

第7课 Hello World服务器

本课将引导你进入Web服务器的世界



Web服务器是啥

服务器定义

Web服务器是专门处理HTTP请求并提供数据的软件或硬件。

基本功能

处理请求并返回数据，确保客户端获得响应。



服务器的核心功能

1. **提供服务：**服务器专门用来回应来自客户端（如电脑、手机等）的请求，提供各种服务（比如网站、文件存储、数据查询等）。
2. **并发处理：**服务器可以同时处理来自多个客户端的请求，就像一个饭馆可以同时为多位顾客做不同的餐。
3. **数据存储与传输：**服务器还负责存储大量信息，并根据请求将数据传输给客户端。

服务器工作原理

1

监听端口

Web服务器监听网络中的特定端口，等待客户端请求。

2

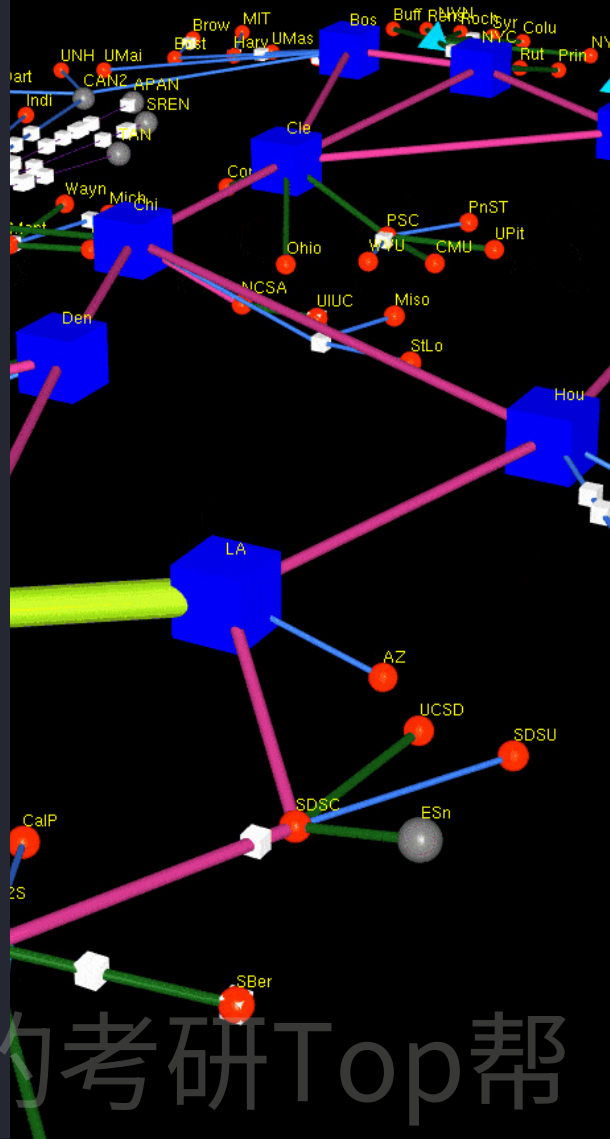
处理请求

解析HTTP请求的详细信息，提取必要数据。

3

发送响应

生成并发送包含请求资源的HTTP响应消息。



Web服务器的组成

静态内容处理

提供文件系统中的文件，如HTML和图像。

动态内容生成

动态生成网页内容，运行应用程序或脚本。

安全管理

加密通信，身份验证，访问控制。

日志记录

记录服务器活动详细信息，便于监控和调试。

服务器类型

专用服务器软件

这类服务器软件专注于处理HTTP请求，提供Web内容服务，并进行相关的安全控制、负载均衡和性能优化等功能。

- Apache HTTP Server：广泛应用的开源Web服务器软件，支持多种操作系统平台。
- Nginx：高性能的Web服务器、反向代理服务器和负载均衡器。
- Microsoft Internet Information Services (IIS)：微软开发的Web服务器产品，主要用于Windows环境。

集成服务器环境

集成服务器环境是一组相互协作的技术组件，构成了完整的Web应用部署平台。

- LAMP：Linux、Apache、MySQL和PHP/Perl/Python的组合。
- MEAN：MongoDB、Express.js、AngularJS和Node.js的组合。

网关是什么

- 默认网关可以看作是**你的设备与外部世界的通道**。当你的计算机或手机想要与局域网外的设备（如互联网中的服务器）通信时，数据会先通过默认网关。通常，这个设备是**路由器**。

功能：

1. 跨网络数据转发：

- 当你在本地网络（如家里的Wi-Fi网络）上访问外部网络（如互联网）时，设备并不知道如何直接找到外部服务器的地址。默认网关会接收你的数据包，并将它们转发到正确的目的地。

2. 充当路由器：

- 默认网关通常是一个路由器，它负责在不同网络之间进行路由选择，决定数据包的最佳传输路径

DHCP服务器是什么

DHCP服务器（Dynamic Host Configuration Protocol，动态主机配置协议服务器）是一种网络服务，用于自动分配网络设备的**IP地址**、**子网掩码**、**网关**、**DNS服务器**等网络参数，使设备能够快速连接到网络而无需手动配置。

DHCP服务器的主要功能：

1. 自动分配IP地址：

- 当一个设备（如电脑、手机等）接入网络时，它会发送一个请求给DHCP服务器，询问是否有可用的IP地址。DHCP服务器会从一个预先配置的IP地址池中分配一个可用的IP地址给这个设备。

2. 分配其他网络参数：

- 除了IP地址，DHCP服务器还会分配网络配置参数，比如：
 - **子网掩码**：定义网络中的IP地址范围。
 - **默认网关**：设备与外部网络（如互联网）的通信出口。
 - **DNS服务器地址**：用于解析域名（如www.google.com）为IP地址。

3. 自动管理IP地址租期：

- DHCP服务器为设备分配IP地址是有“租期”的。当租期快到期时，设备会请求续期，如果租期到期且设备不再需要这个IP地址，DHCP服务器会回收并重新分配给其他设备。

DHCP服务器的工作流程：

1. **DHCP发现：**设备加入网络后，会广播发送一个请求（DHCP Discover），寻找DHCP服务器。
2. **DHCP提供：**DHCP服务器收到请求后，从IP地址池中选择一个可用IP，并通过“DHCP Offer”消息回复设备。
3. **DHCP请求：**设备选择一个IP地址，并发送“DHCP Request”消息来确认接收这个地址。
4. **DHCP确认：**DHCP服务器发送“DHCP ACK”消息，确认设备可以使用这个IP地址，同时提供其他网络配置（如网关和DNS服务器）。

什么是DNS服务器

DNS服务器（Domain Name System，域名系统服务器）是负责将**域名**（如 www.google.com）转换为IP地址（如 172.217.160.68）的服务器。它相当于互联网的“电话簿”，帮助将用户输入的易记的域名翻译为计算机能够理解和定位的数字IP地址。

DNS层次结构

- DNS系统采用分布式、层次化的结构，包含多个不同级别的服务器，例如：
 - **根DNS服务器**：最高级别的服务器，负责将查询定向到特定顶级域（如.com、.org等）的服务器。
 - **顶级域（TLD）DNS服务器**：负责管理特定顶级域下的域名，如.com、.net等。
 - **权威DNS服务器**：负责存储特定域名的IP地址记录，直接回答对该域名的查询。

递归查询 (Recursive Query)

在**递归查询**中，客户端（例如你的计算机）向DNS服务器发送查询请求后，DNS服务器会**完全负责**找到最终的IP地址，并返回结果给客户端。如果DNS服务器不知道答案，它会继续向其他DNS服务器发出查询请求，直到找到正确的IP地址为止。

工作流程：

1. 客户端（如你的计算机）向DNS服务器发送查询请求。
2. 如果该DNS服务器没有缓存这个域名的IP地址，它会递归地查询其他DNS服务器。
3. 该DNS服务器会从根DNS服务器开始，逐级向下查询（例如从根DNS服务器到顶级域DNS服务器，再到权威DNS服务器），直到找到最终的IP地址。
4. 最后，DNS服务器将IP地址返回给客户端。

迭代查询 (Iterative Query)

在**迭代查询**中，客户端向DNS服务器发送查询请求，如果该DNS服务器没有所需的域名解析结果，它会告诉客户端**下一个可以查询的DNS服务器**。客户端再继续向下一个服务器查询，直到找到最终的IP地址。

工作流程：

1. 客户端向本地DNS服务器发送查询请求。
2. 如果该DNS服务器不知道域名的IP地址，它不会代替客户端进行进一步的查询，而是向客户端返回上一级的DNS服务器地址。
3. 客户端再向上一级DNS服务器发送查询，依次进行，直到获得最终的IP地址。
4. 每一级查询后，客户端会得到更多的信息，逐步接近目标IP地址。

DNS两种查询方式总结：

- **递归查询**让DNS服务器承担大部分查询工作，客户端只需与一个DNS服务器交互。
- **迭代查询**则让客户端自己负责逐级查询，DNS服务器只是提供指引。

这两种查询方式通常是结合使用的。例如，客户端会先向本地DNS服务器发起递归查询，而本地DNS服务器在与其他服务器通信时可能采用迭代查询。

常考：互联网通信过程

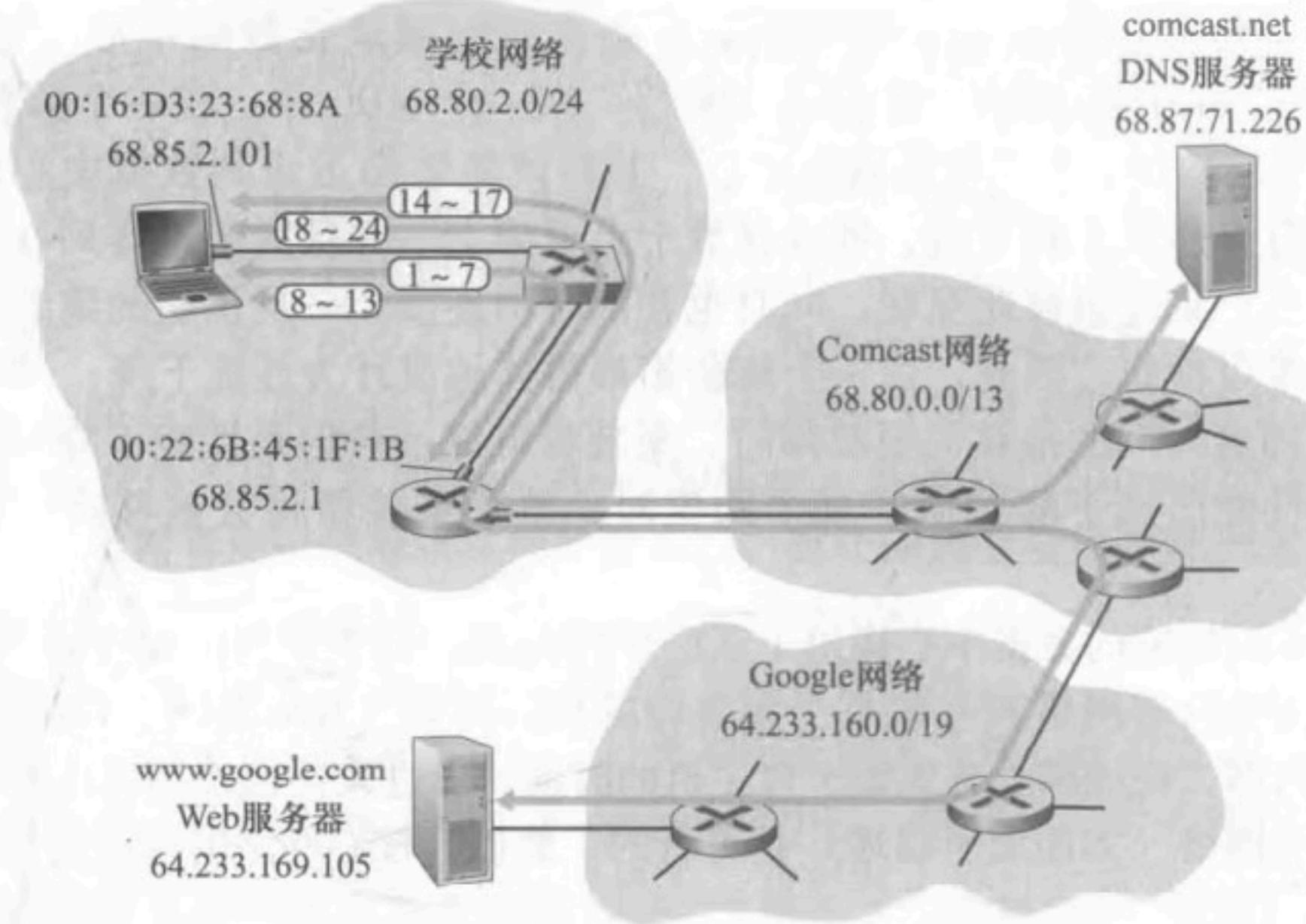


图 6-32 Web 页请求的历程：网络环境和动作

DHCP 初始化（获取IP地址）

请求过程：Bob的便携机连接到网络时，因为没有IP地址，所以首先会向网络发送一个DHCP请求报文（目的端口67，源端口68），这是一个广播请求。

报文内容：报文包含了便携机的MAC地址，以0.0.0.0为源IP，目标IP为广播地址255.255.255.255。

DHCP服务器的响应：网络中的DHCP服务器接收到该请求，提供一个IP地址给Bob的便携机，同时提供子网掩码、默认网关、DNS服务器地址等信息。

IP分配确认：便携机从DHCP服务器收到一个包含分配的IP地址的DHCP ACK报文后，确认使用该IP。

ARP请求（获取默认网关的MAC地址）

ARP查询：

- 为了与外部网络通信（例如发送DNS查询），便携机需要知道默认网关的MAC地址。
- 便携机发送ARP请求报文，目标MAC地址为广播地址FF:FF:FF:FF:FF:FF，询问“谁是IP地址为192.168.1.1（假设的默认网关IP）的主机？”

ARP响应：

- 默认网关收到ARP请求后，发送ARP响应报文，包含自己的MAC地址。
- 便携机接收到ARP响应后，在ARP缓存中记录下默认网关的IP地址和对应的MAC地址。

DNS查询（解析域名）

请求过程：在获取IP地址后，Bob的便携机需要访问www.google.com，但它还不知道这个域名对应的IP地址，因此需要通过DNS进行域名解析。

DNS查询报文：便携机会发送一个DNS查询报文（UDP协议，目的端口53）到DNS服务器的IP地址，该报文包含了需要解析的域名。

DNS服务器的响应：DNS服务器会在收到查询报文后，返回相应的IP地址给Bob的便携机。

TCP三次握手

TCP连接建立：在获取到www.google.com的IP地址后，Bob的便携机需要与该IP建立TCP连接。为了建立连接，便携机会首先发送一个TCP SYN报文。

三次握手过程：SYN报文到达目标服务器后，服务器会返回一个SYN-ACK报文；便携机收到SYN-ACK后，再发送一个ACK报文确认连接的建立。三次握手完成后，TCP连接正式建立。

HTTP请求和响应

HTTP请求：建立TCP连接后，Bob的便携机发送HTTP GET请求来请求网页内容，该请求报文包含了目标URL等信息。

服务器响应：目标服务器接收到HTTP请求后，处理请求并返回一个HTTP响应报文，其中包含所请求的网页内容。

数据传输和关闭连接：便携机接收到HTTP响应后，开始呈现网页内容。

在数据传输完成后，TCP连接会通过四次挥手来关闭连接。



这节课用到的C++知识

M学长的考研Top帮

类型别名 (Type Alias)

在C++编程中，`using`关键字用于创建已存在类型的别名。这一特性有助于：

1. 提升代码可读性：

```
using MyVector = std::vector<int>; // 定义了std::vector<int>类型的别名MyVector
```

2. 简化代码重复：避免在多个地方重复书写相同类型的长而复杂的定义，通过类型别名可以集中管理和引用。

3. 增强抽象与封装：将底层实现细节隐藏起来，用类型别名表示特定的概念或接口，提高代码模块化的程度。

using namespace std;

在C++标准库中，所有标准组件都被定义在 `std` 命名空间内。为了使用这些组件，通常需要明确指定它们来自 `std` 命名空间。然而，如果在某个作用域中包含 `using namespace std;` 语句，则该作用域内的代码可以直接使用 `std` 命名空间的所有组件，无需写出 `std::` 前缀。

```
std::cout << "Hello, World!"; //使用前  
cout << "Hello, World!"; //使用后
```

std::function模板

std::function是C++11标准库引入的一种通用函数对象包装器，位于头文件内。它具有以下核心特征：

1. 存储任意可调用实体：std::function能够容纳任何符合特定签名的**可调用对象**，包括全局函数、成员函数指针、lambda表达式、仿函数以及其他Callable对象。
2. 签名指定：在声明一个std::function实例时，需要明确指出其内部可存储的函数签名，例如：
`std::function<std::string(const std::string&)> requestHandler;` // 可以存储接受一个const std::string&参数并返回一个std::string的函数
3. 延迟绑定和运行时多态：std::function允许在运行时动态决定调用哪个具体的函数或对象，实现了类似于虚函数表的机制。
4. 类型安全性保证：虽然std::function能包含多种类型的可调用实体，但它确保所有存储的对象都能遵循预定义的函数签名进行调用，从而保障类型安全。

代码示例

```
// 声明一个 std::function 实例，表示它可以存储接受两个整数参数并返回一个整数的可调用对象
std::function<int(int, int)> add_function;

// 现在可以将符合这个签名的任何可调用对象赋值给它，如下面的普通函数、lambda 或者绑定到
// 成员函数的对象
add_function = [](int a, int b) { return a + b; }; // lambda 函数

// 如果有一个这样的全局函数：
int global_add(int a, int b) {
    return a + b;
}

// 也可以赋值给上面声明的 std::function 实例
add_function = &global_add;
```



```
atinesh@atinesh-  
atinesh@atinesh-pc:~/D  
Chat server started on  
john is now connected  
michael is now connect  
tracey is now connect  
□
```

```
atinesh@atinesh-  
atinesh@atinesh-pc:~/D  
Connected to remote ho  
Enter your name to ent  
  
Me > hi  
Me > hi  
Me > hi  
Me > hi  
  
Me >  
johnbye > johnbye  
□
```

Socket编程基础

Socket简介

Socket是网络编程中的端点，
用于数据传输。

功能

建立网络连接与数据传输的基
础设施。

M学长的考研Top帮

Socket简介

在网络编程中，socket是一个重要概念，用于描述IP地址和端口，是网络数据传输的端点。通过创建socket，计算机之间可以相互发送和接收数据，实现网络通信。

- 端口是指网络通信中的一个虚拟通道，用于标识不同的服务或应用程序。每个端口都有一个唯一的号码，范围从0到65535。
- 服务器监听特定的端口，通常是80端口（HTTP协议的默认端口）或443端口（HTTPS协议的默认端口）。

Socket类型

流式Socket (SOCK_STREAM)

创建TCP连接，保证数据的完整性和顺序性。

适用于要求可靠传输的应用，如Web服务器、文件传输等。

1

2

数据报Socket (SOCK_DGRAM)

创建UDP连接。

适用于实时应用，在线游戏、实时视频会议等

TCP连接流程详解

服务器端 (Server):

1. 创建套接字 (create): 使用socket()函数创建一个用于网络通信的套接字。
2. 绑定端口号 (bind): 将套接字与特定的IP地址和端口号关联起来。
3. 监听连接 (listen): 使套接字进入被动监听状态, 等待客户端的连接请求。
4. 接受连接请求 (accept): 阻塞并返回一个新的套接字, 用于与发起连接的客户端进行数据交换。
5. 接收/发送数据 (recv/send): 使用新套接字与客户端进行全双工的数据传输。
6. 关闭套接字 (close): 完成数据交互后, 关闭已建立连接的套接字。

客户端 (Client):

1. 创建套接字 (create): 与服务器相同, 客户端也需要先创建一个套接字。
2. 发起连接请求 (connect): 客户端调用connect()函数主动向服务器发起连接请求, 并提供服务器的IP地址和端口号。
3. 发送/接收数据 (send/recv): 一旦连接建立成功, 客户端同样可以使用send()和recv()函数与服务器交换数据。
4. 关闭套接字 (close): 在数据交换完成后, 客户端关闭其使用的套接字以释放资源。

UDP连接流程简介：

由于UDP是无连接的，因此没有“监听”和“接受连接”的概念。但仍有以下基本步骤：

服务器端（Server）：

1. 创建套接字（create）：与TCP一样，首先创建一个UDP套接字。
2. 绑定端口号（bind）：通过bind()函数将套接字绑定到特定的本地IP地址和端口上，以便接收来自客户端的消息。
3. 接收/发送消息（recvfrom/sendto）：对于UDP，服务器和客户端都使用recvfrom()和sendto()函数直接读写数据，它们不仅负责数据传输，还需要处理源和目标地址信息。
4. 关闭套接字（close）：在不需要继续接收或发送数据时，关闭套接字以释放资源。

客户端 (Client):

1. 创建套接字 (create): 同样创建一个UDP套接字。
2. 发送/接收消息 (sendto/recvfrom): 客户端可以直接使用sendto()向任意服务器发送数据, 并使用recvfrom()接收从服务器或其他客户端发来的数据。
3. 关闭套接字 (close): 完成数据交互后, 客户端关闭其套接字。



总结：

- TCP连接涉及三次握手建立连接、数据可靠传输及四次挥手断开连接，而UDP是无连接的，每个数据报文独立传输，不保证顺序和可靠性。
- UDP服务器和客户端不需要监听和接受连接，而是直接通过指定对方地址进行数据报文的收发。

Socket编程流程



创建Socket

利用socket()函数的编程步骤。



绑定及监听

通过bind()和listen()函数将socket与地址绑定并处于监听状态。



接收连接

使用accept()函数，等待客户端连接。



数据交换

通过send()和recv()函数进行通信。



关闭Socket

完成交流后使用close()函数结束连接。

创建socket

```
#include <sys/socket.h>
```

```
int socket(int domain, int type, int protocol);
```

- **domain**: 指定协议族，对于IPv4网络通信，通常设置为**AF_INET**。
- **type**: 指定套接字类型，对于面向连接的流式传输服务（如TCP），设置为**SOCK_STREAM**。
- **protocol**: 指定特定的协议编号，若为0，则会根据**domain**和**type**自动选择最合适的协议（对于**AF_INET**和**SOCK_STREAM**，系统会选择TCP协议）。

函数返回值是一个整数，代表新创建套接字的描述符。如果成功创建套接字，该描述符将用于后续的连接、绑定、接收和发送数据等操作；若失败，则返回-1，并可以通过**errno**获取错误代码。

创建和监听socket

```
int server_fd = socket(AF_INET, SOCK_STREAM, 0);  
struct sockaddr_in address;    //用于互联网的地址结构。  
address.sin_family = AF_INET;  //设置地址族为IPv4  
address.sin_addr.s_addr = INADDR_ANY; //允许服务器接受发往本机任何IP的请求  
address.sin_port = htons(PORT); //设置端口号，htons确保端口格式正确。  
//将服务器 socket server_fd 绑定到 address 所代表的地址和端口上，使服务器能够监听来自  
该地址和端口的连接请求  
bind(server_fd, (struct sockaddr *)&address, sizeof(address));  
listen(server_fd, 3); //让socket进入被动监听状态。
```

bind() 函数:

```
bind(server_fd, (struct sockaddr *)&address, sizeof(address));
```

bind() 函数的作用是将指定的套接字与给定的本地地址进行关联

- server_fd: 是一个已创建好的套接字文件描述符，通过调用 socket() 函数得到。
- (struct sockaddr *)&address: 是指向一个已初始化的 sockaddr_in 结构体（或更通用的 sockaddr 结构体）的指针。这个结构体包含了服务器要绑定的本地地址信息，包括IP地址、端口号等。
- sizeof(address): 表示上述结构体的大小。

listen()函数

- 监听网络请求：

```
listen(server_fd, 3);
```

- listen()：让socket进入被动监听状态。
- 3：等待连接队列的最大长度。具体来说，这里的数字表示服务器可以同时等待的连接请求的最大数量。当服务器正在处理一个连接请求时，如果有其他客户端的连接请求到达，它们将被放入等待队列中，直到服务器有空闲的资源来处理它们。

接收连接过程

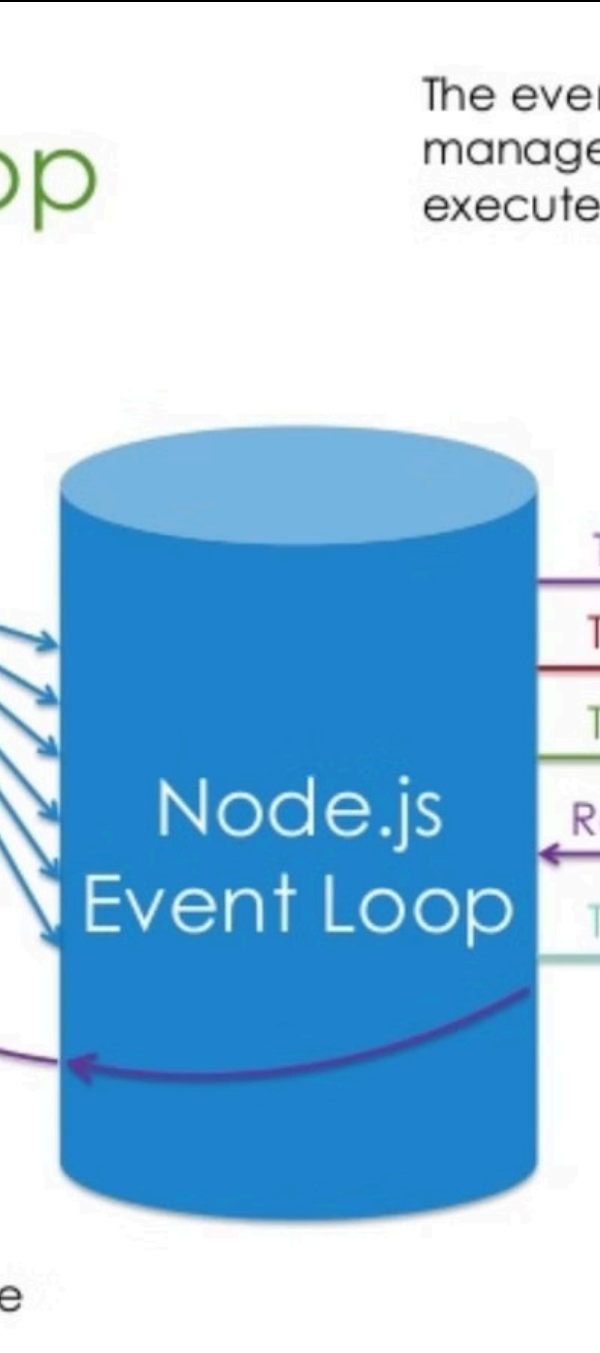
函数	描述
accept()	等待并接受连接请求
new_socket	新的通信socket描述符

accept() 函数：

```
int new_socket = accept(server_fd, (struct sockaddr *)&client_address, &client_len);
```

在服务器端调用 listen() 函数进入监听状态后，使用 accept() 函数等待并接受客户端的连接请求。accept() 会阻塞直到有新的客户端连接到来。

- server_fd：之前创建并进入监听状态的服务器套接字描述符。
- (struct sockaddr *)&client_address：指向客户端地址信息的指针。
- &client_len：存储实际返回的客户端地址结构体的大小。



处理HTTP请求

1

读取请求

使用`read()`函数从客户端socket中读取请求数据。

2

存储数据

Buffer存储客户端发送的请求内容。

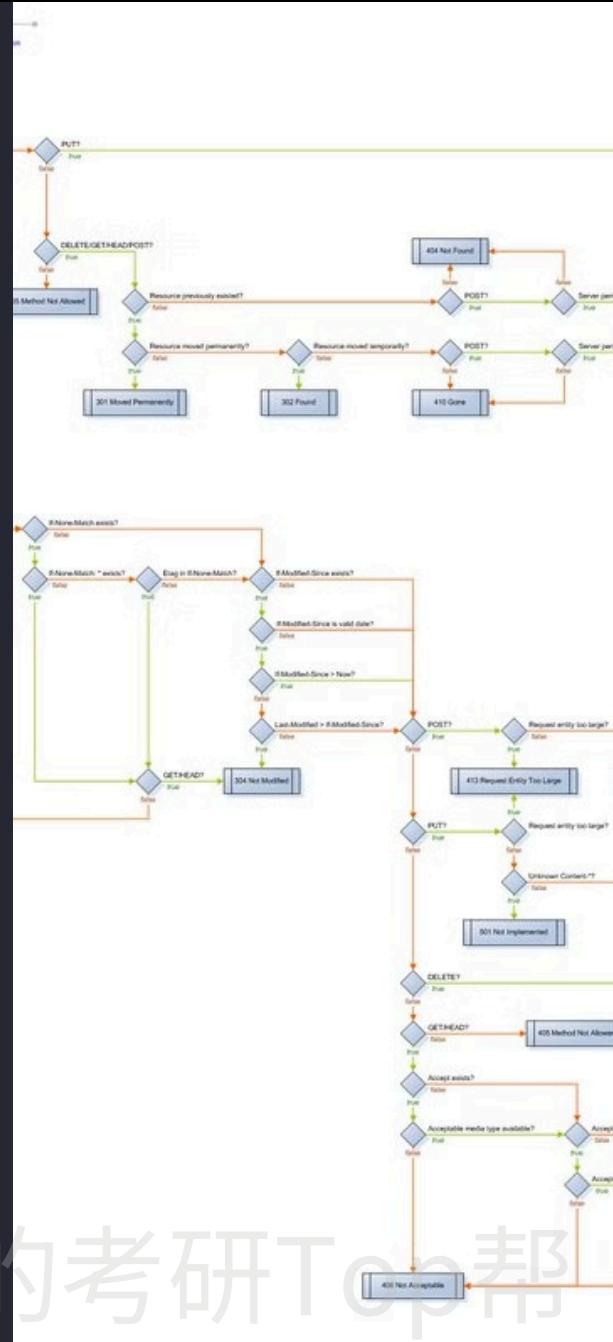
发送HTTP响应

响应方法

通过write()或send()向客户端发送数据响应。

内容包含

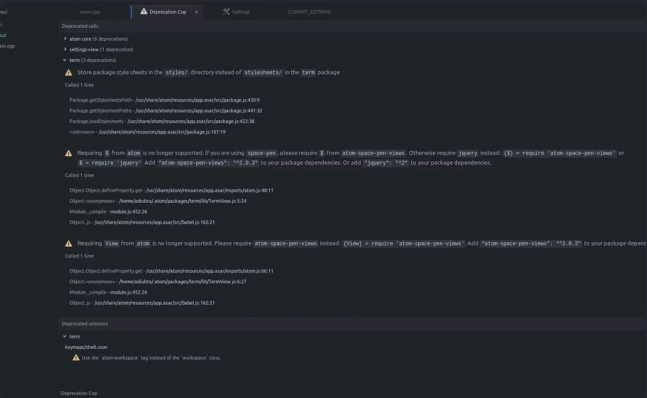
包括HTTP状态码和响应体。



代码实现

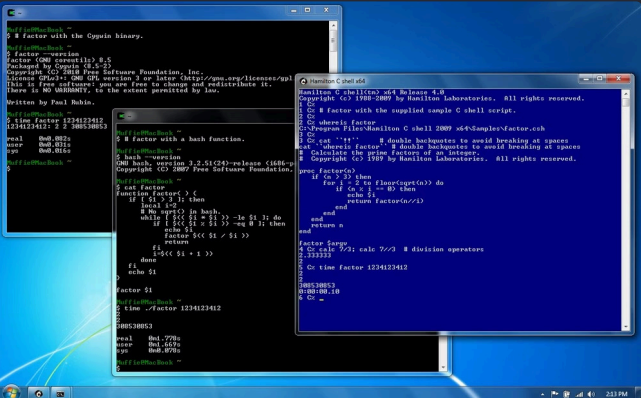


编译和运行



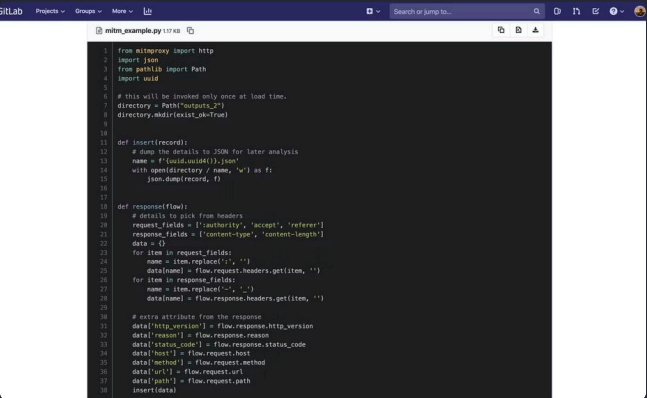
编译

使用g++命令编译代码。



运行

执行编译后的程序。



结果展示

通过浏览器查看运行结果。

```
ckreddy@ckreddy-MXC061:~/elasticsearch-2.1.0$ curl -XDELETE 'http://localhost:9200/varnish_logs/varnish/_query' -d '{
  "query" : {
    "match_all" : {}
  }
}'
{"found":false,"_index":"varnish_logs","_type":"varnish","_id":"_query","_version":1,"_shards":{"total":2,"successful":1,"failed":0}}c
ckreddy@ckreddy-MXC061:~/elasticsearch-2.1.0$
```

测试服务器

1

测试工具

使用浏览器或curl工具进行测试。


`http://localhost:8080`或`curl http://localhost:8080`

2

访问地址

通过指定的本地地址访问服务器。

M学长的考研Top帮

A woman in a blue dress is standing at the front of a modern lecture hall, addressing a group of students seated at curved wooden desks. The room has large windows and a projector on the ceiling. The text '谢谢大家' is overlaid in pink on the left side of the image.

谢谢大家

大家好好消化~~

M学长的考研Top帮