LOG_INFO("========== Server start ==========="); }
while (!isClose_) {
if (timeoutMS_ > 0) { // 如果设置了超时时间
```

```
timeMS = timer_->GetNextTick(); // 获取下一个计时器超时时间
}
int eventCnt = epoller_->Wait(timeMS); // 等待事件发生
for (int i = 0; i < eventCnt; i++) { // 遍历处理每个事件
int fd = epoller_->GetEventFd(i);
uint32_t events = epoller_->GetEvents(i);
if (fd == listenFd_) { // 事件是监听套接字实例
DealListen_(); // 处理监听事件
} else if (events & (EPOLLRDHUP | EPOLLHUP | EPOLLERR)) {
assert(users_.count(fd) > 0);
CloseConn_(&users_[fd]); // 关闭用户连接
性能测试与结果分析
} else if (events & EPOLLIN) { // 可读事件第 29 页 (共 45 页)
assert(users_.count(fd) > 0);
DealRead_(&users_[fd]); // 处理读取事件
} else if (events & EPOLLOUT) { // 可写事件
assert(users_.count(fd) > 0);
DealWrite_(&users_[fd]); // 处理可写事件
} else {
LOG_ERROR("Unexpected event"); // 错误日志
}
}
}
}
```

图21 HTTP GET方法测试结果

从图21中可以看到HTTP请求报文与响应报文的详细内容：请求方法为GET，URL为http://8.137.84.95:1316/；响应报文中状态码为200表示正常工作，返回报文格式为text/html，长度为3148。可知服务器能正确响应HTTP GET请求报文。

基于Linux的WebServer服务器的设计与开发

图22 通过浏览器访问服务器第 30 页 (共 45 页)

通过浏览器访问服务器，结果如图22所示，可知在实际使用中用户也可以通过浏览器访问服务器获取服务，满足Web服务器的基本要求。接着在网页中点击注册按钮，{ 59%：注册一个账号保存到数据库中， }测试POST方法，结果如图23所示。

图23 HTTP POST 方法测试结果

从图23中可以看到请求方法为 POST，URL为http://8.137.84.95:1316/register，状态码为200表示正常响应请求，返回报文格式为text/html，Content-Length表示响应报文中返回的数据长度为3161。可知服务器能正确响应HTTP POST请求报文。

通过本节测试可以得到结论：{ 58%：本文的Web服务器实现了通过HTTP协议进行访问的功能， }{82%：并且支持GET和POST两种请求方法， }{ 59%：用户可以通过浏览器对Web服务器进行访问， }满足了Web服务器的基本设计要求。

5.1.3 服务器吞吐率测试

吞吐率QPS指的是服务器每秒钟处理的请求数或数据量，可以有效的表示服务

器的性能。本节选用Webbench模拟并发连接测试，{87%：Webbench是Linux上著名的网站压力测试工具，}可以展示服务器每秒响应的请求数QPS和每秒钟传及输的数据量以响应失败的请求数。启动UCloud服务器并分配足够的内存，接着安装对应的的服务器与Webbench工具，输入命令开始测试。

Webbench -c 5000 –t 5 http://192.168.142.129:1316/

性能测试与结果分析

如图24展示了Webbench的用法，命令中的5000表示模拟的并发连接数，5表示测试的持续时间，后面跟着要访问的网页URL，等待5s之后就会返回测试每秒钟第 31 页 (共 45 页)

{87%：相应请求数和每秒钟传输数据量。 }

图24 Webbench 性能测试

5.1.4 实验结果分析

实验结果表明：{ 79%：经过webbenchh压力测试可以实现上万的QPS。 }综上可以得出结论：本文设计的服务器事务处理性能表现良好，在以多线程方式工作时能有效提升吞吐率，达到设计要求。

5.2 本章小结

本章对设计的Web服务器的系统吞吐率等性能指标进行了测试，并对实验数据进行了记录整理与分析。Web服务器整体性能良好，在16核16G内存的物理环境下能够满足5M左右的并发连接请求，通过Webbench压力对比测试发现本文设计的服务器吞吐率表现优异。最后得出结论：{ 55%：本文设计的Web服务器性能表现良好， }基本达到了设计要求。

6 总结与展望

基于Linux的WebServer服务器的设计与开发

本篇论文深入探讨了一种基于Linux的高性能Web服务器的原理、设计及实现过程。文章首先分析了服务器中的关键网络协议和网络程序设计的基本知识，通过学习Nginx架构，{ 55%：引发了开发一个轻量级、高效能Web服务器的灵感。 }文章接下来详细介绍了服务器的研究过程和设计策略，对项目需求进行了仔细的分析，并在此基础上展开了全面的系统设计。随后，文中逐一阐述了服务器各个具体模块的实现，包括初始化模块、HTTP协议处理模块、高效IO处理模块、线程池模块、日志记录模块以及定时器模块。最终，通过使用WebBench压力测试工具对服务器性能进行评估，实验数据显示，所设计服务器在事务处理性能上表现出色，基本满足了预定的设计目标。第 32 页 (共 45 页)

由于作者的专业能力和开发时间的限制，本文所开发的轻量级高并发Web服务器[14]还有许多需要改进的地方。在未来的工作中，{ 59%：我计划从以下几个方面进行完善： }

（1）性能优化

随着网络流量的持续增长和数据处理需求的提升，{ 60%：Web服务器的性能优化将成为重点。 }这包括更有效的负载均衡技术、高效的数据缓存机制以及对高并发处理的优化。

（2）安全性强化

网络安全问题日益严峻，设计Web服务器时，更高级的安全措施将被纳入考虑。例如，使用更加复杂的加密技术、实时监控系统和自动化的安全更新。

（3）可扩展性和灵活性

云计算的普及使得Web服务器的可扩展性变得尤为重要。设计时将更加注重服务器的水平扩展能力和容错机制，以及如何快速地适应不同的运行环境和负载变化。

环境友好型设计

随着环保意识的提升，节能减排也将成为Web服务器设计的一个重要方面。这包括优化服务器的能效比，{ 56%：减少数据中心的能耗和碳足迹。 }

支持新技术

随着IoT（物联网）、AI（人工智能）[15]和大数据技术的快速发展，Web服务器需要支持更多先进技术的集成和应用，例如，提供对AI推理的支持或优化大数据分析的处理。

总之，未来的Web服务器设计将趋向于更加智能化、安全化、环保化及高效化，以适应不断演进的技术需求和环境变化。

参考文献

[1]中国互联网络信息中心，第53次中国互联网络发展状况统计报告，2024
[2]聂松松，赵禹，施洪宝，景罗，Nginx底层设计与源码分析，机械工业出版社，2021
[3]金华, 胡书敏，基于Docker的Redis入门与实战，机械工业出版社，2021
[4]小林coding，图解网络，https://www.xiaolincoding.com
[5]马常霞，张占强，TCP/IP网络协议分析及应用，南京大学出版社，2020
[6]杨传栋, 张焕远, 范昊, 徐洪丽，Windows网络编程基础教程，清华大学出版社，2020
总结与展望
[7]李晓红, 唐晓君, 肖鹏，Linux系统及编程基础，Linux系统及编程基础，2021第 33 页 (共 45 页)
[8]柳伟卫，大型互联网应用轻量级架构实战，北京大学出版社，2019
[9]孙鑫，Java无难事:详解Java编程核心思想与技术，电子工业出版社，2023
[10]姚烨, 朱怡安, 张黎翔,计算机网络协议分析与实践,电子工业出版社,2021
[11]陈亮平, 罗南超，Linux下基于epoll+线程池简单web服务器实现，福建电脑，第三十五期：4
[12]陈祥琳，CentOS 8 Linux系统管理与一线运维实战，机械工业出版社，2022
[13]学习webbench的使用，https://www.jianshu.com/p/e9e04fc0fb39
[14]孙海峰，Web安全程序设计与实践，西安电子科技大学出版社，2019
[15]郭军，信息搜索与人工智能，北京邮电大学出版社，2022
基于Linux的WebServer服务器的设计与开发
第 34 页 (共 45 页)
致谢
{ 72%：在本论文完成之际，我首先要感谢我的指导教师朱朝霞老师， }感谢她在研究过程中对我的指导和帮助。朱朝霞老师不仅在学术上给予我极大的支持和启发，{ 56%：也在生活上给予我关心和鼓励， }使我能够顺利完成研究。

{ 55%：同时，我也要感谢计科系的所有教师和同学们。 }在学习和生活中，他们的建议和帮助使我受益匪浅。特别是同班的李斯同学和段婧宇同学，在实验设计和数据分析过程中给予了我大量的帮助。

此外，{ 70%：我还要感谢我的家人对我的学业和生活的全力支持和鼓励。 }他们的理解和爱是我完成学业的坚强后盾。最后向百忙之中评审论文和参加答辩的老师表示深深的感谢。

丁斌斌

2024年5月
致谢
第 35 页 (共 45 页)
附录：
webserver.cpp 服务器初始化
基于Linux的WebServer服务器的设计与开发

```
#include "webserver.h"
using namespace std;
//端口、ET模式、timeoutMs、优雅退出
// sql端口、账号、密码、数据库
//连接池数量、线程池数量、日志开关、日志等级、日志异步队列容量
WebServer::WebServer(
int port, int trigMode, int timeoutMS, bool OptLinger,
int sqlPort, const char *sqlUser, const char *sqlPwd,
```

```
const char *dbName, int connPoolNum, int threadNum,
bool openLog, int logLevel, int logQueSize) :
port_(port), openLinger_(OptLinger), timeoutMS_(timeoutMS), isClose_(false),
timer_(new HeapTimer()), threadpool_(new ThreadPool(threadNum)), epoller_(new Epoller()) {
srcDir_ = getcwd(nullptr, 256); //返回当前工作目录的路径名
assert(srcDir_);
strncat(srcDir_, "/resources/", 16); //将"/resources/"字符串连接到末尾
HttpConn::userCount = 0; //连接的用户数量
HttpConn::srcDir = srcDir_; // HTTP服务器的根目录
SqlConnPool::Instance()->Init("localhost", sqlPort, sqlUser, sqlPwd, dbName,
connPoolNum); //创建一个数据库连接池
InitEventMode_(trigMode); //初始化事件模式为ET模式(3)
if (!InitSocket_()) { isClose_ = true; } //初始化套接字
//日志记录
if (openLog) {
Log::Instance()->init(logLevel, "./log", ".log", logQueSize); //初始化日志
if (isClose_) { LOG_ERROR("========== Server init error!==========="); }第 36 页 (共 45 页)
附录
//如果 isClose_ 为 true，表示服务器初始化出错
else {
LOG_INFO("========== Server init ==========");
LOG_INFO("Port:%d, OpenLinger: %s", port_, OptLinger ? "true" : "false");
LOG_INFO("Listen Mode: %s, OpenConn Mode: %s",
(listenEvent_ & EPOLLET ? "ET" : "LT"),
(connEvent_ & EPOLLET ? "ET" : "LT"));
LOG_INFO("LogSys level: %d", logLevel);
LOG_INFO("srcDir: %s", HttpConn::srcDir);
LOG_INFO("SqlConnPool num: %d, ThreadPool num: %d", connPoolNum, threadNum);
}
}
}
//析构函数
WebServer::~WebServer() {
close(listenFd_); //关闭服务器的监听套接字 listenFd_
isClose_ = true; //表示服务器已关闭
free(srcDir_); //释放存储资源目录路径的内存空间
SqlConnPool::Instance()->ClosePool(); //关闭数据库连接池
}
//初始化事件模式
void WebServer::InitEventMode_(int trigMode) {
```

```
listenEvent_ = EPOLLRDHUP;
connEvent_ = EPOLLONESHOT | EPOLLRDHUP;
switch (trigMode) {第 37 页 (共 45 页)
基于Linux的WebServer服务器的设计与开发
case 0:
break;
case 1:
connEvent_ |= EPOLLET;
break;
case 2:
listenEvent_ |= EPOLLET;
break;
case 3:
listenEvent_ |= EPOLLET;
connEvent_ |= EPOLLET;
break;
default:
listenEvent_ |= EPOLLET;
connEvent_ |= EPOLLET;
break;
}
HttpConn::isET = (connEvent_ & EPOLLET); //判断是否采用边缘触发模式
}
//服务器的主事件循环
void WebServer::Start() {
int timeMS = -1; // epoll等待的超时时间为无限
if (!isClose_) { LOG_INFO("========== Server start ==========="); } //日志
while (!isClose_) {
if (timeoutMS_ > 0) { //如果设置了超时时间
timeMS = timer_->GetNextTick(); //获取下一个计时器超时时间
}
int eventCnt = epoller_->Wait(timeMS); //等待事件发生第 38 页 (共 45 页)
附录
for (int i = 0; i < eventCnt; i++) { //遍历处理每个事件
int fd = epoller_->GetEventFd(i);
uint32_t events = epoller_->GetEvents(i);
if (fd == listenFd_) { //事件是监听套接字实例
DealListen_(); //处理监听事件
} else if (events & (EPOLLRDHUP | EPOLLHUP | EPOLLERR)) {
assert(users_.count(fd) > 0);
```

```
CloseConn_(&users_[fd]); //关闭用户连接
} else if (events & EPOLLIN) { //可读事件
assert(users_.count(fd) > 0);
DealRead_(&users_[fd]); //处理读取事件
} else if (events & EPOLLOUT) { //可写事件
assert(users_.count(fd) > 0);
DealWrite_(&users_[fd]); //处理可写事件
} else {
LOG_ERROR("Unexpected event"); //错误日志
}
}
}
}
//向客户端发送错误信息并关闭连接
void WebServer::SendError_(int fd, const char *info) {
assert(fd > 0);
int ret = send(fd, info, strlen(info), 0); //调用send()函数向客户端发送错误信息
if (ret < 0) { //表示发送错误
LOG_WARN("send error to client[%d] error!", fd);
}第 39 页 (共 45 页)
基于Linux的WebServer服务器的设计与开发
close(fd); //关闭与客户端的连接
}
//用于关闭客户端连接
void WebServer::CloseConn_(HttpConn *client) {
assert(client);
LOG_INFO("Client[%d] quit!", client->GetFd());
epoller_->DelFd(client->GetFd()); //从epoll实例中删除客户端的文件描述符
client->Close(); //关闭客户端连接
}
//添加新的客户端连接
void WebServer::AddClient_(int fd, sockaddr_in addr) {
assert(fd > 0);
users_[fd].init(fd, addr); //初始化新的连接
if (timeoutMS_ > 0) { //添加超时时间
timer_->add(fd, timeoutMS_, std::bind(&WebServer::CloseConn_, this,
&users_[fd])); //添加计时器。
}
epoller_->AddFd(fd,
EPOLLIN | connEvent_); //将文件描述符添加到epoll实例
```

```
SetFdNonblock(fd); //设置为非阻塞模式。
LOG_INFO("Client[%d] in!", users_[fd].GetFd()); //记录信息日志
}
//处理监听套接字实例连接事件
void WebServer::DealListen_() {
struct sockaddr_in addr; //地址信息
socklen_t len = sizeof(addr);第 40 页 (共 45 页)
附录
do {
int fd = accept(listenFd_, (struct sockaddr *) &addr, &len);
if (fd <= 0) { return; }
else if (HttpConn::userCount >= MAX_FD) { //服务器用户连接数已满
SendError_(fd, "Server busy!");
LOG_WARN("Clients is full!");
return;
}
AddClient_(fd, addr); //添加客户端连接
} while (listenEvent_ & EPOLLET); //监听事件采用了边缘触发模式
}
//处理客户端套接字可读事件
void WebServer::DealRead_(HttpConn *client) {
assert(client);
ExtentTime_(client); //更新客户端连接的定时器时间
threadpool_->AddTask(std::bind(&WebServer::OnRead_, this, client));
}
//处理客户端套接字可写事件
void WebServer::DealWrite_(HttpConn *client) {
assert(client);
ExtentTime_(client); //更新过期时间
threadpool_->AddTask(std::bind(&WebServer::OnWrite_, this, client));
}
//更新客户端的活动时间
void WebServer::ExtentTime_(HttpConn *client) {
assert(client);第 41 页 (共 45 页)
基于Linux的WebServer服务器的设计与开发
if (timeoutMS_ > 0) { timer_->adjust(client->GetFd(), timeoutMS_); }
}
//处理客户端的可读事件
void WebServer::OnRead_(HttpConn *client) {
assert(client);
```

```
int ret = -1;
int readErrno = 0;
ret = client->read(&readErrno); //客户端读取数据，存放在读缓冲区中
if (ret <= 0 && readErrno != EAGAIN) {
CloseConn_(client); //关闭客户端连接
return;
}
OnProcess(client); //对读取的数据进行处理
}
//处理客户端读取到的数据
void WebServer::OnProess(HttpConn *client) {
if (client->process()) {//对客户端读取到的数据进行处理
epoller_->ModFd(client->GetFd(), connEvent_ | EPOLLOUT);
} else {
epoller_->ModFd(client->GetFd(), connEvent_ | EPOLLIN);
}
}
//处理客户端套接字的写入事件
void WebServer::OnWrite_(HttpConn *client) {
assert(client);
int ret = -1;第 42 页 (共 45 页)
附录
int writeErrno = 0;
ret = client->write(&writeErrno);
if (client->ToWriteBytes() == 0) { //检查客户端还有待写入的字节数
/* 传输完成 */
if (client->IsKeepAlive()) { //首先检查是否需要保持连接
OnProcess(client); //处理客户端请求
return;
}
} else if (ret < 0) {
if (writeErrno == EAGAIN) { //检查错误码是否为 EAGAIN
/* 继续传输 */
epoller_->ModFd(client->GetFd(), connEvent_ | EPOLLOUT);
return;
}
}
CloseConn_(client); //关闭客户端连接
}
//初始化套接字
```

```
bool WebServer::InitSocket_() {
int ret;
struct sockaddr_in addr; //套接字地址信息
if (port_ > 65535 || port_ < 1024) { //检查端口号
LOG_ERROR("Port:%d error!", port_);
return false;
}
addr.sin_family = AF_INET;
addr.sin_addr.s_addr = htonl(INADDR_ANY);
addr.sin_port = htons(port_); 第 43 页 (共 45 页)
基于Linux的WebServer服务器的设计与开发
struct linger optLinger = {0}; //优雅关闭结构体
if (openLinger_) {
/* 优雅关闭: 直到所剩数据发送完毕或超时 */
optLinger.l_onoff = 1; //启用优雅关闭
optLinger.l_linger = 1; //等待 1秒钟后关闭连接
}
listenFd_ = socket(AF_INET, SOCK_STREAM, 0); //创建了一个套接字实例
if (listenFd_ < 0) {
LOG_ERROR("Create socket error!", port_);
return false;
}
ret = setsockopt(listenFd_, SOL_SOCKET, SO_LINGER, &optLinger, sizeof(optLinger));
if (ret < 0) {
close(listenFd_);
LOG_ERROR("Init linger error!", port_);
return false;
}
int optval = 1;
/* 端口复用 */
/* 只有最后一个套接字会正常接收数据。 */
ret = setsockopt(listenFd_, SOL_SOCKET, SO_REUSEADDR, (const void *) &optval, sizeof(int)); //允许
地址重用
if (ret == -1) {
LOG_ERROR("set socket setsockopt error !");
close(listenFd_);第 44 页 (共 45 页)
return false;
}
ret = bind(listenFd_, (struct sockaddr *) &addr, sizeof(addr));
if (ret < 0) {
LOG_ERROR("Bind Port:%d error!", port_);
```

```
close(listenFd_);
return false;
}
ret = listen(listenFd_, 6); //套接字设置为监听状态
if (ret < 0) {
LOG_ERROR("Listen port:%d error!", port_);
close(listenFd_); //关闭套接字 listenFd_
return false;
}
ret = epoller_->AddFd(listenFd_, listenEvent_ | EPOLLIN);
if (ret == 0) {
LOG_ERROR("Add listen error!");
close(listenFd_);
return false;
}
SetFdNonblock(listenFd_); //将套接字设置为非阻塞模式
LOG_INFO("Server port:%d", port_); //记录日志,服务器已经成功启动
return true;
}
//将指定文件描述符设置为非阻塞模式
int WebServer::SetFdNonblock(int fd) {
assert(fd > 0);
return fcntl(fd, F_SETFL, fcntl(fd, F_GETFD, 0) | O_NONBLOCK);
}
```

第 45 页 (共 45 页)