PÓS WebMob UNOESC

INTRODUÇÃO AO AJAX E JSON

Prof. Roberson Alves

SUMÁRIO

- ✓O que é o AJAX?
- ✓Arquitetura AJAX
- ✓Objeto XMLHttpRequest
- ✓Exemplos
- ✓JSON

O QUE É O AJAX?

- **AJAX é o acrônimo de *Asynchronous javascript and XML***
- **Podemos olhar para esta técnica de duas formas distintas:**
  - **Como sendo um conjunto de tecnologias e padrões**
  - **Como sendo uma arquitetura**

CASOS DE SUCESSO

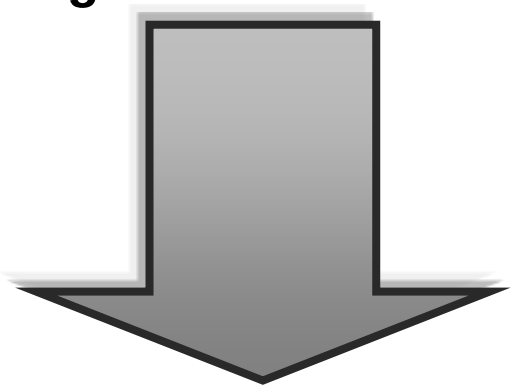
- GMail – http://mail.google.com
- Google Suggest - http://www.google.com/webhp?complete=1
- Start.com portal
- Google Maps - http://maps.google.com/
- MSN Virtual Earth - http://maps.live.com/
- Flickr Photo Sharing website – http://www.flickr.com
- Facebook – http://www.facebook.com

AJAX - AS TECNOLOGIAS

- **O AJAX não é por si só uma tecnologia, mas sim um conjunto de tecnologias combinadas(padrões):**
  - **Utiliza HTML e CSS para a apresentação de conteúdo**
  - **Utiliza o DOM para oferecer páginas interativas e dinâmicas**
  - **Utiliza XML (entre outros) para troca de dados**
  - **As trocas assíncronas de dados são efetuadas utilizando o objeto XMLHttpRequest**
  - **Utiliza o JavaScript como linguagem, que "combina" todas estas tecnologias**

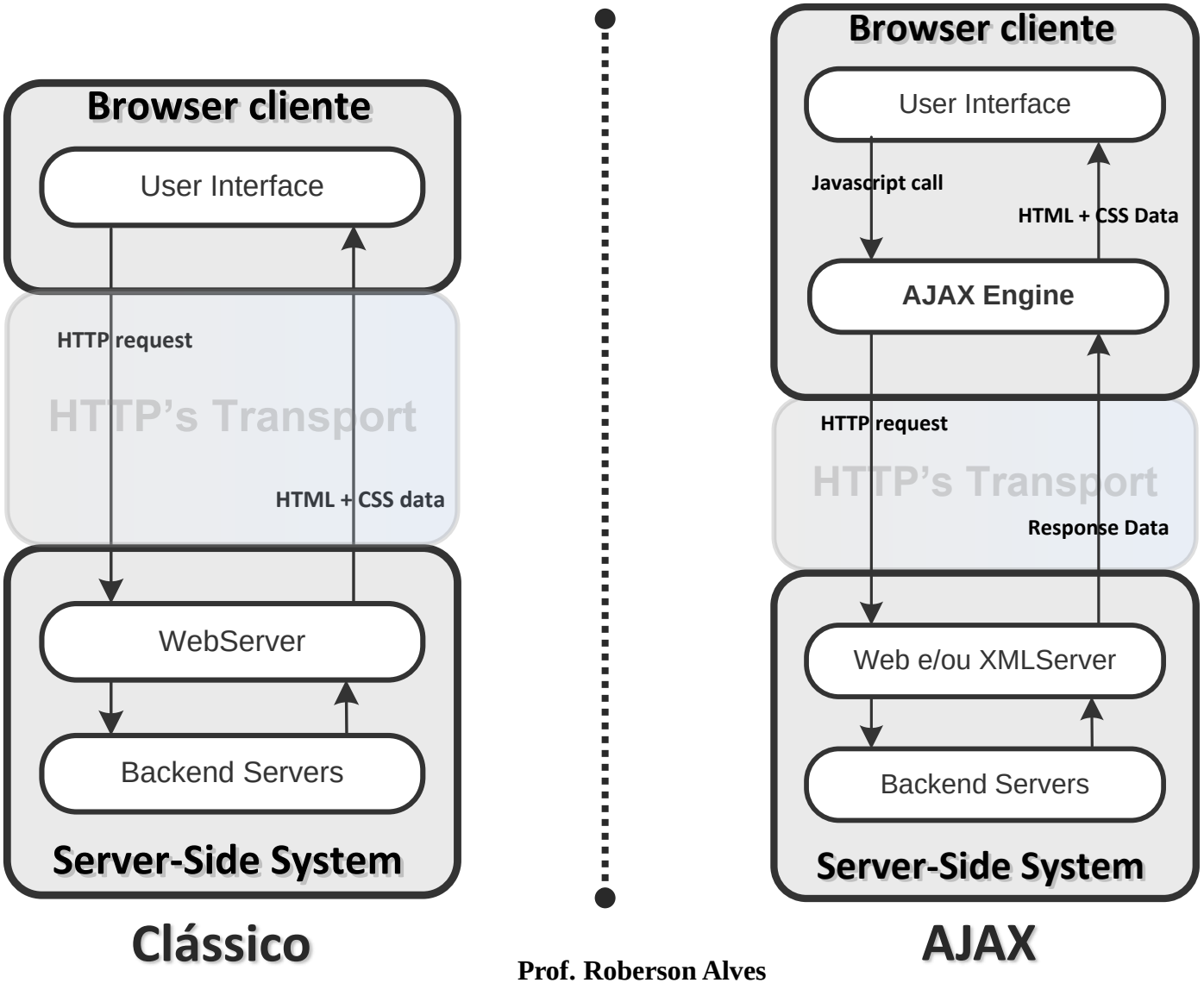
AJAX – A MUDANÇA DE ARQUITETURA

- **O AJAX permitiu atualizar a arquitetura base das aplicações WEB**
  - ***Server Side Events*: Os componentes podem fazer pedidos ao servidor para obter informações, sem forçar o carregamento total da página**
  - ***Assincronismo*: Os pedidos (parciais) ao servidor não bloqueiam a interação com o navegador.**

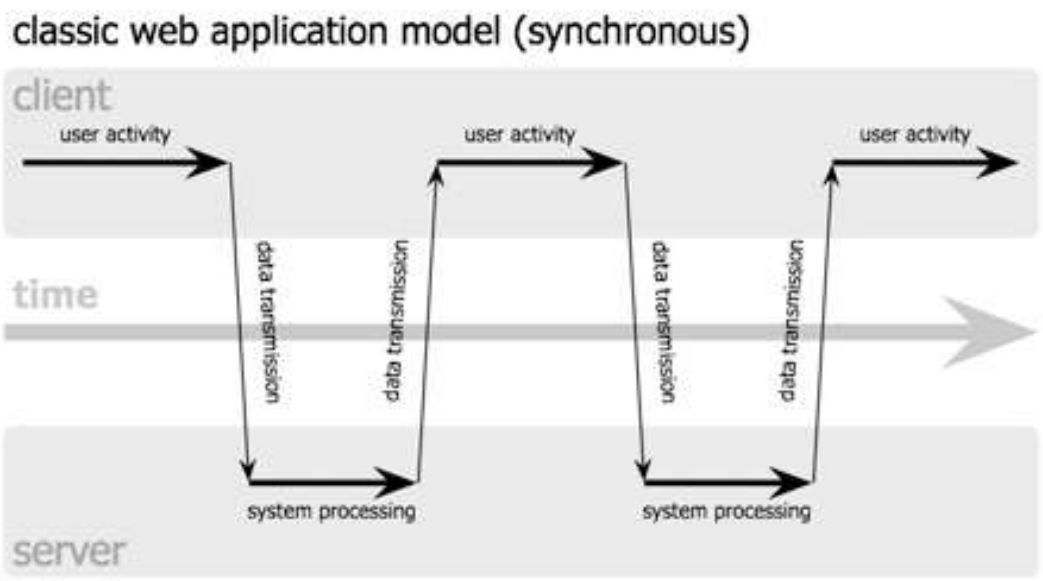


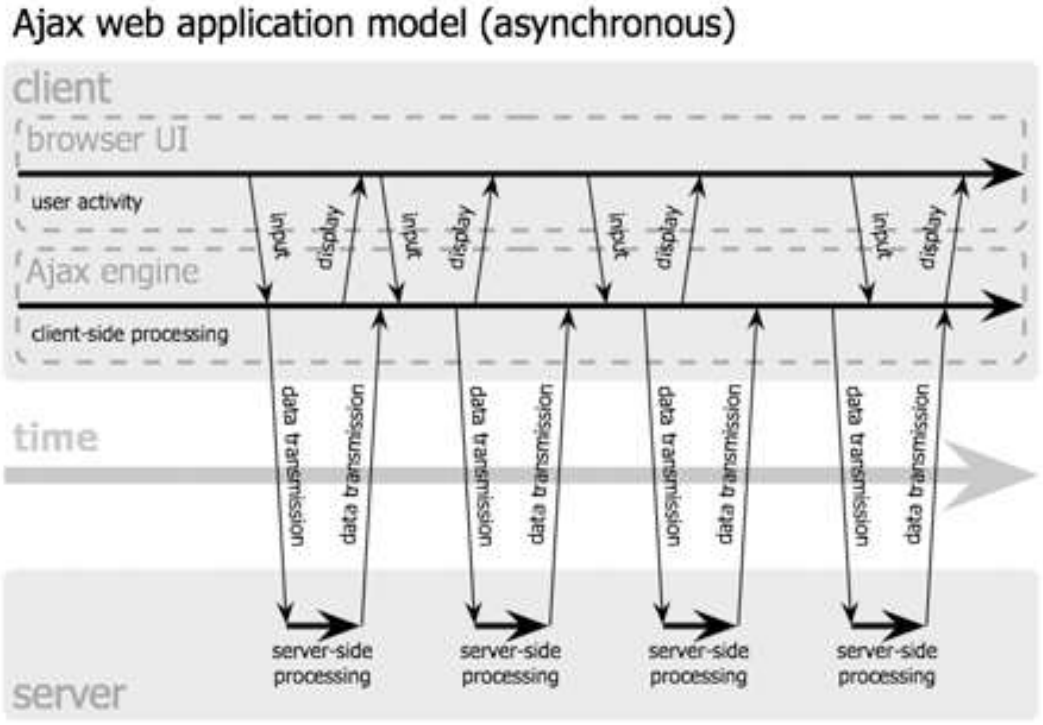
Maior aproximação entre aplicações WEB e Desktop

MODELO CLÁSSICO vs AJAX(IANO)

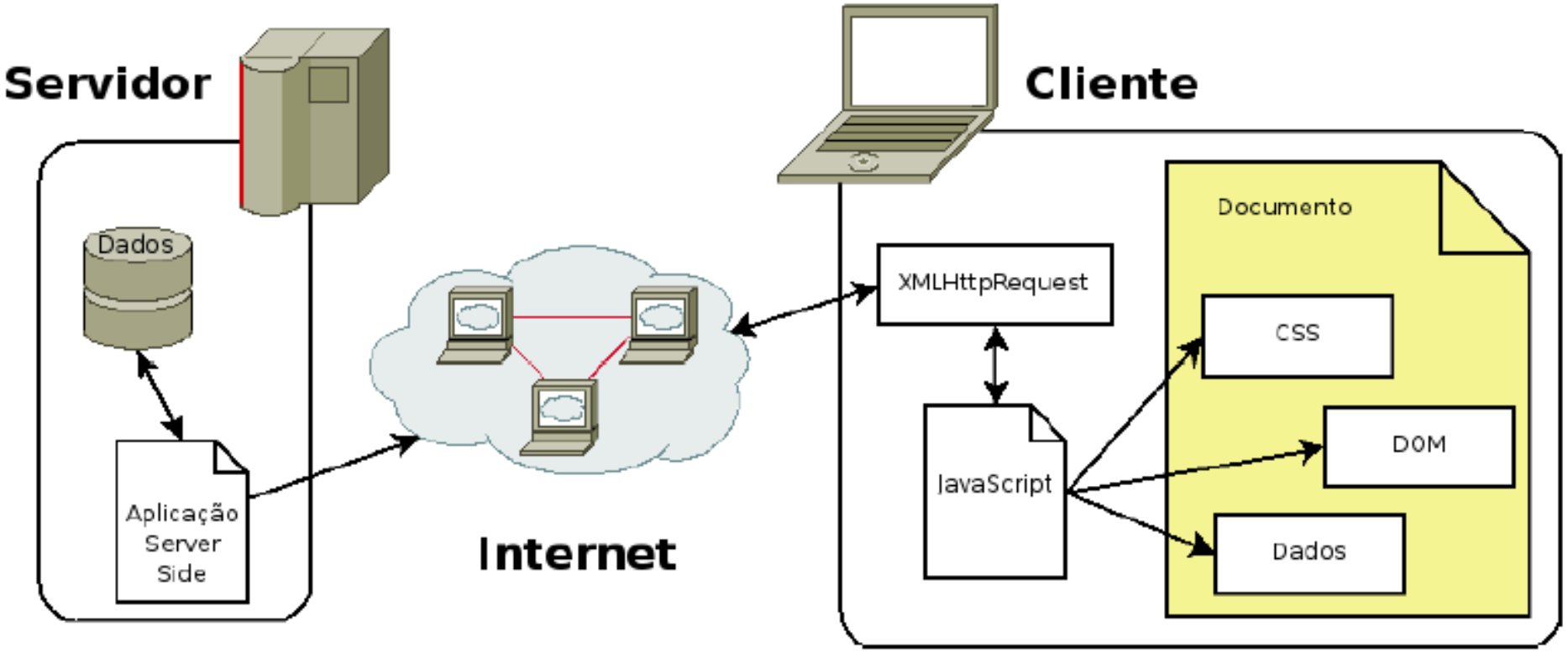


MODELO CLÁSSICO vs AJAX(IANO)





ARQUITETURA AJAX



VANTAGENS

- **Aumento da usabilidade da interface das aplicações WEB;**
- **Possível ter aplicações mais ricas, com mais interações, sem recorrer a *plugins* de terceiros (e.g. Macromedia Flash);**
- **Aplicações requerem menos largura de banda – apenas é descarregado(baixado) o necessário;**
- **Interfaces respondem mais rapidamente (embora estejam mais dependentes da velocidade da rede).**

DESVANTAGENS

- O suporte das diferentes tecnologias utilizadas é dependente do navegador;
- Botão de *back* passa a não funcionar como esperado;
- O URI não se altera com a alteração do estado da aplicação;
- O uso do Javascript aumenta o processamento no cliente;
- Os pedidos apenas podem ser endereçados ao domínio de onde foi originado o pedido inicial (mas é mais seguro !);
- A depuração do código é mais difícil.

XMLHTTPREQUEST

- **É um objeto Javascript, responsável por:**
  - **Enviar pedidos HTTP**
  - **Receber as respostas a esses pedidos**
  - **Efetuar a análise da resposta**
- **Infelizmente, a sua instanciação é dependente do navegador:**

De acordo com a recomendação W3C (Firefox, Opera, …)

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

Internet Explorer (6+)

```
var xhr = new ActiveXObject("Msxml2.XMLHTTP");
```

Internet Explorer (5)

```
var xhr = new ActiveXObject("Microsoft.XMLHTTP");
```

XMLHTTPREQUEST

```
interface XMLHttpRequest {
            attribute EventListener   onreadystatechange;
  readonly attribute unsigned short  readyState;
  readonly attribute DOMString responseText;
  readonly attribute Document responseXML;
  readonly attribute unsigned short  status;
  readonly attribute DOMString statusText;

  void open(in DOMString method, in DOMString url);
  void open(in DOMString method, in DOMString url, in boolean async);
  void open(in DOMString method, in DOMString url, in boolean async, in DOMString
user);
  void open(in DOMString method, in DOMString url, in boolean async,
               in DOMString user, in DOMString password);
  void setRequestHeader(in DOMString header, in DOMString value);
  void send();
  void send(in DOMString data);
  void send(in Document data);
  void abort();
  DOMString getAllResponseHeaders();
  DOMString getResponseHeader(in DOMString header);
};
```

Mais informações: https://developer.mozilla.org/pt-BR/docs/Web/API/XMLHttpRequest

HELLOWORLD

Do lado do cliente

```
…
<span id="span">Try to hit me!</span>
…
```


```
…
span.addEventListener("click",mouseClick,false);
…
function mouseClick(evt){
 xhr.onreadystatechange=processData;
 xhr.open("GET","cgi/HelloAjax.exe");
 xhr.send(null);
}
```

Indicação do *handler* para processar a resposta

Indicação do pedido a efetuar ao servidor

Só aqui o pedido é enviado para o o servidor !

A resposta só está totalmente disponível quando o estado é 4.

Processa-se o conteúdo da resposta

```
function processData()
 {
   if (xhr.status  == 200 &&
       xhr.readyState == 4 /*COMPLETE*/)
    {
     alert(xhr.responseText);
    }
 }
```

HELLOWORLD(CONT.)

Do lado do servidor

```
void main()
{

  cout << "HTTP/1.1 200 OK" << endl;
  cout << "Content-Type: text/plain" << endl;
  cout<<endl;
  cout<<"Hello ajax world!!!";
}
```

Neste caso é indicado que o tipo *MIME* é texto

- Note que a resposta a um pedido pode ser de qualquer tipo;
- No entanto, existem alguns formatos que são mais utilizados, nomeadamente, XML (que lhe deu o nome) e JSON (abordado mais adiante).

XMLHTTPREQUEST – ATRIBUTO READYSTATE

Esta propriedade tem 5 valores possíveis:

- ***0*: (*Unset*) - O método `send()` ainda não foi invocado**
- ***1*: (*Opened*) - o método `send()` foi invocado, encontrando-se o pedido a ser processado**
- ***2*: (*Headers Received*) - o método `send()` foi completado e a resposta recebida**
- ***3*: (*Loading*) - A resposta está sendo processada**
- ***4*: (Complete) - A resposta foi processada e pode ser consultada**

EXEMPLO PRÁTICO

- **Exemplo de chamada Ajax: AloMundo**

```
1  // Firefox, Opera, Chrome, etc...
2  var xhr = new XMLHttpRequest();
3
4
5  function getAloMundo() {
6      xhr.onreadystatechange = processaDados;
7      xhr.open("GET", "/alomundo.html");
8      xhr.send();
9  }
10
11 function processaDados() {
12     if (xhr.status == 200 && xhr.readyState == 4)
13         alert(xhr.responseText);
14 }
```

EXEMPLO PRÁTICO

- **Exemplo de chamada Ajax: AloMundo**

```
1  // Firefox, Opera, Chrome, etc...
2  var xhr = new XMLHttpRequest();
3
4
5  function getAloMundo() {
6      xhr.onreadystatechange = processaDados;
7      xhr.open("GET", "/alomundo.txt");
8      xhr.send();
9  }
10
11 function processaDados() {
12     if (xhr.status == 200 && xhr.readyState == 4)
13         alert(xhr.responseText);
14 }
```

XMLHTTPREQUEST – RESPONSETEXT VS RESPONSEXML

- Ambos os atributos (`responseText` e `responseXML`) são utilizados para obter o resultado do pedido AJAX, embora em situações diferentes
- O atributo `responseText` tem o conteúdo do corpo da resposta, sem nenhum tratamento adicional
  - Funciona com qualquer tipo *MIME*
- O atributo `responseXML` obriga que o tipo *MIME* da resposta seja terminado em XML (text/xml, application/xml, etc.)
  - É feito *parsing* sobre a resposta
  - O resultado é um objeto que implementa a interface `Document`
  - Caso contrário o valor é `null`

HELLOXML

- Note que para obter a resposta a um pedido AJAX utilizando a propriedade `responseXML`, a resposta tem de ser um documento XML bem formado


```
function processData()
{
   if (xhr.status  == 200 &&
       xhr.readyState == 4 /*COMPLETE*/ && xhr.responseXML)
   {
      var div = document.getElementById("div1");
      var root = xhr.responseXML;

       for(var i=0; i < root.childNodes.length; ++i)

        div.appendChild(root.childNodes[i]);

      div.appendChild(document.createElement("p"));
   }
 }
```

```
<span>Data</span>
```

XMLHTTPREQUEST – SÍNCRONO VS ASSÍNCRONO

- O objeto `XMLHttpRequest` também suporta pedidos síncronos!

```
void open(in DOMString method, in DOMString url, in boolean async);
```

A invocação do método `send()` não provoca bloqueio. Embora na recomendação sugira-se a omissão do valor.

`true`

`false`

O método `send()` é bloqueante, não retornando enquanto não for recebida a resposta do servidor.

- Quando não existe bloqueio da interface, é possível serem enviados vários pedidos AJAX;
- É possível abortar um pedido feito, invocando o método `abort()` do objeto `XMLHttpRequest`.

XMLHTTPREQUEST – ENVIO DE DADOS

- Com o objeto XMLHttpRequest é possível enviar dados no corpo do pedido HTTP
  - Invocando método send, passando-lhe como parâmetro, os dados
- É necessário definir o `content-type` do parâmetro, através do método `setRequestHeader`

```
…
var xhr = new XMLHttpRequest ();
var msg="Mensagem Secreta!");
xhr.open("POST","log.ashx",true);

//MPORTANTE: definir o tipo de conteúdo a enviar
xhr.setRequestHeader("Content-Type", "text/plain;charset=UTF-8");
xhr.send(msg);
```

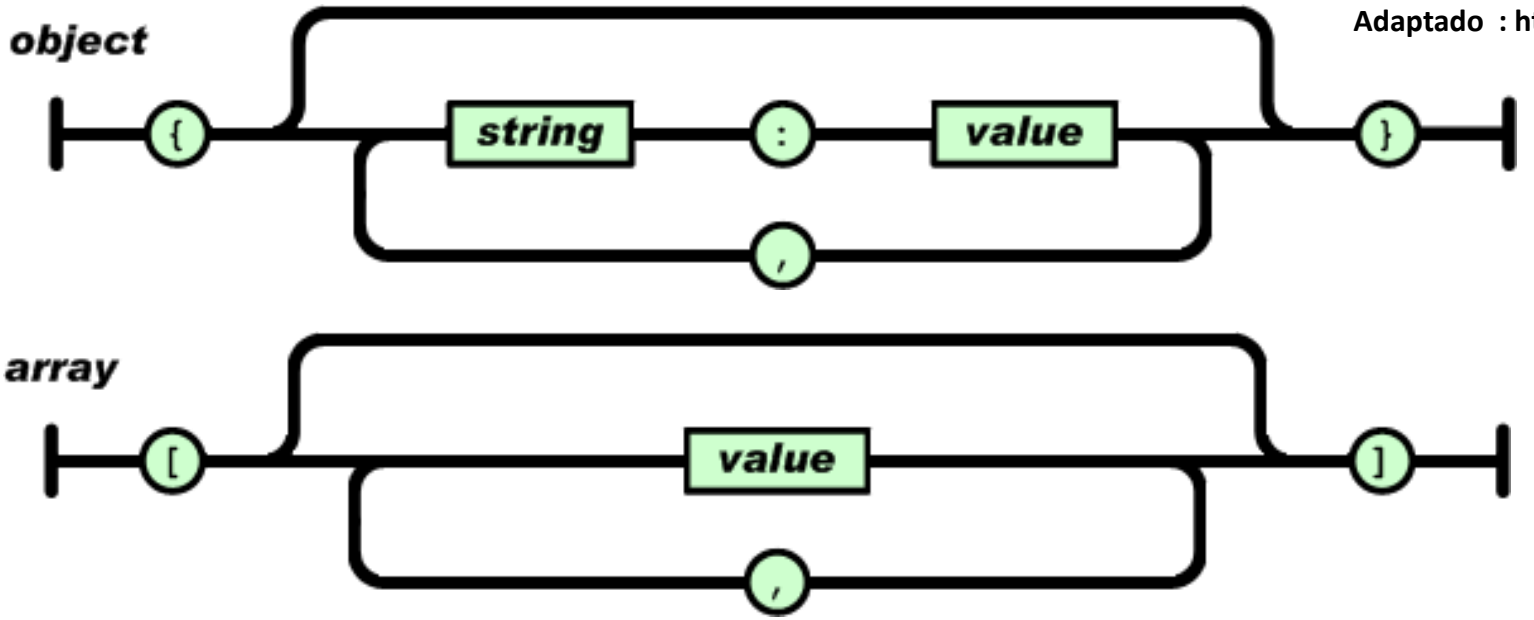
Log.ashx

```
…
public void ProcessRequest (HttpContext context){
        System.IO.TextReader rd = new
System.IO.StreamReader(context.Request.InputStream);
        String s = rd.ReadLine(); //s =  ="Mensagem Secreta!"
…
```

JSON (JAVASCRIPT OBJECT NOTATION)

- **É um formato baseado na notação literal para objetos do Javascript**
  - **Standard ECMA-262 3rd Edition - December 1999**
- **JSON é um formato de texto independente da linguagem**
- **É utilizado para troca de dados**
- **JSON contém duas estruturas base:**
  - **Objeto: Uma coleção de pares nome/valor (tabela de hash)**
  - **Array: Uma lista ordenada de valores**

JSON



Adaptado : http://json.org

JSON Example

```
{"employees":[
    {"firstName":"John", "lastName":"Doe"},
    {"firstName":"Anna", "lastName":"Smith"},
    {"firstName":"Peter", "lastName":"Jones"}
]}
```

JSON EM JAVASCRIPT

- Como o JSON se inspirou na notação literal para objetos do ECMAScript, tudo o que é necessário fazer é avaliar a string recebida pelo objeto XMLHttpRequest:

```
var jsonObj = eval( "(" + xhr.responseText + ")" );
```

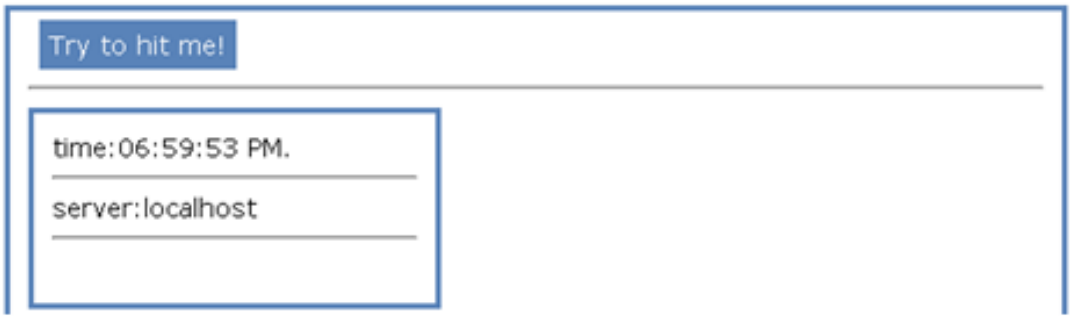
- Por questões de segurança, em vez de se utilizar a função `eval`, deve-se utilizar um parser JSON, o qual só aceitará texto nesse formato

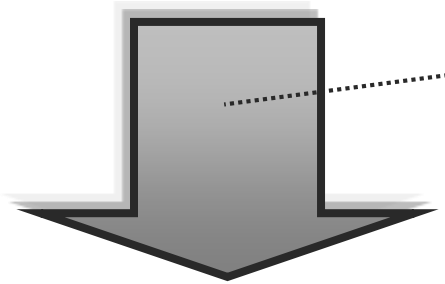
- Para mais informações: `http://www.json.org/`

HELLOJSON

Do lado do servidor

```
{ "time": "06:59:53 PM."
"server":"localhost" }
```





É boa prática que o tipo MIME enviado seja **application/json**

Do lado do cliente

```
function processData()
  {
    if (xhr.status  == 200 && xhr.readyState == 4 /*COMPLETE*/ )
    {
      var jSonObj = eval("(" + xhr.responseText + ")" );
       ...
      for(var p in jSonObj) { /* processar as propriedades */ }
    }
  }
```

EXEMPLO PRÁTICO

- **Exemplo de chamada Ajax: wikipediaSearch(JSON)**

```
1  // Firefox, Opera, Chrome, etc...
2  var xhr = new XMLHttpRequest();
3
4
5  function wikipediaSearch() {
6      xhr.onreadystatechange = processaDados;
7      xhr.open("GET",
   "http://api.geonames.org/wikipediaSearchJSON?
   q=matematica&maxRows=10&username=demo");
8      xhr.send();
9  }
10
11 function processaDados() {
12     if (xhr.status == 200 && xhr.readyState == 4) {
13         document.getElementById("txtResultado").innerHTML
   = xhr.responseText;
14     }
```

EXEMPLO PRÁTICO

- **Exemplo de chamada Ajax: wikipediaSearch(XML)**

```
1  // Firefox, Opera, Chrome, etc...
2  var xhr = new XMLHttpRequest();
3
4
5  function wikipediaSearch() {
6      xhr.onreadystatechange = processaDados;
7      xhr.open("GET",
   "http://api.geonames.org/wikipediaSearch?
   q=matematica&maxRows=10&username=demo");
8      xhr.send();
9  }
10
11 function processaDados() {
12     if (xhr.status == 200 && xhr.readyState == 4) {
13         document.getElementById("txtResultado").innerHTML
   = xhr.responseText;
14     }
```