# 6.The Web

## PPT

## HTTP

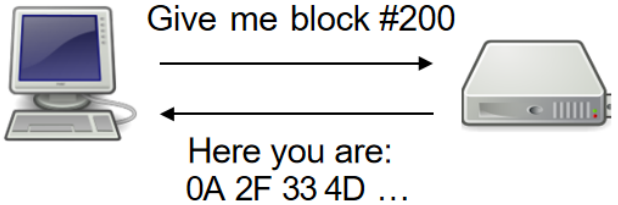
HyperText Transfer Protocol
Tim Berners-Lee, CERN, 1989
version 1.1, RFC 7230-7235

超文本传输协议（HTTP）是一种用于在网络上传输超文本文档的应用层协议。它是万维网的基础之一，允许客户端和服务器之间进行通信和数据交换。HTTP 协议由著名的计算机科学家蒂姆·伯纳斯-李（Tim Berners-Lee）在 1989 年在欧洲核子研究组织（CERN）开发。

HTTP 协议的第一个版本是 1.0，后来在 1997 年发布了版本 1.1，目前广泛使用。HTTP 协议的基本工作原理是客户端（例如浏览器）向服务器发送请求，服务器收到请求后处理并返回相应的资源（例如网页、图像、视频等）。这种请求-响应模型使得在全球范围内获取和共享信息变得更加简单和高效。

HTTP 协议的工作方式和规范被记录在 RFC（请求评论文档）7230-7235 中，这些文档规定了 HTTP 协议的各个方面，包括消息格式、请求方法、状态代码等。

### Protocols



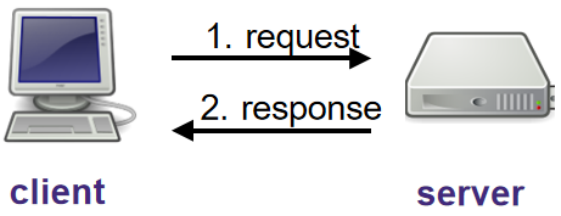
协议是指计算机系统中用于进行通信和数据交换的规则和约定。它定义了数据如何在不同设备或系统之间传输、解释和处理。

在计算机领域，有许多不同类型的协议，涵盖了各种用途和场景。以下是一些常见的协议类型：

1. **网络协议**：用于在计算机网络中传输数据的协议，如 TCP/IP、HTTP、FTP、SMTP 等。
2. **安全协议**：用于确保数据安全性和隐私保护的协议，如 SSL/TLS、IPsec 等。
3. **应用层协议**：用于应用程序之间通信和数据交换的协议，如 HTTP、SMTP、POP3、IMAP 等。
4. **传输层协议**：用于在网络中提供端到端数据传输的协议，如 TCP、UDP 等。
5. **数据链路层协议**：用于在物理链路上传输数据的协议，如 Ethernet、PPP 等。
6. **路由协议**：用于在网络中路由数据包的协议，如 OSPF、BGP 等。

这些协议共同构成了计算机网络和系统之间的基础通信基础设施。通过遵循协议规范，不同的设备和系统能够相互通信和协作，实现数据交换和信息共享。

## Client—Server




## Server

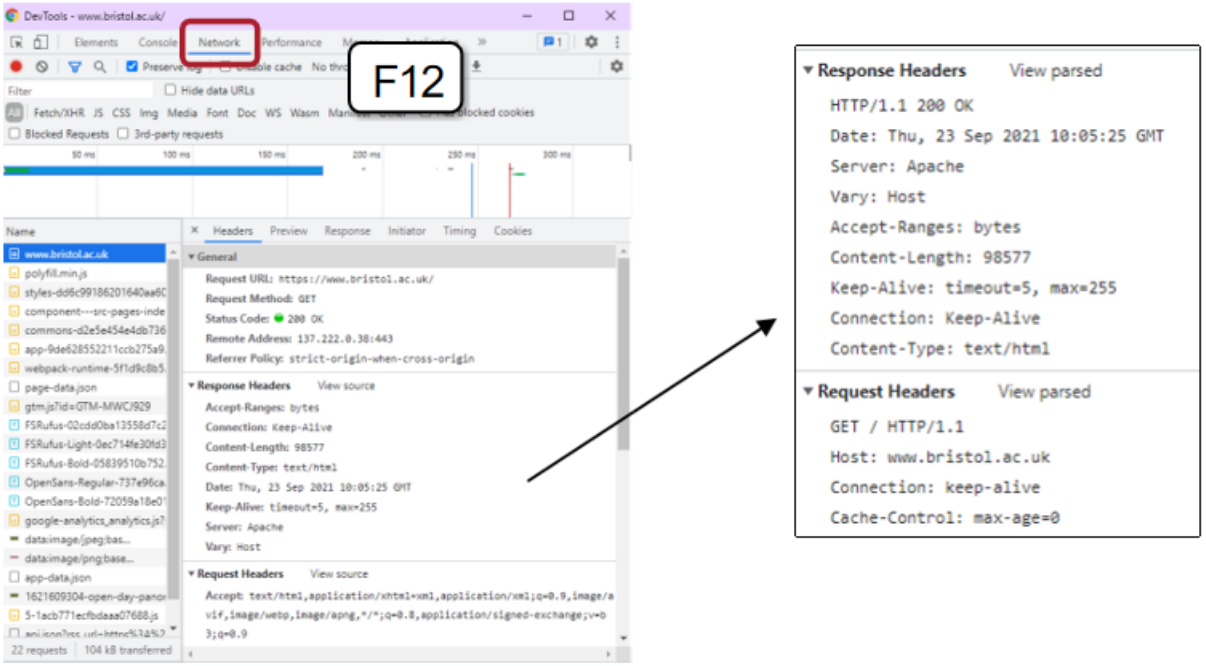
```
Python
```

```
while (1) {
req = read();
resp = serve(req);
write(resp);

}
```

1. **读取请求（req = read()）：** 服务器从网络连接中读取客户端发送的请求。这个请求可能是获取特定资源、执行特定操作等。2. **处理请求并生成响应（resp = serve(req)）：** 服务器对收到的请求进行处理，执行相应的操作，并生成一个响应。这个响应包含了请求的结果或所需资源的数据。3. **发送响应（write(resp)）：** 服务器将生成的响应发送回客户端，以完成请求–响应周期。

## HTTP




## HTTP Request

```
HTTP Request
```

```
GET   /index.html  HTTP/1.1
Host:  www.bristol.ac.uk  Connection:  close
```

**1.请求行（Request Line）：** 第一行是请求行，包含了请求的方法、请求的资源路径以及使用的 HTTP 协议版本。在这个例子中，请求方法是 GET，表示请求获取特定资源；请求的资源路径是/index.html，表示请求获取服务器上的 index.html 文件；HTTP 协议版本为 1.1。

2. **请求头部（Request Headers）：** 请求头部包含了额外的信息，用于描述请求的细节和附加条件。在这个例子中，Host 头部指定了请求的目标主机为 www.bristol.ac.uk，Connection 头部指定了连接关闭后立即关闭连接。

## Methods

1.**GET：** 用于请求指定资源的数据。GET 请求不会修改服务器上的资源，而只是请求数据。例如，当您在浏览器中访问网页时，浏览器发送的请求就是 GET 请求。

2. **HEAD：** 与 GET 方法类似，但服务器只返回请求资源的响应头部信息，而不返回实际资源的内容。HEAD 方法通常用于获取资源的元数据，如文件大小、类型等，而不需要获取实际内容。

3. **POST：** 用于向服务器提交数据，通常用于提交表单数据或上传文件。POST 请求可能会修改服务器上的资源，如创建新资源、更新现有资源等。

4. **PUT：** 用于向服务器上传新的资源，或者替换指定 URL 位置的资源。PUT 请求会完全替换服务器上的资源为请求中提供的新内容。

5. **DELETE：** 用于请求服务器删除指定的资源。DELETE 请求会从服务器上删除指定 URL 位置的资源。

## HTTP Response

Python

```
HTTP/1.1  200  OK
Content-Type:  text/html; charset=UTF-8
Content-Length:  1009

<!DOCTYPE  html>
<html  lang="en">
```

1.**状态行（Status Line）：** 第一行是状态行，包含了 HTTP 协议的版本号、状态码和状态消息。在这个例子中，HTTP 协议版本是 1.1，状态码是 200，表示请求成功；状态消息是 OK，表示请求已成功处理。

2. **响应头部（Response Headers）：** 接下来是一系列的响应头部，用于描述响应的详细信息。每个头部包含了一个属性和对应的值，以冒号分隔。在这个例子中，包含了 Content-Type、Content-Length 等头部，分别描述了响应的内容类型和内容长度。

3. **空行（Blank Line）：** 在所有头部之后是一个空行，用于表示头部的结束。

4. **响应主体（Response Body）：** 最后是响应的主体部分，包含了请求的实际内容。在这个例子中，主体是一个 HTML 文档，用<!DOCTYPE html>标签开始，然后是一个<html>标签，表示 HTML 页面的开始部分。

## Response codes

1xx: 信息性状态码（Informational Status Codes）：表示服务器已经收到请求，并且正在处理中。这些状态码不会直接影响请求的结果。

2xx: 成功状态码（Success Status Codes）：表示服务器已成功接收、理解并接受请求。最常见的成功状态码是 200 OK，表示请求已成功处理并返回所请求的资源。

3xx: 重定向状态码（Redirection Status Codes）：表示客户端需要采取进一步的操作才能完成请求。例如，301 Moved Permanently 状态码表示请求的资源已永久移动到新的 URL，客户端需要使用新的 URL 重新发起请求。

4xx: 客户端错误状态码（Client Error Status Codes）：表示客户端发送的请求包含错误或无法被服务器处理。最常见的客户端错误状态码包括 400 Bad Request（请求语法错误）、403 Forbidden（服务器拒绝请求）和 404 Not Found（未找到请求

的资源）。

5xx: 服务器错误状态码（Server Error Status Codes）：表示服务器在处理请求时发生了错误。这些状态码指示服务器无法完成请求。最常见的服务器错误状态码是 500 Internal Server Error，表示服务器遇到了意外的错误。

## Content-type

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

text/plain,  text/html,   ...
image/jpeg,  application/pdf,  video/mp4   ...
First priority for the browser!
```

Content-Type 是 HTTP 头部的一部分，用于指示发送给客户端的资源的 MIME 类型（Multipurpose Internet Mail Extensions）。它告诉客户端如何解释接收到的数据，并且通常由一个主类型（如 text、image、audio、video、application 等）和一个子类型（如 html、plain、jpeg、pdf、mp4 等）组成。

例如，Content-Type: text/html; charset=UTF-8 表示发送给客户端的数据是 HTML 文档，并且使用 UTF-8 字符编码进行编码。

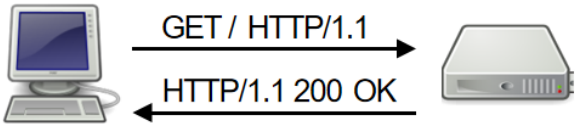
不同的 Content-Type 指示了不同类型的数据。一些常见的 Content-Type 包括：

- text/plain：纯文本数据，不包含任何格式或标记。
- text/html：HTML 文档，用于网页内容。
- image/jpeg：JPEG 格式的图像文件。
- application/pdf：PDF 文档。
- video/mp4：MP4 格式的视频文件。

浏览器在接收到 HTTP 响应时会检查 Content-Type 头部，并根据该头部来决定如何处理接收到的数据。例如，如果 Content-Type 是 text/html，浏览器会将接收到的数据解释为 HTML 文档并在浏览器窗口中显示出来。不同类型的数据可能会由不同的处理程序来处理，以确保用户可以正确地查看或使用接收到的资源。

## Cookies

## Stateless



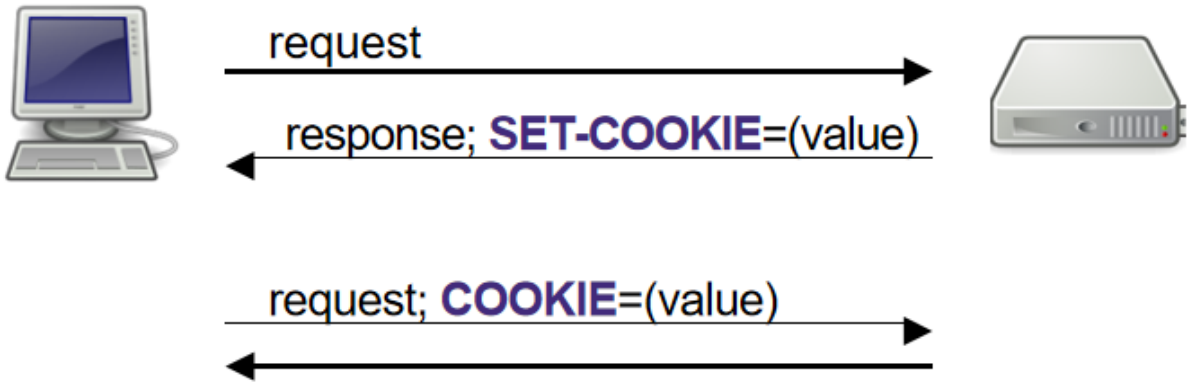

在计算机科学和网络通信中，"无状态"（Stateless）是指系统或协议在处理每个请求时不会保存任何关于前一次请求的信息或状态。换句话说，每个请求都被视为独立的、不依赖于之前的请求。

在 HTTP 协议中，"无状态"的特性意味着每个 HTTP 请求都是独立的，服务器在处理请求时不会存储任何与之前请求相关的信息。这样的设计有利于提高系统的可伸缩性和性能，并简化了服务器的实现和维护。由于 HTTP 协议是无状态的，服务器在接收到每个请求时都需要对请求进行完全处理，并且不会记住之前请求的任何信息。

例如，当客户端发送一个 HTTP 请求（如 GET / HTTP/1.1），服务器会返回一个 HTTP 响应（如 HTTP/1.1 200 OK），并在处理完该请求后立即忘记所有关于该请求的信息。无状态的特性使得每个请求都是独立的，服务器可以轻松地处理大量的并发请求，而无需考虑请求之间的关系或顺序。

虽然 HTTP 是无状态的，但通过使用会话（Session）和 Cookie 等机制，可以在一定程度上实现状态的管理，使得服务器能够识别特定用户的请求，并维护用户的会话状态。但即使使用这些机制，HTTP 仍然是无状态的，因为服务器在每个请求处理完后都会立即忘记相关信息。

## Cookie protocol




(RFC 6265)

HTTP Cookie 协议是一种用于在客户端和服务器之间传递和存储用户状态信息的协议。它允许服务器在客户端的浏览器中存储和检索特定于用户和网站的数据，以便在请求之间保持用户的持续状态。基本流程如下：

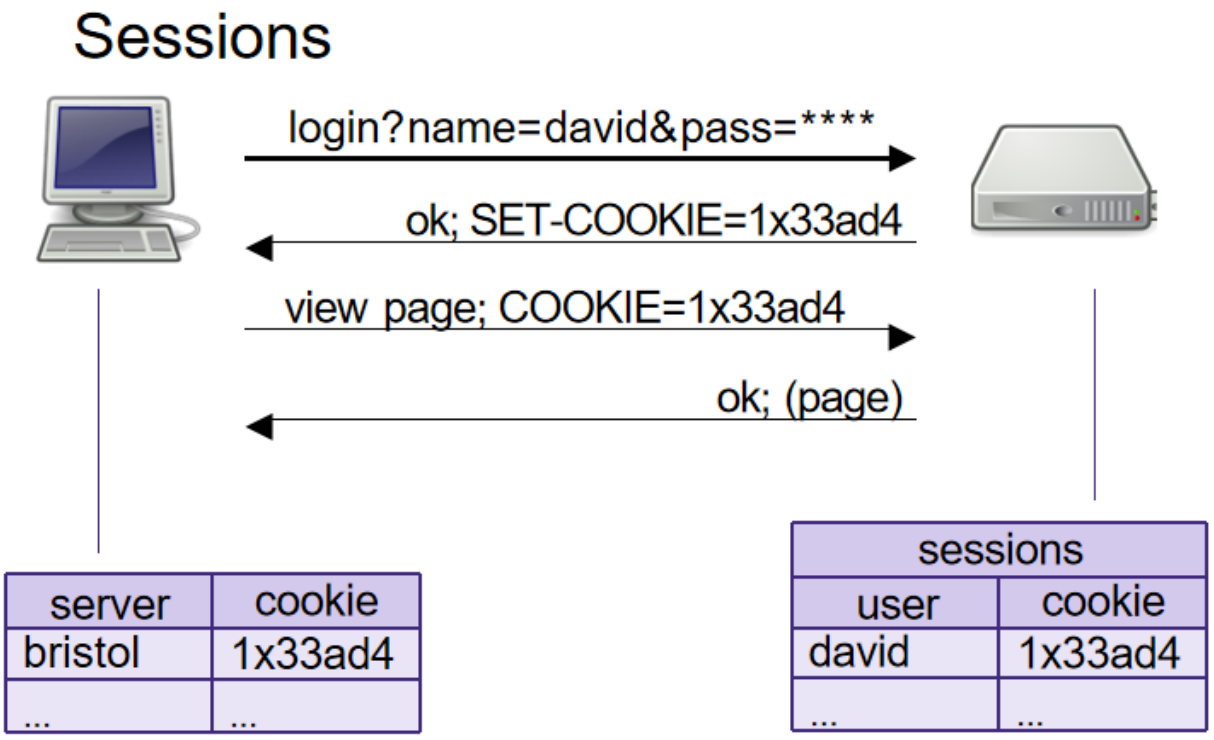
1. **请求（Request）：** 当客户端（例如浏览器）向服务器发送 HTTP 请求时，它可以将存储在本地的 Cookie 信息附加到请求中。这些 Cookie 信息在以前的服务器响应中设置，并由服务器定义和管理。

2. **响应（Response）：** 当服务器接收到客户端的请求并生成 HTTP 响应时，它可以使用 Set-Cookie 标头来在响应中设置 Cookie。这样，服务器可以向客户端发出指令，要求客户端在以后的请求中提供特定的 Cookie。

3. **请求（Request）：** 在客户端的后续请求中，浏览器会自动将存储在本地的 Cookie 信息附加到请求中的 Cookie 标头中。这样，服务器可以根据这些 Cookie 信息来识别特定的用户并维护用户的状态。

Cookie 的值通常包含一些关键信息，如会话标识符、用户首选项、登录凭据等。这些信息可以帮助网站个性化用户体验、实现持久登录、跟踪用户行为等功能。HTTP Cookie 协议的基础规范由 RFC 6265 定义，它提供了一组规则和指南，用于服务器和客户端之间的 Cookie 交换和管理。

## Sessions




会话（Sessions）是指在网络应用中跟踪用户的一系列相关请求和响应的过程。它允许服务器在用户访问网站时保持用户的状态信息，并在用户的不同请求之间保持持久性。

通常，会话通过使用会话标识符（Session ID）来实现。会话标识符是服务器在用户登录时分配给用户的唯一标识符，通常存储在 Cookie 中。一旦用户成功登录，服务器会在响应中发送一个包含该会话标识符的 Cookie，并在之后的每个请求中接收到该 Cookie，从而识别特定的用户。

在您提供的示例中，当用户通过登录页面提供用户名和密码时，服务器可能会验证用户的凭据，并在成功验证后向客户端发送一个包含会话标识符的 Cookie。然后，客户端在之后的每个请求中都会将该会话标识符作为 Cookie 发送给服务器，以便服务器识别用户并保持用户的状态。

在服务器端，会话状态信息通常存储在会话存储中，可以是内存、数据库或其他持久性存储。服务器使用会话标识符来检索与特定会话相关联的状态信息，并根据需要更新或操作这些信息。

通过使用会话，网站可以实现用户登录、个性化用户体验、跟踪用户活动、保持用户登录状态等功能，从而提高用户的交互性和体验。

### Tracking cookies

跟踪 Cookie 是一种在用户浏览网页时由网站设置的小型文本文件，用于跟踪用户的在线活动和行为。这些 Cookie 包含了用于识别特定用户的唯一标识符，通常是会话标识符或持久性标识符。

## The Internet

### Network stack

网络堆栈（Network stack），也称为网络协议栈，是操作系统中负责处理网络通信的一系列软件组件和协议集合。它负责在计算机网络中实现数据的传输、路由和通信。网络堆栈通常由多个层次组成，每个层次都有特定的功能和责任，它们相互协作以确保数据在网络中的有效传输。通常，网络堆栈遵循 OSI（Open Systems Interconnection）或 TCP/IP（Transmission Control Protocol/Internet Protocol）模型中定义的不同层次结构。

1. **物理层（Physical Layer）：** 负责传输比特流，将数据从一个计算机传输到另一个计算机。常见的协议包括以太网、Wi-Fi、蓝牙等。

2. **数据链路层（Data Link Layer）：** 提供了直接的节点间通信，负责将数据帧从一个物理设备传输到相邻的物理设备。常见的协议包括以太网、PPP（Point-to-Point Protocol）等。

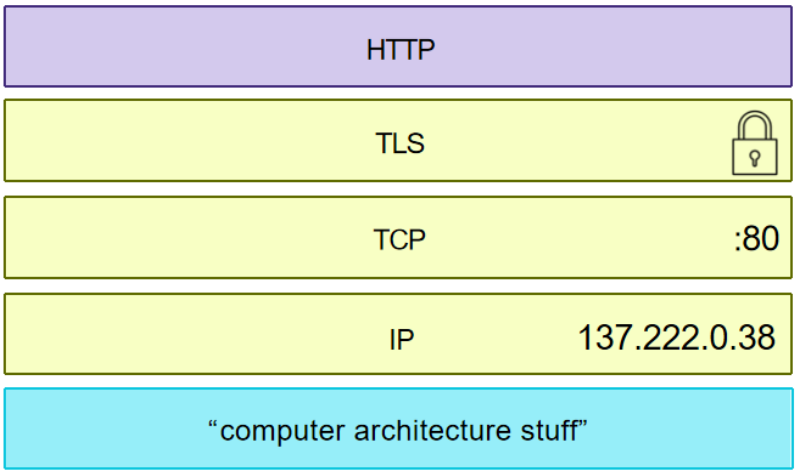
3. **网络层（Network Layer）：** 负责在不同网络之间进行路由和转发，使数据包从源主机传输到目标主机。常见的协议包括 IP（Internet Protocol）、ICMP（Internet Control Message Protocol）等。

4. **传输层（Transport Layer）：** 提供端到端的数据传输服务，负责数据的可靠传输和流量控制。常见的协议包括 TCP（Transmission Control Protocol）、UDP（User Datagram Protocol）等。

5. **会话层（Session Layer）：** 管理用户会话和连接状态，负责建立、维护和终止数据传输的会话。在 TCP/IP 模型中，该功能通常与传输层合并。

6. **表示层（Presentation Layer）：** 负责数据的格式化、加密和压缩，确保数据在不同系统之间的兼容性。在 TCP/IP 模型中，该功能通常与应用层合并。

7. **应用层（Application Layer）：** 提供用户和应用程序访问网络服务和资源的接口。常见的协议包括 HTTP（Hypertext Transfer Protocol）、FTP（File Transfer Protocol）、SMTP（Simple Mail Transfer Protocol）等。




这些是计算机网络中常见的协议，它们构成了网络通信的基础：

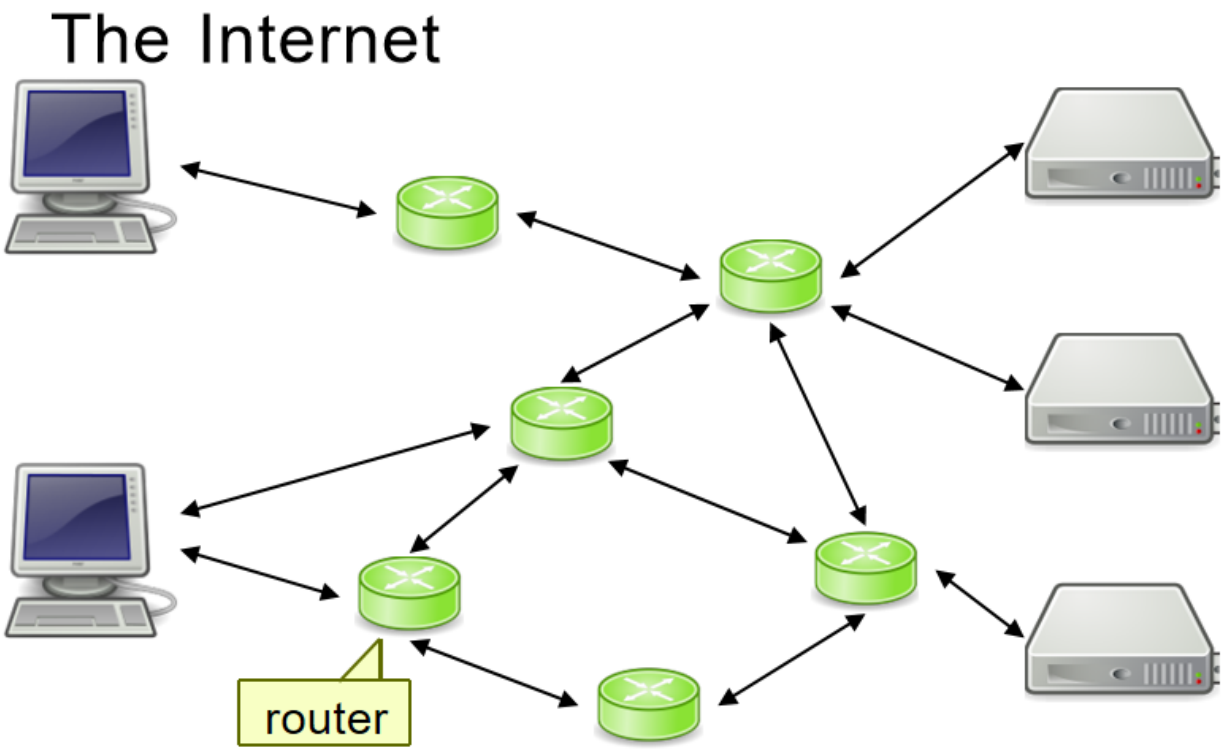
1. **HTTP（Hypertext Transfer Protocol）：** 是一种用于传输超文本数据（如网页、图片、视频等）的应用层协议。它是基于请求-响应模型的，客户端发送 HTTP 请求给服务器，服务器响应相应的数据。HTTP 被广泛用于万维网（World Wide Web）中，是 Web 浏览器和 Web 服务器之间通信的主要协议。

2. **TLS（Transport Layer Security）：** 是一种安全传输协议，用于在网络通信中加密数据并确保通信的安全性和私密性。TLS 协议在传输层提供了加密和身份验证的功能，它的前身是 SSL（Secure Sockets Layer）协议。TLS 常用于保护 Web 浏览器和服务器之间的 HTTP 通信，以及其他网络应用的通信。

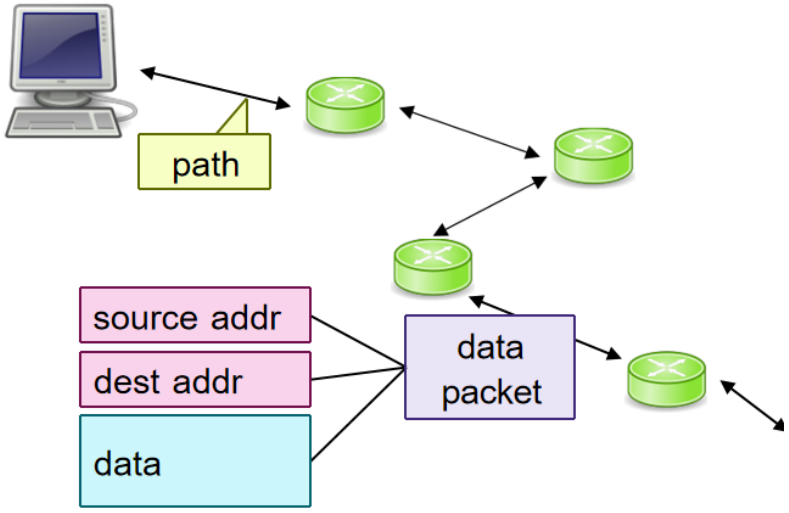
3. **TCP（Transmission Control Protocol）：** 是一种可靠的、面向连接的传输层协议，用于在网络中传输数据。TCP 通过建立连接、分段、重传、确认和流量控制等机制来确保数据的可靠传输。它是 HTTP、FTP、SMTP 等应用层协议的基础，在互联网中被广泛使用。

4. **IP（Internet Protocol）：** 是一种网络层协议，用于在计算机网络中进行数据包的路由和传输。IP 协议定义了数据包的格式、寻址和路由规则，它允许数据在不同网络之间进行传输。IPv4 和 IPv6 是目前最常用的两个 IP 协议版本，它们被用于 Internet 上的数据传输和通信。

## The Internet




## IP: Internet Protoco137.222.0.38




“路径”（path）通常在网络通信中指的是数据包从源节点传输到目标节点所经过的一系列网络节点或设备。这些节点构成了数据包在网络中的传输路径。

在互联网中，数据包通常会经过多个路由器、交换机和其他网络设备，沿着一条路径从发送方传输到接收方。这个路径可能会经过多个网络和子网，可能会经过多个 ISP（Internet Service Provider，互联网服务提供商）的网络。

路径的选择是由网络中的路由器根据路由表和路由协议决定的。路由器会根据目标地址和当前网络拓扑，选择最佳的路径来转发数据包。在选择路径时，路由器会考虑到网络拥塞、延迟、带宽、成本等因素，以确保数据包能够以最快、最稳定的方式到达目标节点。

在网络通信中，数据包（packet）是数据传输的基本单位，它包含了要传输的实际数据以及与传输相关的元数据。数据包通常由两部分组成：

数据（Data）： 这部分包含了要传输的实际信息，可以是文本、图像、音频、视频等任何形式的数据。数据的内容由发送方提供，并在传输过程中被封装在数据包中。

包头（Header）： 这部分包含了关于数据包的控制信息和元数据，用于帮助路由器和网络设备正确地处理和传输数据包。包头通常包括了源地址、目标地址、序列号、校验和等字段，这些字段在网络中起着重要的作用，用于确定数据包的传输路径、检测传输错误等。
源地址（Source Address）： 源地址指示了数据包的发送方的网络地址。它通常是发送方设备的唯一标识符，用于告知接收方数据包的来源。源地址可以是设备的 IP 地址、MAC 地址或其他网络标识符，取决于网络协议和网络层次。

目标地址（Destination Address）： 目标地址指示了数据包的接收方的网络地址。它指定了数据包应该传输到的目标设备或目标网络。目标地址也可以是设备的 IP 地址、MAC 地址或其他网络标识符。

IP（Internet Protocol）是一种网络层协议，用于在计算机网络中进行数据包的路由和传输。在 IP 协议中，每个设备都被分配一个唯一的 IP 地址，用于标识设备在网络中的位置。IP 地址由一系列数字组成，通常以点分十进制（IPv4）或冒号分隔的十六进制（IPv6）表示。

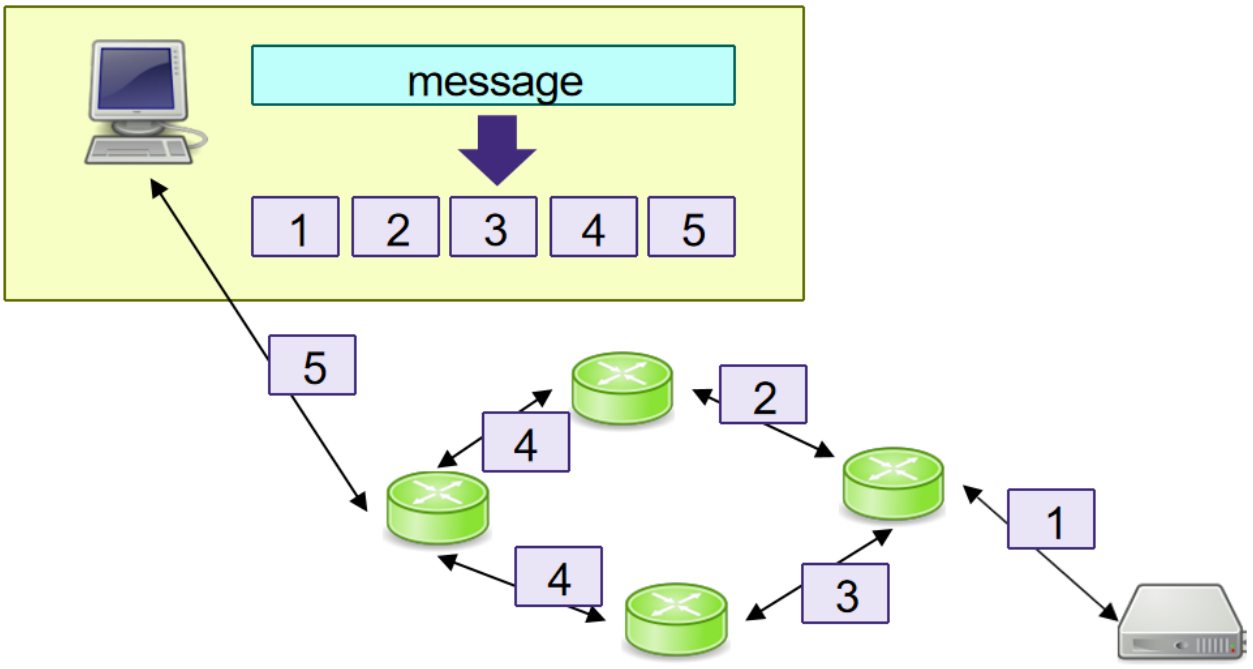
在您提供的示例中，"137.222.0.38" 是一个 IPv4 地址。IPv4 地址由四个十进制数字

组成，每个数字的取值范围为 0 到 255 之间，表示了设备在网络中的唯一位置。

当数据包从一个设备传输到另一个设备时，数据包的源地址和目标地址将分别设置为发送方和接收方的 IP 地址。通过 IP 地址，数据包可以在网络中进行准确的路由和传输，确保数据能够从发送方成功地传输到接收方。

因此，"137.222.0.38"是一个 IPv4 地址，代表了某个设备在网络中的唯一标识。

## TCP: Transmission Control Protocol




四次握手（Four-way Handshake）是 TCP 连接的关闭过程，用于确保通信双方都能够安全、有效地关闭连接。四次握手的过程如下：

1. **第一步（FIN_WAIT_1）：** 主动关闭方（通常是客户端）向被动关闭方（通常是服务器）发送一个 FIN（Finish）报文段，表示它已经完成了数据的发送，并且希望关闭连接。此时，主动关闭方进入 FIN_WAIT_1 状态，等待对方的确认。

2. **第二步（CLOSE_WAIT）：** 被动关闭方接收到 FIN 报文段后，确认收到，并向主动关闭方发送一个 ACK（Acknowledgement）报文段，表示已经收到了关闭请求。此时，被动关闭方进入 CLOSE_WAIT 状态，等待自己的应用程序处理完数据后再发送关闭请求。

3. **第三步（FIN_WAIT_2）：** 被动关闭方的应用程序处理完数据后，向主动关闭方发送一个 FIN 报文段，表示自己也已经完成了数据的发送，希望关闭连接。此时，被动关闭方进入 FIN_WAIT_2 状态，等待对方的确认。

4. **第四步（TIME_WAIT）：** 主动关闭方收到被动关闭方的关闭请求后，发送一个 ACK 报文段作为确认，并进入 TIME_WAIT 状态。在这个状态下，主动关闭方等待一段时间，确保对方收到了自己的确认，并且确保网络中所有的数据包都已经被处理完毕，然后才正式关闭连接。完成四次握手后，TCP 连接正式关闭，双方都释放了连接资源，并且可以重新建立新的连接。四次握手过程中的每一步都是为了确保数据的可靠传输和连接的安全关闭，以保障通信的稳定性和可靠性。

TCP（Transmission Control Protocol，传输控制协议）是一种在计算机网络中实现可靠数据传输的主要协议之一。它属于传输层，负责在网络中建立可靠的、面向连接的数据传输通道。

TCP 协议具有以下特点和功能：

可靠性： TCP 提供了可靠的数据传输机制，通过序列号、确认应答和重传等机制，确保数据在传输过程中不会丢失、损坏或重复。如果数据包在传输过程中丢失或损坏，TCP 会自动进行重传，直到数据被成功接收。

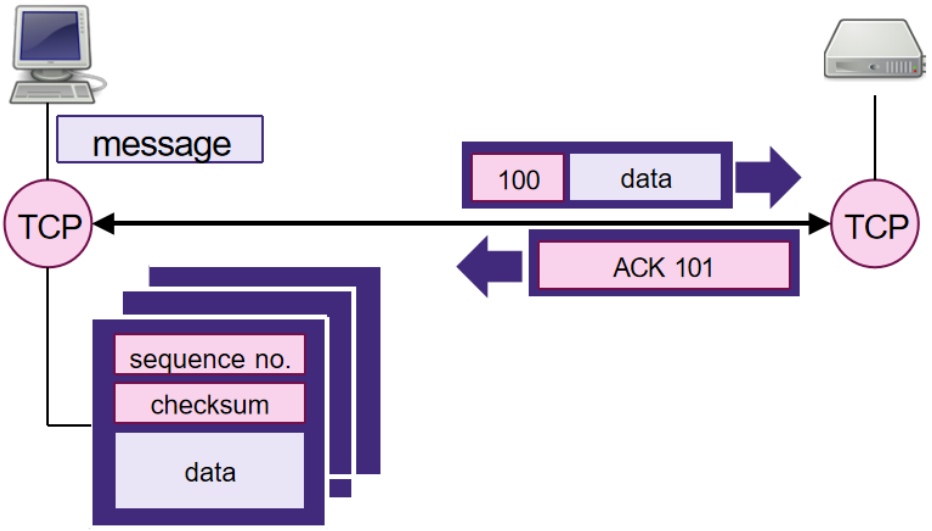
面向连接： TCP 是一种面向连接的协议，通信双方在数据传输之前需要先建立连接。连接的建立包括三次握手的过程，确保通信双方都准备好进行数据传输。连接建立后，数据传输完成后会进行四次挥手的过程，正式关闭连接。

流量控制： TCP 通过流量控制机制来调节数据传输的速度，防止发送方发送数据速度过快导致接收方无法及时处理。通过 TCP 窗口的调整，发送方和接收方可以协商合适的数据传输速度，保持通信的稳定性和平衡性。

拥塞控制： TCP 还具有拥塞控制机制，用于避免网络拥塞导致数据传输效率下降或数据丢失的情况。通过拥塞窗口的调整和慢启动等算法，TCP 可以根据网络的拥塞程度动态调整数据传输速度，确保网络的稳定性和公平性。

面向字节流： TCP 是一种面向字节流的协议，数据在传输过程中被视为连续的字节流。TCP 负责将数据分割成合适的数据包，并在接收方将数据包重新组装成原始数据流。

## TCP




在 TCP 协议中，数据包的头部通常包含了三个重要的字段：

1. **序列号（Sequence Number）：** 序列号用于标识数据包中的数据在数据流中的位置。每个数据包都有一个唯一的序列号，它指示了数据包中数据的起始位置。接收方根据序列号将数据包中的数据按顺序组装成完整的数据流。

2. **校验和（Checksum）：** 校验和是用于检测数据包在传输过程中是否发生错误的一种校验值。发送方根据数据包的内容计算出校验和，并将其添加到数据包的头部。接收方在接收到数据包后，会重新计算校验和，并与接收到的校验和进行比较，以检测数据包是否在传输过程中发生了错误。

3. **数据（Data）：** 数据字段包含了数据包要传输的实际信息。这部分数据可以是文本、图像、音频、视频等任何形式的信息。TCP 将数据分割成合适的数据包，并在网络中进行传输。这三个字段构成了 TCP 数据包的基本结构，它们保证了数据包在

传输过程中的可靠性、完整性和正确性。序列号用于保证数据包的顺序，校验和用于检测传输过程中的错误，数据字段则包含了要传输的实际信息。

TCP（Transmission Control Protocol，传输控制协议）是一种在计算机网络中用于可靠数据传输的主要协议之一。作为传输层的一部分，TCP 负责提供面向连接的、可靠的数据传输服务，以确保数据能够安全、有序地传输到目标设备。

"ACK 101" 是指 TCP 协议中的确认序号（Acknowledgment Number）为 101。在 TCP 的数据传输过程中，当接收方成功接收到数据后，会向发送方发送确认报文（ACK），其中确认序号表示接收方期待下一个数据包的序号。

这里的 "ACK 101" 意味着接收方已经成功接收到了序号为 101 的数据包，并且期待下一个数据包的序号是 102。这样的确认报文用于告知发送方数据传输的进展情况，确保数据包的可靠传输。

## TCP ports

80 HTTP
443 TLS
22 SSH
8000, 8080, … development (unofficial)

TCP 端口是 TCP 协议中用于标识不同网络应用程序或服务的数字标识符。每个 TCP 端口都与一个特定的网络应用程序或服务相关联，用于在网络中唯一标识这些应用程序或服务。以下是一些常见的 TCP 端口及其关联的应用程序或服务：

1. **端口 80：** HTTP（超文本传输协议）的默认端口。HTTP 是用于在 Web 浏览器和 Web 服务器之间传输超文本文档的协议，用于访问网页和其他 Web 资源。

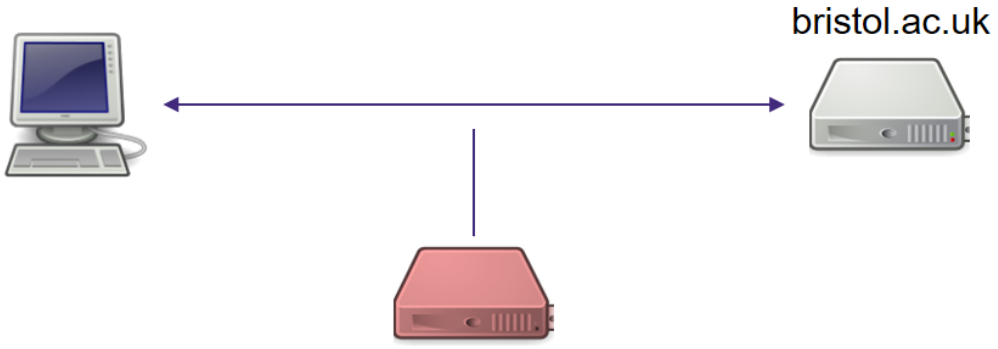
2. **端口 443：** TLS（传输层安全）的默认端口。TLS 是一种用于加密和保护数据传输的安全协议，通常用于在 Web 浏览器和 Web 服务器之间进行安全的 HTTP 通信（HTTPS）。

3. **端口 22：** SSH（安全外壳协议）的默认端口。SSH 是一种用于在网络中安全地远程登录和执行命令的协议，常用于远程管理和文件传输。

4. **端口 8000、8080 等：** 开发环境中常用的端口，用于运行开发服务器、调试应用程序等。这些端口通常是由开发人员根据需要自行选择的，不属于 TCP 协议的官方

端口范围，因此被称为非官方端口。TCP 端口的使用使得不同的网络应用程序能够在同一台计算机上共存并与网络进行通信，同时也方便了网络管理员对网络流量的控制和管理。

## TLS




TLS（Transport Layer Security，传输层安全）是一种用于保护网络通信安全的协议。它建立在 SSL（Secure Sockets Layer，安全套接字层）协议之上，用于在两个通信应用程序之间提供数据保密性和完整性。TLS 协议的主要目标是确保通信的机密性、完整性和认证性，以防止数据在传输过程中被窃听、篡改或伪装。TLS 协议的主要特点包括：

1. **数据加密：** TLS 使用加密算法对数据进行加密，以确保在传输过程中数据不会被未经授权的用户读取。它提供了对称加密、非对称加密和哈希算法等多种加密技术，用于保护通信中的数据。

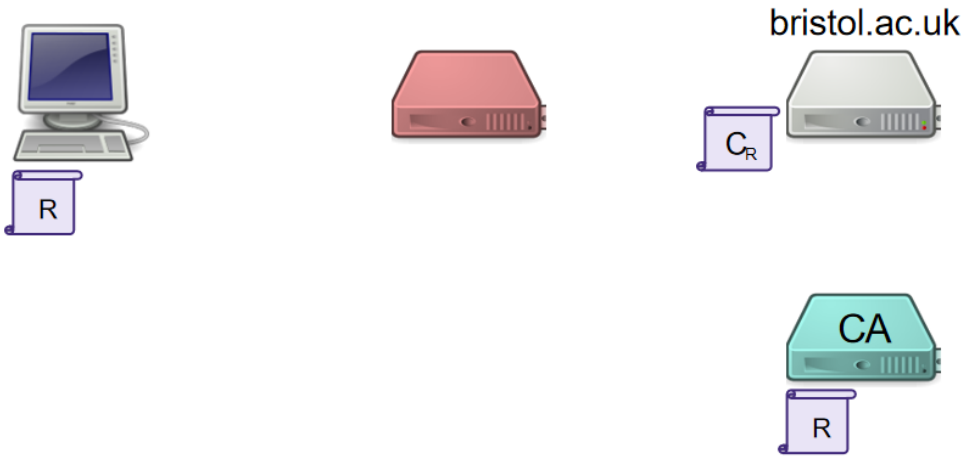
2. **数据完整性：** TLS 使用消息认证码（MAC）等技术来检测数据在传输过程中是否被篡改。接收方可以验证数据的完整性，确保数据在传输过程中没有被修改或损坏。

3. **身份认证：** TLS 使用数字证书来对通信双方进行身份认证，以确保通信双方的真实身份。数字证书包含了公钥和证书持有者的身份信息，可以用于验证通信双方的身份。

4. **会话管理：** TLS 使用会话标识符和会话密钥来管理通信会话，以提高通信的效率和安全性。通信双方可以通过会话标识符和会话密钥来识别和保护通信会话。TLS 协

议被广泛应用于 Web 安全、电子邮件安全、虚拟私人网络（VPN）、即时通讯等领域，为网络通信提供了重要的安全保障。HTTPS（HTTP over TLS，基于 TLS 的 HTTP 安全传输协议）是 Web 安全中最常见的应用之一，用于保护 Web 浏览器和 Web 服务器之间的通信安全。

## TLS




在 TLS（传输层安全）中，"CR" 可能指的是 "Certificate Request"（证书请求）。

在 TLS 握手过程中，当客户端发送 "ClientHello" 消息后，服务器通常会向客户端发送 "ServerHello" 消息，然后会发送一个 "CertificateRequest" 消息。这个消息是可选的，它表示服务器要求客户端提供数字证书以进行身份验证。

"CertificateRequest" 消息通常包含服务器希望客户端使用的加密算法、哈希算法以及服务器信任的 CA 列表。客户端如果收到了 "CertificateRequest" 消息，就会在后续的握手过程中提供相应的数字证书，以便服务器进行验证。

CA（Certificate Authority，证书颁发机构）是负责颁发和管理数字证书的信任机构。数字证书是用于在网络通信中进行身份验证和加密的一种安全机制，由 CA 颁发的数字证书包含了公钥和证书持有者的身份信息，并用 CA 的数字签名进行签名，以确保证书的真实性和完整性。

CA 的主要职责包括：

1. **颁发数字证书：** CA 根据验证的身份信息，向申请者颁发数字证书，证明了该申请者的身份信息和公钥的关联关系。数字证书包含了证书持有者的身份信息、公钥、证书有效期等信息。

2. **证书管理：** CA 负责管理颁发的数字证书，包括证书的签发、更新、撤销等操作。CA 维护证书吊销列表（Certificate Revocation List，CRL），记录已经被撤销的证书信息，确保已经被吊销的证书不再被信任使用。

3. **信任验证：** CA 的数字签名被广泛地信任于公众，因此由 CA 颁发的数字证书也得到了广泛的信任。通信双方可以通过验证证书的签名和证书链的有效性，来确认证书的真实性和可信度，从而实现通信双方的身份验证和数据加密。

4. **根证书管理：** CA 的根证书是用于签署其他证书的最高级别的数字证书，它构成了数字证书信任链的根。CA 负责管理和保护根证书的私钥，以确保证书颁发和验证的安全可靠。CA 在网络安全中扮演着重要的角色，它为网络通信提供了安全基础，保护了通信的隐私性和完整性。HTTPS 等基于 TLS 的安全通信协议就是依赖于 CA 颁发的数字证书来实现通信双方的身份验证和数据加密的。

## URL

URL（Uniform Resource Locator，统一资源定位符）是用于定位和访问互联网上资源的标准格式。它是 Web 浏览器等客户端程序用于定位和获取 Web 资源（如网页、图片、视频等）的一种方式。

1. **协议（Protocol）：** 指定了访问资源所使用的协议类型，例如 HTTP、HTTPS、FTP 等。协议部分通常以 "://" 结尾，例如 "http://"、"https://"。

2. **主机名（Host Name）：** 指定了资源所在的主机或服务器的域名或 IP 地址。主机名部分通常紧跟在协议后面，例如 "www.example.com"。

3. **端口号（Port Number）：** 指定了服务器上提供该资源的端口号。端口号通常用冒号 ":" 后跟一个数字表示，例如 ":80" 表示使用默认的 HTTP 端口号。

4. **路径（Path）：** 指定了服务器上资源的具体路径或位置。路径部分通常跟在主机名或端口号后面，例如 "/index.html"。

5. **查询字符串（Query String）：** 包含了向服务器传递的参数和数值，用于请求特定的资源或执行特定的操作。查询字符串通常以问号 "?" 开始，参数与数值之间用

"&" 分隔，例如 "?id=123&name=John"。

6. **片段标识符（Fragment Identifier）：** 标识了资源中的特定部分或位置。片段标识符通常以井号 "#" 开始，用于在页面内跳转到指定的位置，例如 "#section2"。总的来说，URL 提供了一种统一的方式来定位和访问互联网上的资源，它的格式清晰明了，易于理解和使用。在浏览器地址栏中输入 URL，即可快速访问相应的资源。

## URL

```python
URL

RFC 3986:
.   Uniform
.   Resource
.   Locator (/Identifier - URI)


https://bristol.ac.uk/engineering/departments/computerscience
```

RFC 3986 定义了统一资源定位器（URL）的标准规范。URL 是用于标识和定位互联网上资源的标准格式，它包含了指定资源位置和访问方式的信息。在 RFC 3986 中，URL 被解释为 "Uniform Resource Locator"，意为统一资源定位器。同时，RFC 3986 也将 URL 视为统一资源标识符（Uniform Resource Identifier，URI）的一种特殊形式。在 RFC 3986 中，URL 主要由以下几个组件构成：

1. **方案（Scheme）：** URL 方案部分定义了用于访问资源的协议类型，例如 "http"、"https"、"ftp" 等。

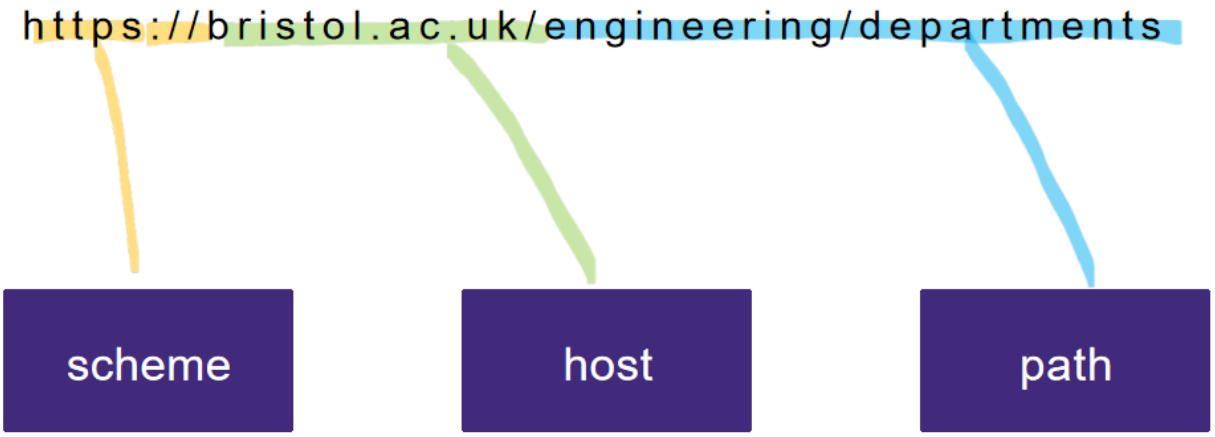
2. **授权信息（Authority）：** URL 授权信息部分包含了资源的访问权限信息，通常包括用户名、密码和主机名等。

3. **路径（Path）：** URL 路径部分指定了资源在服务器上的位置或路径，用于确定要访问的具体资源。

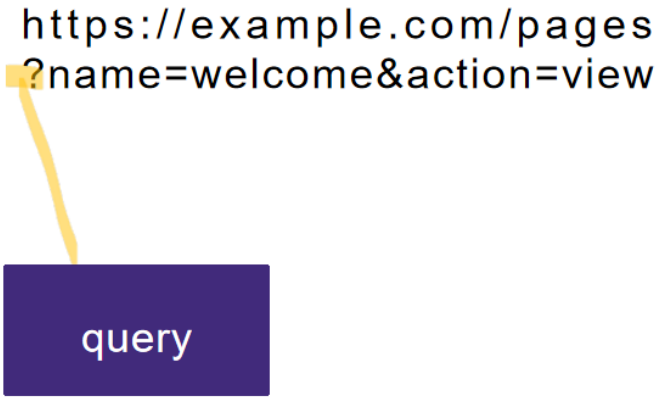
4. **查询字符串（Query）：** URL 查询字符串部分包含了向服务器传递的参数和数值，用于请求特定的资源或执行特定的操作。

5. **片段标识符（Fragment）：** URL 片段标识符部分标识了资源中的特定部分或位置，用于在页面内跳转到指定的位置。RFC 3986 对 URL 的定义和解析提供了详细的规范，使得 URL 在互联网上的使用更加统一和规范。通过遵循 RFC 3986 中的标准规范，可以确保 URL 在不同的网络环境和应用场景下都能正确解析和使用。

## URL parts




## URL parts query




## URI schemes

URI schemes

ftp://ftp.is.co.za/rfc/rfc1808.txt

http://www.ietf.org/rfc/rfc2396.txt

ldap://[2001:db8::7]/c=GB?objectClass?one

mailto:John.Doe@example.com
news:comp.infosystems.www.servers.unix
tel:+1–816–555–1212
telnet://192.0.2.16:80/
urn:oasis:names:specification:docbook:dtd:xml:4.1.2

URI（统一资源标识符）方案是一种用于标识和定位资源的命名方案。它定义了一种统一的语法和格式，使得不同类型的资源可以在互联网上通过唯一的标识符进行访问和定位。

URI 方案通常由一个或多个组成部分组成，包括方案名称（如 http、ftp、mailto 等）、主机名、路径、查询参数和片段标识符。每种 URI 方案都有自己的语法和约定，用于指定资源的位置和访问方式。

在提供的示例中，我们可以看到几种常见的 URI 方案：

1. **ftp**: 用于访问文件传输协议（FTP）服务器上的文件资源。
2. **http**: 用于访问超文本传输协议（HTTP）服务器上的网页资源。
3. **ldap**: 用于访问轻量级目录访问协议（LDAP）服务器上的目录资源。
4. **mailto**: 用于发送电子邮件到指定的邮箱地址。
5. **news**: 用于访问新闻组文章。
6. **tel**: 用于拨打电话号码。
7. **telnet**: 用于远程登录到指定的主机上。
8. **urn**: 用于定义资源的持久唯一名称，但不提供资源的访问方式。

每种 URI 方案都有其特定的用途和语法规则，可根据需要选择合适的方案来标识和定位资源。

## REST

REST（Representational State Transfer，表述性状态转移）是一种软件架构风格，用于设计网络应用程序和服务。它是一种基于 HTTP 协议的通信模式，旨在简化系统之间的通信和资源访问。REST 架构的主要特点包括：

1. **资源（Resources）：** 在 REST 架构中，所有的数据和功能都被视为资源。每个资源都有一个唯一的标识符（URL），用于定位和访问该资源。例如，一个网站中的

文章、评论、用户等可以被视为不同的资源。

2. **表述性（Representation）：** 客户端和服务器之间的通信通过资源的表述进行。资源的表述可以是 JSON、XML、HTML 等格式，客户端可以根据需要选择最合适的表述进行交互。

3. **状态转移（State Transfer）：** 客户端通过发送 HTTP 请求对资源进行操作，服务器通过 HTTP 响应来响应这些请求。在 REST 中，客户端和服务器之间的状态转移是通过 HTTP 方法（GET、POST、PUT、DELETE 等）来实现的。

4. **无状态性（Statelessness）：** REST 架构是无状态的，即服务器不会保存客户端的状态信息。每个请求都包含了足够的信息来完成请求和响应，服务器不需要维护会话状态或上下文信息。

5. **统一接口（Uniform Interface）：** REST 架构中的统一接口指定了客户端和服务器之间的通信方式，包括使用标准的 HTTP 方法和状态码，以及资源的唯一标识符（URL）等。这种统一的接口使得不同系统之间的通信更加简单和可靠。REST 架构风格的设计理念简洁清晰，使得它成为了设计和构建分布式系统的首选方案之一。它广泛应用于 Web API、移动应用程序、微服务架构等领域，为构建可扩展、灵活和易于维护的系统提供了有效的解决方案。

## REST

Representational State Transfer
Fact: HTTP is stateless

. State goes in the request

. Resources have names (URLs)
. use HTTP verbs as intended

这三个要点是 REST 架构设计中的重要原则：

1. **State goes in the request（状态信息在请求中）：** 在 REST 架构中，客户端向服务器发送请求时，应该包含所有必要的状态信息。服务器不会保存客户端的状态，

因此客户端每次请求都应该携带足够的信息来完成所需的操作。这种设计保持了服务器的无状态性，使得系统更加简单和可靠。

2. **Resources have names (URLs)（资源有名称）：** 在 REST 架构中，每个资源都有一个唯一的标识符，即 URL（统一资源定位符）。URL 是用于定位和访问资源的地址，它为每个资源提供了一个唯一的标识符。资源的 URL 应该具有语义上的含义，并且应该通过 URL 来表示资源之间的关系和层次结构。

3. **Use HTTP verbs as intended（使用 HTTP 动词）：** HTTP 协议定义了一组常用的动词（GET、POST、PUT、DELETE 等），用于对资源进行不同的操作。在 REST 架构中，应该按照 HTTP 协议的语义来正确使用这些动词。例如，使用 GET 方法来获取资源的表示，使用 POST 方法来创建新资源，使用 PUT 方法来更新资源，使用 DELETE 方法来删除资源。这种正确使用 HTTP 动词的方式能够使系统的接口设计更加清晰和标准化。这些原则是 REST 架构设计的基石，通过遵循这些原则，可以设计出清晰、可扩展和易于理解的 RESTful API。

REST（Representational State Transfer，表述性状态转移）是一种软件架构风格，用于设计分布式系统和网络应用程序。其中的一项事实是，HTTP（Hypertext Transfer Protocol，超文本传输协议）是一种无状态协议，这意味着每个请求都是独立的，服务器不会记住之前的请求状态。

在 REST 架构中，状态信息通常包含在客户端的请求中，而不是保存在服务器上。客户端通过向服务器发送 HTTP 请求来请求资源，每个资源都有一个唯一的标识符（URL）。因此，资源的名字就是它们的 URL。

REST 架构中，常常使用 HTTP 协议的动词（GET、POST、PUT、DELETE 等）来对资源进行不同的操作。例如，使用 GET 方法来获取资源的表示，使用 POST 方法来创建新资源，使用 PUT 方法来更新资源，使用 DELETE 方法来删除资源。

总的来说，REST 架构的设计理念是通过统一的接口和无状态的通信方式，实现客户端和服务器之间的资源交互和状态转移。这种简洁清晰的设计使得 REST 成为了设计和构建分布式系统的一种重要方式

## REST

Python

```
GET          /files/README.txt
DELETE   /files/README.txt




POST         /files/README.txt?action=delete


GET         /files?name=README.txt
```

尽管这个请求可以用来获取名为 "README.txt" 的文件，但它并不是符合 RESTful 设计原则的典型用法。在 RESTful 架构中，URL 应该代表资源的唯一标识符，而查询参数通常用于对资源进行过滤、排序或分页等操作，而不是用于直接定位资源。因此，对于这个请求来说，更符合 RESTful 设计原则的做法是直接使用资源的完整路径来定位资源，而不是通过查询参数来指定资源的名称。感谢你的澄清和提醒！

这些请求示例展示了在 REST 架构中如何使用 HTTP 动词来对资源进行操作：1. **GET /files/README.txt：** 这是一个获取资源的请求，它指定了要获取的资源的 URL 为 "/files/README.txt"。通过使用 GET 方法，客户端请求获取名为 "README.txt" 的文件资源的表示。

2. **DELETE /files/README.txt：** 这是一个删除资源的请求，它指定了要删除的资源的 URL 为 "/files/README.txt"。通过使用 DELETE 方法，客户端请求删除名为 "README.txt" 的文件资源。

3. **POST /files/README.txt?action=delete：** 这是一个执行特定操作的请求，它使用了 POST 方法，并且在 URL 中传递了一个参数 "action=delete"。这种方式不符合 RESTful 设计的原则，因为 POST 应该用于创建资源或执行非幂等的操作，而不是用于简单的资源删除操作。

4. **GET /files?name=README.txt：** 这是一个根据查询参数获取资源的请求，它指定了要获取的资源的 URL 为 "/files"，并且在查询参数中指定了要获取的文件名为 "README.txt"。这种方式适用于在资源集合中根据特定条件过滤和获取资源。这些请求示例展示了如何在 REST 架构中使用 HTTP 动词和 URL 来对资源进行不同的操作，以及如何根据需要使用查询参数来过滤和定位资源。

# HTML 5

## The Manual

# Web technology references

**Web APIs**

Reference material for each of the individual APIs that comprise the Web's powerful scriptability, including the DOM and all of the related APIs and interfaces you can use to build Web content and apps.

**HTML**

HyperText Markup Language is the language used to describe and define the content of a Web page.

**CSS**

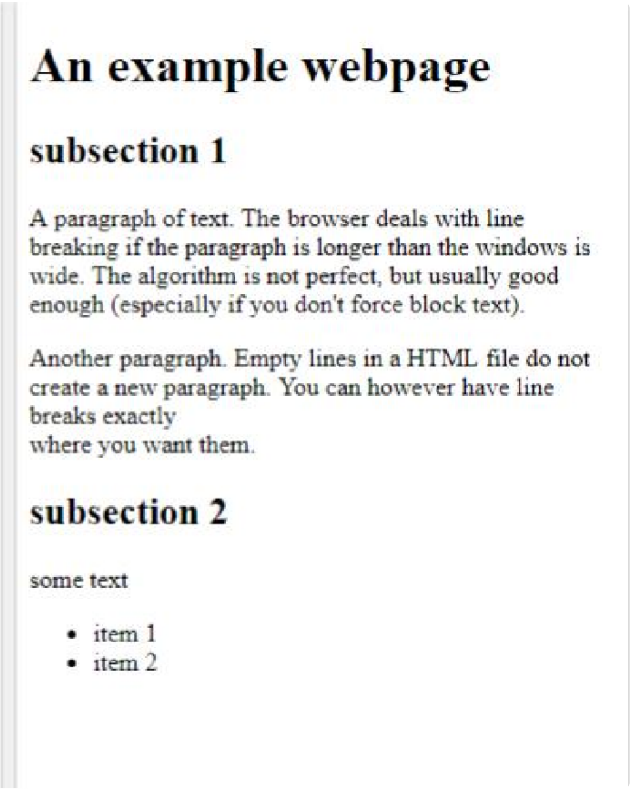
Cascading Style Sheets are used to describe the appearance of Web content.

**JavaScript**

JavaScript is a programming language used to add interactivity to a website.

## HTML5

```html
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8" />
    <title>A web page</title>
  </head>
  <body>
    <h1>An example webpage</h1>
    <h2>subsection 1</h2>
    <p>A paragraph of text. The browser deals
    with line breaking if the paragraph is
    longer than the windows is wide. The
    algorithm is not perfect, but usually
    good enough (especially if you don't force
    block text).</p>
    <p>Another paragraph. Empty lines in a
    HTML file do not create a new paragraph.
    You can however have line breaks
    exactly<br /> where you want them.</p>
    <h2>subsection 2</h2>
    <p>some text</p>
    <ul>
      <li>item 1</li>
      <li>item 2</li>
    </ul>
  </body>
</html>
```




## HTML5 template

Python

```
<!DOCTYPE  html>
<html  lang="en">
<head>
<meta  charset="utf-8"  />
<title>A web page</title> </head>
<body>
your content here
</body>
</html>
```

## Tags

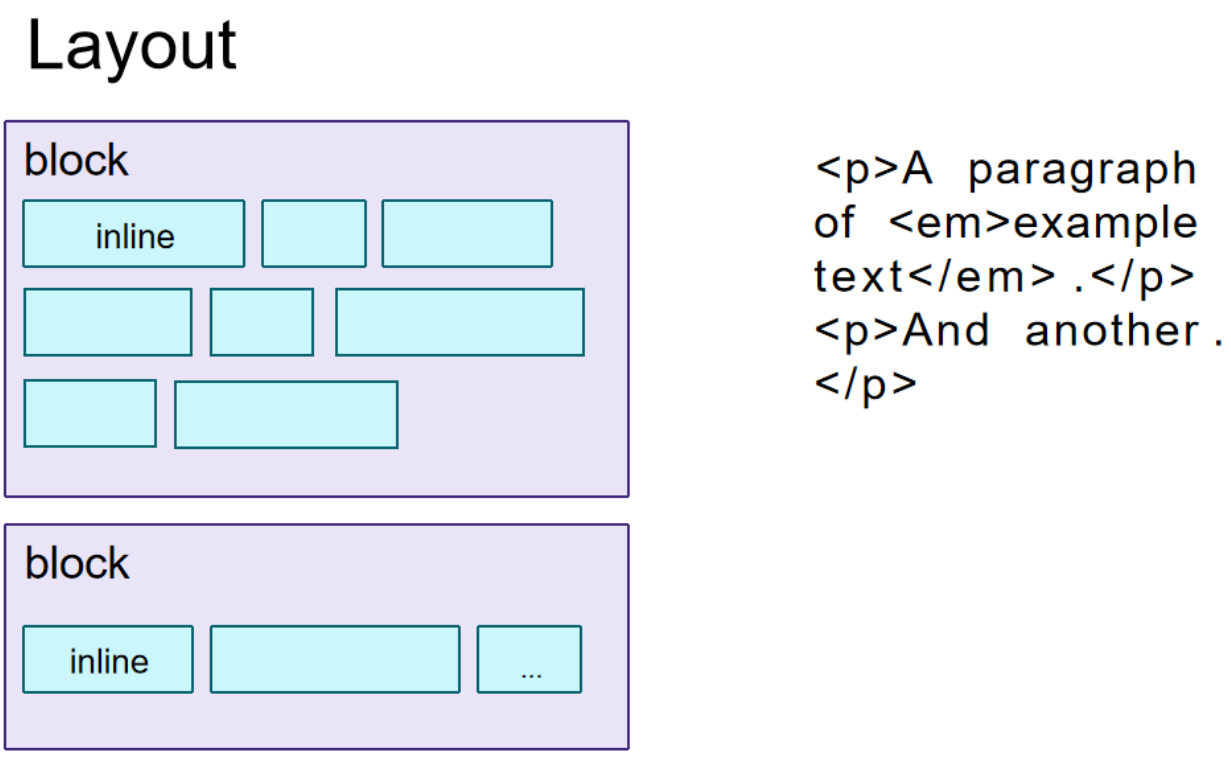
```
<body>
  <h1>An example webpage</h1>
  <h2>subsection 1</h2>
  <p>A paragraph of text. The browser deals
  with line breaking if the paragraph is
  longer than the windows is wide. The
  algorithm is not perfect, but usually
  good enough (especially if you don't force
  block text).</p>
  <p>Another paragraph. Empty lines in a
  HTML file do not create a new paragraph.
  You can however have line breaks
  exactly<br /> where you want them.</p>
  <h2>subsection 2</h2>
  <p>some text</p>
  <ul>
    <li>item 1</li>
    <li>item 2</li>
  </ul>
</body>
```

h1 ... h6 headings

<p> paragraphs
<br /> new line

<ul> unordered list
<ol> ordered list
<li> list item

## Layout




```
<p>A paragraph
of <em>example
text</em> .</p>
<p>And another .
</p>
```

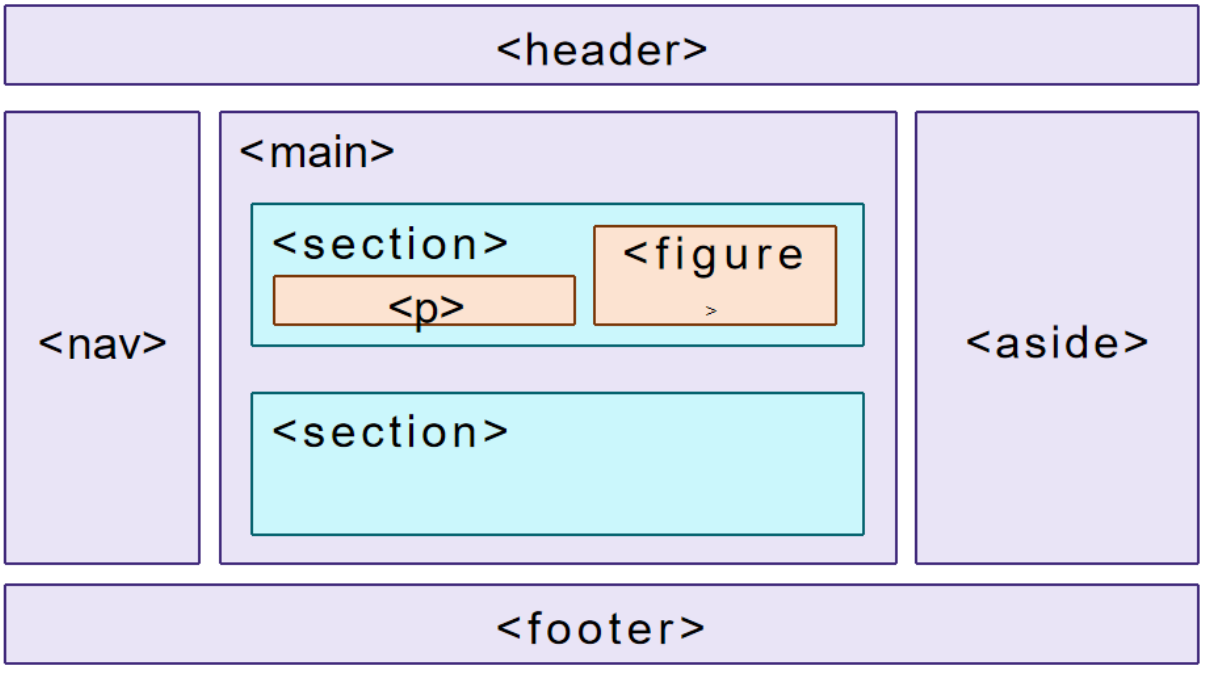
## Semantic Tags

## Semantic Tags

https://developer.mozilla.org/en/docs/Web/HTML/Element

| new | | old | |
|---|---|---|---|
|  | emphasis | <b> | bold |
|  | important | <i> | *italics* |
| <q> | quotation | <u> | underline |
| <cite> | citation | <s> | ~~strike out~~ |
| <var> | variable | <tt> | monospace |
| <code> | source code | | |
| | | <small> | small |

## Semantic Tags




语义标签是指在 HTML 中具有特定含义或语义的标签，这些标签能够更准确地描述其所包含内容的含义和结构。使用语义标签可以提高网页的可访问性、可读性和搜索引擎优化。以下是一些常见的语义标签及其解释：- `<header>`：用于表示文档或页面的页眉部分，通常包含标题、导航链接或其他引导性内容。- `<nav>`：用于表

示导航链接的区域，通常包含页面内或站点内的导航链接。– `<main>`：用于表示文档或页面的主要内容区域，通常包含页面的核心内容。– `<section>`：用于表示文档或页面的一个独立的区域或节，通常包含相关内容的组合。– `<article>`：用于表示文档或页面中的一个独立的文章、博客帖子或新闻稿等内容单元。– `<aside>`：用于表示文档或页面的旁注、侧边栏或附属内容，通常包含与主内容相关的补充信息。– `<footer>`：用于表示文档或页面的页脚部分，通常包含版权信息、联系方式或其他页尾内容。使用这些语义标签能够更清晰地表达网页的结构和内容，有助于提高页面的可理解性、可访问性和可维护性。

## Placeholder Tags

```python
<section>
To structure your content.
<span>
Generic inline tag.
<div>
Generic block tag.
```

这里是对每个 HTML 标记的解释：– `<section>`：用于在文档中创建一个节（section），用于组织和结构化内容，通常表示文档的一部分或一个主题区域。– `<span>`：用于创建一个通用的内联标签，用于对文本的一部分进行样式化或标记，常用于对文本进行修饰或区分。– `<div>`：用于创建一个通用的块级标签，用于对文档内容进行分组，常用于布局和样式化目的，可用于创建盒状模型的容器。

## Attributes

```python
<p  id= "today">28  September</p>
<p  class= "info">Lecture  2</p>
<p  class= "info">QB  0.18</p>
```

这里是对 HTML 中使用的属性的解释：– `id`：用于给 HTML 元素指定一个唯一的标识符。在给定的 HTML 文档中，每个 `id` 属性的值应该是唯一的。在提供了一个唯一标识符后，你可以使用 JavaScript 或 CSS 等技术来操作或样式化该元素。　在给

定的示例中，`id` 属性被用于标识第一个 `<p>` 元素，其值为 `today`。- `class`：用于为 HTML 元素指定一个或多个类名。类名通常用于定义样式规则，多个类名可以通过空格分隔。使用类可以为一组元素应用相同的样式规则。 在给定的示例中，`class` 属性被用于标识两个 `<p>` 元素，它们都有一个名为 `info` 的类。这些属性允许你将额外的信息附加到 HTML 元素上，使其具有更多的语义和样式。

# HTML elements

## Links

```python
<a  href= "/courses">Our  Courses</a>

bristol.ac.uk/students/info.html  :


"/courses"      =>  bristol.ac.uk/courses
"courses"        =>  bristol.ac.uk/students/courses


" ../courses"  =>  bristol.ac.uk/courses
```

这里是对链接以及给定 URL 路径的解释：链接：- `<a>`：HTML 中的锚标签，用于创建超链接。`href` 属性指定链接的目标 URL。给定 URL 路径与其解释：- `"/courses"`：这个路径是相对路径，表示到 `courses` 页面的链接。如果当前页面的 URL 为 `bristol.ac.uk/students/info.html`，那么点击这个链接将导航到 `bristol.ac.uk/courses`。- `"courses"`：这也是一个相对路径，但没有前导斜杠。它表示到当前路径下的 `courses` 页面的链接。如果当前页面的 URL 为 `bristol.ac.uk/students/info.html`，那么点击这个链接将导航到 `bristol.ac.uk/students/courses`。- `"../courses"`：这是一个相对路径，使用 `..` 表示返回到上一级目录。所以如果当前页面的 URL 为 `bristol.ac.uk/students/info.html`，那么点击这个链接将导航到 `bristol.ac.uk/courses`。这些解释基于相对路径和当前页面的 URL 来确定最终导航到的目标 URL。

# Forms


表单（Forms）在网页中用于收集用户输入的数据，以便提交到服务器进行处理或保存。HTML 中的表单由 `<form>` 标签定义，通常包含各种输入字段、按钮和其他控件。以下是一些常见的表单元素和其作用：– `<input>`：用于接受用户输入的单行文本框、密码框、复选框、单选按钮等。– `<textarea>`：用于接受用户输入的多行文本。– `<select>`：用于创建下拉列表框，允许用户从一组选项中选择一个或多个选项。– `<button>`：用于创建按钮，可以用作提交按钮或普通按钮。– `<label>`：用于创建表单控件的标签，通常用于描述输入字段的目的或内容。表单中的输入字段通常由用户填写，然后通过提交按钮将表单数据发送到服务器。服务器端处理程序可以处理提交的数据，并根据需要执行相应的操作，例如保存到数据库、发送电子邮件等。表单还可以使用各种属性来控制其行为和样式，例如 `action` 属性指定表单提交的目标 URL，`method` 属性指定提交方式（通常是 GET 或 POST），`name` 属性指定表单的名称等。通过使用表单，网站可以与用户进行交互，并收集所需的信息，以便提供个性化的服务或执行特定的操作。

Python

```
<form  method= "post"  action="/comment">
<p>
<label  for="name">Name:</label>
<input  id="name"  name="name"/>        </p>
<p>
<button  type="submit">OK</button>   </p>
</form>
```

这个 HTML 表单包含一个 POST 方法和一个 action 属性，用于将表单数据发送到服务器上的 `/comment` 端点。表单包含两个输入字段：1. `<input>` 元素用于用户输入其名称。它有一个 id 属性为"name"，用于与<label>元素关联，以及一个 name 属性为"name"，用于在表单提交时标识该字段的名称。2. `<button>` 元素用于提交表单数据。它的类型为"submit"，表示点击按钮时将提交表单。在用户填写完姓名并点击提交按钮后，表单数据将被打包并发送到服务器的"/comment"端点进行处理。

## Validation


```
<input required type="number">
```

验证（Validation）是在表单提交之前检查用户输入的过程，以确保输入的数据符合预期的格式、范围或其他条件。验证有助于确保用户提供的数据是有效的，从而提高数据的质量和完整性。在 Web 开发中，常见的验证方式包括：1. **客户端验证（Client-Side Validation）**：在用户填写表单时，通过 JavaScript 等客户端脚本来实时验证输入数据。这种验证方式可以提供即时反馈，并且减少不必要的服务器请求。但是客户端验证可以被绕过，因此必须与服务器端验证结合使用。2. **服务器端验证（Server-Side Validation）**：在表单提交到服务器后，在服务器端进行验证。这种验证方式更加安全可靠，因为用户无法绕过服务器端验证。服务器端验证通常在接收到表单数据后，对数据进行验证并根据验证结果返回适当的响应。验证通常涉及以下方面：– **必填字段验证**：确保必填字段不为空。– **数据格式验证**：检查数据是否符合特定的格式，如电子邮件地址、日期、电话号码等。– **数据范围验证**：检查数据是否在指定的范围内，如数字是否在特定范围内、日期是否在有效日期范围内等。– **唯一性验证**：确保数据在系统中唯一，如唯一用户名、电子邮件地址等。– **安全性验证**：检查输入是否包含潜在的恶意内容，如 SQL 注入、跨站点脚本攻击等。综合使

用客户端验证和服务器端验证可以有效地确保用户输入的数据是有效、安全和完整的，从而提高 Web 应用程序的质量和可用性。

## Autocomplete

```Python
<input  type= "text"
autocomplete="name">

name, email, address-line1, country, tel, … cc-name, cc-number, …
```

自动完成（Autocomplete）是一种浏览器提供的功能，用于自动填充输入字段中的值。当用户开始在输入字段中输入内容时，浏览器会显示一个下拉列表，其中包含之前输入过的可能匹配的值。用户可以从下拉列表中选择一个值，然后浏览器会自动填充输入字段。在 HTML 中，可以使用 `autocomplete` 属性来控制浏览器自动完成的行为。这个属性的值通常是一组预定义的值，表示浏览器应该自动填充哪些类型的信息。这些预定义的值通常与常见的表单字段相关，例如名称、电子邮件地址、地址行等。例如，在给定的输入字段中：```html<input type="text" autocomplete="name">````

autocomplete `属性被设置为"name"，表示这个输入字段应该用于输入名称。当用户开始输入时，浏览器会自动填充之前输入过的名称。一些常见的` autocomplete `属性值包括：- name：用于名称字段- email：用于电子邮件地址字段- address-line1：用于地址的第一行- country：用于国家字段- tel：用于电话号码字段- cc-name：用于信用卡持卡人姓名字段- cc-number：用于信用卡卡号字段通过使用`

## input types

这里是对常见的 HTML `<input>` 元素的类型及其用途的解释：– `button`：创建一个普通的按钮，用于执行 JavaScript 操作或触发表单提交等。– `checkbox`：创建一个复选框，允许用户选择一个或多个选项。– `color`：创建一个颜色选择器，允许用户选择颜色。– `date`：创建一个日期选择器，允许用户选择日期。– `datetime-local`：创建一个本地日期和时间选择器，允许用户选择日期和时

间。– `email`：创建一个用于输入电子邮件地址的文本框。– `file`：创建一个文件上传控件，允许用户选择文件。– `hidden`：创建一个隐藏的输入字段，通常用于在表单中传递数据而不显示给用户。– `image`：创建一个图像按钮，类似于普通按钮，但显示为图像。– `month`：创建一个月份选择器，允许用户选择月份。– `number`：创建一个数字输入框，允许用户输入数字。– `password`：创建一个密码输入框，用户输入的文本会被隐藏。– `radio`：创建一组单选按钮，用户只能选择其中的一个选项。– `range`：创建一个滑动条，允许用户在指定范围内选择值。– `reset`：创建一个重置按钮，用于重置表单中的所有字段为初始值。– `search`：创建一个搜索框，用于输入搜索关键字。– `submit`：创建一个提交按钮，用于提交表单数据。– `tel`：创建一个用于输入电话号码的文本框。– `text`：创建一个用于输入单行文本的文本框。– `textarea`：创建一个用于输入多行文本的文本区域。– `time`：创建一个时间选择器，允许用户选择时间。– `url`：创建一个用于输入 URL 地址的文本框。– `week`：创建一个周选择器，允许用户选择周。这些输入类型可以根据不同的需求和场景来选择使用，以便提供更好的用户体验和功能。

## select

```python
<select  name="animal">
<option  value="dog">Dog</option>
<option  value="cat">Cat</option>
</select>
```

## Tables

```python
<table>
<thead>
<tr><th>Name</th><th>ID</th></tr>
</thead>

<tbody>
<tr><td>Sarah</td><td>100</td></tr>
<tr><td>Jon</td><td>101</td></tr>
```

```
</tbody>
</table>
```

表格（Tables）在 HTML 中用于以行和列的形式组织和展示数据。HTML 中的表格由 `<table>` 标签定义，通常包含一系列行（ `<tr>` ）和列（ `<td>` 或 `<th>` ）。以下是 HTML 表格中常见的元素和其作用：- `<table>` ：定义一个表格。- `<tr>` ：定义表格中的一行。- `<td>` ：定义表格中的一个数据单元格（普通单元格）。- `<th>` ：定义表格中的一个表头单元格。通常在表格的第一行或第一列中使用，用于标识列或行的标题。- `<caption>` ：定义表格的标题，通常位于表格之上。表格中的每一行（ `<tr>` ）包含一个或多个数据单元格（ `<td>` 或 `<th>` ）。每个数据单元格可以包含文本、图像或其他 HTML 元素。表头单元格（ `<th>` ）通常用于标识列或行的标题，而普通单元格（ `<td>` ）用于显示数据。下面是一个简单的 HTML 表格示例： `html<table> <caption>Monthly Sales Report</caption> <tr> <th>Month</th> <th>Sales</th> </tr> <tr> <td>January</td> <td>1000</td> </tr> <tr> <td>February</td> <td>1500</td> </tr></table>` 这个示例展示了一个包含月度销售数据的简单表格，其中包括一个标题、两列（月份和销售额）、三行数据（每月的销售数据）。

### Datatable

# 1.2Exploring HTTP

vagrant

```Java
config.vm.network "forwarded_port", guest: 8000, host: 8000
```

## Before you start

```Java
wget localhost:8000

netstat -tan
```

```
netstat -tan | grep 8000
```

## A basic HTTP server and client

Bash

```
#下载
wget https://cs-uob.github.io/COMSM0085/exercises/part2/resources/http

nc -l -p 8000 < http-response

#mac
nc -l 8000 < http-response

###show
HTTP/1.1 200 OK
Content-type: text/plain
Content-length: 18
Connection: close

Hello over HTTP!
###

wget -q -S -O - localhost:8000
```

## Connecting with a web browser

## A web server in C

Bash

```
git clone https://github.com/emikulic/darkhttpd

cd darkhttpd

make
```

```
touch web/hello.txt

#浏览器
localhost:8000/hello.txt
```

# 1.3HTTP/URL research exercises

## URL 的片段部分（Fragment part of a URL）

URL 的片段部分是指 URL 中的 `#` 符号后面的部分，用于指定文档中的特定位置或标识符。它通常在客户端浏览器端使用，而不会被发送到服务器。在网页中，当你在 URL 后添加 `#` 符号以及后面的标识符时，浏览器会自动滚动到对应的位置或执行相应的操作。

以下是一个示例 URL 的片段部分：

Bash

```
https://www.example.com/page.html#section1
```

在这个示例中，`#section1` 是 URL 的片段部分。这个片段部分指定了文档中的特定位置或标识符，可能是文档中的某个部分、锚点或者其他标识符。当用户访问这个 URL 时，浏览器会自动滚动到文档中具有相应标识符的位置，以便用户可以立即查看所需内容。需要注意的是，片段部分不会被发送到服务器，仅在客户端浏览器端使用。

## HTTP 客户端发送的 Accept 头部

Accept 头部指定了客户端（通常是浏览器）所能接受的内容类型。它告诉服务器客户端期望接收什么类型的响应，例如文本、图像、视频等。这有助于服务器根据客户端的偏好来选择合适的响应内容类型。

Bash

```
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/we
```

## HTTP 客户端发送的 User-agent 头部

User-agent 头部包含了客户端（浏览器或其他 HTTP 客户端）的标识信息，用于告知服务器客户端的类型和版本。这允许服务器根据不同的客户端来适应不同的需求。例如，浏览器可能发送类似于 `Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/96.0.4664.110 Safari/537.36` 的 User-agent。

以下是一个示例 HTTP 客户端发送的 User-agent 头部：

Bash

```
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.
```

这个 User-agent 头部告诉服务器，客户端是在 Windows 10 操作系统上运行的 Chrome 浏览器版本 96.0.4664.110，其基于 AppleWebKit 渲染引擎，版本号为 537.36，同时也兼容 Safari 浏览器的一些特性。

## 在 URL 中如何编码包含空格的路径

在 URL 中，空格通常需要被编码成 `%20`。所以，如果你的路径中包含空格，你需要用 `%20` 来替代空格。除了空格之外，其他一些特殊字符也需要进行编码，比如 `?`、`&`、`=` 等。

假设我们有一个包含空格的文件路径，例如：

Bash

```
https://www.example.com/files/my%20file.txt
```

在这个示例中，文件路径是 `/files/my file.txt`，其中空格被编码为 `%20`。这样做是因为在 URL 中，空格不能直接出现，而需要使用 `%20` 进行编码代替。

其他特殊字符也需要进行编码，例如：

- `?` 编码为 `%3F`
- `&` 编码为 `%26`
- `=` 编码为 `%3D`

### 布里斯托大学（University of Bristol）的网站所使用的 Web 服务器

要查看布里斯托大学网站所使用的 Web 服务器，可以通过检查 HTTP 响应头部中的 `Server` 字段来获取。根据这个字段，可以确定服务器的类型和版本。然后，你可以在网上查找这个特定的服务器类型以及维护它的组织。

## 1.4Java Server

### compile and run

```Bash
mvn spring-boot:run
```

### Explore the application

pom.xml 文件告诉 Maven 这是一个 Spring Boot 项目，并指定了项目的名称（softwaretools.server01）。

在 src/main/resources 目录下，你会找到两个文件。首先是 application.properties，这是一个 Spring 配置文件，配置了应用程序在端口 8000 上运行（默认情况下为 8080）。其次是一个 HTML 文件，应用程序可以提供服务。

在 src/main/java 目录下是源代码。虽然只有两个文件，但在应用程序运行时整个 Spring 框架都是活跃的。Server01Application 是主类，但目前只包含样板代码。

Controller.java 才是有趣的部分：在应用程序开发中，Model–View–Controller 是结构化应用程序的几种模式之一，其中 Controller 是一段处理重要工作的代码，例如回复 HTTP 请求。

Spring 使用注解（Annotations），这些特殊类的名称以 @ 符号开头。当服务器启动时，Spring 会扫描所有文件以查找注解，并设置相应的代码来处理它们。在这里我们可以看到：

- `@SpringBootApplication`（在 Server01Application 类上）告诉 Spring 这是要运行的主文件。
- `@RestController` 告诉 Spring 这个类包含处理 HTTP 请求的方法。它被这样命名是因为 Spring 提供了特定的库，使得实现 REST 原则特别容易。
- `@Autowired` 注解在一个字段上告诉 Spring 在应用程序启动时要负责设置这个字段。ResourceLoader 是 Spring 的一部分，它允许你访问 src/main/resources 中的文件。
- `@GetMapping(PATH)` 告诉 Spring 应该调用这个方法来响应提供的路径的 HTTP GET 请求（当然还有 `@PostMapping` 等等）。
- `mainPage` 方法在你加载 `localhost:8000` 时被调用，展示了回复 HTTP 请求的基本方式：设置任何你需要的头部（在这种情况下是内容类型），并返回一个 ResponseEntity，它接受三个参数：要返回的响应的 HTTP 主体（这将显示在你的浏览器中），要设置的 HTTP 头部，以及这里是 200（OK）的响应代码。

htmlPage 方法用于回复对 localhost:8000/html 的请求。在这里，我们想要提供一个文件，所以我们使用资源加载器从类路径中获取它，类路径包括 src/main/resources 文件夹（或者更确切地说，包括最终编译为 JAR 文件的版本）。它还展示了使用建造者模式创建 ResponseEntity 的第二种方式。

## controller

```Bash
package softwaretools.server01;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.core.io.Resource;
import org.springframework.core.io.ResourceLoader;
import org.springframework.web.bind.annotation.RestController;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.http.ResponseEntity;
import org.springframework.http.HttpHeaders;
import org.springframework.http.HttpStatus;

@RestController
public class Controller {

    @Autowired
    ResourceLoader loader;

    @GetMapping("/")
    public ResponseEntity<String> mainPage() {
        HttpHeaders headers = new HttpHeaders();
        headers.set(HttpHeaders.CONTENT_TYPE, "text/plain");

        return new ResponseEntity<String>("Hello from Spring", head
    }

    @GetMapping("/html")
    public ResponseEntity<Resource> htmlPage() {
        Resource htmlfile = loader.getResource("classpath:web/page.
        return ResponseEntity
            .status(200)
            .header(HttpHeaders.CONTENT_TYPE, "text/html")
            .body(htmlfile);
    }

    @GetMapping("/bad")
    public ResponseEntity<String> badPage(){
      return ResponseEntity.status(HttpStatus.NOT_FOUND).body("404'
    }
}
```

- server

```Bash
package softwaretools.server01;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplicatior

@SpringBootApplication
public class Server01Application {

  public static void main(String[] args) {
    SpringApplication.run(Server01Application.class, args);
  }
}
```

## 2.1 Basic HTLM5

```Bash
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1
    <title>COMS10012 软件工具</title>
</head>
<body>
    <h1>COMS10012 软件工具</h1>
    <p>本单元教您Web开发的基础知识。</p>

    <h2>内容</h2>
    <p>内容是来自Mozilla开发者网络的视频讲座和阅读作业的组合。</p>
    <p>您可以在GitHub上找到该单元页面。</p>

    <h2>学习成果</h2>
    <p>完成本单元后，您将能够使用以下功能：</p>
```

```
    <ul>
        <li>HTML5</li>
        <li>CSS</li>
        <li>JavaScript</li>
    </ul>
</body>
</html>
```

## 2.2 Templating

文件 Templates.java 配置了 Thymeleaf 模板引擎。@Component 注解告诉 Spring 管理这个类；其他需要模板的类可以通过声明一个带有 @Autowired 注解的正确类的字段来请求它，就像您在 Controller.java 中看到的那样。
主页仍然从 src/main/resources/web 的类路径中作为 HTML 文件提供。

当请求 /units 时，请看一下整个过程。
首先，方法 unitsPage 访问数据库并加载单位列表。
接下来，它创建一个 Context，这是一个 Thymeleaf 对象，让您可以向模板传递值。在这里，我们添加了一个带有键 units 的值，其中包含单位列表。
最后，我们使用上下文渲染模板并返回结果。如果 Spring 控制器中的方法返回一个字符串，那么假定该字符串包含一个 HTML 页面，并且 Spring 会正确设置 HTTP 状态码和头信息。

模板本身位于 src/main/resources/units.html。任何带有 th: 前缀的内容都会由 Thymeleaf 渲染：

Bash

```
<ul th:each="unit : ${units}">
    <li>
        <a th:href="'/unit/' + ${unit.code}" th:text="${unit.code}"></
        <span th:text="${unit.title}"></span>
    </li>
</ul>
```

th:each 属性相当于 Java 中的 for 循环，在这种情况下相当于 for (Unit unit : units) – 在 Thymeleaf 中，与普通 Java 不同的是，您不需要为循环变量指定类型，但在访问上下文中的变量时需要使用 ${...} 语法。th:each 属性会在每次循环遍历时渲染其自身标签一次（在这种情况下为 <ul>），然后在每次循环中渲染标签内的所有内容一次，因此您会得到每个单位一个 <li> 项。

要创建属性，您需要在属性名称前面加上 th:，所以 th:href 创建了链接的 href 属性（在浏览器中检查结果）。Thymeleaf 属性语法要求您在字符串周围使用单引号，使用 + 类似于 Java 来连接字符串，并在变量周围使用 ${...}。

语法 ${instance.attribute} 通过调用其 getter 方法，从 Java 类的实例中加载属性。在这种情况下,${unit.code} 最终会在 unit 实例上调用 .getCode()，而不是直接访问字段。

th:text 属性是特殊的，它创建了一个标签的内容。例如，这里的 span 标签创建了一个 ，其内容（在开放和闭合标签之间）包含单位的标题。与一些其他模板系统不同，Thymeleaf 不会让您在 HTML 页面的任何位置包含变量 – 一些模板引擎允许您在 HTML 页面中的任何位置只写 ${variable}，但是 Thymeleaf 只允许您将变量放在标签的属性中，所以如果您只想显示一些文本，您需要使用一个占位标签（这里 span 是一个很好的选择），并将变量放在其 th:text 属性中。

如果您点击一个单位，您将进入该单位的详细页面（目前该页面除了代码和标题外没有更多的细节）。当您发送一个对 /unit/UNITCODE，例如 /unit/COMSM0085 的 HTTP 请求时，这将由 unitDetailPage 方法处理。

请注意，@GetMapping 包含一个模式 /unit/{code}，其中带有一个参数，用于匹配所有这种形式的 URL，并且该参数作为一个参数传递给方法，并使用 @PathVariable 注解告诉 Spring 如何处理它，即从 HTTP 请求的路径中填充其值。

该方法首先在数据库中查找单位，并在没有该代码的单位时返回一个 404 错误页面。（在真实的数据库中，将会有一个“按代码查找单位”的方法，您不必加载所有单位才能找到其中一个。）

之后，我们调用 Thymeleaf 渲染 unit.html 模板页面（这是没有 's' 的模板名称），以渲染单位页面。

TLS

CA（Certificate Authority，证书颁发机构）是负责颁发和管理数字证书的信任机构。数字证书是用于在网络通信中进行身份验证和加密的一种安全机制，由 CA 颁发的数字证书包含了公钥和证书持有者的身份信息，并用 CA 的数字签名进行签名，以确保证书的真实性和完整性。CA 的主要职责包括：1. **颁发数字证书：** CA 根据验证的身份信息，向申请者颁发数字证书，证明了该申请者的身份信息和公钥的关联关系。数字证书包含了证书持有者的身份信息、公钥、证书有效期等信息。2. **证书管理：** CA 负责管理颁发的数字证书，包括证书的签发、更新、撤销等操作。CA 维护证书吊销列表（Certificate Revocation List，CRL），记录已经被撤销的证书信息，确保已经被吊销的证书不再被信任使用。3. **信任验证：** CA 的数字签名被广泛地信任于公众，因此由 CA 颁发的数字证书也得到了广泛的信任。通信双方可以通过验证证书的签名和证书链的有效性，来确认证书的真实性和可信度，从而实现通信双方的身份验证和数据加密。4. **根证书管理：** CA 的根证书是用于签署其他证书的最高级别的数字证书，它构成了数字证书信任链的根。CA 负责管理和保护根证书的私钥，以确保证书颁发和验证的安全可靠。CA 在网络安全中扮演着重要的角色，它为网络通信提供了安全基础，保护了通信的隐私性和完整性。HTTPS 等基于 TLS 的安全通信协议就是依赖于 CA 颁发的数字证书来实现通信双方的身份验证和数据加密的。

URI schemes

REST

Tables

Datatable datatables.net