



com JQuery

AJAX

ALLAN GUIMARÃES



ALLAN GUIMARÃES

Web Developer Full Stack com conhecimentos em Node.js e Laravel. Formado em Análise e Desenvolvimento de Sistemas, é fluente em inglês, espanhol e italiano. Aspirante a uma pós-graduação em IA e Machine Learning, Allan busca intensificar seu conhecimento para realizar a integração de sistemas, visando otimizar o trabalho humano em diversas áreas. Ele está sempre olhando para um futuro mais harmonioso entre máquinas e seres humanos.

ÍNDICE

Apresentação	01
O que é AJAX	02
Requisição Simples	03
Parâmetros da requisição \$.get()	03
Conheça o \$.getJSON()	04
O poder do \$.ajax()	05
Múltiplas requisições	06
Resumo	07
Tabela de definições	07



Olá, Dev! 😎

Bem-vindo ao nosso eBook sobre requisições AJAX com jQuery! Vamos partir do princípio que você já tem uma noção básica de JavaScript e jQuery. Vou te guiar desde os conceitos mais básicos até exemplos mais avançados de uma maneira simples e divertida.

O que é Ajax? 🤔

Imagine que você está pedindo uma pizza. Na forma tradicional, você teria que esperar a pizza chegar para continuar fazendo qualquer outra coisa (chato, né?).

O Ajax permite que você continue fazendo outras coisas enquanto a pizza está a caminho. Em vez de esperar pela resposta e atualizar a página inteira, ele atualiza apenas as partes necessárias, deixando tudo mais rápido e dinâmico. Isso significa uma experiência melhor para quem usa o site e menos trabalho para os servidores. Bem legal, né? 🍕

Vamos começar inserindo o script JQuery. Pode ser no fim do body ou no head, porém com o atributo defer.

```
1 <script src="https://code.jquery.com/jquery-3.6.0.min.js"></script>
```

Começando pelo mais simples. 😊

Este é o método mais simples de requisição AJAX. Ele é bem fácil de usar, mas oferece bem menos controle sobre a requisição. Ainda sim, é um ótimo ponto de partida!

```
1 <script>
2     $.get("https://api.example.com/userinfo", { id: 12345 })
3         .done(function (data) {
4             console.log("Dados recebidos:", data);
5         })
6         .fail(function (jqXHR, textStatus, errorThrown) {
7             console.error("Erro na requisição: " + textStatus, errorThrown);
8         })
9         .always(function () {
10            console.error("Executarei sempre");
11        });
12 </script>
```

Entendendo os parâmetros no .fail 🕵️

O jqXHR é o aprimoramento do objeto XMLHttpRequest nativo do navegador, com várias opções extras.

status: O código de status HTTP da resposta. Sabe aquele "404 Not Found" ou "500 Internal Server Error"? Então, é isso!

statusText: O texto da mensagem de status HTTP. Tipo um "Oops, deu ruim!".

responseText: O corpo da resposta em forma de string. Imagina receber uma carta e ler o conteúdo.

responseJSON: O corpo da resposta como um objeto JSON. Quando a resposta é JSON, o jQuery faz a mágica de transformar isso em um objeto pra você.

getAllResponseHeaders(): Retorna todos os cabeçalhos de resposta como uma string. Como pegar a lista de ingredientes de uma receita.

getResponseHeader(name): Retorna o valor de um cabeçalho de resposta específico.

O textStatus é uma string que descreve o tipo de erro que ocorreu. Pode ser um dos seguintes valores:

"timeout": A requisição ultrapassou o tempo limite.

"error": Um erro genérico ocorreu.

"abort": A requisição foi abortada.

"parsererror": A análise da resposta JSON falhou.

O errorThrown é um objeto ou string que contém a exceção capturada pelo navegador ou uma mensagem de erro textual. É opcional e pode ser undefined dependendo do tipo de erro.

Um passo além...

Esse próximo método ainda é simples, mas já oferece um pouco mais de controle. Aqui, estamos seguros de que a resposta será um JSON. Este método é isolado, não há um \$.postJSON. Caso tenha interesse em usar o POST deverá optar pelo uso do \$.ajax.

```
1 <script>
2     $.getJSON("https://api.example.com/userinfo")
3         .done(function (data) {
4             $("#nomeUsuario").text(data.nome);
5             $("#emailUsuario").text(data.email);
6         })
7         .fail(function (jqXHR, textStatus, errorThrown) {
8             $("#errorMsg").text("Erro ao carregar informações do usuário.");
9         });
10 </script>
```

Controle total com `$.ajax` 😎

O método `$.ajax` oferece mais flexibilidade e controle, permitindo que você configure a requisição de forma muito mais detalhada. Abaixo está um exemplo com quase todas as possibilidades de configurações, apenas para fins de exposição das possibilidades.

No final do eBook, você encontrará uma breve explicação da utilidade de cada configuração.

```
1 <script>
2     $.ajax({
3         url: 'https://api.example.com/data',
4         type: 'GET',
5         async: false,
6         headers: {
7             'Authorization': 'Bearer token123',
8             'X-Custom-Header': 'Value'
9         },
10    })
```

```
10     beforeSend: function (xhr) {
11         xhr.setRequestHeader('Authorization', 'Bearer your-token-here');
12         $('#loading').show();
13         xhr.setRequestHeader('Cache-Control', 'no-cache');
14         xhr.setRequestHeader('Pragma', 'no-cache');
15     },
16     timeout: 5000,
17     contentType: 'application/json',
18     statusCode: {
19         404: function () {
20             console.log('Página não encontrada');
21         },
22         500: function () {
23             console.log('Erro interno do servidor');
24         }
25     },
26     dataType: 'json',
27     contents: {
28         xml: /xml/,
29         html: /html/,
30         json: /json/
31     },
32     cache: false,
33     success: function (response) {
34         console.log(response);
35     },
36     error: function (xhr, status, error) {
37         console.error('Error:', error);
38     },
39     complete: function (xhr, status) {
40     }
41 });
42 </script>
```

Este método é útil quando você precisa fazer múltiplas requisições AJAX de forma assíncrona e deseja realizar alguma ação somente após todas as requisições terem sido concluídas com sucesso.



```
1 <script>
2     $.when($.ajax("/endpoint1"), $.ajax("/endpoint2"))
3         .done(function (response1, response2) {
4             console.log("Ambas as requisições completadas");
5             console.log("Resposta 1:", response1);
6             console.log("Resposta 2:", response2);
7         });
8 </script>
```

Resumo das Diferenças: 😎

\$.get: Mais simples, para requisições GET básicas.

\$.getJSON: Para requisições GET que esperam respostas JSON, com sintaxe específica para esse formato.

\$.ajax: Mais flexível e complexo, permitindo configurar qualquer tipo de requisição e controlar todos os aspectos dela.

Tabela das coisas ! 🕵️

beforeSend ⇒ Mostrar indicadores de carregamento, adicionar cabeçalhos, modificar dados de requisição, validar ou cancelar a requisição.

async ⇒ Default é true e mantém a requisição assíncrona. Definir false irá torná-la síncrona. O que é fortemente desencorajado devido ao desempenho.

accepts ⇒ Conjunto de chave e valor que informará ao servidor qual tipo de response será aceita no return. Ex: application/json.

cache ⇒ Define se o navegador irá armazenar ou não cache da requisição. Default é true, funcionará bem com requisições HEAD e GET. Outra opção é apenas acrescentar aos parametros GET “_= {timestamp}”. Ex: url: 'https://api.example.com/data?_= ' + new Date().getTime() .

complete ⇒ É uma função chamada quando a requisição Ajax é completada, independentemente se a requisição foi bem-sucedida ou resultou em erro. Essa função é chamada após o callback success ou error.

contents ⇒ Permite especificar como o jQuery irá analisar a resposta da requisição, com base em seu tipo de conteúdo.

contentType ⇒ Permite especificar o tipo de conteúdo que será enviado para o servidor. Se você não especificar um tipo de conteúdo, o padrão "application/x-www-form-urlencoded; charset=UTF-8" será utilizado, o que é adequado para enviar dados em formulários HTML padrão.

timeout ⇒ Define um limite de tempo (em milissegundos) para esperar por uma resposta da requisição antes de considerá-la como falha. Se a resposta não for recebida dentro desse limite de tempo, a requisição Ajax será cancelada e o callback de erro será chamado.

statusCode ⇒ Define manipuladores de status personalizados para diferentes códigos de status HTTP retornados pela resposta da requisição.

headers ⇒ É útil quando você precisa enviar cabeçalhos personalizados junto com a requisição Ajax. Isso é comum ao trabalhar com APIs que requerem autenticação ou para fornecer informações adicionais ao servidor.

Referências:

Texto:

<https://jquery.com>

Imagens geradas por IA:

<https://chatgpt.com>

<https://br.freepik.com>

