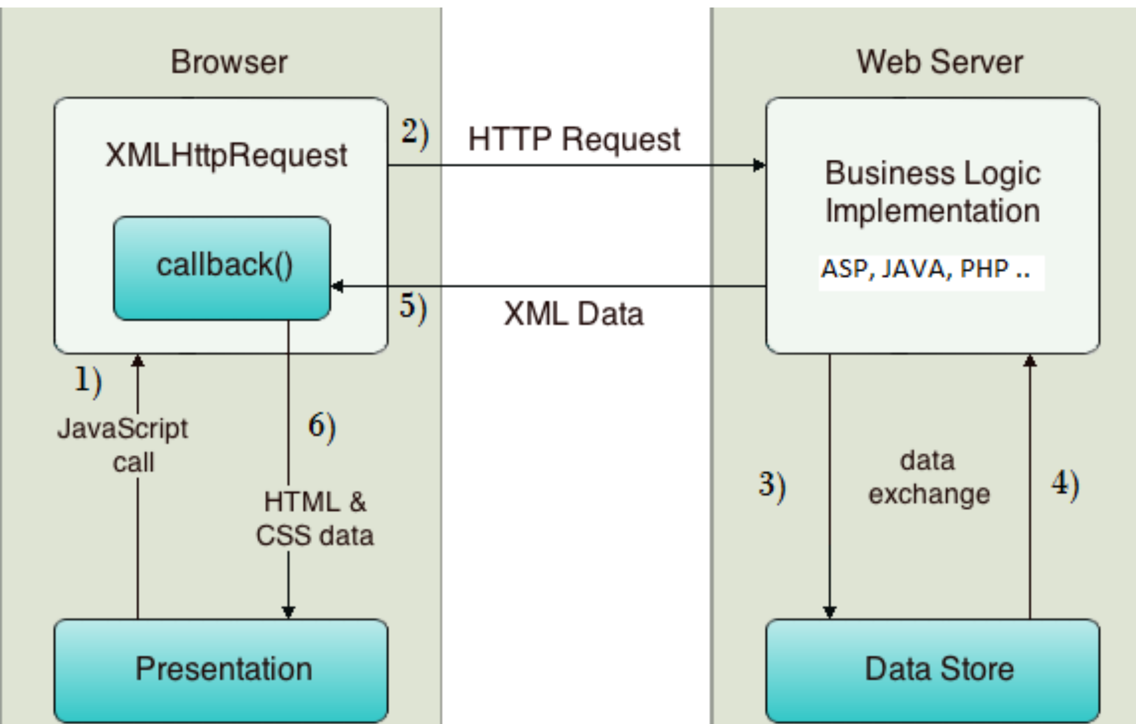


DCM

5954025 - Sistemas Distribuídos

Aula 08 - Comunicação via HTTP: XHR

Prof. Dr. Denis M. L. Martins
DCM | FFCLRP | USP

Objetivos de Aprendizagem

Compreender como o objeto XMLHttpRequest permite a troca assíncrona de dados em sistemas distribuídos, eliminando a necessidade de recarregar páginas web.

- Entender o que é XHR
- Comparar com outras abordagens (fetch, gRPC)
- Implementar exemplos práticos

Evolução da Comunicação Web

- Nos anos 90, qualquer interação exigia o recarregamento total da página.
- O XMLHttpRequest (XHR) permitiu que o JavaScript fizesse requisições HTTP em segundo plano.
- Desenvolvido pela Microsoft para Internet Explorer.
- Surgiu a técnica AJAX (Asynchronous JavaScript and XML) para interfaces dinâmicas.
- A comunicação tornou-se mais fluida e eficiente para o usuário final.

Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

O formato XML

- Extensible Markup Language (XML)
  - Derivada da linguagem SGML
- Padronizado pela W3C para troca de dados.
- Foco em estrutura de dados, não visualização.
- Extensão : `.xml`

> Imagem ao lado: Jon Bosak, que liderou a criação da XML.

O formato XML (cont.)

- Elementos definidos por tags (`<tag>`).
- Atributos dentro das tags (ex: `id="..."`).
- Hierarquia de elementos (pai e filhos).
- Sensível a maiúsculas e minúsculas.

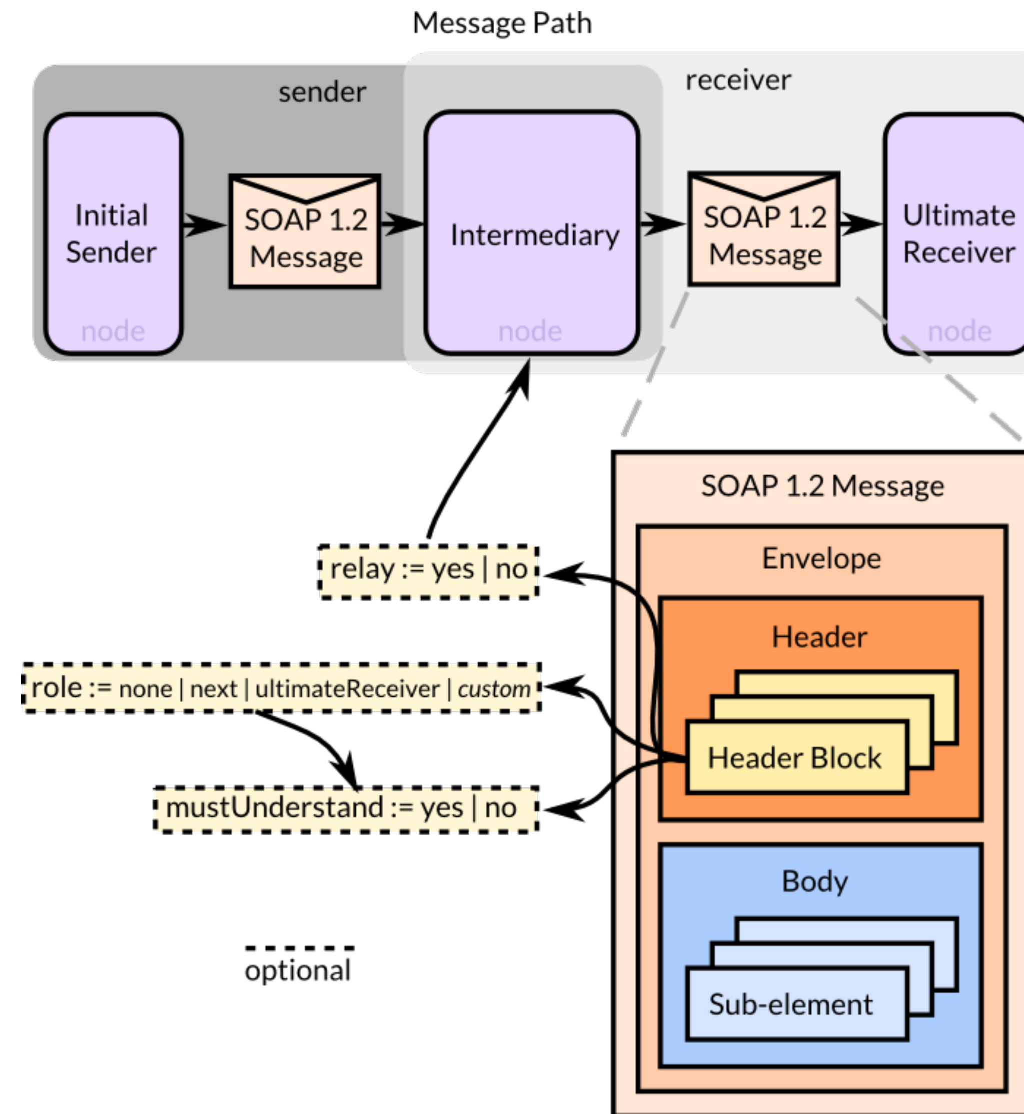
```xml
<produtos>
  <categoria id="eletronicos">
    <item nome="Celular"/>
  </categoria>
</produtos>
```

Aplicações de XML

- Web Services via Simple Object Access Protocol (SOAP).
- Configuração de servidores distribuídos.
- Feeds RSS para agregação de conteúdo.
- Troca de dados entre bancos de dados heterogêneos.

```
<soap:Envelope>
  <soap:Body>
    <statusTransacao>APROVADO</statusTransacao>
  </soap:Body>
</soap:Envelope>
```

> Fonte da imagem: The Customize Windows



Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

O que é XHR?

- XMLHttpRequest (XHR): API para requisições HTTP.

```
let xhr = new XMLHttpRequest();
```

- Comunicação assíncrona em aplicações web
- Quando foi adicionada ao Internet Explorer, foi permitido fazer coisas com JavaScript que eram bem difíceis anteriormente.
  - **Exemplo**: lista de sugestões enquanto o usuário digitava algo em um campo de texto
- Modelo de comunicação simples: Cliente solicita dados e Servidor responde

> Apesar do nome, pode processar qualquer formato de dado, como JSON ou texto puro.

Funcionamento da XHR

Criar objeto XHR $\rightarrow$ Abrir conexão $\rightarrow$ Enviar requisição $\rightarrow$ Receber resposta

```
var xhr = new XMLHttpRequest();
xhr.open("GET", "data/cd_catalog", false);
xhr.send();
console.log(xhr.status, xhr.statusText);
console.log(xhr.responseText);
```

Nota: Para executar o exemplo acima, crie um arquivo **exemplo.html** e uma pasta **data**. Faça o download to arquivo **cd_catalog.xml** em https://www.w3schools.com/xml/cd_catalog.xml e salve-o na pasta **data**.

Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Funcionamento da XHR (cont.)

```
xhr.open(method, URL, [async, user, password])
```

- **method** – método HTTP. Geralmente "GET" ou "POST".
- **URL** – a URL para solicitar, uma string, pode ser objeto URL.
- **async** – se definido explicitamente como false, a requisição é síncrona.
- **user**, **password** – login e senha para autenticação básica HTTP (se necessário).

Importante:

- A chamada **open**, ao contrário do nome, não abre a conexão. Ela apenas configura a requisição.
- A atividade de rede começa apenas com a chamada de **send**.

Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Funcionamento da XHR (cont.)

- O método `open()` inicializa os parâmetros básicos da conexão.
  - Define o método HTTP (GET para busca, POST para criação, etc.).
  - Permite definir se a chamada será assíncrona (padrão) ou síncrona.
- O método `send()` inicia a atividade na rede e abre a conexão.
  - Em requisições POST, o corpo dos dados (como um JSON) é enviado como argumento.
  - Para requisições GET, o parâmetro costuma ser nulo ou vazio.

Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Exemplo

```
<body>
<h1>Lista de Produtos</h1>
<button onclick="carregar()">Carregar Dados</button>
<ul id="lista"></ul>
<script>
function carregar() {
    const lista = document.getElementById("lista");
    lista.innerHTML = "<li>Carregando...</li>";
    const xhr = new XMLHttpRequest();
    xhr.open("GET", "https://api.restful-api.dev/objects", true);
    xhr.send();
    if (xhr.status === 200) {
        const dados = JSON.parse(xhr.responseText);
        lista.innerHTML = "";
        dados.forEach(item => {
            const li = document.createElement("li");
            let info = item.name;
            if (item.data) {
                info += " | " + JSON.stringify(item.data);
            }
            li.innerText = info;
            lista.appendChild(li);
        });
    } else {
        lista.innerHTML = "<li>Erro ao carregar</li>";
    }
}
</script>
</body>
```


Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Estados da Requisição

Constantes de Estado `readyState`:

- **UNSENT = 0**: estado inicial
- **OPENED = 1**: método **open** chamado
- **HEADERS_RECEIVED = 2**: cabeçalhos da resposta recebidos
- **LOADING = 3**: resposta sendo carregada (pacote de dados recebido)
- **DONE = 4**: requisição completa

```
xhr.onreadystatechange = function() {
  if (xhr.readyState == 3) {
    // loading
  }
  if (xhr.readyState == 4) {
    // request finished
  }
};
```

> **Nota:** Esses valores indicam o progresso da requisição HTTP. O estado muda automaticamente durante a execução.

Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Respostas e Códigos de Status

- O **status** retorna o código HTTP do servidor (ex: 200 para sucesso, 404 para erro).
- **responseText** ou **response** contém o conteúdo retornado pelo servidor.
- É possível definir o **responseType** para o navegador já converter o dado (ex: "json").

```
try {
  xhr.send();
  if (xhr.status != 200) {
    alert(`Error ${xhr.status}: ${xhr.statusText}`);
  } else {
    alert(xhr.response);
  }
} catch(err) { // instead of onerror
  alert("Request failed");
}
```

Tratamento de Erros e Robustez

O evento `onerror` é disparado em falhas de rede ou URLs inválidas.

```
xhr.onerror = function() {
    console.log("Erro na requisição");
};
```

A propriedade `timeout` evita que o cliente espere infinitamente por uma resposta.

```
xhr.timeout = 2000;

xhr.ontimeout = function() {
    console.log("Tempo excedido");
};
```

> **Nota**: O método `abort()` permite cancelar requisições desnecessárias em andamento.

XHR e JSON

- Formato comum de dados
- Fácil de manipular

Exemplo:

```
let xhr = new XMLHttpRequest();
xhr.open("GET", "https://api.restful-api.dev/objects");
xhr.onload = function() {
    if (xhr.status === 200) {
        const data = JSON.parse(xhr.responseText);
        console.log(data);
    }
};
xhr.send();
```


Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Comparação: XHR vs Fetch

XHR:

- mais antigo
- mais verboso

Fetch:

- mais moderno
- usa Promises

```
fetch("https://api.restful-api.dev/objects")
  .then(response => response.json())
  .then(data => console.log(data));
```


Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Questões para Estudo

1. XHR *garante* consistência dos dados?
2. O que caracteriza o modelo de comunicação assíncrona do XHR e qual sua vantagem para sistemas distribuídos?
3. Um desenvolvedor configurou um XHR para buscar dados de um banco distribuído, mas a tela trava até a resposta chegar. Qual parâmetro do método `open()` foi configurado incorretamente? Explique o porquê.
4. Você está projetando um sistema de monitoramento de sensores em tempo real. Como você implementaria o tratamento de falhas usando as propriedades `timeout`, `status` e o evento `onerror` para garantir que o sistema não fique inconsistente?

Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Conclusão

XHR: Interface clássica para requisições HTTP no navegador.

- **Assincronia:** Essencial para não travar a interface do usuário.
- **Dados:** XML estruturado vs JSON leve (preferência moderna).
- **Estados:** `readyState` indica o progresso da requisição.

Próximos Passos:

- Faça o lab prático em https://denmartins.github.io/labs/2026-04-22-sd-xhr
- Estude **Fetch API**, interface mais moderna e limpa que o XHR
- Crie **REST APIs:** como padrão atual para comunicação cliente-servidor

Dúvidas e Discussão

Prof. Dr. Denis M. L. Martins | martins.denis@usp.br