O Protocolo HTTP: Fundamentos e Evolução

Pedroso

UFPR

26 de junho de 2025

1. Introdução ao HTTP

- O **HTTP (Hypertext Transfer Protocol)** é o protocolo de camada de aplicação para transferência de hipertexto da World Wide Web.
- O *hipertexto* se refere a um texto não linear e que conté4m conexões (links) para outras informações. Versões:
  - ✔ HTTP/1.0 (RFC 1945): A primeira versão, estabelecia uma nova conexão TCP para cada requisição.
  - ✔ HTTP/1.1 (RFC 2616): Versão mais utilizada por décadas, possibilita conexões persistentes (*keepalive*), pipelining de requisições, cache aprimorado e novos métodos, melhorando significativamente a eficiência.
  - ✔ HTTP/2 (RFC 7540): Lançado em 2015, trouxe melhorias de desempenho através de multiplexação (várias requisições/respostas sobre uma única conexão TCP), compressão de cabeçalhos e *serverpush* (servidor pode enviar recursos proativamente).
  - ✔ HTTP/3 (RFC 9114): Versão mais recente, usa QUIC (Quick UDP Internet Connections) em vez de TCP como seu protocolo de transporte, visa reduzir a latência e melhorar o desempenho, especialmente em redes com perda de pacotes.

2. Especificação do HTTP

- **RFC 2616 (HTTP/1.1):** A especificação mais amplamente utilizada e fundamental.
- **Protocolo Cliente-Servidor:** O cliente (geralmente um navegador) envia requisições e o servidor responde.
- **Sem Estado (Stateless):** Cada requisição é independente das anteriores. O servidor não guarda informações sobre o cliente entre as requisições (mas sessões podem ser gerenciadas com cookies).
- **Baseado em Texto:** Mensagens são legíveis por humanos.

3. Formato do Header HTTP

O cabeçalho HTTP contém informações meta sobre a requisição ou resposta.

Estrutura básica:

```
Campo-Nome: Valor
```

Exemplos de campos comuns:

- **Host:** `www.example.com` (requisição)
- **User-Agent:** `Mozilla/5.0 (...)` (requisição)
- **Content-Type:** `text/html` (resposta)
- **Content-Length:** `1234` (resposta)
- **Accept:** `text/html,application/xhtml+xml,...` (requisição)
- **Date:** `Tue, 24 Jun 2025 19:20:00 GMT` (resposta)

4. Principais Mensagens (Métodos HTTP)

Os métodos HTTP indicam a ação desejada a ser realizada para um recurso identificado:

- **GET**. Solicita um recurso hipertexto.
  - **Características:** Dados são enviados na URL (query string). Idempotente (múltiplas requisições GET têm o mesmo efeito).
  - **Exemplo de uso:** Acessar uma página web (ver próximo slide).
- **POST**. Enviar dados a um recurso especificado para serem processados:
  - **Características:** Dados são enviados no corpo da requisição. Não é idempotente (múltiplas requisições POST podem ter efeitos diferentes).
  - **Exemplo de uso:**
    - Cliente: `POST /usuarios` (para criar um novo usuário).
    - Servidor: `201 Created` (servidor criou).
- **PUT**. Atualizar um recurso completo:
  - **Características:** Idempotente.
  - **Exemplo de uso:**
    - Cliente: `PUT /usuarios/123` (para atualizar o usuário com ID 123).
    - Servidor: `200 OK` (servidor atualizou).

5. Exemplo de Mensagem HTTP

Requisição HTTP (GET):

```
GET /index.html HTTP/1.1
Host: www.example.com
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
    AppleWebKit/537.36 (KHTML, like Gecko) Chrome/123.0.0.0
    Safari/537.36
Accept:
    text/html,application/xhtml+xml,application/xml;q=0.9,image/avi
Accept-Encoding: gzip, deflate, br
Accept-Language: en-US,en;q=0.9,pt-BR;q=0.8,pt;q=0.7
Connection: keep-alive
```

5. Exemplo de Mensagem HTTP

```
HTTP/1.1 200 OK
Date: Tue, 24 Jun 2025 19:20:00 GMT
Server: Apache/2.4.41 (Ubuntu)
Last-Modified: Mon, 23 Jun 2025 10:00:00 GMT
ETag: "1a2b3c4d5e6f"
Accept-Ranges: bytes
Content-Length: 123
Content-Type: text/html; charset=UTF-8

<!DOCTYPE html>
<html>
<head>
    <title>Minha Página</title>
</head>
<body>
    <h1>Bem-vindo!</h1>
    <p>Esta é uma página de exemplo.</p>
</body>
</html>
```

6. Formato HTML (HyperText Markup Language)

O HTML define a estrutura e o conteúdo de páginas web usando um sistema de **tags**.

Tags HTML

- **Elementos:** Construídos com tags de abertura e fechamento.
  - ✔ Exemplo: `<p>Este é um parágrafo.</p>`
- **Atributos:** Fornecem informações adicionais para as tags.
  - ✔ Exemplo: `<a href="https://www.google.com»Google</a>`
- **Tags comuns:** `<html>`, `<head>`, `<body>`, `<h1>` a `<h6>`, `<p>`, `<a>`, `<img>`, `<ul>`, `<ol>`, `<li>`, `<div>`, `<span>`.

6. Exemplo de Código HTML

```html
<!DOCTYPE html>
<html>
<head>
    <title>Exemplo de HTML</title>
</head>
<body>
    <h1>Minha Primeira Página HTML</h1>
    <p>Este é um <b>parágrafo</b> com texto em <i>negrito</i> e
        <em>itálico</em>.</p>
    <a href="https://www.example.com">Visite o Exemplo.com</a>
    <img src="logo.png" alt="Logo Exemplo" width="100">
    <ul>
        <li>Item 1</li>
        <li>Item 2</li>
    </ul>
</body>
</html>
```

7. Páginas Dinâmicas

- Páginas dinâmicas são geradas no servidor em tempo real, permitindo conteúdo personalizado e interativo.
- A revolução das aplicações Web: A transição de aplicações locais (desktop) para aplicações web representa uma das maiores transformações na computação moderna, oferecendo vantagens que redefinem a forma de uso e distribuição de software, representa uma evolução estratégica que beneficia usuários e empresas de diversas formas:
  - ✔ Acesso de qualquer lugar, a aualquer hora.
  - ✔ Independência de sistema operacional.
  - ✔ Colaboração simplificada.
  - ✔ Reduz custos e necessidade de equipes para manutenção das instalações.

CGI-BIN (Common Gateway Interface)

- **Conceito:** Um padrão para que servidores web executem programas externos (scripts) para gerar conteúdo dinâmico.
- **Funcionamento:** O servidor passa dados da requisição HTTP para o script CGI e envia a saída do script de volta ao cliente.
- **Limitações:** Pode ser ineficiente devido à criação de um novo processo para cada requisição.
- **Linguagens:** Perl, C/C++, Python, Shell Script (Bash, Sh), PHP, Ruby, Tcl, Visual Basic (em ambientes Windows), AppleScript;

7.4. Páginas Dinâmicas em C (via CGI)

É possível gerar páginas dinâmicas usando a linguagem C, tipicamente através do protocolo **CGI (Common Gateway Interface)**.

- O servidor web executa um programa C compilado (binário) quando uma requisição CGI é feita.
- O programa C lê os dados da requisição (via variáveis de ambiente e/ou entrada padrão `stdin`).
- Ele imprime o conteúdo da página web (incluindo cabeçalhos HTTP) na saída padrão (`stdout`), que é então enviada de volta ao cliente pelo servidor web.
- **Vantagens:** Alto desempenho (após a inicialização do processo), controle granular sobre o sistema.
- **Desvantagens:** Mais complexo de desenvolver e depurar, cada requisição geralmente inicia um novo processo (o que pode ser ineficiente para alta carga).

7.4.1. Exemplo de Página Dinâmica em C

Codigo C (`meu_cgi.c`):

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int main() {
    // Imprime o cabeçalho HTTP necessário
    printf("Content-Type: text/html\n\n");

    // Início do documento HTML
    printf("<!DOCTYPE html>\n");
    printf("<html>\n");
    printf("<head>\n");
    printf("    <title>Pagina Dinamica em C</title>\n");
    printf("</head>\n");
    printf("<body>\n");
    printf("    <h1>Ola do Servidor C!</h1>\n");

    // Gera conteudo dinamico
    time_t rawtime;
    struct tm *info;
    char buffer[80];

    time(&rawtime);
    info = localtime(&rawtime);
    strftime(buffer, 80, "Data e Hora Atual: %d/%m/%Y %H:%M:%S", info);

    printf("    <p>%s</p>\n", buffer);
```

7.4.1. Exemplo de Página Dinâmica em C

Codigo C (`meu_cgi.c`):

```c

    // Exemplo de leitura de variável de ambiente (do servidor web)
    char *metodo_requisicao = getenv("REQUEST_METHOD");
    if (metodo_requisicao) {
        printf("    <p>Método da Requisição: %s</p>\n", metodo_requisicao);
    }

    // Fim do documento HTML
    printf("    <p>Esta pagina foi gerada dinamicamente por um programa C.</p>\n");
    printf("</body>\n");
    printf("</html>\n");

    return 0;
}
```

7.4.2. Compilação e Configuração (Conceitual)

Para que o exemplo anterior funcione, os passos conceituais seriam:

1. **Compilar o Código C:**
   - ✔ Compile `meu_cgi.c` para criar um executável:
     `gcc meu_cgi.c -o meu_cgi.cgi`
   - ✔ **Colocar o Executável no Diretório CGI-BIN:**
     - ✓ Mova `meu_cgi.cgi` para o diretório configurado no seu servidor web como `cgi-bin` (ex: `/var/www/cgi-bin/`).
     - ✓ Certifique-se de que o executável tem permissões de execução (ex: `chmod +x meu_cgi.cgi`).
   - ✔ **Acessar via Navegador:**
     - ✓ Via uma URL como:
       `http://seuservidor.com/cgi-bin/meu_cgi.cgi`

7.4.2. Compilação e Configuração (Conceitual)

Para que o exemplo anterior funcione, os passos conceituais seriam:

1. **Compilar o Código C:**
   - ✔ Compile `meu_cgi.c` para criar um executável:
     `gcc meu_cgi.c -o meu_cgi.cgi`
   - ✔ **Colocar o Executável no Diretório CGI-BIN:**
     - ✓ Mova `meu_cgi.cgi` para o diretório configurado no seu servidor web como `cgi-bin` (ex: `/var/www/cgi-bin/`).
     - ✓ Certifique-se de que o executável tem permissões de execução (ex: `chmod +x meu_cgi.cgi`).
   - ✔ **Acessar via Navegador:**
     - ✓ Via uma URL como:
       `http://seuservidor.com/cgi-bin/meu_cgi.cgi`

Nota: A configuração exata do CGI varia entre servidores web (Apache, Nginx, etc.).

Linguagens para Desenvolvimento Web Dinâmico

- **PHP (Hypertext Preprocessor):**
  - ✔ Linguagem de script de código aberto amplamente utilizada para desenvolvimento web.
  - ✔ Código PHP é embutido no HTML e executado no servidor.
- **Perl (Practical Extraction and Report Language):**
  - ✔ Inicialmente popular para scripts CGI, ainda utilizada, mas menos dominante em web do que PHP.
  - ✔ Forte em manipulação de texto.
- **Outras:** Python (Django, Flask), Ruby (Ruby on Rails), Node.js, ASP.NET.

7.3. Exemplo em PHP

```
<!DOCTYPE html>
<html>
<head>
    <title>Pagina PHP Dinamica</title>
</head>
<body>
    <h1>Bem-vindo!</h1>
    <?php
        $nome = "Usuário";
        echo "<p>Olá, " . $nome . "! A data de hoje é: " . date(
            "d/m/Y") . "</p>";
    ?>
    <p>Este conteúdo foi gerado dinamicamente pelo PHP.</p>
</body>
</html>
```

8. Uso de JavaScript

JavaScript é uma linguagem de script executada no **lado do cliente** (no navegador).

- **Interatividade:** Permite criar páginas web interativas sem recarregar a página.
- **Manipulação do DOM:** Modifica o conteúdo, estilo e estrutura de uma página HTML em tempo real.
- **Requisições Assíncronas (AJAX):** Permite buscar dados do servidor sem recarregar a página inteira.

8. Exemplo de JavaScript

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4      <title>Exemplo de JavaScript</title>
5  </head>
6  <body>
7      <h1 id="titulo">Clique no botão!</h1>
8      <button onclick="mudarTexto()">Mudar Texto</button>
9
10     <script>
11         function mudarTexto() {
12             document.getElementById("titulo").innerHTML = "O
                   texto foi alterado!";
13         }
14     </script>
15 </body>
16 </html>
```

Exemplos

- Clique no link para ver o resultado.
- Veja o código fonte da página.
  - ✔ Exemplo *Javascript*.
  - ✔ Exemplo *PHP*.
  - ✔ Exemplo *HTML5*.

9. Padrão XML (Extensible Markup Language)

O **XML** é uma linguagem de marcação que define um conjunto de regras para codificar documentos de uma forma que seja legível por humanos e por máquinas.

- **Propósito:** Foco na estrutura e significado dos dados, não na apresentação.
- **Uso:** Troca de dados entre sistemas, configuração de aplicações, armazenamento de dados.
- **Estrutura:** Usa tags para definir elementos, semelhantes ao HTML, mas as tags são definidas pelo usuário.

9. Exemplo de Uso de XML

```xml
<?xml version="1.0" encoding="UTF-8"?>
<livros>
    <livro id="1">
        <titulo>O Senhor dos Anéis</titulo>
        <autor>J.R.R. Tolkien</autor>
        <ano>1954</ano>
    </livro>
    <livro id="2">
        <titulo>1984</titulo>
        <autor>George Orwell</autor>
        <ano>1949</ano>
    </livro>
</livros>
```

9.1. Vantagens do XML

O XML oferece diversas vantagens para a troca e organização de dados:

- **Legibilidade:** Tanto por humanos quanto por máquinas, facilitando a depuração e o entendimento.
- **Extensibilidade:** Permite criar tags personalizadas, adaptando-se a qualquer tipo de dado.
- **Hierarquia:** Estrutura os dados de forma hierárquica (árvore), o que é natural para muitos tipos de informação.
- **Independência de Plataforma:** Não está vinculado a nenhuma tecnologia ou sistema operacional específico.
- **Padrão Aberto:** Amplamente adotado e mantido por um consórcio internacional (W3C).
- **Separção de Dados e Apresentação:** O XML foca nos dados, enquanto a apresentação é feita por outras tecnologias (CSS, XSLT).

9.2. XML vs. HTML: Principais Diferenças

Embora ambos sejam linguagens de marcação e usem tags, HTML e XML têm propósitos distintos:

HTML (HyperText Markup Language)

- **Propósito:** Exibição de conteúdo.
- **Tags Pré-definidas:** Usa um conjunto fixo de tags (ex: `<h1>`, `<p>`, `<a>`).
- **Tolerante a Erros:** Navegadores tentam renderizar mesmo com erros de sintaxe.
- **Foco:** Como o conteúdo deve *parecer* no navegador.

XML (Extensible Markup Language)

- **Propósito:** Armazenamento e transporte de dados.
- **Tags Definíveis pelo Usuário:** Permite criar tags personalizadas para descrever dados (ex: `<livro>`, `<autor>`).
- **Estrito:** Requer sintaxe perfeita (bem-formado).
- **Foco:** O que os dados *são*.

9.3. Tecnologias Relacionadas ao XML

O XML não atua sozinho; diversas tecnologias complementam seu uso:

- **DTD (Document Type Definition) / XML Schema:**
  - ✔ Definem a estrutura e o conteúdo válidos para documentos XML.
  - ✔ XML Schema é mais moderno e poderoso que DTD.
- **XSL (Extensible Stylesheet Language):**
  - ✔ **XSLT (XSL Transformations):** Transforma documentos XML em outros formatos (HTML, texto, outro XML).
  - ✔ **XPath:** Linguagem para selecionar nós em um documento XML.
  - ✔ **XSL-FO (XSL Formatting Objects):** Formatação para saída impressa.
- **SAX / DOM:** APIs para parsing e manipulação de documentos XML em linguagens de programação.
- **AJAX (Asynchronous JavaScript and XML):** Uso de XML (ou JSON) para troca de dados assíncrona entre cliente e servidor.

9.4. Validação de XML: DTD e XML Schema

Para garantir que um documento XML segue uma estrutura esperada, usamos esquemas de validação.

- **DTD (Document Type Definition):**
  - ✔ Forma mais antiga de definir a estrutura de um XML.
  - ✔ Sintaxe não-XML.
  - ✔ Limitado em tipos de dados e namespaces.
- **XML Schema (XSD - XML Schema Definition):**
  - ✔ Mais moderno e poderoso que DTD.
  - ✔ Escrito em XML, o que facilita o processamento.
  - ✔ Suporta tipos de dados (inteiro, string, data, etc.), namespaces e estruturas complexas.
  - ✔ Permite definir restrições mais detalhadas.

9.4.1. Exemplo de DTD

Documento XML:

```
<!DOCTYPE biblioteca SYSTEM "biblioteca.dtd">
<biblioteca>
    <livro id="L001">
        <titulo>Dom Casmurro</titulo>
        <autor>Machado de Assis</autor>
    </livro>
</biblioteca>
```

biblioteca.dtd:

```
<!ELEMENT biblioteca (livro+)>
<!ELEMENT livro (titulo, autor)>
<!ATTLIST livro id CDATA #REQUIRED>
<!ELEMENT titulo (#PCDATA)>
<!ELEMENT autor (#PCDATA)>
```

9.4.2. Exemplo de XML Schema (XSD)

biblioteca.xsd:

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
3
4    <xs:element name="biblioteca">
5      <xs:complexType>
6        <xs:sequence>
7          <xs:element name="livro" maxOccurs="unbounded">
8            <xs:complexType>
9              <xs:sequence>
10               <xs:element name="titulo" type="xs:string"/>
11               <xs:element name="autor" type="xs:string"/>
12             </xs:sequence>
13             <xs:attribute name="id" type="xs:ID" use="required"/
                   >
14           </xs:complexType>
15         </xs:element>
16       </xs:sequence>
17     </xs:complexType>
18   </xs:element>
19
20 </xs:schema>
```

10. HTML5

HTML5 é a quinta e última revisão principal da linguagem HTML, introduzindo novos recursos e melhorias significativas.

- **Novas Tags Semânticas:** `<header>`, `<footer>`, `<nav>`, `<article>`, `<section>`, `<aside>`, `<main>` - Melhoram a estrutura e a acessibilidade.
- **Multimídia Natiiva:** `<audio>` e `<video>` - Suporte nativo para áudio e vídeo sem a necessidade de plugins (como Flash).
- **Novos Tipos de Input em Formulários:** `email`, `url`, `number`, `date`, `range`, `color` - Melhoram a validação e a experiência do usuário.

10. HTML5 (continuação)

- **APIs Gráficas:** `<canvas>` (para desenhos e animações) e SVG (Scalable Vector Graphics).
- **Armazenamento Local:** `localStorage` e `sessionStorage` - Permitem armazenar dados no navegador do cliente.
- **Web Workers:** Execução de scripts em segundo plano.
- **Web Sockets:** Comunicação bidirecional em tempo real.

10. Exemplo de HTML5 (com tags semânticas)

```
1  <!DOCTYPE html>
2  <html lang="pt-BR">
3  <head>
4      <meta charset="UTF-8">
5      <meta name="viewport" content="width=device-width, initial-
           scale=1.0">
6      <title>Exemplo de HTML5</title>
7  </head>
8  <body>
9      <header>
10         <h1>Meu Site HTML5</h1>
11         <nav>
12             <ul>
13                 <li><a href="#home">Home</a></li>
14                 <li><a href="#sobre">Sobre</a></li>
15                 <li><a href="#contato">Contato</a></li>
16             </ul>
17         </nav>
18     </header>
19
20     <main>
21         <section id="home">
22             <h2>Bem-vindo!</h2>
```

11. Tendências Futuras do HTTP

O HTTP continua a evoluir para atender às crescentes demandas da web, com foco em desempenho, segurança e novas capacidades.

- **HTTP/3 e QUIC:**
  - ✔ Substituição do TCP pelo protocolo **QUIC** (Quick UDP Internet Connections) como camada de transporte.
  - ✔ Benefícios: Redução da latência (zero-RTT ou 1-RTT para handshakes), melhor desempenho em redes instáveis (sem "Head-of-Line Blocking"para múltiplos streams), e migração de conexão (mantém a conexão IP mesmo mudando de rede).
- **WebTransport:**
  - ✔ Uma API web que expõe os recursos do QUIC para desenvolvedores de aplicações web.
  - ✔ Permite comunicação bidirecional e multiplexada de baixa latência entre o cliente e o servidor, ideal para jogos, vídeo ao vivo e aplicativos em tempo real.

11. Tendências Futuras do HTTP (continuação)

- **HTTP/2 e HTTP/3 com Server Push:**
  - ✔ Capacidade do servidor de enviar proativamente recursos (CSS, JS, imagens) para o cliente antes mesmo que o navegador os solicite explicitamente.
  - ✔ Otimiza o tempo de carregamento da página, pois os recursos já estão disponíveis quando o navegador precisa deles.
- **Segurança e Privacidade (HTTPS Everywhere):**
  - ✔ A adoção do **HTTPS** (HTTP sobre TLS/SSL) é a norma, garantindo comunicação criptografada e autenticada.
  - ✔ Tendências incluem HTTP Strict Transport Security (HSTS) para forçar HTTPS, e o foco contínuo em eliminar o HTTP não seguro.
- **WebAssembly (Wasm):**
  - ✔ Não é diretamente uma mudança no HTTP, mas impacta como as aplicações são entregues e executadas na web.
  - ✔ Permite executar código de linguagens como C++, Rust, Go com desempenho quase nativo no navegador, abrindo portas para aplicações web muito mais complexas e exigentes.