# 摘要

WWW(万维网)发展至今已25年有余，其使用之广泛性，在网络世界中有着不可代替的地位。如今，我们使用的大部分网络应用，如web应用，app应用，甚至游戏等，其后台都需要web服务器为数据传输做支撑。更随着用户的与日俱增，web服务器也面临着诸如更高并发，高大规模计算等挑战。所以，对web服务器工作原理，运行机制的深入研究是非常有必要的。

本文通过对TCP/IP到HTTP协议以及Linux I/O模型的研究，对商用web服务器tomcat的分析，采用Java语言设计一款小型web服务器。

关键词：web，Http协议，Java，服务器

# 绪论

## 1.1 研究背景与意义

从1980年[Tim](https://zh.wikipedia.org/wiki/%E8%92%82%E5%A7%86%C2%B7%E4%BC%AF%E7%BA%B3%E6%96%AF-%E6%9D%8E) Berners-Lee构建的ENQUIRE项目至今，万维网已经走过了28年之久。随着网络技术的高速发展，如今每个人都通过互联网获取信息数据。而这些海量数据的传输，交互，背后都是由web服务器作为支撑。在全民互联网时代的今天，web服务器也面临着新的挑战。如天猫双十一几十万每秒的超高并发，以及海量数据的超大计算等对服务器性能的要求也越来越高。而web服务器使用HTTP协议作为通信协议，HTTP又工作在TCP协议之上。所以，对网络协议的了解以及对web服务器整个工作流程以及运行机制的研究是非常有必要的。

## 1.2 课题研究的目的及内容

本文先对从TCP到HTTP的工作原理以及对其协议报文进行概述，了解web服务器的通信协议基础；然后通过对Linux五种I/O模型进行分析对比，了解I/O通信过程中同步与异步，阻塞与非阻塞的概念以及在通信过程中的优劣；再分析商用web服务器tomcat的I/O以及线程模型；再简述Java编程语言以及JVM平台的特点；最后再分析怎么用socket编程开发一款web服务器。本文旨在剖析一款web服务器从网络协议，到操作系统I/O通信，到编码实现阶段所遇到以及要解决的种种问题。

## 1.3 论文组织结构

本文共分六个章节。即第二章介绍web服务器通信中的网络协议；第三章介绍操作系统I/O通信中的问题；第四章服务器架构和编码设计实现；第五章：服务器测试与性能优化。对服务器进行功能和性能上的测试；第六章总结。

# 网络协议基础

## 2.1 OSI网络模型

开放式系统互连通信参考模型(Open System Interconnection Reference Model,缩写为OSI)，简称OSI模型(OSI Model)，一种概念模型，有国际化标准组织(ISO)提出，一种试图使各种计算机在世界范围内互联为网络的标准框架。

#### 2.1.1 OSI七层模型划分

1983年，ISO发布了著名的**ISO/IEC 7489**标准，它定义了网络互连的7层框架，也就是开放式系统互连参考模型。如下表：

|  |  |  |  |
| --- | --- | --- | --- |
| **OSI 模型** | | | |
|  | **数据单元** | **层** | **功能** |  |
| **主机层** | 数据(Data) | 7. 应用层 | 网络进程到应用程序。针对特定应用规定各层协议、时序、表示等，进行封装 。在端系统中用软件来实现，如HTTP等 |  |
| 6.表示层 | 数据表示形式，加密和解密，把机器相关的数据转换成独立于机器的数据。规定数据的格式化表示 ，数据格式的转换等 |  |
| 5.会话层 | 主机间通讯，管理应用程序之间的会话。规定通信时序 ；数据交换的定界、同步，创建检查点等 |  |
| 数据段(Segment) | 4.传输层 | 在网络的各个节点之间可靠地分发数据包。所有传输遗留问题；复用；流量；可靠 |  |
| **媒介层** | 数据包(Packet) | 3.网络层 | 在网络的各个节点之间进行地址分配、路由和（不一定可靠的）分发报文。路由（ IP寻址）；拥塞控制。 |  |
| 数据帧(Frame) | 2. 数据链路层 | 一个可靠的点对点数据直链。检错与纠错（CRC码）；多路访问；寻址 |  |
| 比特(Bit) | 1. 物理层 | 一个（不一定可靠的）点对点数据直链。定义机械特性；电气特性；功能特性；过程特性 |  |

#### 2.1.2各层功能定义

这里我们只对OSI各层进行功能上的大概阐述，不详细深究，因为每一层实际都是一个复杂的层。而我们所关心，并且和服务器密切相关的也只在传输层和应用层。

应用层

OSI参考模型中最靠近用户的一层，是为计算机用户提供应用接口，也为用户直接提供各种网络服务。我们常见应用层的网络服务协议有：HTTP，HTTPS，FTP，POP3、SMTP等。

表示层

表示层提供各种用于应用层数据的编码和转换功能,确保一个系统的应用层发送的数据能被另一个系统的应用层识别。如果必要，该层可提供一种标准表示形式，用于将计算机内部的多种数据格式转换成通信中采用的标准表示形式。数据压缩和加密也是表示层可提供的转换功能之一。

会话层

会话层就是负责建立、管理和终止表示层实体之间的通信会话。该层的通信由不同设备中的应用程序之间的服务请求和响应组成。

传输层

传输层建立了主机端到端的链接，传输层的作用是为上层协议提供端到端的可靠和透明的数据传输服务，包括处理差错控制和流量控制等问题。该层向高层屏蔽了下层数据通信的细节，使高层用户看到的只是在两个传输实体间的一条主机到主机的、可由用户控制和设定的、可靠的数据通路。我们通常说的，TCP UDP就是在这一层。端口号既是这里的“端”。

网络层

本层通过IP寻址来建立两个节点之间的连接，为源端的运输层送来的分组，选择合适的路由和交换节点，正确无误地按照地址传送给目的端的运输层。就是通常说的IP层。这一层就是我们经常说的IP协议层。IP协议是Internet的基础。

数据链路层

将比特组合成字节,再将字节组合成帧,使用链路层地址 (以太网使用MAC地址)来访问介质,并进行差错检测。

数据链路层又分为2个子层：逻辑链路控制子层（LLC）和媒体访问控制子层（MAC）。MAC子层处理CSMA/CD算法、数据出错校验、成帧等；LLC子层定义了一些字段使上次协议能共享数据链路层。 在实际使用中，LLC子层并非必需的。

物理层

实际最终信号的传输是通过物理层实现的。通过物理介质传输比特流。规定了电平、速度和电缆针脚。常用设备有（各种物理设备）集线器、中继器、调制解调器、网线、双绞线、同轴电缆。这些都是物理层的传输介质。

## 2.2 TCP/IP协议

互联网协议族（英语：Internet Protocol Suite，缩写IPS）[1]是一个网络通信模型，以及一整个网络传输协议家族，为互联网的基础通信架构。它常被通称为TCP/IP协议族（英语：TCP/IP Protocol Suite，或TCP/IP Protocols），简称TCP/IP[2]。因为该协议家族的两个核心协议：TCP（传输控制协议）和IP（网际协议），为该家族中最早通过的标准[3]。由于在网络通讯协议普遍采用分层的结构，当多个层次的协议共同工作时，类似计算机科学中的堆栈，因此又被称为TCP/IP协议栈（英语：TCP/IP Protocol Stack）[4][5] 。这些协议最早发源于美国国防部（缩写为DoD）的ARPA网项目，因此也被称作DoD模型（DoD Model）[6]。这个协议族由互联网工程任务组负责维护。

### 2.2.1 TCP协议定义

传输控制协议（英语：Transmission Control Protocol，缩写为 TCP）是一种面向连接的、可靠的、基于字节流的传输层通信协议，由IETF的RFC 793定义。在简化的计算机网络OSI模型中，它完成第四层传输层所指定的功能，用户数据包协议（UDP）是同一层内另一个重要的传输协议。

在因特网协议族（Internet protocol suite）中，TCP层是位于IP层之上，应用层之下的中间层。不同主机的应用层之间经常需要可靠的、像管道一样的连接，但是IP层不提供这样的流机制，而是提供不可靠的包交换。

应用层向TCP层发送用于网间传输的、用8位字节表示的数据流，然后TCP把数据流分区成适当长度的报文段（通常受该计算机连接的网络的数据链路层的最大传输单元（MTU）的限制）。之后TCP把结果包传给IP层，由它来通过网络将包传送给接收端实体的TCP层。TCP为了保证不发生丢包，就给每个包一个序号，同时序号也保证了传送到接收端实体的包的按序接收。然后接收端实体对已成功收到的包发回一个相应的确认（ACK）；如果发送端实体在合理的往返时延（RTT）内未收到确认，那么对应的数据包就被假设为已丢失将会被进行重传。TCP用一个校验和函数来检验数据是否有错误；在发送和接收时都要计算校验和。

### 2.2.2 TCP报文格式

下表为TCP数据包结构：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **偏移** | **位0–3** | **4–7** | **8–15** | **16–31** |
| **0** | 来源连接端口 | | | 目的连接端口 |
| **32** | 序列号码 | | | |
| **64** | 确认号码 | | | |
| **96** | 报头长度 | 保留 | 标志符 | 窗口大小 |
| **128** | 校验和 | | | 紧急指针 |
| **160** | 选项字段 | | | |
| **160/192+** | 数据 | | | |

* 来源连接端口（16位长）－辨识发送连接端口
* 目的连接端口（16位长）－辨识接收连接端口
* 序列号（seq，32位长）
  + 如果含有同步化旗标（SYN），则此为最初的序列号；第一个数据比特的序列码为本序列号加一。
  + 如果没有同步化旗标（SYN），则此为第一个数据比特的序列码。
* 确认号（ack，32位长）—期望收到的数据的开始序列号。也即已经收到的数据的字节长度加1。
* 报头长度（4位长）—以4字节为单位计算出的数据段开始地址的偏移值。
* 保留—须置0
* 标志符
  + URG—为1表示高优先级数据包，紧急指针字段有效。
  + ACK—为1表示确认号字段有效
  + PSH—为1表示是带有PUSH标志的数据，指示接收方应该尽快将这个报文段交给应用层而不用等待缓冲区装满。
  + RST—为1表示出现严重差错。可能需要重现创建TCP连接。还可以用于拒绝非法的报文段和拒绝连接请求。
  + SYN—为1表示这是连接请求或是连接接受请求，用于创建连接和使顺序号同步
  + FIN—为1表示发送方没有数据要传输了，要求释放连接。
* 窗口（WIN，16位长）—表示从确认号开始，本报文的接受方可以接收的字节数，即接收窗口大小。用于流量控制。
* 校验和（Checksum，16位长）—对整个的TCP报文段，包括TCP头部和TCP数据，以16位字进行计算所得。这是一个强制性的字段。
* 紧急指针（16位长）—本报文段中的紧急数据的最后一个字节的序号。
* 选项字段—最多40字节。每个选项的开始是1字节的kind字段，说明选项的类型。
  + 0：选项表结束（1字节）
  + 1：无操作（1字节）用于选项字段之间的字边界对齐。
  + 2：最大报文段长度（4字节，Maximum Segment Size，MSS）通常在创建连接而设置SYN标志的数据包中指明这个选项，指明本端所能接收的最大长度的报文段。通常将MSS设置为（MTU-40）字节，携带TCP报文段的IP数据报的长度就不会超过MTU，从而避免本机发生IP分片。只能出现在同步报文段中，否则将被忽略。
  + 3：窗口扩大因子（4字节，wscale），取值0-14。用来把TCP的窗口的值左移的位数。只能出现在同步报文段中，否则将被忽略。这是因为现在的TCP接收数据缓冲区（接收窗口）的长度通常大于65535字节。
  + 4：sackOK—发送端支持并同意使用SACK选项。
  + 5：SACK实际工作的选项。
  + 8：时间戳（10字节，TCP Timestamps Option，TSopt）
    - 发送端的时间戳（Timestamp Value field，TSval，4字节）
    - 时间戳回显应答（Timestamp Echo Reply field，TSecr，4字节）

### 2.2.3 TCP协议通信流程

### 2.2.4 TCP协议通信过程中的问题

## 2.3 HTTP协议

超文本传输协议（HTTP，HyperText Transfer Protocol)是互联网上应用最为广泛的一种网络协议。所有的WWW文件都必须遵守这个标准。设计HTTP最初的目的是为了提供一种发布和接收HTML页面的方法。1960年美国人Ted Nelson构思了一种通过计算机处理文本信息的方法，并称之为超文本（hypertext）,这成为了HTTP超文本传输协议标准架构的发展根基。Ted Nelson组织协调万维网协会（World Wide Web Consortium）和互联网工程工作小组（Internet Engineering Task Force ）共同合作研究，最终发布了一系列的RFC，其中著名的RFC 2616定义了HTTP 1.1。

### 2.3.1 HTTP协议定义

### 2.3.2 HTTP协议报文格式

### 2.3.3 HTTP协议通信流程

### 2.3.4 HTTPS以及webSocket

# I/O模型与多线程编程

## 3.1 Linux I/O模型分析

一：select模型

服务器的几个主要动作如下：

创建监听套接字，绑定，监听；

创建工作者线程；

创建一个套接字数组，用来存放当前所有活动的客户端套接字，每accept一个连接就更新一次数组；

接受客户端的连接。

二：WSAAsyncSelect 异步选择模型

服务器的几个主要动作如下：

在WM\_CREATE消息处理函数中，初始化Windows Socket library，创建监听套接字，绑定，监听，并且调用WSAAsyncSelect函数表示我们关心在监听套接字上发生的FD\_ACCEPT事件；

自定义一个消息WM\_SOCKET，一旦在我们所关心的套接字（监听套接字和客户端套接字）上发生了某个事件，系统就会调用WndProc并且message参数被设置为WM\_SOCKET；

在WM\_SOCKET的消息处理函数中，分别对FD\_ACCEPT、FD\_READ和FD\_CLOSE事件进行处理；

在窗口销毁消息(WM\_DESTROY)的处理函数中，我们关闭监听套接字，清除Windows Socket library

特点：需要建立一个窗口用于接收消息。

三：WSAEventSelect 模型

服务器的几个主要动作如下：

感兴趣的一组网络事件创建一个事件对象，再调用WSAEventSelect函数将网络事件和事件对象关联起来。

当网络事件发生时，winsock使响应的事件对象受信，在事件对象上等待的函数就会立即返回。

调用WSAEnumNetworkEvents函数便可获得到底发生了什么网络事件（FD\_READ/FD\_ACCEPT/FD\_CLOSE等等）。

特点：最多可以支持WSA\_MAXIMUM\_WAIT\_EVENTS个对象,他的大小是64。

四：Overlapped I/O 事件通知模型

Overlapped I/O 事件通知模型和WSAEventSelect模型在实现上非常相似，主要区别在”Overlapped”，Overlapped 模型是让应用程序使用重叠数据结构(WSAOVERLAPPED)，一次投递一个或多个 Winsock I/O请求。这些提交的请求完成后，应用程序会收到通知。什么意思呢？就是说，如果你想从 socket上接收数据，只需要告诉系统，由系统为你接收数据，而你需要做的只是为系统提供一个缓冲区 。

五：Overlapped I/O 完成例程模型

Overlapped I/O 完成例程要求用户提供一个回调函数，发生新的网络事件的时候系统将执行这个函数。

特点：由I/O来完成socket的拆包工作，实现异步。

六：IOCP 完成端口模型

特点：动用一个合理数量的线程来接受信息，然后把信息投送到应用程序。

“Windows NT小组注意到这些应用程序的性能没有预料的那么高。特别的，处理很多同时的客户请求意味着很多线程并发地运行在系统中。因为所有这些线程都是可运行的 [没有被挂起和等待发生什么事]， Microsoft意识到NT内核花费了太多的时间来转换运行线程的上下文[Context]，线程就没有得到很多CPU时间来做它们的工作。大家可能也都感觉到并行模型的瓶颈在于它为每一个客户请求都创建了一个新线程。创建线程比起创建进程开销要小，但也远不是没有开销的。我们不妨设想一下：如果事先开好 N个线程，让它们在那hold[堵塞 ]，然后可以将所有用户的请求都投递到一个消息队列中去。然后那N 个线程逐一从消息队列中去取出消息并加以处理。就可以避免针对每一个用户请求都开线程。不仅减少了线程的资源，也提高了线程的利用率。理论上很不错，你想我等泛泛之辈都能想出来的问题， Microsoft又怎会没有考虑到呢?”—– 摘自nonocast的《理解I/O Completion Port》

## 3.2 Tomcat I/O模型与线程模型分析

一个或多个Acceptor线程，每个线程都有自己的Selector，Acceptor只负责accept新的连接，一旦连接建立之后就将连接注册到其他Worker线程中

多个Worker线程，有时候也叫IO线程，就是专门负责IO读写的。一种实现方式就是像Netty一样，每个Worker线程都有自己的Selector，可以负责多个连接的IO读写事件，每个连接归属于某个线程。另一种方式实现方式就是有专门的线程负责IO事件监听，这些线程有自己的Selector，一旦监听到有IO读写事件，并不是像第一种实现方式那样（自己去执行IO操作），而是将IO操作封装成一个Runnable交给Worker线程池来执行，这种情况每个连接可能会被多个线程同时操作，相比第一种并发性提高了，但是也可能引来多线程问题，在处理上要更加谨慎些。tomcat的NIO模型就是第二种。

## 3.3 I/O模型与线程模型的设计与实现

本服务器采用多路复用I/O模型，以及Java NIO语言的实现

# Web服务器设计与实现

## 4.1 Java语言与JVM平台概述

Java是一门面向对象编程语言，不仅吸收了C++语言的各种优点，还摒弃了C++里难以理解的多继承、指针等概念，因此Java语言具有功能强大和简单易用两个特征。Java语言作为静态面向对象编程语言的代表，极好地实现了面向对象理论，允许程序员以优雅的思维方式进行复杂的编程[1] 。

Java具有简单性、面向对象、分布式、健壮性、安全性、平台独立与可移植性、多线程、动态性等特点[2] 。Java可以编写桌面应用程序、Web应用程序、分布式系统和嵌入式系统应用程序等。

JVM是Java Virtual Machine（Java[虚拟机](https://baike.baidu.com/item/%E8%99%9A%E6%8B%9F%E6%9C%BA)）的缩写，JVM是一种用于计算设备的规范，它是一个虚构出来的计算机，是通过在实际的计算机上仿真模拟各种计算机功能来实现的。

[Java语言](https://baike.baidu.com/item/Java%E8%AF%AD%E8%A8%80)的一个非常重要的特点就是与平台的无关性。而使用Java虚拟机是实现这一特点的关键。一般的高级语言如果要在不同的平台上运行，至少需要编译成不同的[目标代码](https://baike.baidu.com/item/%E7%9B%AE%E6%A0%87%E4%BB%A3%E7%A0%81)。而引入Java语言虚拟机后，Java语言在不同平台上运行时不需要重新编译。Java语言使用Java虚拟机屏蔽了与具体平台相关的信息，使得Java语言[编译程序](https://baike.baidu.com/item/%E7%BC%96%E8%AF%91%E7%A8%8B%E5%BA%8F)只需生成在Java虚拟机上运行的目标代码（[字节码](https://baike.baidu.com/item/%E5%AD%97%E8%8A%82%E7%A0%81)），就可以在多种平台上不加修改地运行。Java虚拟机在执行字节码时，把字节码解释成具体平台上的[机器指令](https://baike.baidu.com/item/%E6%9C%BA%E5%99%A8%E6%8C%87%E4%BB%A4)执行。这就是Java的能够“一次编译，到处运行”的原因。

## 4.2 服务器整体架构

## 4.3 服务器编码与实现

# 服务器测试与性能优化

## 5.1黑盒测试

## 5.2性能优化

# 总结

## 6.1 总结

毕业设计最初的时候，主要是对相关资料的收集和理论知识的学习。在这个阶段，最好是同时结合资料和源码一起来看，效果会比较好，学习效率较高。看资料和教程是从细节和基础上去学习知识，而看相关程序的源码则是从整体和实现上去了解一个系统。这样才能做到“见树又见林”。学习理论知识可以使我们掌握最基础的知识，能更深入的了解设计的底层实现。当在具体实现的时候，可以以模块或分层次的思想来分析系统。重点掌握核心的模块，其他模块可以采用现有的类库或开源的实现，这样可以提高开发的效率。软件开发其实对于代码量的积累是很重要的。当积累了一定的代码量后，看问题就会比较有程序的思想，能够从层次，模块的角度来分析问题，这样思路就比较清晰了。 整个毕业设计的过程其实就是经历了一个项目的生命周期。从最初的选题确定后，开始进行相关资料的收集和理论知识的学习，接着确定自己的方案设计和系统整体结构，然后开始编码实现，调试代码，直至顺利运行，再进行性能测试，最后写出论文。这些步骤其实和一个软件项目的开发是很类似的。软件的开发同样会有这些步骤，需求分析，设计，编码，测试，发布，文档撰写等。 当完成了整个毕业设计后，对如何把握一个项目的整体有了一点基本的认识。同时从中体会到时间控制和进度安排都是很重要的，任何任务和项目都是有时间期限的，自己的想法和设计都是得基于按时完成这个前提的。 四年的大学时光即将结束，心中还是有不舍。回顾四年的学习时光，感觉自己还是过的蛮充实的。做过很多有意思的事，也认识了一帮好友与同窗。无论是做人还是学习，我的老师和同学朋友们都给了我很大的帮助，我非常感谢他们。作为即将踏上工作、步入社会的我，我想我会更加努力奋斗，不让我的家人、朋友、老师们失望。最后，愿大家在今后的日子里，一帆风顺，身体健康。