

HTTP请求处理与响应

HTTP协议是网络通信的基石。了解其运作方式与不同版本的演变对互联网发展有着重要意义。



课程目标

1

理解协议结构

掌握HTTP协议的基本组成及其工作原理。

2

解析HTTP请求

学习如何解码GET和POST请求中的数据。

3

实际操作

通过实际操作，学会构建响应，实现有效的通信。



HTTP-HyperText Transfer Protocol

超文本传输协议

M学长的考研Top帮

超文本传输协议

超文本传输协议（HTTP）顾名思义，包含“超文本”、“传输”和“协议”三部分。

“超文本”指的是包含多种类型内容并支持超链接的文本格式，它可以包含文本、图像、视频等多媒体内容，并通过超链接连接到其他文档或资源，代表着HTTP的扩展性。

“传输”指的是HTTP 的主要功能是传输超文本数据，它定义了浏览器和服务器之间如何请求和响应数据。

“协议”指的是通信规则。HTTP 定义了一组规则，用于规定客户端与服务器之间如何建立连接、如何传输数据、以及请求和响应的格式。



HTTP基本知识

HTTP——超文本传输协议，是连接客户端与服务器的纽带，实现网络资源的互联与交换的基础。

基于TCP/IP协议的应用层通信协议，主要用于万维网（WWW）中客户端与服务

器之间的通信。

它是互联网上数据交换的基础，允许用户从WWW服务器传输超文本文件到本地浏览器。

HTTP的主要功能是定义了客户端如何向服务器请求网页内容以及服务器如何响应这些请求。

HTTP协议特点

请求/响应模型

HTTP采用客户端/服务器模式工作

无状态性

HTTP不保留两次请求的任何上下文信息。

丰富媒介

支持传输多种数据格式，丰富网络交互内容。

方法灵活

GET、POST、PUT、DELETE、HEAD、OPTIONS

HTTP协议规定的内容

1

请求/响应模型

采用客户端-服务器模式，实现请求与响应的交互。

2

请求方法

定义如GET、POST等方法，匹配不同的操作需求。

3

状态码

通过数字码展示处理结果，例如200代表成功。

4

消息头

包括了关于请求或响应的附加信息，如内容类型、缓存控制等。

5

消息体

用于传输实际的数据，如HTML页面、图片或其他媒体文件。

基本HTTP交互流程

1

构造请求

客户端封装HTTP请求，包括请求行、头部以及消息体。

2

服务器处理

服务器解析请求报文，找出对应资源，生成响应。

3

客户端解析

客户端接收响应报文，展示相应的数据或执行后续操作。

不同版本的HTTP协议

HTTP 1.0

最早期的HTTP版本，提供基本功能，每次连接后立即关闭。

HTTP 1.1

引入持久连接和更丰富的缓存控制机制，优化性能。

HTTP/2

支持多路复用和头部压缩，极大提升了传输效率。

HTTP 1.0

HTTP 1.0是HTTP协议的第一个正式版本，于1996年发布。主要有以下几个特征：

连接管理：每个请求都会创建一个新的TCP连接并立即关闭，被称为非持久连接。这种"请求-响应-关闭"的模式对于资源密集型的网站来说效率低下。

流水线处理：虽然HTTP 1.0引入了近似流水线的处理，允许在单个连接上发送多个请求而无需等待响应，但它并没有得到广泛支持

缓存：HTTP 1.0提供了基本的缓存控制指令，如Expires和Pragma，但不如后续版本那样全面。

头部压缩：HTTP 1.0不支持头部压缩，导致头部信息占用较大的网络带宽。

连接管理

- **非持久连接**：在 HTTP 1.0 中，每个请求都会创建一个新的 TCP 连接，完成请求-响应后立即关闭。这意味着每次获取一个资源（如网页、图片、样式文件等）都需要进行一次新的 TCP 连接建立和关闭。这种机制称为**非持久连接**。

总结：HTTP 1.0 的连接管理模式在小型网站上勉强适用，但对于现代复杂的 Web 应用，它的效率极低。因此，HTTP 1.1 引入了**持久连接**（Connection: keep-alive）以优化连接管理。

- **TCP 连接的建立和关闭**：每次请求都需要执行三次握手来建立连接，并在响应完成后通过四次挥手关闭连接。这个过程会带来显著的延迟，尤其是在获取复杂网页时，多个资源文件需要多个连接。
- **性能影响**：对于资源密集型的网站，如包含大量图片、CSS、JavaScript 的页面，频繁的 TCP 连接建立和关闭会严重影响性能。每个请求都需要经历建立连接的时间消耗，增加了服务器负载和网络延迟。

流水线处理

- **基本概念：**流水线处理（Pipelining）允许在同一个 TCP 连接上，客户端可以在未收到第一个请求的响应之前发送多个请求。这种机制旨在减少网络延迟，因为多个请求可以并发发送，无需每次等待响应后再发送下一个请求。
 - **HTTP 1.0 的支持情况：**虽然 HTTP 1.0 开始尝试引入类似的流水线处理，但它的实现极为有限，客户端通常会等待服务器的每个响应，不能充分利用流水线带来的优势。
 - **问题：**在 HTTP 1.0 中，由于服务器响应的顺序性要求（即按请求顺序处理响应），流水线处理可能导致**队头阻塞**问题。即，某个响应处理时间较长时，后续的请求必须等待其完成

缓存

- **缓存的作用：**缓存是 HTTP 协议的关键优化机制之一，旨在减少服务器负载和网络带宽的使用。客户端和代理服务器可以通过缓存机制存储资源的副本，以便在将来相同资源请求时直接返回缓存内容，而无需重新向服务器发起请求。
- **HTTP 1.0 的缓存机制：**HTTP 1.0 提供了基础的缓存控制指令，如：
 - **Expires 头部：**用于指定资源的过期时间，客户端可以在资源未过期的时间段内直接从缓存中获取资源，而不需要重新请求服务器。
 - **Pragma 头部：**提供了客户端和代理服务器之间的缓存控制，**Pragma: no-cache** 可用于强制客户端不缓存资源。
- **问题：**
 - **过于基础：**HTTP 1.0 的缓存机制较为简单，主要依赖资源的过期时间，缺乏更精确的控制手段。
 - **不支持条件请求：**HTTP 1.0 缺少现代缓存机制中的 **ETag** 和 **Last-Modified** 这样的条件请求支持，无法在资源变更时有效验证缓存内容是否依然有效。



HTTP 1.1

持久连接

提供Keep-Alive特性，减少连接建立的成本。

缓存控制

新增缓存头部字段，提高缓存的有效性。

分块编码

分割大对象进行传输，提升了响应速度。

持久连接 (Persistent Connection)

原理

- **HTTP 1.1** 引入了**持久连接**（默认启用），也称为 **Keep-Alive** 连接。通过这种方式，客户端和服务端之间的 TCP 连接在完成一次请求后不会立即关闭，而是保持一段时间，等待后续的请求。这意味着多个请求和响应可以通过同一个连接传输，避免了每次请求都需要重新建立连接的成本。

优点

- **减少延迟**：每次 TCP 连接都要经历三次握手，这会增加延迟。而通过持久连接，多个请求可以复用同一个连接，避免了多次握手的开销，从而降低延迟。
- **减少服务器负载**：保持连接的状态减少了连接的频繁建立和关闭，降低了服务器端管理连接的开销，特别是对于资源密集型网站和高并发场景，持久连接能显著提高性能。

缓存控制 (Cache-Control)

- **缓存的作用：**客户端或代理服务器可以存储已经请求过的资源，并在后续请求中直接返回缓存的内容，而无需再次向服务器请求，降低带宽消耗，加快资源加载速度
- **HTTP 1.1 中的缓存机制：**
 - **Cache-Control 头部字段：**HTTP 1.1 引入了 Cache-Control 头部，它提供了更多的缓存指令，允许客户端和服务对缓存行为进行细粒度控制。

分块传输编码 (Chunked Transfer Encoding)

分块传输编码是 HTTP 1.1 为了解决大数据传输效率问题而引入的传输机制。它允许服务器将大数据对象分割成多个小块进行传输，而不是一次性传输整个数据对象。

原理

- 在传统的 HTTP 数据传输中，服务器需要在响应开始之前确定整个响应体的大小，并在 `Content-Length` 头部中指明。但有时候，服务器无法提前知道内容的总大小（例如，动态生成的内容或实时流媒体传输）。
- **分块传输编码**允许服务器在发送响应体时不需要预先知道响应体的长度。相反，服务器可以将响应体分割成若干块，每一块包含其长度信息，然后逐块发送给客户端。客户端接收到这些分块后可以依次解析并处理数据，直到接收到长度为 `0` 的块，表示传输结束。

HTTP/2

多路复用

通过单一连接并行处理多个请求，效率大幅提升。

二进制分帧

使用**二进制格式**而不是文本格式传输数据

将数据分解成更小的单元，称为**帧**

头部压缩

引入了 HPACK 压缩算法对头部进行压缩，降低传输开销。请求和响应头被分帧传输，不同请求之间相同的头部只需发送一次



HTTP协议的演进

HTTP的演进过程显示了互联网技术的不断迭代，并且随着用户对网速需求的增长，HTTP/2的改进对提升Web性能有显著影响。



HTTPS

HTTPS \approx HTTP + 加密层 (SSL/TLS)

安全传输

HTTPS通过SSL/TLS层对数据加密，保障传输安全。

身份验证

HTTPS使用证书验证服务器身份，提升信任度。

信任标志

带有安全锁的HTTPS，表示连接更加可靠。

HTTPS的优势

与HTTP相比，HTTPS在现代Web应用中更受推崇，特别是对于敏感数据的交换，提供了加密和认证的强力保障。

1

数据加密

确保用户信息在传输过程中不被泄露。

2

服务器认证

防止用户接触到假冒网站。

3

信任加强

借助安全标志提高用户信任。

HTTP请求结构

请求行

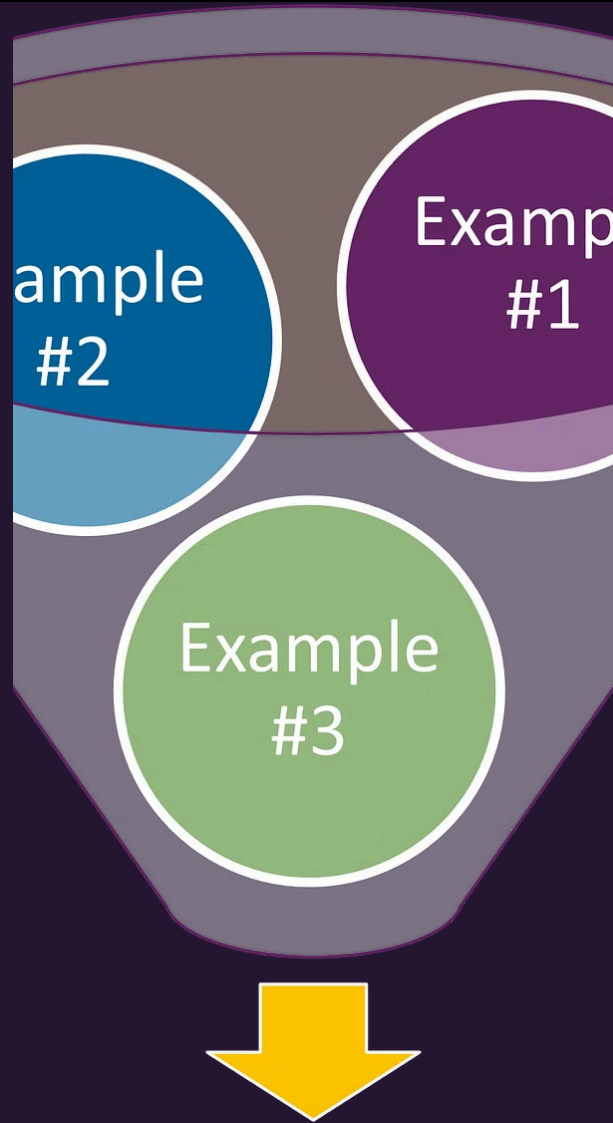
描述HTTP方法、URI和版本的重要部分。

请求头

提供请求的附加信息，影响处理方式。

请求体

可选部分，常用于POST请求中传输数据。



请求行 示例

例如: "GET /index.html HTTP/1.1", 代表用GET方法请求/index.html资源, 遵循HTTP 1.1版本。

1

HTTP方法

识别请求操作类型, 如GET、POST等。

2

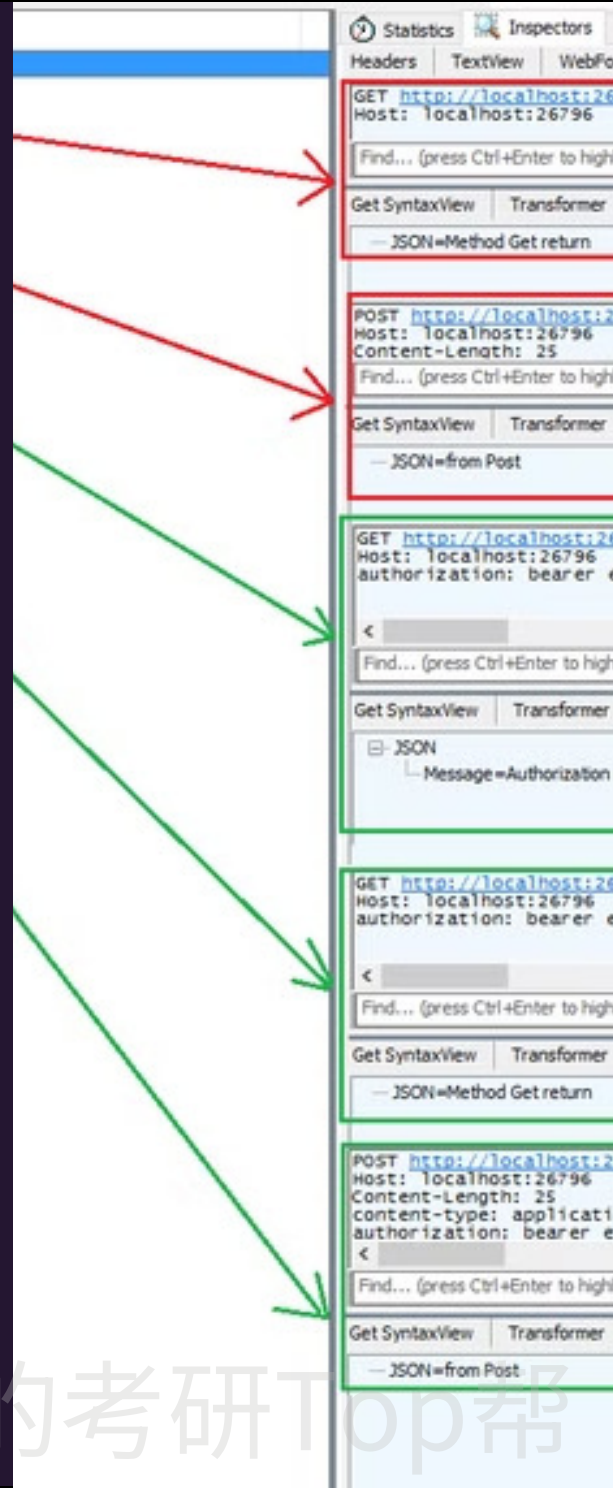
URI

指明请求资源的路径。

3

HTTP版本

表明所使用的HTTP协议版本。



请求头 示例

```
Content-Type: application/json  
User-Agent: Mihoyo/5.0
```

1

Content-Type

定义发送数据的格式，如application/json。

2

User-Agent

描述请求发起者的环境，如浏览器类型。

请求体 使用场景

请求体在POST请求（如表单提交数据）中使用较多，取决于Content-Type头部字段的定义。

1

携带数据

传输需要提交的信息内容，如JSON或文件数据。

2

与请求头配合

依据请求头部的描述来处理请求体数据。

GET与POST请求

GET请求

请求获取服务器资源，参数出现在URL中。

POST请求

提交数据给服务器，数据包含在请求体内。

GET请求:

用于请求服务器上的资源。

参数通常附加在URI后，如/search?q=yuanshen，表示请求搜索关键词为"yuanshen"的结果。

```
GET /search?q=yuanshen HTTP/1.1  
Host: www.example.com  
User-Agent: Mihoyo/5.0
```

HTTP方法是GET，表示请求获取资源。

URI是/search?q=openai，包含了参数q，其值为openai，表示搜索关键词为"openai"的结果。

Host字段指定了服务器的主机名。 User-Agent字段表示请求来自Mihoyo浏览器。

POST请求 结构

- 用于向服务器提交数据，如表单提交。
- 数据放置在请求体中，不出现在URI中。

```
POST /submit-form HTTP/1.1
Host: www.example.com
User-Agent: Mihoyo/5.0
Content-Type: application/x-www-form-urlencoded
Content-Length: 21
```

```
username=johndoe&password=12345
```

请求解析

请求解析涉及提取方法、URI以及处理请求体中的数据，对于POST请求特别重要。

1

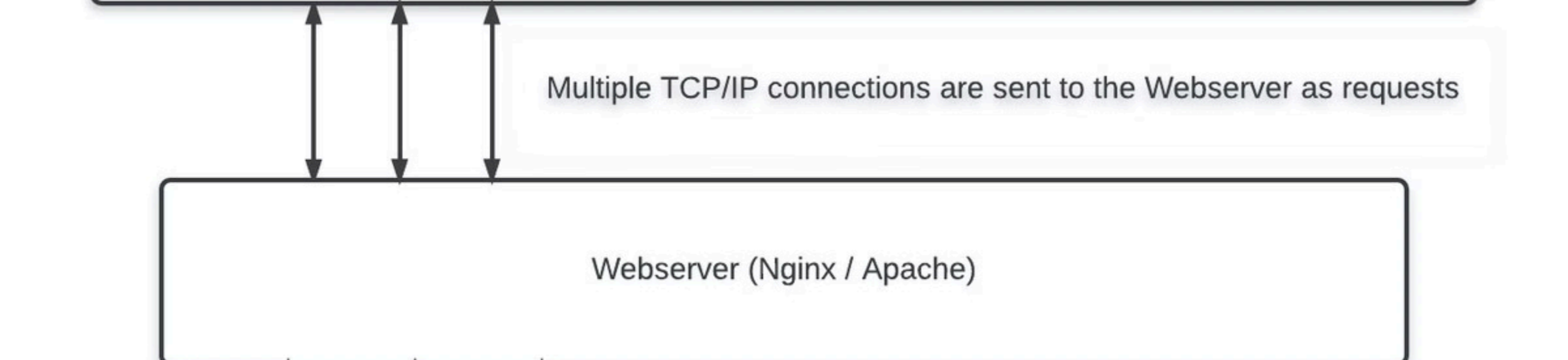
解析方法

提取请求中的HTTP方法和资源路径。

2

处理数据

对POST请求体内容进行提取和使用。



Multiple TCP/IP connections are sent to the Webserver as requests

The diagram illustrates a client sending multiple requests to a webserver. Three vertical double-headed arrows on the left represent the connections. The webserver is represented by a large white box with a black border. The text 'Multiple TCP/IP connections are sent to the Webserver as requests' is in a light gray box above the webserver.

Webserver (Nginx / Apache)

实例：解析HTTP请求

1

请求行解析

确定请求类型及资源路径。

2

请求头解析

额外信息，如服务器主机名及客户端信息。

M学长的考研Top帮

实例

```
GET /index.html HTTP/1.1
Host: www.example.comUser-Agent: Mihoyo/5.0
```

- 解析请求：
 - 请求行为GET /index.html HTTP/1.1，表示这是一个GET请求，请求的资源为/index.html。
 - 请求头包含Host: www.example.com和User-Agent: Mihoyo/5.0，分别表示请求的服务器地址和客户端信息。
 - 由于是GET请求，因此没有请求体。

HTTP响应结构

状态行

包括HTTP版本、状态码和状态消息，为响应的开头。

响应头

附加信息如Content-Type和Server，说明响应的性质。

响应体（可选）

响应体包含了服务器向客户端发送的实际数据

状态行

状态行是HTTP响应的第一部分，包含了HTTP版本、状态码和状态消息。其格式如下：

```
HTTP版本 状态码 状态消息  
HTTP/1.1 200 OK
```

- HTTP版本：标识了服务器返回响应时所使用的HTTP协议版本。
- 状态码：三位数字代码，指示请求处理的结果。如200表示成功，404表示未找到资源，500表示服务器内部错误等。
- 状态消息：对状态码的简短描述，如"OK"、"Not Found"等。

响应头

Content-Type: text/html; charset=UTF-8

Cache-Control: max-age=3600

Server: Apache/2.4.41

- Content-Type：指定了响应体的内容类型及编码。
- Cache-Control：指导客户端如何缓存响应内容。
- Server：标明处理请求的服务器及其软件版本。

响应体 (Response Body, 可选)

响应体包含了服务器向客户端发送的实际数据，根据响应头中的Content-Type字段解析。响应体可以是HTML文档、JSON数据、图片或其他任何类型的数据。

GET与POST响应示例

GET响应

返回请求的资源，如HTML页面。

POST响应

确认信息提交，返回操作结果，如创建资源状态。

GET

```
GET /index.html HTTP/1.1
Host: www.example.com
User-Agent: Mihoyo/5.0
HTTP/1.1 200 OK
Content-Type: text/html; charset=UTF-8
Content-Length: 1024
```

```
<!DOCTYPE html...
```

```
POST /submit-form HTTP/1.1
Host: www.example.com
User-Agent: Mihoyo/5.0
Content-Type: application/x-www-form-urlencoded
Content-Length: 21
username=johndoe&password=12345
```

```
HTTP/1.1 201 Created
Location: /profile/johndoe
Content-Type: application/json
Content-Length: 56
```

```
{"status": "success", "message": "Form submitted"}
```

响应解析

1

解析状态码和状态消息

从状态行中提取相关信息，了解请求执行的成功与否。

2

处理响应体

根据响应头中的Content-Type字段确定响应体格式，并从中获取具体数据进行进一步处理。例如，如果是JSON格式，则将其解析为JSON对象。

实操内容：解析HTTP请求和处理

1

解析HTTP请求

- 从接收到的HTTP请求中解析出请求方法和URI。
- 对于POST请求，解析请求体。

2

处理GET和POST请求

- 针对GET请求，提取URI参数并进行相应处理。
- 对于POST请求，获取请求体数据。

3

生成并返回响应

- 根据处理结果，生成HTTP响应。
- 设置响应头和响应体，并将响应返回给客户端。

实战示例：简单HTTP服务器

1

服务器初始化

- 创建TCP socket，绑定端口，监听。

2

解析HTTP请求

- 接收客户端HTTP请求。
- 使用parseHttpRequest函数解析请求方法和URI。

3

处理请求

- 根据URI调用相应处理函数。
- 针对GET/POST执行逻辑。

4

发送响应

- 构建HTTP响应，发送回客户端。

6

- Specifies architecture-independent data transfer format
- Encodes and decodes data; Encrypts and decrypts data; Compresses and decompresses data

Layer 5: Session Layer

- Manages user sessions and dialogues
- Controls establishment and termination of logical links between users
- Reports upper layer errors

5

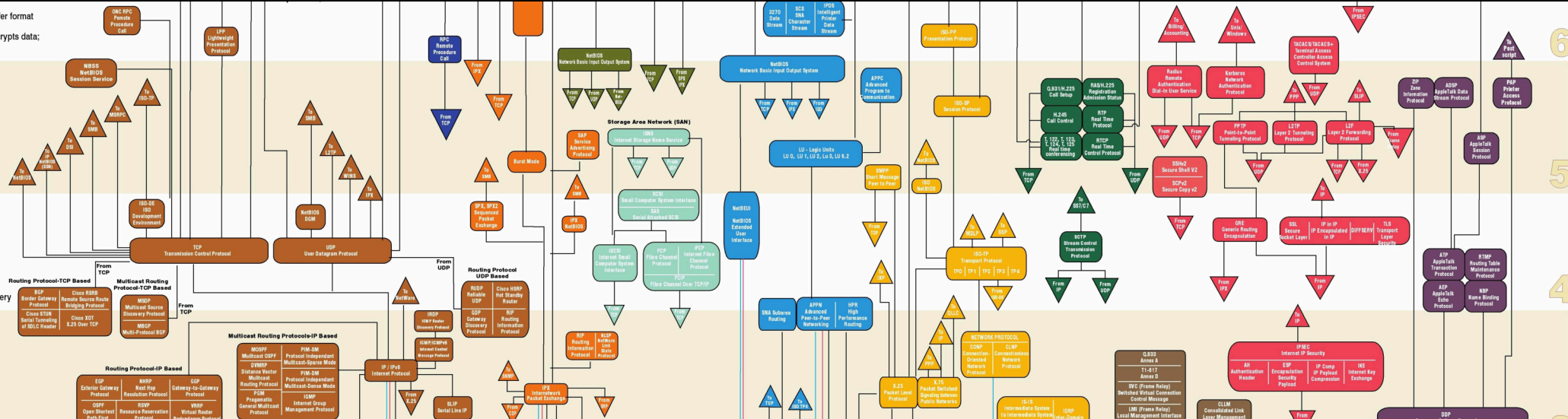
Layer 4: Transport Layer

- Manages end-to-end message delivery in network
- Provides reliable and sequential packet delivery through error recovery and flow control mechanisms
- Provides connectionless oriented packet delivery

4

Layer 3: Network Layer

- Determines how data are transferred among network devices
- Routes packets according to unique network addresses
- Provides flow and congestion control to



谢谢大家

M学长的考研Top帮