安阳工学院
ANYANG INSTITUTE OF TECHNOLOGY

本 科 毕 业 论 文

基于 HTTP 协议的 Web 服务器的设计与实现

The Design and Implementation of Web Server Based on HyperText Transfer Protocol

学院名称：计算机科学与信息工程学院

专业班级：软件工程 15-1

学生姓名：李勇峰

学生学号：15031210127

指导教师姓名：冯 贺

指导教师职称：讲 师

2019 年 5 月

毕业设计（论文）原创性声明和使用授权说明

原创性声明

本人郑重承诺：所呈交的毕业设计（论文），是我个人在指导教师的指导下进行的研究工作及取得的成果。尽我所知，除文中特别加以标注和致谢的地方外，不包含其他人或组织已经发表或公布过的研究成果，也不包含我为获得安阳工学院及其它教育机构的学位或学历而使用过的材料。对本研究提供过帮助和做出过贡献的个人或集体，均已在文中作了明确的说明并表示了谢意。

作 者 签 名：_____　　日　　期：_____

指导教师签名：_____　　日　　期：_____

使用授权说明

本人完全了解安阳工学院关于收集、保存、使用毕业设计（论文）的规定，即：按照学校要求提交毕业设计（论文）的印刷体和电子版本；学校有权保存毕业设计（论文）的印刷体和电子版，并提供目录检索与阅览服务；学校可以采用影印、缩印、数字化或其他复制手段保存论文；在不以赢利为目的前提下，学校可以公布论文的部分或全部内容。

作者签名：_____　　日　　期：_____

目　录

基于 HTTP 协议的 Web 服务器的设计与实现

摘要：现在，我们所处的这个时代，是一个真正属于互联网的时代。我们获取信息的方式已经从传统媒体（电视、广播、杂志、报纸）这种单方面传输的方式中脱离。现在，我们获取信息，已经不再满足于这种传统媒体，而是主观的从网络中选择自己感兴趣的信息。Web服务其实就是一种使我们能够主动获取信息的一种技术。信息提供者分门别类的把信息放到Web站点上，而我们则可以主观的从Web服务站点去筛选，通过URL（统一资源定位符）去拿到我们感兴趣的信息。因为每个人的兴趣爱好不同，移动设备的普及，网络速度的提高，网络获取信息的便捷性等种种原因，Web服务已经成功取代了传统媒体，成为这个时代不可缺少的一部分。

关键词：C/C++；Linux；epoll；HTTP；线程池；有限状态自动机；Web 服务器

The Design and Implementation of Web Server Based on HyperText Transfer Protocol

Abstract: Now, we are living in an era that truly belongs to the Internet.The way we obtain information has been separated from the traditional media (television, radio, magazines, newspapers) which transmit information unilaterally.Nowadays, we are no longer satisfied with the traditional media to obtain information, but choose the information we are interested in from the network subjectively.Web services are technologies that enable us to proactively obtain information.Information providers categorize information on Web sites, while we can subjectively filter it from Web service sites and get the information we are interested in through the URL (Uniform Resource Locator).Because everyone has different interests, the popularity of mobile devices, the improvement of network speed, the convenience of network access to information and other reasons, Web services have successfully replaced the traditional media and become an indispensable part of this era.

Key words: C/C++; Linux; epoll; HTTP; Thread Pool; Finite State Automaton; Web Server

引　言

计算机与互联网技术的发展改变了我们的生活。在获取信息方面，我们从传统的报纸、杂志、广播或电视机这种被动接收信息的方式，到现在的很轻松的通过使用移动设备或者是 PC 机，从万维网上主动的，有选择的去获取我们感兴趣的信息。与传统的传播方式比较，其方便快捷、信息全面、可主动选择性等优点早已取代了传统的信息传播方式。真正的达到了“秀才不出门，尽知天下事”的程度。这种现代信息传播方式依赖于万维网技术，在信息传播领域，万维网已成为最受欢迎的传播信息的方式。

万维网获取信息由于其便捷性，让人们足不出户就可以浏览获取各种各样的信息。并且从中随意选择自己感兴趣的信息。这种方式其实是信息的发布者通过再万维网中的 Web 站点挂在信息，信息的获取者则通过 URL 去获取到该站点上的信息。站点是怎么得知用户请求的是什么呢？这就是依靠 HTTP 协议传输数据包到指定的 Web 站点服务器上，然后由该服务器上的一个服务进程去解析这个数据包中用户请求的信息，然后将这个信息发回给用户。其中这个 Web 站点上用来解析用户请求的进程就是 Web 服务进程。对应的程序就是一个 Web 服务程序。这种成熟的 Web 服务程序再市面上有很多，比如：Nginx、Apache、微软的 IIS、淘宝的 Tengine（一款基于 Nginx 开发出来的 Web 服务程序）、Lighttpd 等等。这些都是市面上常见的 Web 服务程序。

当然，对于我来说，要开发出一款跟这些成熟的 Web 服务程序一样功能强大的 Web 服务程序还是十分困难的，但是实现一个功能简单的类似的 Web 服务程序还是可以做到的。所以接下来就来实现一个功能简单的 Web 服务程序。对于一个 Web 服务程序，其核心就是解析用户发来的 HTTP 请求报文，然后对其做出相应的 HTTP 响应。由于完整功能的 Web 服务程序比较复杂，这里开发的 Web 服务程序就只支持 HTTP 请求报文中的 GTE 静态请求，其基本涵盖了开发完整 Web 服务程序的核心思想，后期如果有兴趣可以对其功能进行完善。

第 1 章 概述

计算机与互联网技术的发展改变了我们的生活。在获取信息方面，我们从传统的报纸、杂志、广播或电视机这种被动接收信息的方式，到现在很轻松的通过使用移动设备或者是 PC 机，从万维网上主动的，有选择的去获取我们感兴趣的信息。与传统的传播方式比较，其方便快捷、信息全面、可主动选择性等优点早已取代了传统的信息传播方式。这种现代信息传播方式依赖于万维网技术，在信息传播领域，万维网已成为最受欢迎的传播信息的方式。万维网使有一个个的 Web 站点连接起来的，而 Web 站点服务器中必须有 Web 服务程序用来解析 HTTP 请求报文。一个 Web 服务程序的性能好坏决定了在网络高峰期时，用户的请求被响应的速度快慢，这直接影响用户对该网站使用时的用户体验。所以研究一个高性能的 Web 服务器程序是非常有必要的课题。

1.1 课题背景

从计算机与网络技术的发展开始，到现在进入移动互联的时代。现在互联网在全球范围内基本已经普及，使用的人也越来越多。在国内，基本上每一个成年人都用于一部属于自己的移动设备。人们每天查阅资料、观看新闻、搜索问题等等都是在从网络中获取信息，这几乎成为我们生活中不可或缺的一部分，而使用万维网传播信息已成为最受欢迎的传播信息的方式。万维网在传播信息的过程中必须要通过 Web 站点中的 Web 服务进程解析发来的客户端 HTTP 请求报文。当前，市面上由很多成熟的 Web 服务程序。这种 Web 服务程序不但性能高效，而且可以同时支持多用户的高并发处理，是经得住现网大流量考验的。正是这种 Web 服务程序，才让我们可以很轻松的通过 URL 到指定的 Web 站点去获取我们想要的信息。它为我们流畅的、高效的获取信息提供的关键性的作用。

如今，在这个互联网高速发展的年代，快速的、稳定的获取我们想要的信息成为了我们作为使用者最在乎的问题。Web 站点中使用一个性能优越的 Web 服务程序是对客户端访问速度的提升是必须的。将一个性能优越的 Web 服务程序运行在 Web 服务站点的服务器机器上，来快速高效的接收解析用户发送的 HTTP 请求报文，并快速的将处理后的 HTTP 响应数据包发送回用户。用户通过浏览器程序通过 URL 快速的访问 Web 站点上的信息，然后迅速的得到 Web 站点上返回的响应信息。

1.2 课题意义

随着计算机与互联网技术的发展，网民数量呈现爆炸式增长，这就导致了每时每刻

Web 服务站点都会接收到大量的信息请求，这对 Web 服务程序的性能有了很大的考验。淘宝曾经还因为在双十一承受不住这种大流量请求而服务器宕机。相同的硬件配置的情况下，一个性能极差的 Web 服务程序可能会导致用户接收信息的时间大大延长，更严重的情况，用户无法请求到数据，站点崩溃。所以编写一个简单的 Web 服务程序来研究 Web 服务程序的实现，性能优化等等是非常有必要的。通过编写一个而简单的 Web 服务程序来研究 Web 服务程序的工作原理，可以让我们更加清晰的认识到 Web 服务程序性能提升的关键点在什么地方，有什么好的优化方式，以及优化的瓶颈在是什么。这对以后优化 Web 服务程序，或者编写性能更加高效的 Web 服务程序有着重大意义。

1.3 课题内容

就目前来说，市面上有很多成熟的 Web 服务程序。它们有一个共同的特点，就是都追求高效、稳定的。为了更好的研究 Web 服务程序，本课题准备编写一个简单的 Web 服务程序，从而更好地研究 Web 服务程序的细节。本 Web 服务程序提供了对 HTTP 请求中 GET 静态请求的解析，并且组织 HTTP 响应报文将其发送回客户端。其中使用了线程池和请求队列技术，用于提供并发执行的功能。另外在处理 HTTP 请求报文的时候，使用有限状态自动机来高效的解析 HTTP 请求报文。而且本程序提供了日志输出功能以保证程序后期维护以及错误检查和访问检查。本服务程序还提供了配置文件的功能，以便用户自己配置一些数据。使用 Makefile 来进行项目管理，编译程序。最后本 Web 服务程序还提供了一个脚本工具用于对进程的管理。

总的来说，本 Web 服务程序基本实现了一个正常服务程序的最基本功能。虽然仅仅支持 HTTP 请求的 GET 静态请求功能，但是这是一个 Web 服务程序最核心的功能，其他功能的思想基本一致，细节处理可能稍有不同。其他功能以后有时间去慢慢增加完善。争取做成一个功能齐全的 Web 服务程序。

1.4 本章小结

本章主要介绍了课题研究的背景、课题研究的主要内容、和涉及到的一些相关技术，为本 Web 服务程序的设计和实现提供了必要的理论知识基础。

第 2 章 系统分析

在开发研究一个项目之前，对项目进行全面的系统分析是必须的。首先要对项目进行一个需求分析，来了解该项目的基本功能模块有哪些，分析完基本功能后，要对项目的这些功能模块进行可行性分析，可行性分析包括经济可行性、技术可行性、运行可行性。经济可行性就是指完成该项目需要的成本能否在经济能力范围之内。技术可行性是指在技术层面上来粗略的验证一下整个项目的基本功能能否实现出来。运行可行性是指项目能否再现有的设备环境下顺利的执行起来。

2.1 需求分析

对比于现有的成熟的 Web 服务程序，经过对这些成熟的 Web 服务程序进行研究分析，得知了一个 Web 服务程序的基本逻辑功能，而我们开发的 Web 服务程序是一个简单版，只需要实现 Web 服务程序最基本的功能即可。但是这个功能包含了每一个成熟 Web 服务程序的核心处理思想。我们这个简单的 Web 服务程序应具有的功能如下：

1. 接收浏览器客户端发来的 HTTP 的 GET 静态请求。
2. 解析 HTTP 请求数据报。
3. 对请求做出相应的 HTTP 响应报文发送回去。
4. 当多个客户端同时请求时，Web 服务程序应支持其并发执行。
5. 对 I/O 的处理应高效。
6. 由于服务程序为守护进程，所以应支持日志功能。
7. 应为提供配置文件，方便一些信息的配置，增加程序灵活性。
8. 提供一个 Makefile 用来管理编译项目文件。
9. 提供一个进程管理工具，方便进程管理。

2.2 可行性分析

在一个项目开始之前，进行可行性分析是必须的，因为一个项目开发之前需要做好充分的准备，以免开发中发现程序的某些功能无法实现导致项目失败。所以我们一定要在项目着手开发之前进行可行性分析。

2.2.1 经济可行性分析

完成本Web服务程序需要一台装有Linux操作系统的虚拟机的电脑就可以了，或者也可以使用阿里云、腾讯云等租借的一台Linux远程服务器。现在电脑非常流行，几乎每家都有，所以非常普遍。远程服务器的话其实也很便宜，阿里云的一般配置的Linux系统的

远程服务器学生租借一年才￥110 多，也比较便宜。开发环境的话在Linux上使用vim开发或者VScode，Sublime Text等其它编辑工具也可以。至于编译器，一般Linux的操作系统默认都会安装有gcc和g++，就算没有可以直接下载安装一个，都是免费的。另外就是电脑上装有一个浏览器程序，方面后期运行测试。这些在经济上完全可行。

2.2.2 技术可行性分析

开发一个Web服务程序首先为其选用一门语言，因为Web服务程序需要比较高的性能，所以C/C++语言无疑是首选语言。而且由于系统环境为Linux操作系统，需要使用Linux系统API，多路I/O复用技术直接使用Linux下的高效epoll系列系统调用，Web服务程序的高并发支持使用线程池来实现，解析模块使用高效的有限状态自动机。配置文件选用JSON格式，使用轻量级JSON解释器cJSON函数库来进行解析。进程管理工具使用bash脚本来写。这些在技术上是完全可行的。

2.2.3 运行可行性分析

该Web服务程序运行在Linux操作系统上，可以是远程Linux的设备或者本地Linux虚拟机，占用资源极少，程序启动后后台运行，所以运行方面没有问题。

2.3 本章小结

本章主要是对该Web服务程序进行需求分析、可行性分析以及对本Web服务器程序开发需要的技术进行简单研究分析，为本服务程序的设计和实现提供理论基础。

第 3 章 总体设计

在第 2 章中已经完成了系统分析，下面就要开始着手项目的实现了。在对项目进行实现之前，肯定要对整个项目进行总体设计和分析，进一步对每一个功能模块进行详细的分析设计。

3.1 服务程序整体架构设计分析

本Web服务器程序采用的是分模块开发的模式，对各个模块进行功能编写。本Web服务器程序主要实现对HTTP的GET请求进行处理，而且仅支持静态请求，动态请求其实类似，对HTTP请求进行解析，但是需要解析出请求的参数信息，在处理上还要将这些参数发送给CGI程序去做处理，相对于静态请求来说麻烦一点，所以这里就不做动态请求的处理了。按照服务器的基本架构，无论是B/S模型的服务程序还是C/S模型的服务程序，一个最基本的服务器程序都大致的包含这几个模块：I/O处理模块、请求队列和逻辑处理模块。本Web服务器程序也不例外。本Web服务器主要包含了以下几个模块：I/O处理模块、逻辑处理模块、请求队列和线程池、日志功能模块、配置功能模块。每一个模块各司其职而又相互关联，组成整个系统的基本架构，使整个项目正常平稳的运行起来。整个项目的架构如图 3-1 所示。

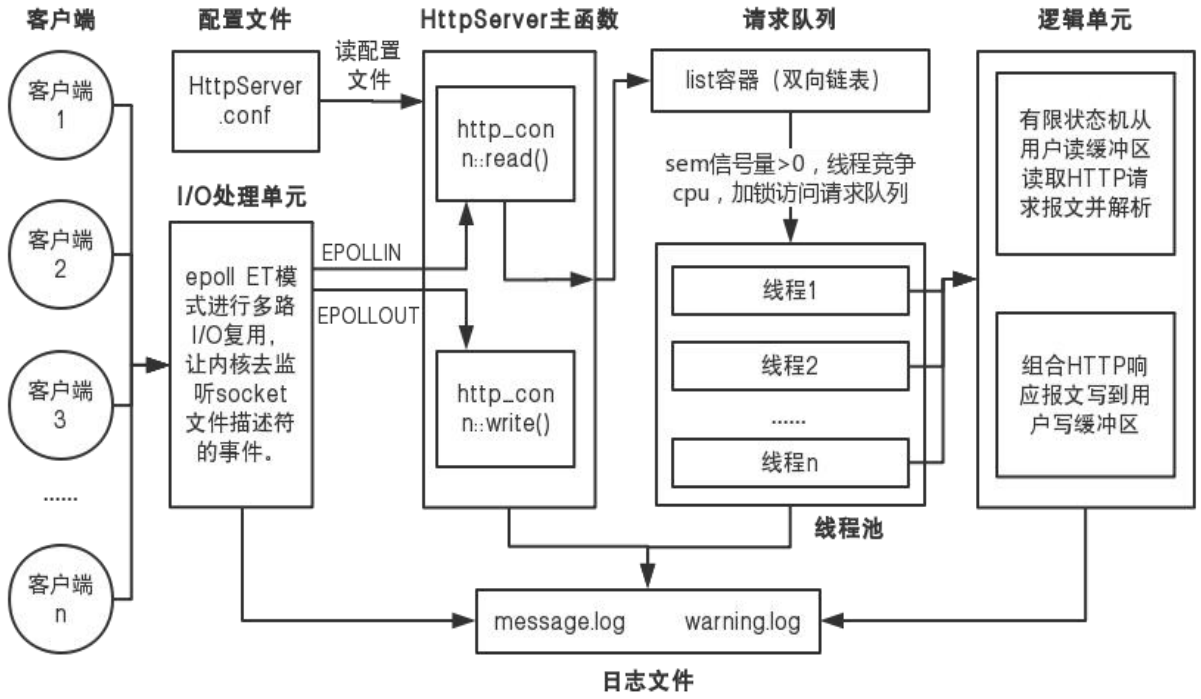

图 3-1 整体架构设计流程图

3.1.1 I/O 模块分析

由于本服务程序运行在Linux操作系统下，而在Linux操作系统下的I/O复用的系统调用有三种：select系统调用、poll系统调用和epoll系列系统调用。下面讲解一下在本程序中关于I/O复用的三个问题。

1. 为什么要使用I/O复用技术呢？如果是传统的意义下使用高并发多线程或多进程处理多个客户端连接，那么当一个客户端连接成功时，我们需要使用一个进程（或线程）去处理这个通信，监听该通信套接字上的内核缓冲区中是否有可读、可写或异常事件发生（一般情况下一个文件描述符发生的事件为这三种，当然还有一些不常见的），当上百万或者更多的客户端连接进来时，那么服务程序就要开上百万个这样的进程（或线程）去监听所有客户端的通信套接字，这么多的进程（或线程）会导致CPU占用过高，而且这种方式下，主进程需要自己阻塞的进行监听所有客户端，效率变得低下。下面就是传统多线程（或进程）高并发的图示，如图 3-2 所示。

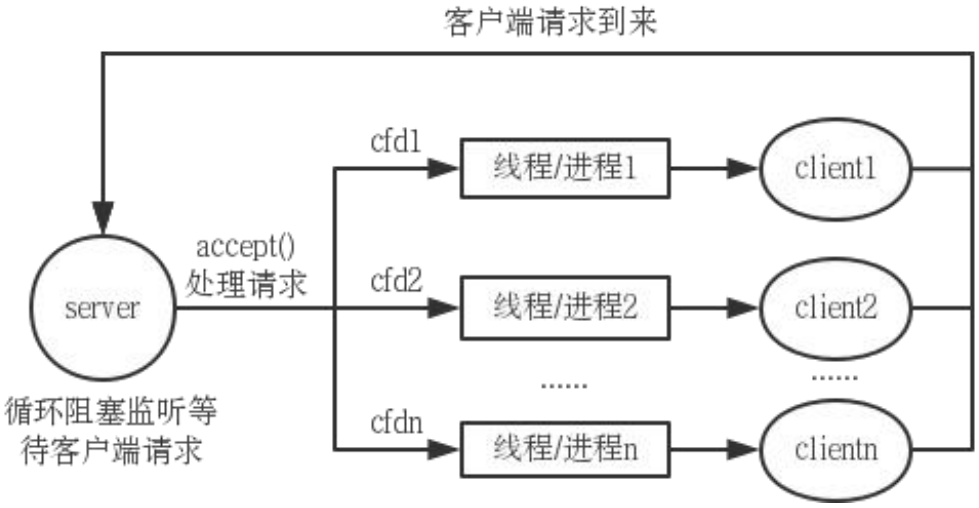

图 3-2 传统高并发

所以就诞生了I/O复用技术，用户进程委派Linux的内核去监听所有客户端的通信套接字。当有读写或异常事件发生时，内核会通知用户进程，让用户进程去对该事件做出相应的处理，这对提高服务程序的性能至关重要。但是有一点需要说明，多路I/O复用技术其实并不是并发的，当内核监听的某个文件描述符上有可读、可写或异常事件发生时，内核将发送事件的文件描述符返回给用户进程，用户进程拿到文件描述符后对其进行发生的事件进行逐个的处理，对于用户程序其实这还是单线程的串行处理，所以多路I/O复用技术并不能实现高并发。这里要想实现高并发，最终还是必须要借助多线程或者多进程去实现。

基本上所有的高并发都要借助多线程或多进程才能实现（在Golang语言中的协程好像也可以实现高并发，这部分我也不是很了解，有兴趣可以查询一下）。本程序在的高并发实现会在后面线程池模块的小节中做详细讲解。下面就是使用I/O复用技术的多线程（或进程）高并发的图示，如图 3-3 所示。

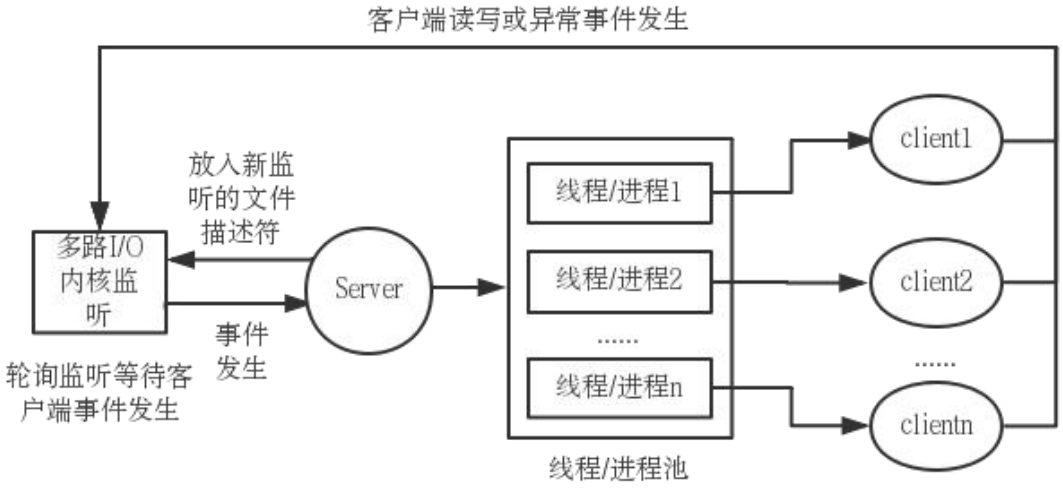

图 3-3 多路I/O实现的高并发

2. 为什么要选用epoll系列系统调用作为服务器的多路I/O复用处理？对比与select系统调用和poll系统调用，epoll系列的系统调用有很多的优点。首先epoll系列的系统调用在实现上跟select系统调用和poll系统调用有着很大的差异。epoll系列系统调用把用户关心的文件描述符（即服务端与客户端的通信套接字）和监听套接字上的事件通过epoll_ctl系统调用放在内核里的一个事件表中，从而无须像select和poll那样每次调用都要重复从用户空间传人传出文件描述符集到内核空间。不过epoll需要使用一个额外的文件描述符，用来唯一标识内核中的这个事件表。另外，epoll系列系统调用内部维护的内核事件表的内部实现为一颗红黑树，这样即使我们调用epoll_ctl函数放入上百万个文件描述符，由于红黑树是二叉搜索树的变体，所以我们依然可以迅速的遍历到这个发生事件的文件描述符所在的位置。此外epoll系列系统调用内部还维护了一个list链表，用于存放发生事件的文件描述符。当调用epoll_wait函数时，只需要将list中发生事件的socket文件描述符从内核返回到用户进程的events传出参数中。而select系统调用和poll系统调用在每次调用之前要使用一个数组将要监听的文件描述符放到数组中，该数组作为一个传入传出参数使用。在函数执行完毕后，将发生事件的文件描述符放入到该数组中，然后将其返回给用户进程。用户进程拿到后，通过遍历该数组去获取发生事件的文件描述符。对比epoll系列的系统调用，select系统调用和poll系统调用在性能上差了很多。首先select系统调用和poll系统调用需要频繁的

将事件数组传递给内核，处理完之后内核还要将其返回给用户缓冲区。在这中间，数组数据需要频繁的在内核和用户空间中进行上下文你切换，而epoll系列的系统调用则是使用内核维护的事件表（红黑树），只要对其进行插入和删除等操作即可，无需对整个事件数组进行操作。另外在返回值遍历的时候由于epoll采用的是红黑树结构的内核事件表，其本质是一个平衡二叉搜索树的一个变体，而select系统调用和poll系统调用需要遍历其传入传出参数的数组，从二叉搜索树中寻找一个值比起从数组中遍历查找在时间复杂度上要快。所以这就导致epoll系列系统调用在遍历速度比起select系统调用和poll系统调快的多。下面是关于epoll调用的详细使用过程图解，如图 3-4 所示。

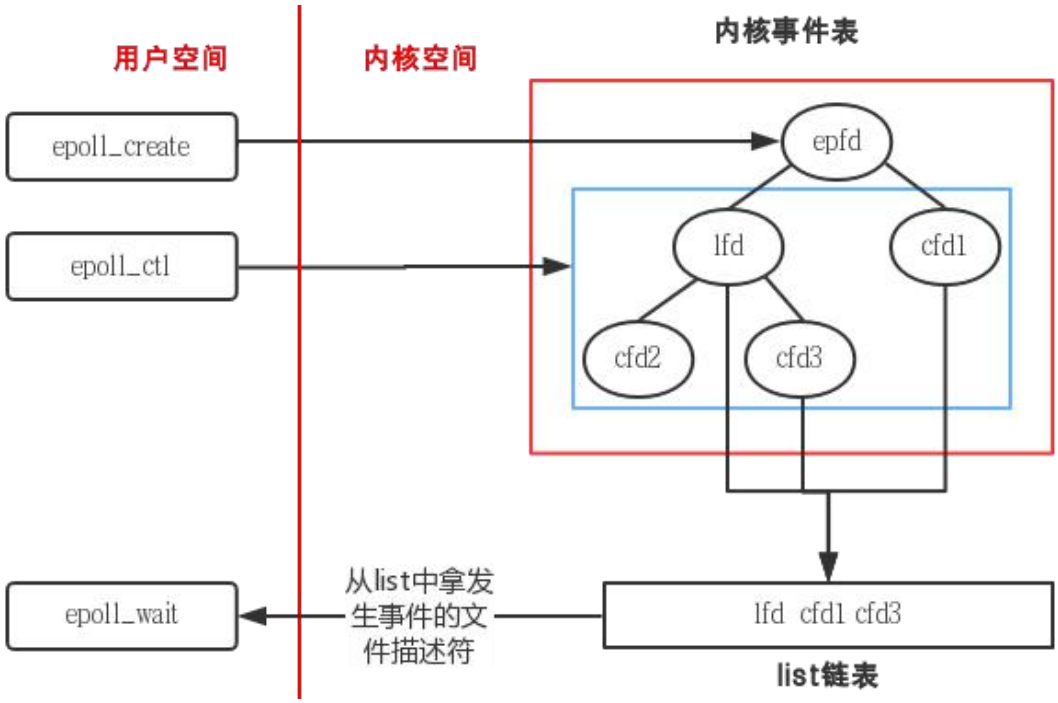

图 3-4 epoll系列系统调用过程

3. 内核监听的读写或异常事件什么时候发生？当网卡接收到从网络中发来的数据时，存入其网卡硬件缓冲区，这时会触发中断信号来通知CPU，CPU接收到中断信号后，会调用相应的驱动接口函数从网卡的硬件缓冲区中把数据读到内核接收缓冲区中（其间由于网卡数据属于网络层数据报，中间需要解包到应用层），当内核接收缓冲区中的字节数大于等于最低水位标记SO_SNDLOWAT时，就会触发读事件。然后用户进程调用read函数把数据从内核接收缓冲区复制到进程的用户缓冲区中。同理，用户进程通过write函数从进程的用户缓冲区复制到内核的发送缓冲区中，当内核发送缓冲区中的字节数大于或等于其低水位标记SO_RCVLOWAT时，就会触发写事件，而异常事件触发时机为由带外数据到来，所谓带外数据，是给应用程序提供的一种紧急方式，其大致可以理解为在报文的TCP协议

头中的紧急指针和紧急指针标志字段填充适合的数据，服务端接收到该报文后就会处理这个报文的带外数据。读写事件发生时机的图解分析如图 3-5 所示。

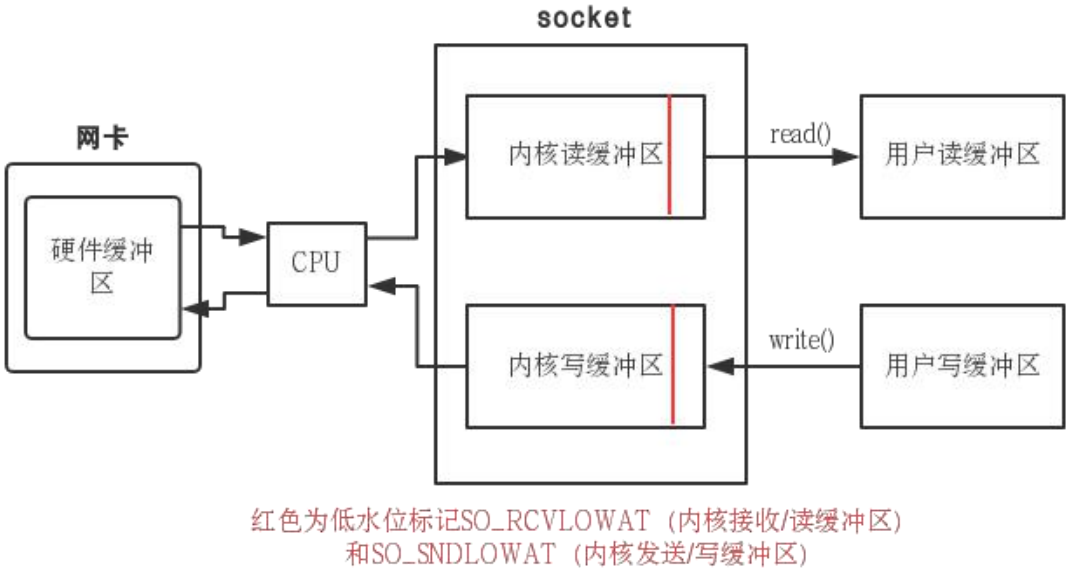

图 3-5 读写事件触发原理

3.1.2 逻辑处理模块分析

由于本服务器为一个Web服务器程序，所以其逻辑处理大概分为这两部分：1. 获取HTTP请求报文并对该HTTP请求报文进行解析。2. 组织发送HTTP响应报文。在对HTTP请求进行解析的部分，本程序采用有限状态自动机（Finite State Automaton）模型这一高效的逻辑处理模型，有限状态自动机模型大概分为两种：不带状态转移的有限状态自动机和带状态转移的有限状态自动机。本程序使用的有限状态机为带状态转移的有限状态自动机。通过状态之间的转移，逐步去分析HTTP请求报文，从而一步一步的解析出请求行中的请求类型，请求数据，HTTP协议版本，请求头中的所有键值对等。

3.1.3 请求队列和线程池

本服务器程序使用线程池来实现高并发，前面 3.1.1 节中提到过，多路I/O能大大提高服务程序的性能但服务程序仍然不是高并发的。要想使服务程序高并发，只能通过多线程或者多进程的手段实现。

为什么使用线程池来实现高并发？首先来讲讲传统的多线程模式（多进程模式与之类似），当客户端有连接请求时，服务端为该客户端创建一个线程来与之通信，当有很多客户端连接请求时，服务程序会创建很多的线程来与每个客户端进行通信。在来看看线程池方式（进程池的方式与之类似），在程序启动后服务程序就创建好了n个线程，是他们处于等待状态，当由新的客户端连接请求时，这些创建好的n个线程去争夺CPU，然后去与

该客户端进行通信，通信结束后继续处于等待状态，等待下一个请求到来。当大量客户端请求到来时，如果n个线程都已经处于工作状态，那么新的请求将进入到请求队列中等待有空闲线程时在做处理（这种方式是本服务程序线程池的处理方式，其实在对大量请求到来时的情况，也可以有别的处理方式，比如：线程池中初始化一个监听线程，让其轮询地监听线程池中工作线程的数量，当其工作线程占全部线程的一个比例，如 80%时，让服务程序在创建一批空闲线程，以备大量连接来时线程数不够用，当工作线程占比低于一个比例，如 20%时，让服务程序销毁一部分空闲线程）。

对比传统的多线程模式和线程池模式，可以很明显的发现线程池的一个优势，即线程的创建在客户端请求到来之前，这样可以节省掉服务程序处理客户端请求时创建线程所消耗的时间。另外，在应对大量客户端请求到来时使用多线程的方式仍然会发生前面 3.1.1 节提到的线程创建过多，CPU占用严重的问题。而使用线程池时，可以对请求数量做一个限制，让其超过这个限制时，进入请求队列中等待处理。其对比仍可以参看上面的图 3-2 和图 3-3 所示。

3.1.4 日志功能模块

为什么需要编写日志模块？由于服务程序一般是脱离控制终端的守护进程，当程序由于某种原因，导致程序崩溃时，无法向终端输出错误信息。针对这一问题，大部分的服务程序都拥有打印日志的功能，将一些运行信息和出错信息写到日志中去，便于后期运维和排查错误。所以本Web服务程序也提供了日志模块用于输出日志信息。

日志输出的原理是什么？其实这个输出日志原理很简单，首先在服务程序启动时，在程序中打开日志文件，然后当打印错误或者写一些必要的正常输出信息时，向该日志文件中写入数据。一般对于写日志，都会封装一些写日志函数，便于服务程序中的调用。一般情况下，还会对日志分级别，将一些正常的打印信息与异常错误告警信息区分开，放在不同的日志文件中。

3.1.5 配置功能模块

为什么要使用配置文件？对于不同的用户需求场景，可能会有一些细微的不同要求，如果不使用配置文件，可能在一些细微的不同要求下，要修改程序的源代码，来去匹配这些细微要求。这样会造成很多不便，所以这时就可以使用配置文件来解决这一问题，将这些细微的不同要求放在配置文件中作为一个配置项，当程序启动时，自动加载解析这些配置项，这样就可以很容易的应对这些要求，当一些参数需要改变时，只需要修改配置文件，然后重启服务即可，而不再需要对程序源码做出更改。大大方便了用户使用，也使程序变

得更加灵活。

本程序在对配置文件进行选择时使用了JSON格式的文件作为配置文件。JSON为一种格式化数据语言。其实向JSON这种格式化数据语言有很多，XML，JSON，INI，YAML等，INI格式经常被用在Windows操作系统下的配置文件。

为什么要选用JSON作为配置文件呢？...，其实说实话JSON文件并不是作为配置文件的一个好的选择，甚至说这是一个坏的主意。先说说JSON为何不适合作为配置文件，首先选用JSON作为配置文件有一个最大的缺点，那就是JSON文件不支持注释，也就是说对于JSON中的配置项，你不能使用注释的方式对其进行解释说明。另外JSON严格的语法要求很容易导致在修改配置文件后服务程序在读取配置文件时发生错误。那为什么还要选择JSON做配置文件呢，其实，...，这个原因是，我对其他的格式化数据语言的语法和其解释器程序库并不了解，另外C/C++中拥有一个非常轻盈的小型JSON解释器程序库，可以直接拿到自己的服务程序中调用，非常简单，所以本程序中选用JSON格式作为配置文件的格式。

3.2 本章小结

本章的主要任务是简述本服务程序的整体设计结构和各个模块的简单介绍以及通过画图和文字结合的方式进行了一些原理分析解释，为下一个阶段架构搭建和各个模块的具体实现做一个铺垫。

第 4 章 详细设计与实现

前面已经第 3 章已经讲过本Web服务程序的架构和各个模块的简单分析和实现原理等，现在我们就对各个模块进行具体的代码实现和对代码进行简单分析。

4.1 I/O处理模块

本Web服务程序在处理I/O事件时使用多路I/O复用技术去处理，从而提高服务程序的性能。对于多路I/O复用的原理分析，在上面第 3.1.1 节中已经讲述的非常详细，这里就不再赘述。

首先是main.cpp中epoll系列系统调用使用的核心代码，下面来看下代码中epoll系列系统调用的代码实现。

```
epoll_event events[ MAX_EVENT_NUMBER ];
int epollfd = epoll_create( 5 );
addfd( epollfd, listenfd, false );
http_conn::m_epollfd = epollfd;
while( true ) {
    int number = epoll_wait( epollfd, events, MAX_EVENT_NUMBER, -1 );
    for ( int i = 0; i < number; i++ ) {
    int sockfd = events[i].data.fd;
    if( sockfd == listenfd ) {
        struct sockaddr_in client_address;
        socklen_t client_addrlength = sizeof( client_address );
        int connfd = accept( listenfd, ( struct sockaddr* )&client_address, &client_addrlength );
        if ( connfd < 0 ) {
            continue;
        }
        if( http_conn::m_user_count >= MAX_FD ) {
            continue;
        }
        users[connfd].init( connfd, client_address );
    } else if( events[i].events & ( EPOLLRDHUP | EPOLLHUP | EPOLLERR ) ) {
```

```
                users[sockfd].close_conn();
            } else if( events[i].events & EPOLLIN ) {
                if( users[sockfd].read() ) {
                    pool->append( users + sockfd );
                } else {
                    users[sockfd].close_conn();
                }
            } else if( events[i].events & EPOLLOUT ) {
                if( !users[sockfd].write() ) {
                    users[sockfd].close_conn();
                }
            }
        }
```

我们对上面的代码做简单的分析。在main.cpp中使用epoll_create系统调用创建了epoll的内核监听事件表（即红黑树）的根节点，并使用自己封装的addfd函数（其实现内部调用epoll_ctl系统调用）将Web服务程序的监听文件描述符listenfd加入到epoll的内核监听事件表中，然后循环去调用epoll_wait系统调用去取出内核内部维护的list链表中的已触发事件的文件描述符，接着循环判断这些已触发事件的文件描述符是否为新客户端连接，如果是新的客户端，将调用accpet系统调用从等待连接队列中取出第一个连接，即该客户端与服务端的通信描述符。然后将该文件描述符也加入到epoll的内核监听事件表中。如果不是新客户端连接，判断该发生事件的文件描述符的事件类型，本Web服务程序只对读写事件进行处理，其它事件默认关闭客户端连接。如果发生读事件，去进行进行读操作，从内核缓冲区中读取HTTP请求报文数据。如果是写事件，则进行写操作，将组织好的HTTP响应报文数据写入内核写缓冲区中。

4.2 逻辑处理模块

本Web服务程序在逻辑处理方面使用了带有状态转移的有限状态自动机模型，可以说它是一种算法思想或者说它是一种程序设计的一种模型。其可以很清晰的表示出程序运行过程中每一个状态运行的结果和其对应的下一个状态要去处理的任务。在上面 3.1.2 节也进行了简单的介绍，下面来看看用来解析HTTP请求报文的有限状态自动机的实现。

下面列出了主状态自动机状态的枚举值，如下。

```
enum HTTP_CODE { NO_REQUEST, GET_REQUEST, BAD_REQUEST, NO_RESOURCE,
FORBIDDEN_REQUEST, FILE_REQUEST, INTERNAL_ERROR, CLOSED_CONNECTION };
```

主状态自动机是解析HTTP请求报文的核心，下面来展示一下主状态自动机的代码实现，如下。

```
http_conn::HTTP_CODE http_conn::process_read()
{
    LINE_STATUS line_status = LINE_OK;
    HTTP_CODE ret = NO_REQUEST;
    char* text = 0;
    while ( ( ( m_check_state == CHECK_STATE_CONTENT ) && ( line_status == LINE_OK  ) )
|| ( ( line_status = parse_line() ) == LINE_OK ) ) {
        text = get_line();
        m_start_line = m_checked_idx;
        switch ( m_check_state ) {
            case CHECK_STATE_REQUESTLINE:
            {
                ret = parse_request_line( text );
                if ( ret == BAD_REQUEST ) {
                    return BAD_REQUEST;
                }
                break;
            }
            case CHECK_STATE_HEADER:
            {
                ret = parse_headers( text );
                if ( ret == BAD_REQUEST ){
                    return BAD_REQUEST;
                } else if ( ret == GET_REQUEST ) {
                    return do_request();
```

```
                    }
                    break;
                }
                case CHECK_STATE_CONTENT:
                {
                    ret = parse_content( text );
                    if ( ret == GET_REQUEST ) {
                        return do_request();
                    }
                    line_status = LINE_OPEN;
                    break;
                }
                default:
                    return INTERNAL_ERROR;
            }
        }
        return NO_REQUEST;
    }
```

上面列出了主状态机的代码，接下来简单讲述一下有限状态自动机的大概思想。首先设置主状态机的初始状态为NO_REQUEST，然后开始执行状态机，循环条件设置为到达终止状态，然后判断当前状态是哪一个状态，进而去做相应状态的处理，一轮处理完后再次判断当前状态，如果为终止状态，状态机执行结束，否则再根据当前的状态去做该状态的相应处理。一轮一轮的执行下去，直到达到状态机的终止状态为止。

4.3 线程池和请求队列

本Web服务程序使用线程池来进行高并发处理，其线程池的基本原理和构造在前面3.1.3 节中进行了详细的介绍，下面来简单分析一下线程池的代码，线程池声明代码如下。

```
template< typename T >
class threadpool
{
```

```
public:
    threadpool( int thread_number = 8, int max_requests = 10000 );
    ~threadpool();
    bool append( T* request );
private:
    static void* worker( void* arg );
    void run();
private:
    int m_thread_number;
    int m_max_requests;
    pthread_t* m_threads;
    std::list< T* > m_workqueue;
    locker m_queuelocker;
    sem m_queuestat;
    bool m_stop;
};
```

我们对上面线程池类的声明部分代码做简单分析。上面线程池类使用了C++的模板类技术，这样使用该线程池的对象可以是任何对象，这是一个适用于任何对象的线程池。线程池中构造函数使用默认参数，即如果不传入参数线程池中线程个数，则默认为 8 个，默认请求队列中的最大请求个数为 10000 个。当客户端请求数量大于线程数 8 个的时候，该线程池类中使用list双向链表容器来存放这些请求，待线程池中有空闲线程的时候，在从请求队列中去取出这些等待的请求去做处理。另外，线程池中有一个互斥锁变量m_queuelocker和信号量变量m_queuestat。在线程操作list请求队列时，作为公共资源，需要加互斥锁去访问。另外，信号量用来标记工作队列中是否还有待处理的请求，当没有请求时，让线程池中的线程阻塞在信号量处。

4.4 日志功能模块

为了服务器的运维和后期检查，本Web服务程序提供了日志功能，将HTTP请求报文数据和HTTP响应报文数据输出到信息日志message.log中。如果服务程序运行过程中出现异常错误，将这些错误输出到告警日志warning.log中。下面关于log日志模块输出日志的

实现代码做简单分析。

```
int msglog(int mtype, const char *outfmt, ...)
{
        time_t now_time;
        va_list ap;
        char logprefix[1024], tmpstring[1024];
        time(&now_time);
        if (mtype & MSG_INFO) {
                strftime(logprefix, sizeof(logprefix), msgdatefmt, localtime(&now_time));
                strcat(logprefix, " ");
                if (msgopt&LOG_PROCNAME) {
                        strcat(logprefix, ident_name);
                        strcat(logprefix, " ");
                }
                if (msgopt&LOG_PID) {
                        sprintf(tmpstring, "[%6d]", getpid());
                        strcat(logprefix, tmpstring);
                }
                fprintf(msgfile, "%s: ", logprefix);
                va_start(ap, outfmt);
                vfprintf(msgfile, outfmt, ap);
                va_end(ap);
                fprintf(msgfile, "\n");
        }
        if (mtype & MSG_WARN) {
                strftime(logprefix, sizeof(logprefix), wandatefmt, localtime(&now_time));
                strcat(logprefix, " ");
                if (wanopt & LOG_PROCNAME) {
                        strcat(logprefix, ident_name);
                        strcat(logprefix, " ");
```

```
            }
            if (wanopt & LOG_PID) {
                sprintf(tmpstring, "[%6d]", getpid());
                strcat(logprefix, tmpstring);
            }
            fprintf(wanfile, "%s: ", logprefix);
            va_start(ap, outfmt);
            vfprintf(wanfile, outfmt, ap);
            va_end(ap);
            fprintf(wanfile, "\n");
            if (wanopt & LOG_PERROR) {
                fprintf(stderr, "%s: ", logprefix);
                va_start(ap, outfmt);
                vfprintf(stderr, outfmt, ap);
                va_end(ap);
                fprintf(stderr, "\n");
            }
        }
        return 0;
    }
```

上面这段代码为日志输出函数的实现代码，将日志信息输出到日志文件中。其实现为先判断日志级别，是普通信息日志（MSG_INFO）还是告警日志（MSG_WARN），然后在判断是否输出进程名（LOG_PROCNAME），是否输出进程的PID（LOG_PID）。最后按照日志的级别，将日志输出到不同的日志文件message.log和warning.log中。

4.5 配置文件功能模块

为了让程序更加灵活，我们将常用的参数提取出来，写到配置文件中，便于用户在不同环境下的做出不同的调整。本服务程序使用JSON格式作为配置文件的格式化数据存储方式，在主程序中对配置文件进行去读取解析，然后程序运行时使用这些参数信息作为程序的参数。本程序中使用C/C++下的微型解释器程序库cJSON库，调用cJSON库函数对

JSON配置文件进行解析。下面是配置文件内容和读取解析配置文件的代码实现。

下面将展示本Web服务器程序JSON格式的配置文件，配置文件内容如下。

```
{
    "server" : {
        "IP" : "192.168.2.121",
        "Port" : "9999",
        "Path" : "/var/www/html",
        "PthreadNum" : "8"
    },
    "log" : {
        "MessageLog" : "/var/www/log/message.log",
        "WarningLog" : "/var/www/log/warning.log"
    }
}
```

使用配置文件时，程序运行的第一步肯定是打开并读取配置文件的内容，下面是读取配置文件的函数实现。

```
void read_config(char *buf)
{
    FILE *fp = fopen("../conf/HttpServer.conf", "r");
    if (fp == NULL) {
        printf("open HttpServer.conf error!\n");
        return ;
    }
    int i = 0;
    char ch;
    while ((ch = fgetc(fp)) != EOF) {
        buf[i++] = ch;
    }
}
```

读取完JSON格式的配置文件后，需要对该配置文件进行解析，解析配置文件的各个

配置项的值，下面是解析配置文件的函数实现。

```
void analysis_json(char *jsonStr, char *ip, char *port, char *path, char *pthread_num, char *mes_log,
char *war_log)
{
    cJSON *root = NULL;
    cJSON *server = NULL;
    cJSON *log = NULL;
    cJSON *item = NULL;
    root = cJSON_Parse(jsonStr);
    if (!root) {
        printf("Error before: [%s]\n", cJSON_GetErrorPtr());
        return ;
    } else {
        server = cJSON_GetObjectItem(root, "server");
        item = cJSON_GetObjectItem(server, "IP");
        strcpy(ip, cJSON_Print(item));
        sscanf(ip, "\"%[^\"]", ip);
        item = cJSON_GetObjectItem(server, "Port");
        strcpy(port, cJSON_Print(item));
        sscanf(port, "\"%[^\"]", port);
        item = cJSON_GetObjectItem(server, "Path");
        strcpy(path, cJSON_Print(item));
        sscanf(path, "\"%[^\"]", path);
        item = cJSON_GetObjectItem(server, "PthreadNum");
        strcpy(pthread_num, cJSON_Print(item));
        sscanf(pthread_num, "\"%[^\"]", pthread_num);
        log = cJSON_GetObjectItem(root, "log");
        item = cJSON_GetObjectItem(log, "MessageLog");
        strcpy(mes_log, cJSON_Print(item));
        sscanf(mes_log, "\"%[^\"]", mes_log);
```

```
            item = cJSON_GetObjectItem(log, "WarningLog");
            strcpy(war_log, cJSON_Print(item));
            sscanf(war_log, "\"%[^\"]", war_log);
        }
    }
```

上面展示了配置文件的内容、读取配置文件和解析配置文件的函数实现。首先读取配置文件函数read_config很简单，打开配置文件，对其进行循环读操作，将其数据读入指定的用户缓冲区中。然后解析配置文件函数analysis_json也很简单，先创建根、子节点和键值元素的指针，然后使用cJSON库函数cJSON_Parse解析json数据获取根，然后根据其子节点的键，拿到子节点位置，最后根据元素的键去获取值的信息。最后使用sscanf函数去掉信息字符串两端的引号。

另外，由于JSON格式的配置文件无法写入注释，所以详细注释请见Readme.txt文件中。

4.6 Makefile

在一般的Linux下，如果是使用源码安装的程序情，都会存在Makefile，对源码进行编译。Makefile是一种脚本，用来管理项目工程文件的。当程序仅有简单的一两个源文件的情况下，或许还可以使用gcc或g++手动进行编译，但是当项目庞大到一定程度时，在手动使用gcc或g++进行编译显然是不现实的，所以就有了Makefile的出现，来实现自动化编译程序。下面就来简单介绍一下Makefile的一般格式和本程序中的Makefile源码。

Makefile文件有者固定的编写格式，Makefile的一般格式如下。

```
目标 ： 依赖条件
    （前面必须有 tab 缩进）命令
```

Makefile中有三个常用的自动变量，如下。

```
$<: 规则中的第一个依赖(:后面的第一个依赖)
$@: 规则中的目标
$^: 规则中的所有依赖，组成一个列表, 以空格隔开, 如果这个列表中有重复的项
```

Makefile中有两个常用的函数，如下。

```
wildcard 函数：
例：src=$(wildcard ~/test/*.c)              查找指定目录下，指定类型的文件
patsubst 函数：
```

```
例：obj=$(patsubst %.c,%.o,$(src))          匹配替换函数
```

上面大概讲述了Make file的一般格式和常用的函数和自动变量，下面展示本程序中使用的Makefile的内容，如下。

```
LDFLAGS = -lpthread -lm
CPPFLAGS = -I ./include
SRCDIR = ./src
OBJDIR = ./obj
src = $(wildcard $(SRCDIR)/*.cpp)
obj = $(patsubst $(SRCDIR)/%.cpp, $(OBJDIR)/%.o, $(src))
target = ./bin/HttpServer
$(target):$(obj)
    g++ $(obj) -o $@ $(CPPFLAGS) $(LDFLAGS)
$(OBJDIR)/%.o:$(SRCDIR)/%.cpp
    g++ -c $< -o $@ $(CPPFLAGS) $(LDFLAGS)
.PHONY:clean
clean:
    rm -f $(target) $(obj)
```

上面展示了Makefile的源码内容，接下来对Makefile的源码进行分析。首先是设置链接库，链接头文件，源文件路径，对象文件路径等自动变量信息，然后使用wildcard查找函数获取全部的.cpp文件信息，使用patsubst匹配替换函数获取.o文件信息，以便后面编译时使用，设置目标文件名。准备任务完成，接着就是Makefile的一般格式，根据依赖条件，通过命令生成目标文件。最后的.PHONY:clean为声明的一个伪目标，用来删除Makefile编译产生的文件，重置环境，以便重新进行编译。

4.7 进程管理工具

本Web服务程序为了提高用户使用的舒适度，提供了一些工具供用户对进程进行管理使用。其是一个bash脚本，在支持使用bash解释器的Linux操作系统下可以直接使用。进程管理工具tool.sh脚本中包含两个主要的辅助函数：检查HttpServer进程是否正在运行、检查文件./../bin/HttpServer文件是否存在，和四个由主函数调用的函数。下面分别对其源代码中的函数进行简单的分析。

下面这些为tool.sh脚本中使用的几个本地变量，如下。

```
FILENAME="HttpServer"
FILEPATH="../bin/HttpServer"
RUNFILE="./../bin/HttpServer"
```

下面列出tool.sh脚本中检查HttpServer进程是否存在的函数，如下。

```
is_run()
{
    ps_out=$(ps -ef | grep $FILENAME | grep -v 'grep')
    result=$(echo $ps_out | grep "$FILENAME")
    if [[ "$result" != "" ]]; then
        return 0
    else
    return 1
    fi
}
```

这里对上面检查HttpServer进程是否存在的函数进行分析讲解。该函数主要使用Linux下的命令ps -ef | grep $FILENAME | grep -v ‘grep’来实现判断进程是否在运行。ps -ef为系统命令，输出所有系统运行的进程，然后通过管道作为grep的输入，通过grep工具判断是否拥有$FILENAME（即HttpServer，$的作用为取出变量的值）的那条结果，grep -v grep表示默认不处理使用grep命令的那个进程输出。

下面列出tool.sh脚本中检查文件../bin/HttpServer文件是否存在的函数，如下。

```
is_exist()
{
    if [ -x $FILEPATH ]; then
        return 0
    else
        return 1
    fi
}
```

这里对上面检查文件../bin/HttpServer文件是否存在的函数进行分析。该函数主要使用

[] 命令，来判断$FILEPATH（即../../bin/HttpServer）文件是否存在，其实也可以使用test命令实现同样的效果。

下面列出tool.sh脚本中检查HttpServer服务运行状态的函数，如下。

```
state_process()
{
    if is_run; then
        echo "HttpServer state: Activity"
    else
        echo "HttpServer state: Termination"
    fi
}
```

这里对上面函数进行分析，该函数主要根据is_run()函数的结果去打印出进程状态。

下面列出tool.sh脚本中启动HttpServer服务的函数，如下。

```
start_process()
{
    if is_run; then
        echo "HttpServer is running ..."
    else
        if is_exist; then
            $($RUNFILE)
            if is_run; then
                echo "Start HttpServer Sucess !!!"
            else
                echo "Start HttpServer Fail !!!"
            fi
        else
            echo "First, please use Makefile to build HttpServer !!!"
        fi
    fi
}
```

这里对启动HttpServer服务的函数进行分析，该函数先判断进程的运行状态，如果不在运行，则判断程序文件是否存在，然后通过$()去启动进程（或者``也可以）。

下面列出tool.sh脚本中，停止HttpServer服务的函数，如下。

```
stop_process()
{
    PROCESS=$(ps -ef | grep "$FILENAME" | grep -v grep | grep -v PPID | awk '{ print $2}')
    for i in $PROCESS
    do
        kill -9 $i
    done
    if is_run; then
        echo "Stop HttpServer Fail !!!"
    else
        echo "Stop HttpServer Sucess !!!"
    fi
}
```

这里对停止HttpServer服务的函数进行分析，该函数通过ps -ef和grep命令找到指定程序名的进程，然后通过awk工具命令找到其进程对应的PID，然后通过kill -9 命令去杀死指定PID的进程。

下面列出tool.sh脚本中，重新启动HttpServer服务的函数，如下。

```
restart_process()
{
    stop_process
    sleep 3
    start_process
}
```

这里对重启HttpServer服务的函数进行分析，该函数其实关闭进程后等待三秒然后重启。

在本tool.sh脚本中，主函数通过用户传入参数值的比较，执行不同得操作。下面列出tool.sh脚本中的主函数，如下。

```
if [ $# -eq 0 ];then
        echo "使用格式："
        echo "./tool Option"
        echo "Option:"
        echo "    state -  查看 HttpServer 服务运行状态"
        echo "    start -  启动 HttpServer 服务"
        echo "    restart -  重启 HttpServer 服务"
        echo "    stop -  停止 HttpServer 服务"
        exit 0
fi
case $1 in
        state)
                state_process
                ;;
        start)
                start_process
                ;;
        restart)
                restart_process
                ;;
        stop)
                stop_process
                ;;
esac
```

这里对的主函数进行分析。主函数通过拿到用户传入的参数，然后在程序中对其进行参数值对比，根据匹配值去执行不同的函数调用，从而执行不同的操作。需要注意的是，在使用tool.sh脚本时，需要跟一个参数，主程序通过该参数的值来判断要执行的操作。在没有跟参数时会打印该脚本使用格式以及各个参数值所代表的含义。如果在使用tool.sh脚本工具时，后面跟了多个参数，这时程序只会默认的使用第一个参数进行操作，其他参数将不会起到任何作用。

4.8 本章小结

本章主要是对基于 HTTP 协议的 Web 服务器程序的整体架构实现做了一个详细的介绍，对于本 Web 服务器程序的各个模块的实现的核心代码也做出展示，并且对这些核心的代码做了简单的代码分析。另外，对于每一个模块所涉及的技术点，也做了粗略的分析介绍。

第 5 章 Web服务程序测试

对于一个公司来说，一个项目完成之后上线之前，都会由测试人员进行大量的测试，找出程序中的BUG，在基本保证程序没有什么大的问题时，才会让项目上线，所以对于一个项目来说，进行测试是必须的。

5.1 测试目的

测试是一个项目开发完成上线前的最后一个必不可少的步骤。它的目的主要在于发现项目中的不足，以及存在的BUG，如果没有测试环节，万一项目中存在重大BUG那么在上线之后将会产生巨大的损失。另外，测试也可以检验我们最终开发出来的项目是否符合我们的预期，以免产生开发完成的项目跟目标系统不一致的尴尬。所以对本Web服务程序的测试是必不可少的。

5.2 测试方案

在进行程序测试之前，我们应该先分析一下程序使用操作和运行流程。因为本程序为一个Web服务程序，目的是为浏览器提供静态请求访问的，因此测试需要为该Web服务站点提供一些必须的静态访问资源，所以本程序在服务器站点下挂在了一些静态的html页面，主页为index.html。页面挂在的路径为服务程序配置文件Path字段中指定的路径。有了这些静态html文件后，我们先登录服务器，编译代码文件生成我们的服务器程序HttpServer，然后使用工具tool.sh，通过命令./tool.sh start启动进程，接着我们就可以打开本地的浏览器或者统一局域网下其他PC机下的浏览器（这样必须保证服务器下的指定端口是允许其他机器访问的），在其地址栏输入服务器IP和对应服务程序的端口号以及请求的静态页面，例如：192.168.2.121:9999/index.html。浏览器会根据地址栏中的请求内容，给指定的服务器发送请求数据包，服务器上的服务程序收到该数据包后，通过程序中的有限状态自动机解析出请求数据，保留一份数据写入正常日志文件message.log，然后根据请求的内容判断请求的文件是否存在，存在的话服务程序组织响应报文，将请求的文件数据发送给客户端，同时也保留一份数据写入正常日志文件message.log。如果中途程序出现BUG，会将错误信息写入到告警日志文件warning.log中。

这样，如果服务程序正常，我们将可以看到浏览器收到请求的html页面，并将其展示在浏览器上。另外，在服务器上的正常日志文件message.log中，也会有数据信息写入，我们可以查看分析日志文件中的内容，根据日志内容信息，判断程序是否正常执行成功。测试方案如表 5-1 所示。

表 5-1 测试方案表

测试功能	测试要求说明	测试结果
程序Makefile编译测试	1. 登录远程服务器，将本服务程序压缩包解压到自定义目录。	测试成功
	2. 进入程序文件目录中，找到Readme.txt，并阅读。	
	3. 在conf/目录下找到HttpServer.conf文件，并修改保存配置项，确保配置项中的路径有效。	
	4. 将静态html测试文件放入到配置文件中指定的路径下。	
	5. 使用Makefile进行程序编译。	
进程管理工具tool.sh使用测试	1. 进入/tools目录下，使用./tool.sh start命令进行HttpServer程序启动。	测试成功
	2. 使用./tool.sh state命令查看HttpServer进程是否启动成功，成功则显示HttpServer state: Activity。	
	3. 使用./tool.sh stop关闭HttpServer进程。	
	4. 再次使用./tool.sh state命令查看HttpServer进程是否关闭成功，成功则显示HttpServer state: Termination。	
浏览器访问测试	1. 使用tool.sh工具将HttpServer进程启动。	测试成功
	2. 打开PC端浏览器，在地址栏输入IP:端口/index.html。这里的IP和端口是你在配置文件中指定的。并且要确保所在的Linux服务器的端口是开放可以访问的。成功则显示页面。	
日志	1. 进入日志目录，看日志文件是否生成。	测试成功
	2. 打开日志文件，看日志内容是否正确写入。	

5.3 测试结果

在PC上打开浏览器，在其地址栏输入IP:端口/index.html或者在本地hosts文件中根据指定格式，自己编写一个域名指定其对应IP和端口为要访问的服务器的IP和端口。这样就可以在地址栏中输入域名/index.html这样的格式来访问了。其原理是根据DNS域名解析流程，DNS解析先读取本地hosts文件，如果能找到对应域名，就直接根据这个域名拿到解析的IP，如果不存在，会去递归查询本地指定的DNS服务器，如果本地指定的DNS服务器上的区文件中不存在该域名，则本地指定的DNS服务器会迭代查询根域名服务器，然后逐步找到查询的域名对应的IP。这里有点跑题了，关于DNS解析如果想了解更多，自己去查询资料。在浏览器上进行请求后，如果浏览器中成功显示了请求到的界面，表示浏览器测

试成功。如图 5-1 所示。

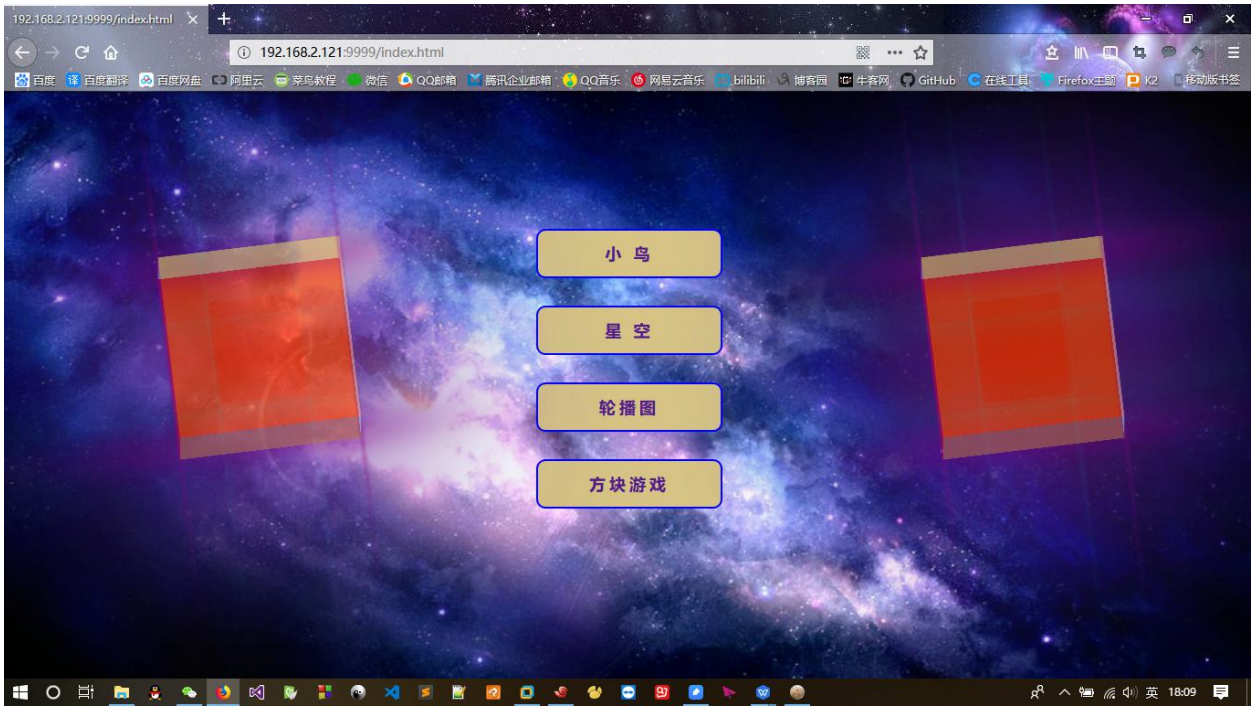

图 5-1 浏览器测试成功截图

然后，登录服务器，找到日志文件，打开正常日志文件message.log，查看日志信息，根据日志信息来判断服务程序是否正常。如图 5-2 所示。

```
1139
1140 ================== Http request ==================
1141
1142 05-02 18:09:13 HttpServer [ 61245]: GET /xingkong.css HTTP/1.1
1143 05-02 18:09:13 HttpServer [ 61245]: Host: 192.168.2.121:9999
1144 05-02 18:09:13 HttpServer [ 61245]: User-Agent: Mozilla/5.0 (Windows NT 10.0; Win6
     4; x64; rv:66.0) Gecko/20100101 Firefox/66.0
1145 05-02 18:09:13 HttpServer [ 61245]: Accept: text/css,*/*;q=0.1
1146 05-02 18:09:13 HttpServer [ 61245]: Accept-Language: zh-CN,zh;q=0.8,zh-TW;q=0.7,zh
     -HK;q=0.5,en-US;q=0.3,en;q=0.2
1147 05-02 18:09:13 HttpServer [ 61245]: Accept-Encoding: gzip, deflate
1148 05-02 18:09:13 HttpServer [ 61245]: Referer: http://192.168.2.121:9999/index.html
1149 05-02 18:09:13 HttpServer [ 61245]: Connection: keep-alive
1150 05-02 18:09:13 HttpServer [ 61245]: Cookie: Hm_lvt_bfc6c23974fbad0bbfed25f88a973fb
     0=1555210680,1555605040,1555855845,1556206559
1151 05-02 18:09:13 HttpServer [ 61245]:
1152 05-02 18:09:13 HttpServer [ 61245]:
1153
1154 ================== Http response ==================
1155
1156 05-02 18:09:13 HttpServer [ 61245]: HTTP/1.1 200 OK^M
1157 Content-Length: 4173^M
1158 Content-Type: text/css^M
1159 Connection: keep-alive^M
1160 ^M
```

图 5-2 日志结果截图

经上面的全面测试，初步确认程序没有重大BUG，各项功能也都可以正常运行。虽

然个别地方可能还不是很完美，但是基本上可以判断该项目已经可以让用户正常使用了，初步到达了本服务程序设计的预期要求。

5.4 本章小结

本章主要是对整个项目的行测试分析，测试方案的分析，以及实战测试的结果展示。根据整个项目的测试流程对项目的基本功能做了比较完善的测试，保证了项目基本不存在重大的BUG问题。

结　论

在这次毕业设计的前期准备之中，我了解到，当今世界互联网技术的飞速发展，网民数量的极速增加，导致了对大流量处理的急切需求。所以，研究高并发、高性能的 Web 服务程序是很有必要的。

在本Web服务程序的开发中，使用了很多技术，比如处理高并发的时候使用线程池技术，处理I/O的时候使用Linux下的epoll系列系统调用实现多路I/O复用。对HTTP请求进行解析的时候，使用高效率的有限状态自动机模型，在传输请求的文件数据的时候，为了提高效率，使用mmap内存映射，避免数据在硬盘和内存之间进行多次拷贝，在进程管理工具编写的时候使用bash脚本，使用cJSON函数库去解析JSON格式的配置文件，Makefile编写格式、输出日志功能模块实现等等。

另外，我也发现了自己的技术能力的不足，好多功能实现所需要的技术都是现用现学的，比如cJSON库的使用、Makefile编写时一些自带函数的使用等等。通过本次的开发我充分的认识到了知道了自己有很多不足之处。在对本服务程序进行需求分析的时候，由于是根据自己对Web服务程序的功能分析和查阅资料得来的，所以在开发的过程之中，也有一些问题出现在了的设计和开发的过程之中。发现自己对操作系统内核的知识还是有很多欠缺之处，在今后的学习生活中我也会尽力去补充自己的短板之处。因为我们毕业设计作品的总体时间有限，所有我只是对Web服务程序解析HTTP请求的静态GET请求作了实现，还有很多功能有待补充。之后有时间我会对这个程序进行功能的完善和改进，尽量做到跟正常Web服务程序功能相差不会太远。

致　谢

首先我要感谢我的毕业设计指导老师冯贺老师，他是一位和蔼可亲的老师，他再对我进行毕业设计指导的过程中非常耐心，遇到问题时，总是不遗余力的为我进行讲述以及帮助，从来不会因为我的知识缺乏而发怒或者不耐烦。我真的很荣幸能成为您的学生。另外，也非常感谢您为我们讲解论文书写规范和需要注意的事项，您是一位值得我发自内心尊敬的老师。

在这里也感谢在大学中教导过我的老师们，在大学四年中对于你们的教导，我表示真心的感谢。是你们让我从当初由一个对计算机的一无所知的学生，现在变成了一个合格的程序员。虽然中间经历了很多很多困难，但是我也很感谢那些磨难，它们让我学到了许多东西。另外，在这里需要特别感谢它，那就是学校的图书馆，这里是我提升自己，丰富知识的地方。每次在这里我都深深的感受到了知识的力量，仿佛置身于知识的海洋，让我欲罢不能。正是因为这里，让我深深的被计算机技术所吸引，让我接触了很多学校课程中学不到的知识，让我的知识面变得更宽，知识深度也变得更深。感谢这里，它承载了我大学期间最美好最激情最狂热的时光。

最后感谢我的大学的同学们，感谢你们在我的开发过程中提出的宝贵意见，感谢你们的无私的帮助。在这个互联网的时代，希望我们在今后的道路上，都能找到一个自己喜欢的研究方向，不断的提升自己的技术水平，相信在未来我们都将成为一个技术大牛。

参考文献

[1] 游双，Linux 高性能服务器编程[M]，机械工业出版社，2013.5

[2] 理查德·史蒂文斯、拉戈，UNIX 环境高级编程[M]，人民邮电出版社，2006.5

[3] 鸟哥，鸟哥的 Linux 私房菜[M]，人民邮电出版社，2010.7

[4] 德鲁尼尔，Vim 实用技巧[M]，人民邮电出版社，2014.5

[5] 刘遄，Linux 就该这么学[M]，人民邮电出版社，2013.1

[6] 陶辉，深入理解 Nginx：模块开发与架构解析（第 2 版）[M]，机械工业出版社，2016.2

[7] Ellie Quigley，UNIX.shell 范例精解（第 4 版）[M]，清华大学出版社，2007.6

[8] 史蒂芬斯、芬纳，UNIX 网络编程卷 1：套接字联网 API（第 3 版）[M]，人民邮电出版社，2010.5

[9] Stanley B. Lippman，C++Primer（第 5 版）[M]，人民邮电出版社，2013.9

[10] 上野宣，图解 HTTP[M]，人民邮电出版社，2014.1

[11] 陈皓，跟我一起写 Makefile[M]，人民邮电出版社，2016.5

[12] 竹下隆史、村山公保，图解 TCP/IP（第 5 版）[M]，人民邮电出版社，2013.7

[13] Scott Meyers，Effective C++（第 3 版）[M]，电子工业出版社，2006.7

[14] 林锐、韩永泉，高质量程序设计指南-C++/C 语言（第三版）[M]，电子工业出版社，2007.5

[15] James F.Kurose、Keith W.Ross，计算机网络-自顶向下方法[M]，机械工业出版社，2009.6

[16] Dale Dougberty，sed 与 awk（修订第三版）[M]，机械工业出版社，2015.8

[17] 布鲁姆、布雷斯纳汉，Linux 命令行与 shell 脚本编程大全（第 3 版）[M]，人民邮电出版社，2016.8

[18] 博韦、西斯特，深入理解 LINUX 内核（第三版）[M]，中国电力出版社，2007.3

[19] 张天飞，奔跑吧 Linux 内核 入门篇[M]，人民邮电出版社，2019.2

[20] 朱文伟、李建英，Linux C 与 C++ 一线开发实践[M]，清华大学出版社，2018.12

[21] 赵炯，Linux 内核完全剖析——基于 0.12 内核[M]，机械工业出版社，2008.10

[22] Scott Granneman，Linux 命令速查手册（第 2 版）[M]，清华大学出版社，2017.1