Tecnologias Web

Aula 11 – Requisições Assíncronas

Yuri Dirickson

yuri.dirickson@faculdadeimpacta.com.br

http://github.com/ImpactaTecWeb

Objetivos da Aula

- Entender o que é e para que serve o **AJAX**
- Como o JavaScript cuida do AJAX.
- Como fazer o Django entender e responder uma requisição AJAX.

AJAX - O que é?

- **AJAX** é um acrônimo para **A**synchronous **J**avaScript **A**nd **X**ML.

- AJAX é uma maneira de deixar o JavaScript fazer uma requisição HTTP para o servidor, no lugar do navegador.

- Requisições AJAX são assíncronas por natureza, ou seja, não bloqueiam a execução do navegador.

- Requisição AJAX não recarrega a página como o GET e o POST do navegador.

AJAX - O que é?

- Mesmo com o **X** no nome, não é necessário usar o XML para passar dados.

- A maneira mais comum de passar dados via AJAX é por JSON (JavaScript Object Notation - veremos adiante).

- **Não é tecnologia, é especificação:**
  - Dispara um objeto XMLHttpRequest (o mesmo que o navegador usa) para o servidor.
  - JavaScript recebe a resposta e já interpreta os dados.

AJAX - Como funciona?

- JavaScript prepara a requisição:

```
var xhttp = new XMLHttpRequest();
```

- Abre-se a requisição para o servidor com o método, URL e um condicional de sincronicidade:

```
xhttp.open(METODO,URL,ASSINCRONO)

xhttp.open("GET","buscar",true) // Um GET assíncrono para 'buscar'

xhttp.open("POST","buscar",false) // Um POST síncrono para 'buscar'
```

- Ao configurar tudo que for necessário, basta chamar a função **send** para enviar a requisição.

```
xhttp.send()
```

AJAX - Como funciona?

- Para requisições **GET**, os parâmetros vão na URL ao abrir a requisição:

```
xhttp.open("GET","buscar?nome=Yuri&idade=30",true)
```

- Para requisições **POST**, precisamos *serializar* as informações e colocar como parâmetro do **send**, informando como o servidor deve entender os parâmetros:

```
xhttp.open("POST","buscar",true)
xhttp.setRequestHeader("Content-type","application/x-www-form-urlencoded");
xhttp.send("nome=Yuri&idade=30")
```

AJAX - Async

- O terceiro parâmetro do **open** identifica se a chamada será assíncrona ou síncrona:
  - Chamadas síncronas: durante a execução da requisição, todo o restante do código JavaScript esperará a requisição retornar antes de continuar:

```
xhttp.open("GET","buscar?nome=Yuri&idade=30",false)
xhttp.send();
alert("Acabou!");
```

  - Chamadas assíncronas deixam o restante do JavaScript continuar a sua execução enquanto faz a requisição ao servidor. Como executar algo ao acabar então?

AJAX - Async

- Eventos! O **XMLHttpRequest** possui um evento que avisa quando a requisição muda de estado: o **onreadystatechange**
- Esse evento é disparado toda vez que a requisição muda de estado. O estado de uma requisição é dado pela propriedade **readyState** da requisição (xhttp.readyState)
  - 0: Requisição ainda não inicializou
  - 1: Conexão com o servidor estabelecida.
  - 2: Requisição recebida pelo servidor.
  - 3: Requisição sendo processada.
  - **4: Requisição terminada e resposta recebida**

Ajax - Async

- Exemplo de chamada assíncrona:

```
xhttp.open("GET","buscar?nome=Yuri&idade=30",false)
xhttp.onreadystatechange = function() {
        if (this.readyState == 4 && this.status == 200) {
                alert("Acabou!");
        }
};
xhttp.send();
```

- Para exibir um alerta quando acabar a requisição, precisamos esperar o **readyState = 4**.

Ajax - Propriedades

- O objeto XMLHttpRequest possui as seguintes propriedades:
  - **readyState:** Estado da requisição.
  - **status:** Código HTTP (200, 404, 500 etc.).
  - **statusText:** Texto do status (ok ou algum erro).
  - **responseText:** se não houve erro, a resposta em formato de texto (HTML,JSON).

AJAX - JSON

- Por mais que o AJAX tenha sido criado para utilizar o XML, hoje o formato mais utilizado na Web é o **JSON - JavaScript Object Notation**.

- O JSON é uma notação para objetos genéricos do JavaScript, feita com o objetivo de ser uma formatação leve para troca de dados entre servidores.

- O formato JSON é muito similar aos **dicionários** do Python.

- Fundamentalmente é composto por uma sequência de pares **chave-valor** separados por vírgula.

AJAX - Usando com JSON

- Para **enviar** um JSON via POST, podemos montar um objeto JSON com os dados e transformá-lo em String.
- Exemplo: imagine o seguinte formulário de login:

```
<form method=”POST” action=”logar” id=”form1”>
     Usuário: <input name=”usuario” type=”text” />
     Senha: <input name=”senha” type=”password” />
     <input type=”submit” value=”Enviar” />
</form>
```

- Ao invés de usarmos a submissão via formulário comum, vamos interceptar essa requisição e fazer via AJAX. A ideia é logar o usuário e avisar ele sem recarregar a página, mudando uma região com o nome do usuário.

AJAX - Enviando JSON

- Para interceptar a submissão do formulário, vamos usar o evento **onsubmit** e construir o nosso JSON com os dados do formulário:

```
var form = document.getElementById(“form1”)
form.onsubmit = function(event){
     event.preventDefault(); // Mata o evento padrão
     var dados = { } //Cria um JSON vazio, podemos usar o new Object()
     for (var i = 0; i < form.elements.length; i++){
          var input= form.elements[i];
          if(input.name){
               dados[input.name] = input.value;
          }
     }
    // Constrói a requisição
     var xhr = new XMLHttpRequest();
     xhr.onreadystatechange(... fazer algo ...); // Veremos adiante
     xhr.open(form.method, form.action, true); // Abre a requisição
     xhr.setRequestHeader(“Content-Type”, “application/json; charset=UTF-8”);
    // Envia configurando o JSON como uma String/serializando
     xhr.send(JSON.stringify(dados));
}
```

JSON - Enviando o JSON

- Teste a requisição e veja o console do navegador (na parte **network**), veja como os dados foram enviados.

- O utilitário **JSON** do JavaScript possui algumas funções para lidar com o formato JSON. Uma delas é o **stringify**, que converte um objeto JSON em uma String:

```
JSON.stringify({ “nome” : “Yuri”, “idade” : 30 })
// Isso vira
“{ \”nome\”:\”Yuri\”,\”idade\”:30 }”
```

AJAX - Recebendo JSON

- E como o servidor recebe o JSON?

- Cada linguagem de programação tem a sua forma de entender o JSON, usando outras bibliotecas para traduzi-lo. Lembrando que o JSON é um formato específico, mas seus dados são enviados como String.

- No Django, o mais comum é traduzirmos o JSON para um dicionário. Para fazer isso usamos o utilitário **json** do Django.

AJAX - Recebendo o JSON

- Veja a view **logar**:

```
import json
def logar(request):
        login = json.loads(request.body.decode("utf-8"))
        # faz a lógica de login, aqui apenas uma impressão
       print (login)
        responseData = { # Cria a resposta com os dados do usuário logado
                "nome": "Yuri",
                "email" : "yuri@email.com"
        }
        return HttpResponse(json.dumps(responseData),content_type="application/json")
```

- Primeiro usamos a função **json.loads** para traduzir uma String no formato JSON para um dicionário do Python.
- Fazemos a lógica qualquer de login e montamos uma resposta com os dados do usuário logado (print).
- Para converter novamente o dicionário para a String no formato JSON, usamos a função **json.dumps** (precisamos setar o *content_type* para **application/json**, pois ele sempre espera voltar HTML nessas requisições).

AJAX - Retornando o JSON

- Depois do processamento no Django, retornamos novamente um JSON para o JavaScript, como trabalhar com ele?
- Dentro da evento **onreadystatechange** da requisição, esse JSON estará na propriedade **responseText**. Quando ele retornar (**status 200** e **readyState 4**), podemos traduzi-lo para um objeto JSON e usar normalmente para atualizar a tela.
- Para esse exemplo, vamos atualizar uma div com id **usuario**.

AJAX - Retornando o JSON

- Retornando ao JavaScript anterior:

```
var form = document.getElementById(“form1”)
form.onsubmit = function(event){
     event.preventDefault(); // Mata o evento padrão
     // montando os dados e construindo a requisição
     xhr.onreadystatechange = function(){
          if (xhr.readyState == 4 && xhr.status === 200) {
          var dados = JSON.parse(xhr.reponseText);
          var div = document.getElementById(“usuario”);
               div.innerHTML = “Usuário: ”+dados.nome+” EMAIL: “+dados.email;
          }
     }
     // enviando os dados
}
```

- Quando a requisição terminar, a div terá o seguinte texto:

  “Usuário: Yuri EMAIL: yuri@email.com”

AJAX - Onde usar?

- Vimos a funcionalidade principal do AJAX - Chamadas assíncronas feitas pelo JavaScript.

- A pergunta que fica, quando devemos usá-lo.

- **Resposta: vai depender do seu sistema ou projeto.**

AJAX - Desvantagens

- Vimos a principal vantagem do AJAX: chamar o servidor sem travar a tela do usuário. Mas como nem tudo é alegria, aqui vão algumas desvantagens para se levar em conta:
  - Retrocompatibilidade: navegadores antigos (IE antes do 6) não possuem essa funcionalidade completa.
  - Limites do HTTP 1.1: se tudo for ajax no seu sistema, você pode esbarrar nos limites de requisição do HTTP (2 por domínio).
  - Assíncrono mas nem tanto: Nem todos os navegadores operam da mesma forma com chamadas assíncronas, em específico o Safari tem muitos bug's sem resolução ainda.
  - Muito código: Você não precisa escrever um código para receber um HTML no navegador, mas cada AJAX precisa ser re-escrito.

AJAX - Usos Comuns

- Frameworks para construir **SPA - Single Page Applications** - usam AJAX para tudo (Ex: Angular e React).

- Google Suggestion: campos que tentam autocompletar o que o usuário está digitando com textos que estão no banco (veja também o buscar dos ecommerces).

- Atualização de carrinho sem sair da página: alguns ecommerces você pode incluir um produto no carrinho e continuar sua navegação normalmente, mas a figura do carrinho é atualizada.

Faculdade
IMPACTA
TECNOLOGIA