Асинхронни заявки (AJAX)

Жоро Пенчев
Съвременно уеб-програмиране с PHP 2013

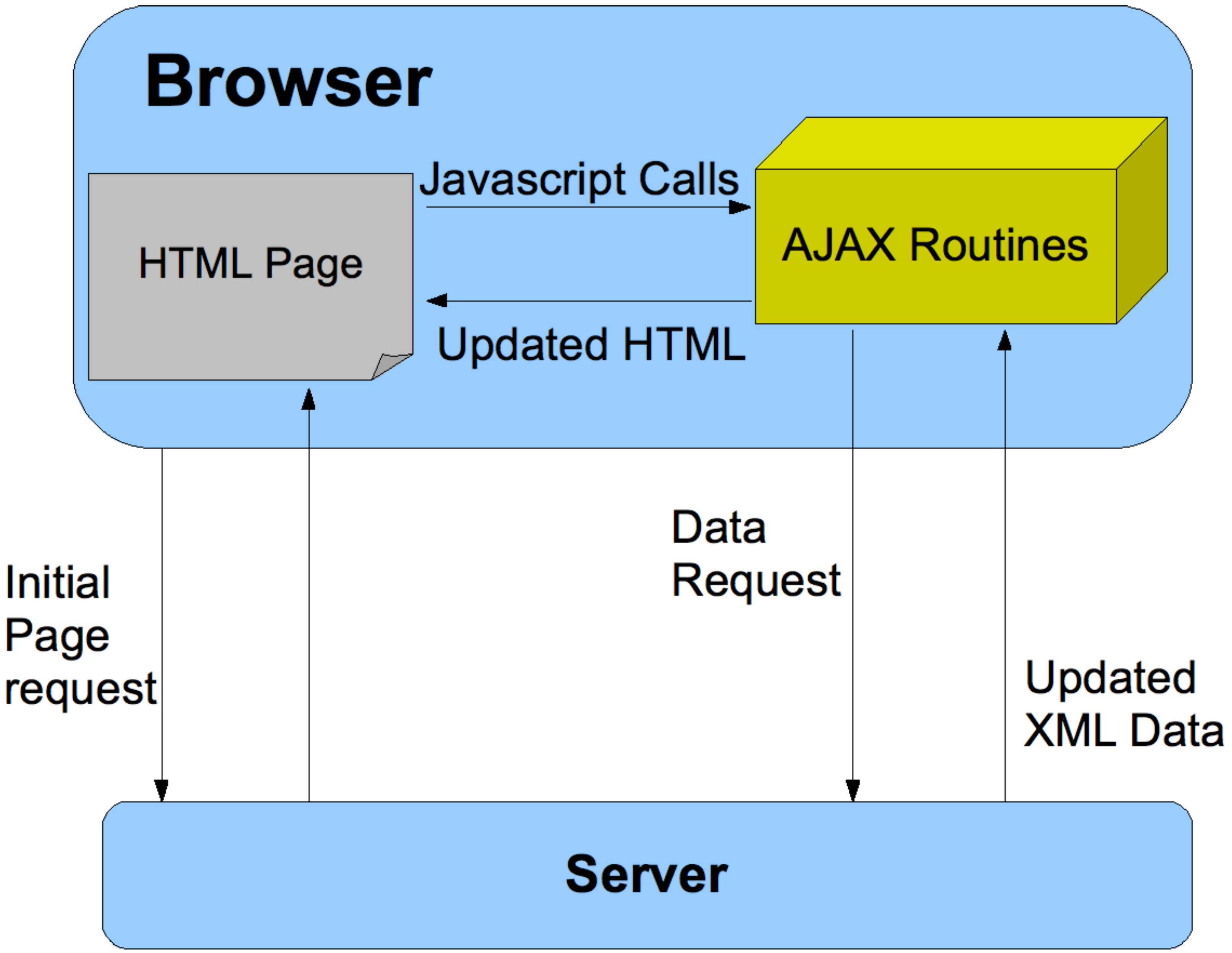
Какво е AJAX?

- Asynchronous
- JavaScript
- and XML (not)

Защо ни е AJAX?

- Основно защото сме 2013
- Но сериозно:
  - по-интуитивно за потребителя
  - богатство на features
  - AJAX е отговорен за популярността на web
  - уеднаквен интерфейс за данни със сървъра
  - по-малко трансфер
- Проблеми
  - по-сложно за писане
  - проблеми с браузъри (jQuery to the resque!)
  - BACK BUTTON

Как работи?


Browser
HTML Page
Javascript Calls
AJAX Routines
Updated HTML
Initial Page request
Data Request
Updated XML Data
Server

На практика

```
var request = false;
try {
	request = new XMLHttpRequest();
} catch (ms) {
	try {
		request = new ActiveXObject("Msxml2.XMLHTTP");
	} catch (ms2) {
		request = new ActiveXObject("Microsoft.XMLHTTP");
	}
}
```

Пращане на заявка

- GET

```
function myAjaxFunction() {
    request.open("GET", "ajax.php", true);
    request.onreadystatechange = updatePage;
    request.send(null);
}
```

- POST

```
function myAjaxFunction() {
    request.open("POST", "resource.php", true);
    var data = "book_name=" + book;
    request.onreadystatechange = updatePage;
    request.send(data);
}
```

Получаване на отговор

```
function updatePage() {
      if (request.readyState == 4 && request.status == 200) {
      var answer = request.responseText;
            //do something useful with the response
      }
}
```

- States
- HTTP Codes
- Какво да правим като получим отговор

```
document.getElementById("to-update").value = answer;
document.getElementById("to-update").innerHTML = answer;
```

REST + JSON

- Какво е JSON?
  - по-добрият XML
  - може да се **eval**-не до javascript обект
- Алтернатива: връщаме HTML
- Какво е REST?
  - "всичко е ресурс"
  - просто и универсално
  - невинаги е идеалното решение
  - HTTP методите са заявките към сървъра

Стандартният REST интерфейс

- GET /todo/ => index
- GET /todo/:id => show
- POST /todo/ => insert
- PUT /todo/:id => update
- DELETE /todo/:id => delete

- ВНИМАНИЕ: браузърите не поддържат сами PUT & DELETE

jQuery AJAX

- Удобство
- Cross-browser

$.get(url,[data], [success(data, textStatus, jqXHR)],[dataType])

Пример:

```
$.get('ajax.php?book_name=Great', function(data)
    {
        $('#suggestion).html(data);
        alert('Load was performed');
    });
```

POST & LOAD & GETJSON

```
$.post( url,[ data ],[ success(data, textStatus, jqXHR) ],[ dataType ] )

$.post('ajax.php', {book_name: 'Great'}, function(data)
      {
            $('#suggestion).html(data);
      });

//load the result in div#suggestion
$('#suggestion).load('ajax.php', {book_name: 'Great'});

//when the result is encoded in JSON
$.getJSON( "ajax.php", function( json )
  {
      console.log( "JSON Data: " + json.users[3].name );
  });
```

The MOTHER of THEM ALL

$.ajax(url, settings)

```
$.ajax({
	type: "POST",
	url: "ajax.php",
	data: {book_name: data},
	dataType: "html",
	success: function (result)
		{
			alert(result);
		}
});
```

`url`	String	The URL for the request.
`type`	String	The HTTP method to use. Usually either POST or GET. If omitted, the default is GET.
`data`	Object	An object whose properties serve as the query parameters to be passed to the request. If the request is a GET, this data is passed as the query string. If a POST, the data is passed as the request body. In either case, the encoding of the values is handled by the `$.ajax()` utility function.
`dataType`	String	A keyword that identifies the type of data that's expected to be returned by the response. This value determines what, if any, post-processing occurs upon the data before being passed to callback functions. The valid values are as follows: ■ `xml`—The response text is parsed as an XML document and the resulting XML DOM is passed to the callbacks. ■ `html`—The response text is passed unprocessed to the callbacks functions. Any `<script>` blocks within the returned HTML fragment are evaluated. ■ `json`—The response text is evaluated as a JSON string, and the resulting object is passed to the callbacks. ■ `jsonp`—Similar to `jason` except that remote scripting is allowed, assuming the remote server supports it. ■ `script`—The response text is passed to the callbacks. Prior to any callbacks being invoked, the response is processed as a JavaScript statement or statements. ■ `text`—The response text is assumed to be plain text.

`timeout`	Number	Sets a timeout for the Ajax request in milliseconds. If the request does not complete before the timeout expires, the request is aborted and the error callback (if defined) is called.
`global`	Boolean	Enables (if `true`) or disables (if `false`) the triggering of so-called global functions. These are functions that can be attached to elements that trigger at various points or conditions during an Ajax call. We'll be discussing them in detail in section 8.8. If omitted, the default is to enable the triggering of global functions.
`contentType`	String	The content type to be specified on the request. If omitted, the default is *application/x-www-form-urlencoded*, the same type used as the default for form submissions.
`success`	Function	A function invoked if the response to the request indicates a success status code. The response body is returned as the first parameter to this function and formatted according to the specification of the `dataType` property. The second parameter is a string containing a status value—in this case, always *success*.
`error`	Function	A function invoked if the response to the request returns an error status code. Three arguments are passed to this function: the XHR instance, a status message string (in this case, always *error*), and an optional excep-

`error`	Function	A function invoked if the response to the request returns an error status code. Three arguments are passed to this function: the XHR instance, a status message string (in this case, always *error*), and an optional exception object returned from the XHR instance.
`complete`	Function	A function called upon completion of the request. Two arguments are passed: the XHR instance and a status message string of either *success* or *error*. If either a success or error callback is also specified, this function is invoked after the callback is called.
`beforeSend`	Function	A function invoked prior to initiating the request. This function is passed the XHR instance and can be used to set custom headers or to perform other pre-request operations.
`async`	Boolean	If specified as `false`, the request is submitted as a synchronous request, By default, the request is asynchronous.
`processData`	Boolean	If set to `false`, prevents the data passed from being processed into URL-encoded format. By default, the data is URL-encoded into a format suitable for use with requests of type *application/x-www-form-urlencoded*.
`ifModified`	Boolean	If `true`, allows a request to succeed only if the response content has not changed since the last request according to the *Last-Modified* header. If omitted, no header check is performed.

Въпроси???