# 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，对网络协议的学习，以及对HTTP协议的通信原理，对web服务器的运行机制进行深入研究是极其重要的。

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" \o "蒂姆·伯纳斯-李) Berners-Lee构建的ENQUIRE项目至今，万维网已经走过了28年之久。从互联网时代的开起到移动互联大潮兴起，每天基于http协议所传输的数据不计其数。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。在互联网浪潮下的今天，web服务器也面临着更加严峻的挑战。几万甚至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。web服务器使用HTTP协议作为应用层通信协议，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

本文采用自底向上的方式，先从网络底层协议开始进行介绍，对IP等内核级协议做简单介绍，对tcp和http协议进行详细说明。然而一款高性能，高可用服务器的瓶颈不仅仅在网络通信上，对操作系统I/O模型的选择，对线程模型的处理同样是重中之重。所以我们需要对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；再分析商业http服务器Jetty的I/O模型和线程模型。然后对Java编程语言以及JVM平台的特点进行简单说明；最后剖析我们自己开发的http服务器的运行原理。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。

* 第一章：绪论
* 第二章：自底向上介绍网络协议基础
* 第三章：介绍操作系统I/O模型以及多线程编程模型
* 第四章：介绍服务器架构和编码设计实现
* 第五章：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

TCP是一个面向连接的协议。无论哪一方向另一方发送数据之前，都必须现在双方之间建立一条连接，所以需要了解TCP是如何建立连接以及通信结束后是如何终止的。

TCP的连接过程也称为三次握手：

1) 请求端（客户端）发送一个SYN段指明打算连接的服务端的端口号，以及初始序列号(ISN)。这个SYN段为报文段1。

2) 服务端发回包含服务端的初始序列号的SYN报文段，作为应答。同时，將确认序号设置为客户端的ISN加1以对客户端的SYN报文段进行确认。一个SYN將占用一个序号。

3) 客戶端必須將确认序号设置为服务端的ISN加1以对服务端的SYN报文段进行确认。

这三个过程即可完成连接的建立。

TCP终止连接的过程也称为四次挥手：

1) 进行关闭的一方执行主动关闭，发送一个FIN报文段到被动关闭方。

2) 被动关闭一方收到这个FIN，将发回一个ACK，确认序号为收到的序号加1

3) 被动关闭方发送一个FIN报文段到主动关闭一方。

4)主动关闭方收到这个FIN，并发回一个ACK，确认序号为收到的序号加1

以上为TCP建立连接和终止连接的全过程。下图描述了整个连接过程以及TCP的状态迁移。



### 2.2.4 TCP协议通信过程中的问题

当然，网络传输通常是不可靠的，而TCP协议是可靠地，在网络传输过程中会出现各种各样的问题，例如网络拥塞，数据超时，丢包，半包，粘包等。所以了解TCP是通过什么手段和方法保证了网络的可靠传输是非常关键且必要的。

#### 2.2.4.1 流量控制

#### 2.2.4.2 拥塞控制

## 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

HTTP（HyperText Transport Protocol）是超文本传输协议的缩写，它用于传送WWW方式的数据，关于HTTP协议的详细内容请参考RFC2616。HTTP协议采用了请求/响应模型。客户端向服务器发送一个请求，请求头包含请求的方法、URL、协议版本、以及包含请求修饰符、客户信息和内容的类似于MIME的消息结构。服务器以一个状态行作为响应，响应的内容包括消息协议的版本，成功或者错误编码加上包含服务器信息、实体元信息以及可能的实体内容。

### 2.3.2 HTTP协议报文格式

HTTP客户端和服务器通过发送纯文本（ASCII）消息进行通信。客户端向服务器发送请求，服务器发送响应。

HTTP请求报文主要有请求行，请求头部和请求正文三部分组成



1. 请求行：由三部分组成，请求方法、URL、协议版本号、之间由空格分隔

请求方法：GET、POST、PUT、HEAD、DELETE、OPTIONS、TRACE以及扩展方法。

URL：[www.example.com/example](http://www.example.com/example)

协议版本号：HTTP/主版本号.次版本号，例如：HTTP/1.0和HTTP/1.1

1. 请求头部：请求头部为请求报文添加了一些附加信息有K/V对组成，没对一行，Key和Value之间用冒号分隔，如下图：

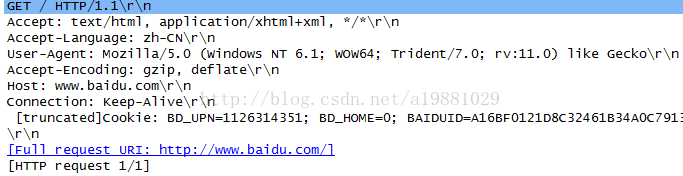
|  |  |
| --- | --- |
| 请求头 | 说明 |
| Host | 接收请求的服务器地址，可以是IP:PORT或者域名 |
| User-Agent | 发送请求的应用程序名称，通常为各浏览器内核名称 |
| Connection | 指定与连接相关的属性，如Connection:Keep-Alive |
| Accept-Charset | 通知服务端可以发送的编码格式 |
| Accept-Encoding | 通知服务端可以发送的数据压缩格式 |
| Accept-Language | 通知服务端可发送的语言 |

请求头部的最后一行为空行，表示请求头部结，接下来是正文，且这一行非常重要，必不可少。

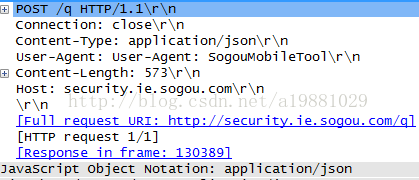
1. 请求正文：

可选部分，如GET请求没有正文，而POST请求有

GET请求：



POST请求：



HTTP响应报文格式：

HTTP响应报文格式同样由状态行、响应头部、响应正文三部分组成。



1. 状态行：由三部分组成，分别是：协议版本，状态吗，状态描述符，之间由 空格隔开。

常见状态码列举：

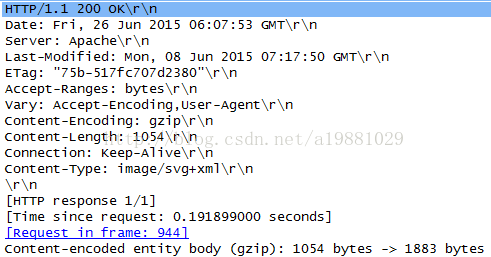
|  |  |
| --- | --- |
| 状态码 | 说明 |
| 200 | 响应成功。 |
| 302 | 跳转，跳转地址通常在Location中指定。 |
| 400 | 客户端请求由语法错误，无法被服务端识别。 |
| 403 | 服务端收到请求，但拒绝提供服务。 |
| 404 | 请求资源不存在。 |
| 500 | 服务器内部错误。 |

1. 响应头部：与请求头部类似，为响应报文添加一些附加信息。

常见响应头部如下：

|  |  |
| --- | --- |
| 响应头 | 说明 |
| Server | 服务器应用软件的名称和版本 |
| Content-Type | 响应正文类型 |
| Content-Length | 响应正文长度 |
| Content-Charset | 响应正文使用的编码 |
| Content-Encoding | 响应正文使用的数据压缩格式 |
| Content-Language | 响应正文使用的语言 |

示例：



1. 响应正文：服务端发送的HTML，CSS,JS或者图片等资源文件。

### 2.3.3 HTTP协议通信流程

HTTP协议为应用程协议，其工作在传输层协议TCP以上，所以HTTP协议的整个通信流程建立在TCP协议双方建立连接的基础上。整个通信流程如下：

1. 客户端(一般为各浏览器)发送HTTP请求报文给服务器
2. TCP协议封装HTTP请求报文，作为TCP数据包
3. 双方经过TCP建立连接。
4. 服务器解析TCP报文并返回相应的HTTP响应报文，并将响应报文封装为TCP数据包发回客户端
5. 客户端接收服务端返回的HTTP响应报文，并与浏览器解析并展示响应的资源数据。
6. 双方执行TCP关闭连接过程。

### 2.3.4 HTTPS与Web安全

# I/O模型与多线程编程

要编写一款高性能HTTP服务器，关键就在于在高并发请求下的处理能力。而对于操作系统I/O方式的选择以及高可用的多线程并发编程设计是一款服务器性能好坏，处理并发能力强弱的重中之重。所以了解操作系统的I/O模型和熟练使用多线程编程是非常重要的。

## 3.1 Linux I/O模型分析

在了解I/O模型之前，需先理解关于同步，异步，阻塞，非阻塞这几个概念。

### 同步，异步，阻塞，非阻塞

**同步与异步**：同步与异步关注的是消息通信机制(synchronous communication/ asynchronous communication)。

所谓同步，就是发出一个调用时，在没有得到结果之前，该调用就一直不返回，。一旦调用返回，就得到该返回值了。即，调用方主动等待这个调用结果。

而异步则相反，在调用发出之后，这个调用就直接返回了，所以没有返回结果。换句话说，当一个异步过程调用发生之后，调用者不会立刻得到结果。而是在调用发出后，被调用者通过状态通知来通知调用者，或者通过回调函数来处理这个调用。

**阻塞与非阻塞**：阻塞与非阻塞关注的是程序在等待调用结果（消息，返回值）是的状态。阻塞调用是指调用结果返回之前，当前线程被挂起。调用线程只有得到结果之后才返回。非阻塞调用是指在不能立刻得到结果之前，该调用不会阻塞当前线程。

了解完这几个概念之后，就需要了解Linux下的几种I/O模型了。

### 3.1.2 Linux下的五种I/O模型

一般来说，程序进行输入操作有两步：

1：等待有数据可读；

2：将数据从系统内核拷贝到程序的数据区。

对于一个对Socket的输入操作，第一步一般来说是等待数据从网络上传到本地。当数据包到达的时候，数据将会从网络层拷贝到内核的缓冲中；第二步是把内核中的数据拷贝到程序的数据区中。

* **阻塞式I/O**

阻塞I/O模式是最普遍使用的I/O 模式。大部分程序使用的都是阻塞模式的I/O。缺省的，一个套接字建立后所处于的模式就是阻塞I/O模式。 对于一个 UDP套接字来说，数据就绪的标志比较简单：

* 已经收到了一整个数据报
* 没有收到。

而TCP这个概念就比较复杂，需要附加一些其他的变量。

在下图中，一个进程调用recvfrom，然后系统调用并不返回知道有数据报到达本地系统，然后系统将数据拷贝到进程的缓存中。（如果系统调用收到一个中断信号，则它的调用会被中断）我们称这个进程在调用 recvfro一直到从recvfrom返回这段时间是阻塞的。当recvfrom 正常返回时，我们的进程继续它的操作。



* **非阻塞式I/O**

当我们将一个套接字设置为非阻塞模式，我们相当于告诉了系统内核：“当我请求的I/O操作不能够马上完成，你想让我的进程进行休眠等待的时候，不要这么做，请马上返回一个错误给我。”

我们可以参照下图来描述非阻塞模式I/O。

我们开始对 recvfrom 的三次调用，因为系统还没有接收到网络数据，所以内核马上返回一个EWOULDBLOC的错误。第四次我们调用recvfrom函数，一个数据报已经到达了， 内核将它拷贝到我们的应用程序的缓冲区中，然后recvfrom正常返回，我们就可以对接收到的数据进行处理了。当一个应用程序使用了非阻塞模式的套接字，它需要使用一个循环来不停的测试是否一个文件描述符有数据可读（称做polling）。应用程序不停的polling内核来检查是否I/O操作已经就绪。这将是一个极浪费CPU资源的操作。这种模式使用中不是很普遍。



* **I/O复用(select，poll,epoll)**

在使用I/O多路技术的时候，我们调用 select()函数和poll()函数，在调用它们的时候阻塞，而不是我们来调用recvfrom（或 recv）的时候阻塞。

下图说明了它的工作方式。

当我们调用select函数阻塞的时候，select函数等待数据报套接字进入读就绪状态。当select函数返回的时候，也就是套接字可以读取数据的时候。这时候我们就可以调用recvfrom 函数来将数据拷贝到我们的程序缓冲区中。和阻塞模式相比较，select()和 poll()并没有什么高级的地方，而且，在阻塞模式下只需要调用一个函数：读取或发送，在使用了多路复用技术后，我们需要调用两个函数：先调用select()函数或 poll()函数，然后才能进行真正的读写。多路复用的高级之处在于，它能同时等待多个文件描述符，而这些文件描述符（套接字描述符）其中的任意一个进入读就绪状态，select()函数就可以返回。

 假设我们运行一个网络客户端程序，要同时处理套接字传来的网络数据又要处理本地的标准输入输出。在我们的程序处于阻塞状态等待标准输入的数据的时候，假如服务器端的程序被kill（或是自己Down掉了），那么服务器程端的TCP 协议会给客户端（我们这端）的TCP协议发送一个FIN数据代表终止连接。但是我们的程序阻塞在等待标准输入的数据上，在它读取套接字数据之前（也许是很长一段时间），它不会看见结束标志，我们就不能够使用阻塞模式的套接字。 IO多路技术一般在下面这些情况中被使用：

* 当一个客户端需要同时处理多个文件描述符的输入输出操作的时候（一般来说是 标准的输入输出和网络套接字）， I/O多路复用技术将会有机会得到使用。当程序需要同时进行多个套接字的操作的时候。
* 如果一个 TCP 服务器程序同时处理正在侦听网络连接的套接字和已经连接好的套接字。
* 如果一个服务器程序同时使用TCP和UDP协议。
* 如果一个服务器同时使用多种服务并且每种服务可能使用不同的协议（比如 inetd 就是这样的）。

I/O 多路服用技术并不只局限与网络程序应用上。几乎所有的程序都可以找到应用 I/O 多路复用的地方。

* **信号驱动式I/O(SIGIO)**

我们可以使用信号，让内核在文件描述符就绪的时候使用SIGIO信号来通知我们。

我们将这种模式称为信号驱动I/O模式。使用这种模式，我们首先需要允许套接字使用信号驱动I/O，还要安装一个SIGIO的处理函数。在这种模式下，系统调用将会立即返回，然后我们的程序可以继续做其他的事情。当数据就绪的时候，系统会向我们的进程发送一个SIGIO信号。这样我们就可以SIGIO信号的处理函数中进行I/O操作（或是我们在函数中通知主函数有数据可读）。

对于信号驱动I/O模式，它的先进之处在于它在等待数据的时候不会阻塞程序可以做自己的事情。当有数据到达的时候，系统内核会向程序发送一个 SIGIO 信号进行通知，这样我们的程序就可以获得更大的灵活性，因为我们不必为等待数据进行额外的编码。

 信号I/O可以使内核在某个文件描述符发生改变的时候发信号通知我们的程序。异步I/O可以提高我们程序进行I/O读写的效率。通过使用它，当我们的程序进行I/O操作的时候，内核可以在初始化I/O操作后立即返回，在进行I/O操作的同时，我们的程序可以做自己的事情，直到 I/O 操作结束，系统内核给我们的程序发消息通知。

基于Berkeley接口的Socket 信号驱动 I/O 使用信号SIGIO。有的系统SIGPOLL信号， 它也是相当于SIGIO的。

为了在一个套接字上使用信号驱动I/O操作，下面这三步是所必须的。

（1）一个和SIGIO信号的处理函数必须设定。

（2）套接字的拥有者必须被设定。一般来说是使用fcntl函数的 F\_SETOWN参数来进行设定拥有者。

（3）套接字必须被允许使用异步I/O。一般是通过调用 fcntl 函数的 F\_SETFL命令，O\_ASYNC为参数来实现。

* **异步I/O(POSIX的aio\_系列函数)**

当我们运行在异步I/O模式下时，我们如果想进行I/O操作，只需要告诉内核我们要进行I/O操作，然后内核会马上返回。具体的I/O和数据的拷贝全部由内核来完成，我们的程序可以继续向下执行。当内核完成所有的I/O操作和数据拷贝后，内核将通知我们的程序。

异步I/O和信号驱动I/O的区别是：

* + 信号驱动I/O模式下，内核在操作可以被操作的时候通知给我们的应用程序发送SIGIO消息。
  + 异步I/O模式下，内核在所有的操作都已经被内核操作结束之后才会通知我们的应用程序。

如下图，当我们进行一个 IO 操作的时候，我们传递给内核我们的文件描述符，我们的缓存区指针和缓存区的大小，一个偏移量offset，以及在内核结束所有操作后和我们联系的方法。这种调用也是立即返回的，我们的程序不需要阻塞住来等待数据的就绪。我们可以要求系统内核在所有的操作结束后（包括从网络上读取信息，然后拷贝到我们提供给内核的缓存区中）给我们发一个消息。



* **几种I/O模式的比较**

下面这个表格对这几种I/O模式进行了对比。

 以上就是Linux下五种I/O模式，以及五种I/O模式的区别，开发一款高性能服务器，选择合适的I/O模式是最关键的因素之一。，

## Tomcat I/O模型与线程模型分析

上节分析了Linux下的五种I/O模型，在熟悉了Linux下的各种I/O模式之后，们想要开发一款高性能，高可用的商用服务器还需要深厚的多线程编程功底。于是，不妨了解一款成熟的商用服务器，研究其源码。学习其I/O模式，多线程模型以及程序架构，有助于帮助我们更好的了解服务器。

Tomcat拥有两种I/O模型，BIO以及NIO。两种I/O分别应用于不用的场景。

BIO即阻塞式I/O，用于在并发数相对低的场景。每个socket对应一个线程，用来接收socket(http)请求以及进行数据处理，再将响应发送回客户端。由线程池统一管理所有线程资源。如下图所示：

NIO即非阻塞I/O，也就是多路复用I/O。对于Java语言来说，只需使用NIO对应的JDK即可，其底层对于不同的操作系统有不同的处理方式(Linux2.6之前是select,poll,2.6之后为epoll，Windows是IOCP)。其可用于并发数相对较高的场景下。

## 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

整个项目架构分为9个模块:dam-annotation,dam-bridge,dam-synamic-syntax,dam-exception,dam-http,dam-io,dam-server,dam-start,dam-util。

每个模块的作用 说明如下：

* dam-annotatio：所有注解定义，开发webapp所需要的注解类全部在这个模块中。
* dam-bridge：桥梁模块，用于连接server与app之间通信的桥梁。
* dam-synamic-syntax：动态脚本解析模块，用于解析app返回的动态脚本。
* dam-exception：所有的自定义异常类。
* dam-http：定义http请求，响应所需要的数据结构，以及根据RFC2616所定义http协议消息枚举，常量等。
* dam-io：采用多路复用IO，监听TCP请求，以及将server返回的数据写回客户端。
* dam-server：处理并解析tcp-http消息，并将消息封装交给handler链进行一系列的处理。
* dam-start：启动模块。
* dam-util：工具模块。

### 4.2.1 I/0模型以及线程模型

·

### 4.2.2 整体架构以及流程

### 4.2.3 模块说明

## 4.3 服务器编码与实现

### 4.3.1 生命周期管理

### 4.3.2 NIO编码实现

### 4.3.3 多Reactor模式编码实现

### 4.3.4 观察者模式与Handler链实现

### 4.3.5工厂模式创建app返回结果处理器

### 4.3.6 策略模式与动态脚本解析

## 4.4服务器运行

## 4.5 在服务器上进行webApp开发

### 4.5.1注解说明

### 4.5.2 开发webApp

# 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。