AJAX

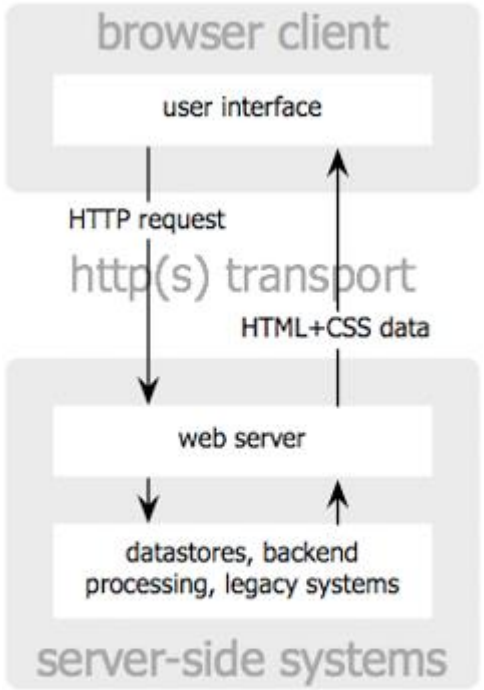
- AJAX → Asynchronous JavaScript and XML)

- Termo surgiu em 2005, por Jesse James Garret (https://web.archive.org/web/20150910072359/http://adaptivepath.org/ideas/ajax-new-approach-web-applications/)

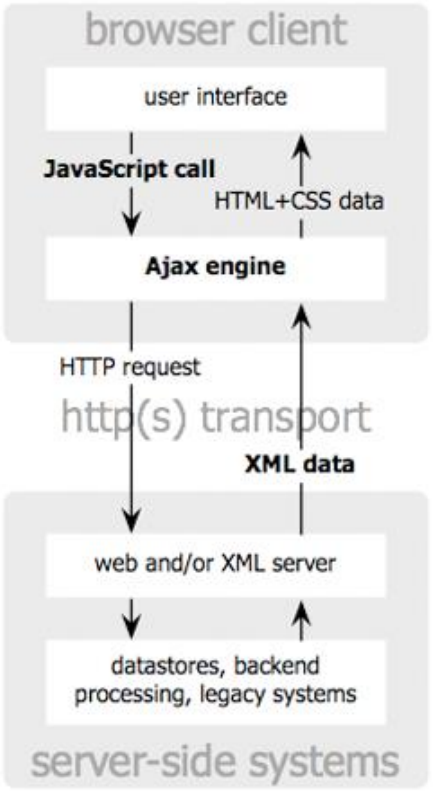
- Resulta da combinação de tecnologias existentes para enriquecer os mecanismos de construção e aplicações Web
  - HTML/XHTML e CSS
  - DOM
  - XML e XSLT
  - JavaScript
  - **XMLHttpRequest**

Modelos de execução Web

(por Jesse James Garret)



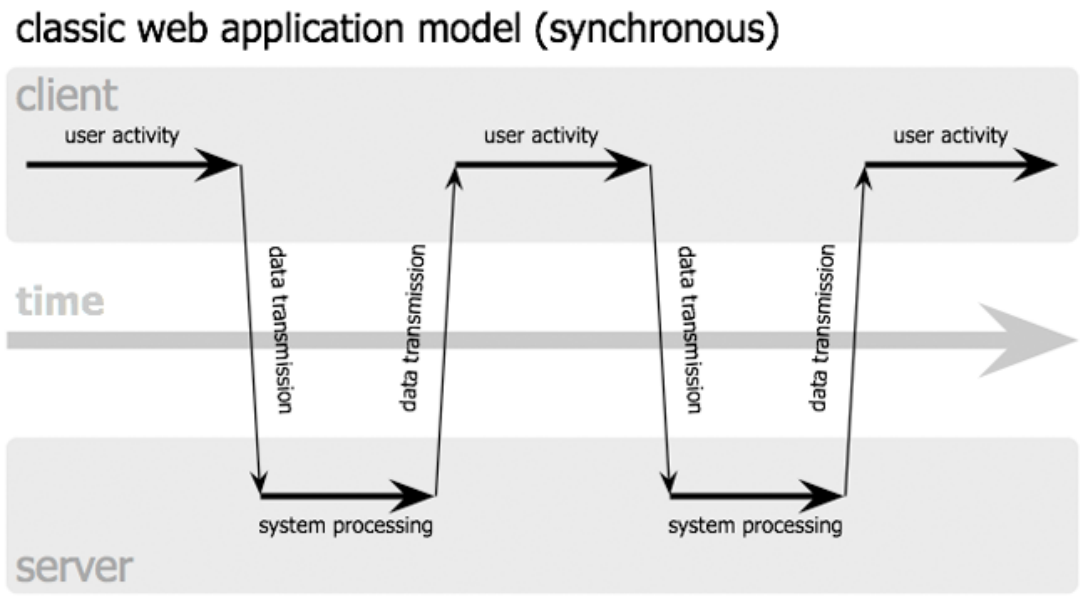
classic
web application model

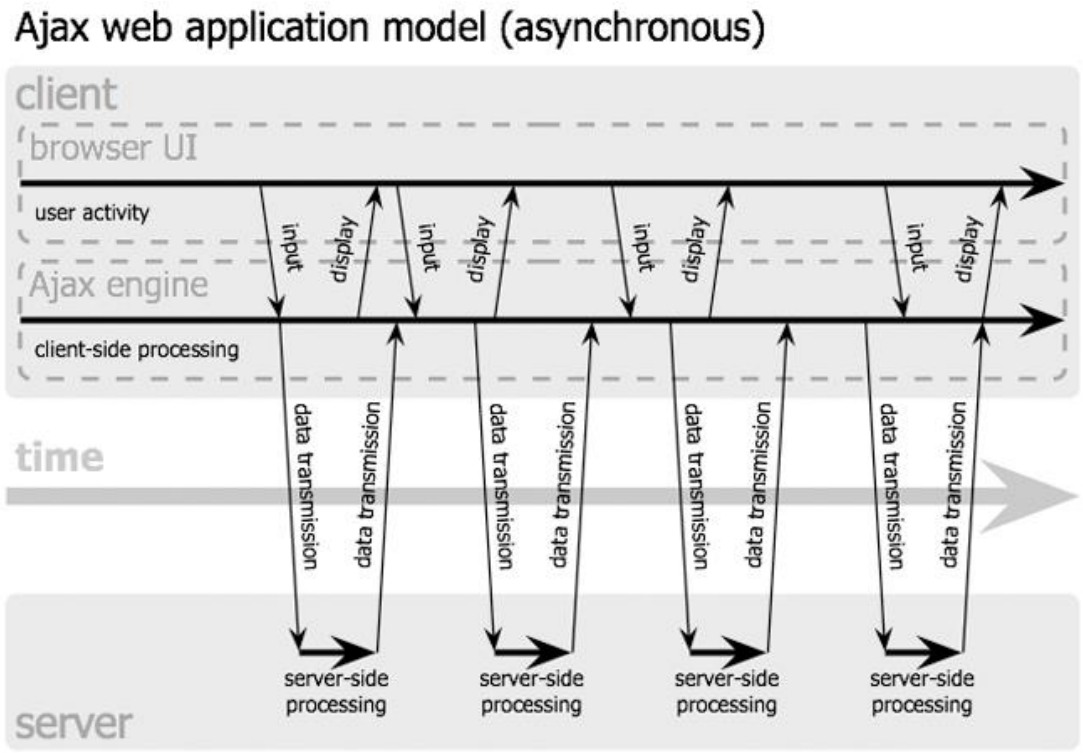


Ajax
web application model

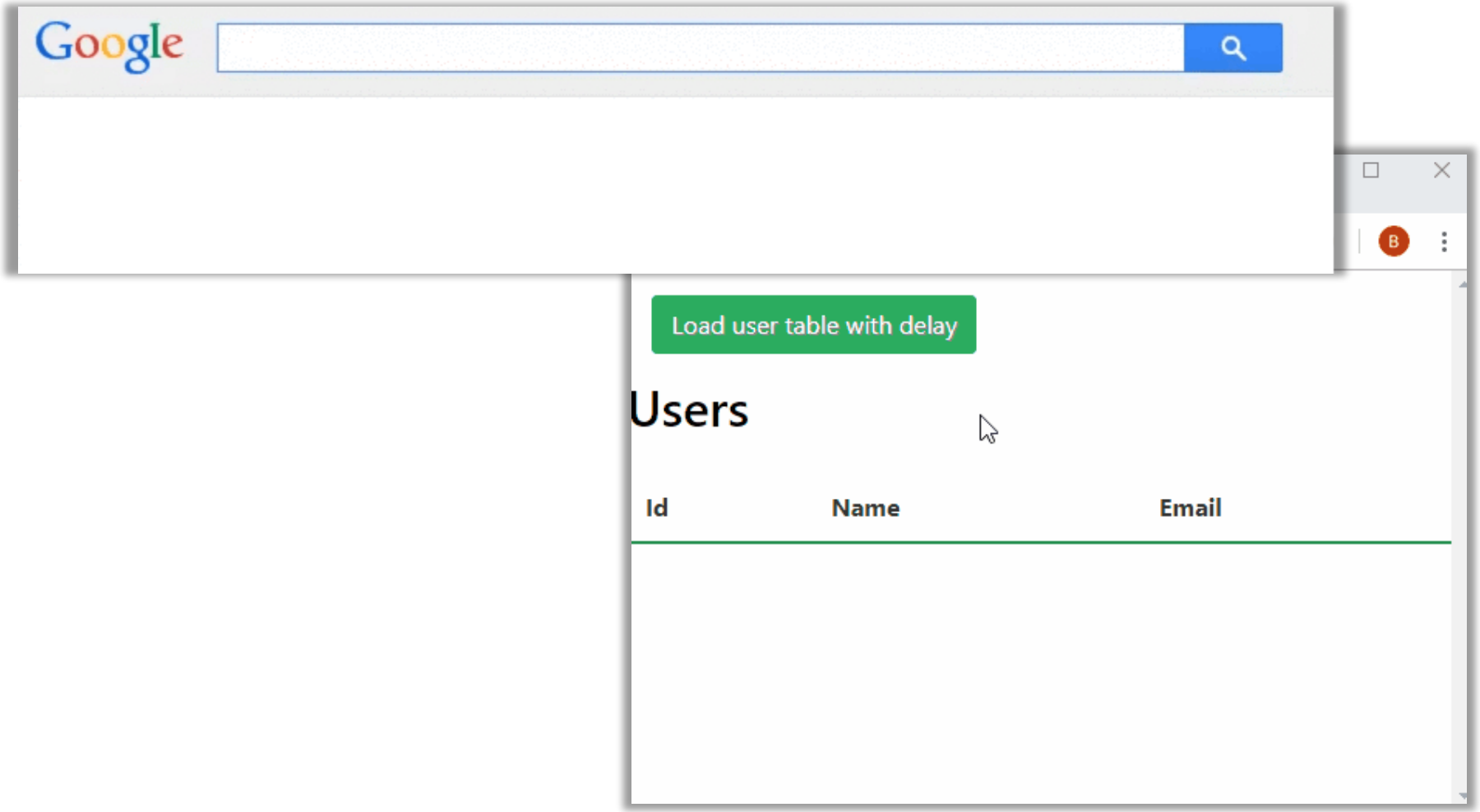
Modelos de execução Web

(por Jesse James Garret)





Exemplos de utilização

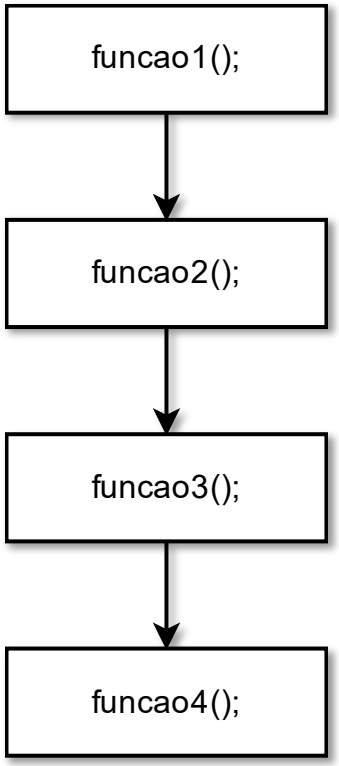


Execução AJAX

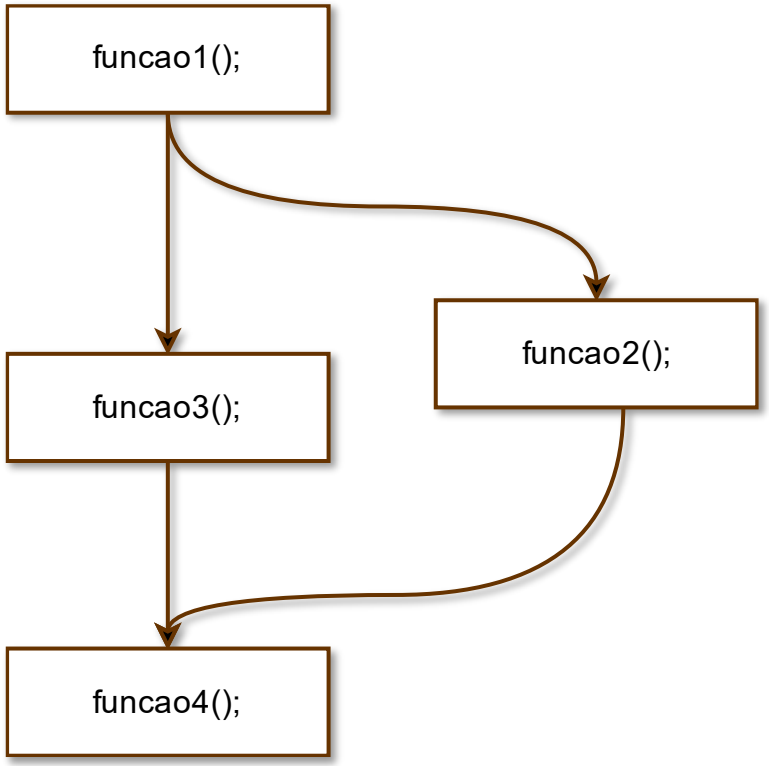
- Permite comunicação *assíncrona* entre cliente (browser) e servidor
- O cliente não fica bloqueado durante o pedido
- Suporta diferentes formatos de dados além do XML
  - O formato JSON (JavaScript Object Notation) é o dos mais comuns
    https://www.json.org/json-pt.html

Execução Síncrona vs Assíncrona

Síncrona



Assíncrona



Objecto XMLHttpRequest

- Enviar pedidos de dados a um servidor após a página ter sido carregada
- Receber resposta pedidos de dados após a página ter sido carregada
- Atualizar uma página web sem recarregá-la
- Enviar dados para um servidor em background
- Suportado em praticamente todos os browsers
  - https://caniuse.com/?search=XMLHttpRequest

Objecto XMLHttpRequest - Métodos

- `open(metodo, URL, isAsync);`
  - Iniciar um novo pedido ou reiniciar um novo existente

- `send(payload);`
  - Processa o envio de *`payload`* para o servidor indicado em `open()`

- `abort()`
  - Cancelar o pedido em execução

Objecto XMLHttpRequest - Métodos

- `getResponseHeader("header")`
  - Retorna uma string com o valor campo de cabeçalho HTTP
  - Exemplo: `“content-type”`

- `getAllResponseHeaders()`
  - Retorna uma string com todos os cabeçalhos HTTP da resposta

https://developer.mozilla.org/pt-BR/docs/Web/API/XMLHttpRequest#M%C3%A9todos

Objecto XMLHttpRequest - Propriedades

- `status`
  - Código de resposta HTTP enviado pelo servidor
  - **Exemplo**: 200 OK, 404 Not Found, etc.
  - https://developer.mozilla.org/pt-PT/docs/Web/HTTP/Status

- `statusText`
  - Texto do estado

- `readyState`
  - Estado do pedido → 0=**`UNSENT`**; 1=**`OPENED`**; 2=**`HEADERS_RECEIVED`**; 3=**`LOADING`**; 4= **`DONE`**

Objecto XMLHttpRequest - Propriedades

- `responseText`
  - texto da resposta sem qualquer tipo de tradução/tratamento

- `responseXML`
  - resposta é tratada e carregada numa estrutura DOM se **“`content-type`”** for **“`text/html`”**

- `onreadystatechange`
  - função de *event handler* que é invocada sempre que o valor de `readyState` altera

https://developer.mozilla.org/pt-BR/docs/Web/API/XMLHttpRequest#Propriedades

AJAX em 4 passos

1. Criar uma instância do objecto XML HTTP Request

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

2. Especificar de onde e como se quer obter o recurso

```
xmlhttp.open('GET', 'foo.jsp', true);
```

3. Especificar a função que vai tratar a resposta

```
xmlhttp.onreadystatechange = function() {
  // Código para tratar a resposta
}
```

4. Executar o pedido

```
xmlhttp.send([payload] | [null]);
```

AJAX em 4 passos
1. Criar um instância do objecto XML HTTP Request

- Firefox, Opera, Safari, IE 7+, Chrome, Edge

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

Para versões antigas do IE (< 7)

```
var xmlhttp = new ActiveXObject(MSXML_ProgID);
```

MSXML_ProgID = string que identifica a DLL Microsoft XML Core Services (MSXML). Exemplo "Msxml2.XMLHTTP.4.0", "Msxml2.XMLHTTP"

AJAX em 4 passos
2. Especificar de onde e como se quer obter o recurso

```
xmlhttp.open('GET', 'foo.jsp', true);
```

- `open(metodo, URL, isAsync);`
  - *`metodo`* : método HTTP, por ex. *'GET'*, *'POST'*
  - *`URL`* : endereço do recurso a obter
  - *`isAsync`*
    - *`true`* - execução assíncrona
    - *`false`* ou omitido - execução síncrona

Nota: este método não executa nenhum pedido

AJAX em 4 passos

3. Especificar a função que vai tratar a resposta

```
xmlhttp.onreadystatechange = function() {
  // Código para tratar a resposta
}
```

- Cada instância do objecto `XMLHttpRequest` tem uma propriedade `readyState` que mantém o estado da resposta do servidor.
- O event handler `onreadystatechange` é invocado sempre que o valor de `readyState` muda.

AJAX em 4 passos
3. Especificar a função que vai tratar a resposta

- Valores possíveis de `readyState`

Valor	Estado	Descrição
0	UNSENT	O XMLHttpRequest foi criado. Mas o método open() não foi chamado ainda.
1	OPENED	O método open() foi invocado. Durante esse estado, os headers do pedido podem ser inseridos usando o método setRequestHeader() e o método send() pode ser chamado, iniciando o pedido.
2	HEADERS_RECEIVED	O método send() foi chamado e os cabeçalhos de respostas foram recebidos.
3	LOADING	A resposta do pedido está a ser recebida. Se o responseType for "text" ou um texto em branco, responseText terá o texto parcial da resposta durante o carregamento.
4	DONE	A Operação está completa. Isso pode significar que a transferência foi concluída com êxito ou que falhou.

AJAX em 4 passos
4. Executar o pedido

```
xmlhttp.send([payload] | [null]);
```

- Executa a ligação ao *URL* especificado em `open()`
- Se o pedido for assíncrono, passa à linha seguinte do script
- Se o pedido for síncrono, bloqueia a execução até que a totalidade do resultado do pedido tenha sido recebido
- Em pedidos do tipo *'POST'* a string com conteúdo a enviar (`payload`) é enviado no corpo do pedido
- Se não for para submeter conteúdo ou o tipo de pedido for *'GET'* deve ser enviado `null`

AJAX em 4 passos
4. Executar o pedido

- Em pedidos do tipo *‘POST’*, é necessário indicar o tipo do conteúdo que vai ser enviado para que o saber como o processar
- Essa informação é adicionada ao pedido através do campo do cabeçalho ”Content-type“
  - Por exemplo, para emular o envio de um formulário:

```
xmlhttp.setRequestHeader(‘content-type’,
                         'application/x-www-form-urlencoded;charset=UTF-8;');
xmlhttp.send('param1=value1&param2=value2’);
```

Outros campos de cabeçalho podem ser adicionados ou alterados através do uso de `setRequestHeader()`

Tipos de pedidos HTTP - HTTP verbs

- *GET*
  - Consultar/recolher recursos
  - Para comparação, em base de dados seria a cláusula SQL SELECT

- *POST*
  - Criar novos recursos
  - Para comparação, em base de dados seria a cláusula SQL INSERT

Tipos de pedidos HTTP - HTTP verbs

- *PUT*
  - Atualizar recursos existentes
  - Para comparação, em base de dados seria a cláusula SQL UPDATE

- *DELETE*
  - Apagar recursos
  - Para comparação, em base de dados seria a cláusula SQL DELETE

https://restfulapi.net/http-methods/

Tipos MIME

- Os tipos MIME determinam o formato de troca de informação entre cliente e servidor
- O cabeçalho *“content-type”* deve ser descrito utilizando as *strings* de formato. Por exemplo:
  - *“text/json”* → formato JSON
  - *“text/xml”* → formato XML
  - *“multipart/form-data”* → conteúdo de um formulário web

https://developer.mozilla.org/pt-BR/docs/Web/HTTP/Basico_sobre_HTTP/MIME_types

Recursos

Living Standard - https://xhr.spec.whatwg.org/

https://www.w3schools.com/js/js_ajax_intro.asp

https://developer.mozilla.org/pt-BR/docs/Web/Guide/AJAX

https://javascript.info/xmlhttprequest

https://caniuse.com/?search=XMLHttpRequest