

AJAX

In diesem Teil in die Programmiertechnik des Web 2.0 eingeführt. Dazu wird das Konzept von Ajax erklärt und an einem einfachen Beispiel demonstriert.

Inhalt

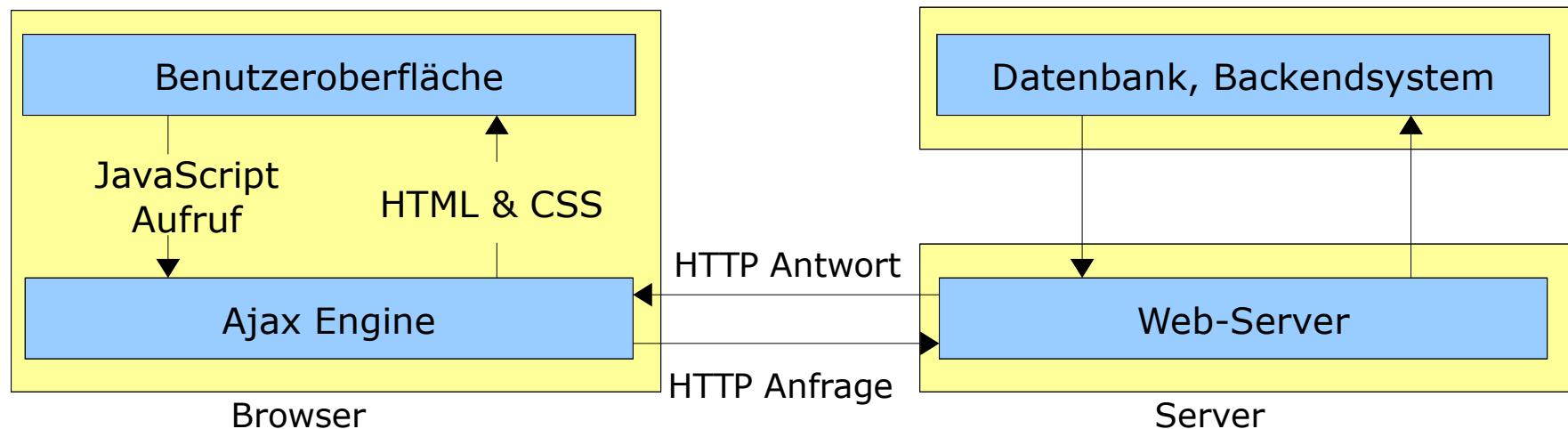
1Überblick.....	2
2XMLHttpRequest Objekt	3
3Bewertung von Ajax.....	16
4AJAX Frameworks.....	17

1 Überblick¹

AJAX beschreibt eine Technologie, mit der Daten zwischen Client und Server mittels JavaScript ausgetauscht werden können, ohne dass die gesamte Web-Seite neu geladen werden muss.

Der Begriff AJAX geht auf den im Februar 2005 veröffentlichten Artikel Ajax: „A New Approach to Web Applications“ von Jesse James Garrett von der Firma Adaptive Path zurück. Dort wird die bis dahin XMLHttpRequest genannte Technologie (von Microsoft 2001 eingeführt) unter dem Namen AJAX (**A**synchronous **J**avascript **a**nd **X**ML) vorgestellt.

Durch diese Technologie ist es möglich, dass eine Webseite ohne dass sie erneut geladen wird, Daten von einem Server abfragt und auf einem Bereich der Seite darstellt. Ermöglicht wird dies durch eine im Hintergrund des Browsers ablaufenden Thread.



¹ Ausarbeitung basiert auf: „<http://ajax.get-the-code.de/>“

2 XMLHttpRequest Objekt

Im Folgenden wird schrittweise eine Ajax-Anwendung entwickelt, die auf der Seite „willkommen.php“ einen **Seitenzähler** anzeigt. Der gesamte Code inklusive der Ajax-Engine ist in dieser Datei enthalten.

```
$cat willkommen.php
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
<meta content="text/html; charset=ISO-8859-1">
<title>Willkommen</title>
<script language="JavaScript">
```

Die **Ajax Engine** wird implementiert durch ein **JavaScript-Programm**, das ein **XMLHttpRequest-Objekt** verwendet. Unglücklicherweise muss man hier zwischen unterschiedlichen Browserstypen unterscheiden, da der Internet-Explorer ein solches Objekt anders instanziiert als der Rest der Browserwelt.

Am einfachsten ist es, wenn man dazu eine eigene JavaScript Funktion erstellt, die das Erzeugen des XMLHttpRequest-Objekts kapselt:

```
// global xmlhttprequest object
var xmlhttp = false;
/** AJAX functions **/
// constants
```

XMLHttpRequest Objekt

```
var REQUEST_GET    = 0;
var REQUEST_POST   = 2;
var REQUEST_HEAD   = 1;
var REQUEST_XML    = 3;
/**
 * instantiates a new xmlhttprequest object
 * @return xmlhttprequest object or false
 */
function getXMLRequester() {
    var xmlhttp = false;
    // try to create a new instance of the xmlhttprequest object
    try {
        // Internet Explorer
        if( window.ActiveXObject ) {
            for( var i = 5; i; i-- ) {
                try {
                    // loading of a newer version of msxml dll (msxml3 - msxml5) failed
                    // use fallback solution
                    // old style msxml version independent, deprecated
                    if( i == 2 ) {
                        xmlhttp = new ActiveXObject( "Microsoft.XMLHTTP" );
                    }
                    // try to use the latest msxml dll
                } else {
                    xmlhttp = new ActiveXObject( "Msxml2.XMLHTTP." + i + ".0" );
                }
            }
        }
    }
}
```

```
        break;
    }
    catch( excNotLoadable ) {
        xmlhttp = false;
    }
}
}
// Mozilla, Opera und Safari
else if( window.XMLHttpRequest ) {
    xmlhttp = new XMLHttpRequest();
}
}
// loading of xmlhttp object failed
catch( excNotLoadable ) {
    xmlhttp = false;
}
return xmlhttp ;
}
```

Nachdem ein **XMLHttpRequest-Objekt instanziiert** ist, kann es verwendet werden, um Daten an den Server zu senden und Daten vom Server zu erhalten. Hierzu wird das HTTP-Protokoll verwendet. Es können also die bekannten Übertragungsmethoden (POST, GET, PUT, HEAD) verwendet werden. Die **Daten**, die an den **Server gesendet** werden, werden in dem bekannten

Format geschickt. Die zu sendende URL besteht also aus einem Zielpfad und gegebenenfalls einem Querystring. Zusätzlich besteht die Möglichkeit, Request-Header zu setzen.

Aus **Sicherheitsgründen** können mittels AJAX nur **Dateien** von der **gleichen Domain angefordert werden**, von der die entsprechende HTML-Seite stammt. Die URL muss identisch mit der, der aktuellen HTML-Seite sein.

Um eine Kommunikation mit dem Server zu starten muss mit der Methode 'open' eine Verbindung geöffnet und mit 'send' gesendet werden.

```
// open the connection
xmlHttp.open( 'GET', strURL, true );

// send request to server
xmlHttp.send( null );    // param = POST data
```

Überr den dritten Parameter der open-Methode kann festgelegt werden, ob die Kommunikation **synchron** ablaufen, der weitere Programmablauf also blockiert werden soll, bis die angeforderten Daten angekommen sind, oder ob sie **asynchron** sein soll. Wird die letztere und empfohlene Möglichkeit genutzt, muss zusätzlich eine **Funktion registriert** werden, die **die ankommenden Daten entgegennimmt und verarbeitet**. Diese Funktion kann als Event-Handler mittels 'onreadystatechange' registriert werden.

```
/**
 * sends a http request to server
 * @param strSource, String, datasource on server, e.g. data.php
```

XMLHttpRequest Objekt

```
* @param strData, String, data to send to server, optionally
* @param intType, Integer,
*     request type, possible values: REQUEST_GET, REQUEST_POST,
*     REQUEST_XML, REQUEST_HEAD default REQUEST_GET
* @param strData, Integer, ID of this request, will be given to
*     registered event handler onreadystatechange', optionally
* @return String, request data or data source
*/
function sendRequest( strSource, strData, intType, intID ) {
    if( !strData )
        strData = '';

    // default type (0 = GET, 1 = xml, 2 = POST )
    if( isNaN( intType ) )
        intType = 0; // GET

    // previous request not finished yet, abort it before sending a new request
    if( xmlHttp && xmlHttp.readyState ) {
        xmlHttp.abort( );
        xmlHttp = false;
    }

    // create a new instance of xmlhttprequest object

    // if it fails, return
    if( !xmlHttp ) {
```

XMLHttpRequest Objekt

```
xmlHttp = getXMLRequester( );
if( !xmlHttp )
    return;
}

// parse query string
if( intType != 1 && ( strData && strData.substr( 0, 1 ) == '&' ||
                    strData.substr( 0, 1 ) == '?' ) )
    strData = strData.substring( 1, strData.length );

// data to send using POST
var dataReturn = strData ? strData : strSource;

switch( intType ) {
    case 1: // xml
        strData = "xml=" + strData;
    case 2: // POST
        // open the connection
        xmlHttp.open( "POST", strSource, true );
        xmlHttp.setRequestHeader( 'Content-Type',
                                   'application/x-www-form-urlencoded' );
        xmlHttp.setRequestHeader( 'Content-length', strData.length );
        break;
    case 3: // HEAD
        // open the connection
        xmlHttp.open( "HEAD", strSource, true );
```

XMLHttpRequest Objekt

```
    strData = null;
    break;
default: // GET
    // open the connection
    var strDataFile = strSource + (strData ? '?' + strData : '');
    xmlhttp.open( "GET", strDataFile, true );
    strData = null;
}
// set onload data event-handler
xmlhttp.onreadystatechange =
    new Function( "", "processResponse(" + intID + ")" ); ;

// send request to server
xmlhttp.send( strData );    // param = POST data

return dataReturn;
}
```

Eine Eigenschaft von AJAX ist, dass **nicht nur Daten an den Server** gesendet und von diesem empfangen werden können, sondern **gleichzeitig auch der Status der Anfrage** und der Status der Server-Antwort **abgefragt** werden können. Sollte beispielsweise eine Datei vom Server abgerufen werden, die dort nicht vorhanden ist, wird der Status 404 zurückgesendet und es kann entsprechend darauf reagiert werden.

Die Daten, die vom Server empfangen worden sind, können entweder mittels '**responseXML**' oder '**responseText**' abgefragt werden. Letzteres enthält die Daten als String, 'responseXML' als XML-Daten. Wurden die Daten vom Server nicht als XML-Daten abgeschickt oder der MIME-Type nicht entsprechend im Header gesetzt ("text/xml"), liefert 'responseXML' null zurück.

Über die **Eigenschaft 'onreadystatechange'** des XMLHttpRequest-Objekts kann eine Funktion als **Event-Handler registriert** werden, die die Daten vom Server entgegen nimmt. War die Anfrage erfolgreich, können die Daten jetzt weiterverarbeitet werden, andernfalls Fehlerbehandlung durchgeführt werden.

Eine solche Event-Handler-Funktion könnte folgendermaßen aussehen:

```
/**
 * process the response data from server
 * @param intID, Integer, ID of this response
 */
function processResponse( intID ) {
    // status 0 UNINITIALIZED open() has not been called yet.
    // status 1 LOADING send() has not been called yet.

    // status 4 COMPLETED Finished with all operations.
    switch( xmlHttp.readyState ) {
        // uninitialized
        case 0:
            // loading
        case 1:
```

XMLHttpRequest Objekt

```
// loaded
case 2:
  // interactive
case 3:
  break;
  // complete
case 4:
  // check http status
  if( xmlhttp.status == 200 )    // success
  {
    processData( xmlhttp, intID );
  }
  // loading not successfull, e.g. page not available
  else {
    if( window.handleAJAXError )
      handleAJAXError( xmlhttp, intID );
    else
      alert( "ERROR\n HTTP status = " +
            xmlhttp.status + "\n" + xmlhttp.statusText ) ;
  }
}
} // processResponse
/** End AJAX functions **/
```

Mit den o.a. Funktionen ist die Ajax-Engine vollkommen definiert. Nun müssen die **Anwendungsspezifischen Funktionen** realisiert werden, hier nun die Funktion **processData**, die die empfangenen Daten (einen Seitenzähler) auf der Seite mittels div-Elementen anzeigt.

```
/** real application functions */  
// process data from server  
function processData( xmlHttp, intID ) {  
    // process text data  
    //alert(xmlHttp.responseText);  
    updateCounter( xmlHttp.responseText );  
}  
  
// process data from server  
function updateCounter( strData ) {  
    if( strData ) {  
        var obj=document.getElementById("cnt");  
        var oldValue = obj.childNodes[0].nodeValue;  
        if (oldValue != strData) {  
            obj.childNodes[0].nodeValue=strData;  
            obj.style.visibility = "visible";  
        } else  
            obj.style.visibility = "hidden";  
    }  
}
```

XMLHttpRequest Objekt

```
function getData() {  
    var strURL = "getCounter.php";  
    // alert("sending request"); // just to test  
    sendRequest( strURL );  
    setTimeout("getData()",10000);  
}  
</script>
```

Nun noch die Seite, die angefordert wird, und die o.a. JavaScript Funktionen beinhaltet:

```
</head>  
<body onload="getData();" >  
    Hallo Kurs!  
    <?php  
        require_once("counter.inc");  
$cnt=read_cnt()+1;  
write_cnt($cnt);  
echo "<div id='cnt' class='pagecount' style='float: right; visibility: visible;'  
>".$cnt."</div>";  
?>  
</body>  
</html>  
$
```

Ende der Datei "willkommen.php"

Der Rest ist nur noch das **php-Programm**, das **asynchron** von der **Ajax-Engine aufgerufen wird**:

```
$cat counter.inc
$file="./willkommen.cnt";
function write_cnt($counter) {
    global $file;
    $fp = @fopen($file, "w");
    if (!$fp) {
        return;
    }
    fwrite($fp, $counter,strlen($counter));
    fwrite($fp,"\n",1);
    fclose($fp);
}
function read_cnt() {
    global $file;
    $fp = @fopen($file, "r");
    if (!$fp)
        return 0;
    $cnt=fgets($fp, 100);
    fclose($fp);
    return $cnt;
}
$
$cat getCounter.php
```

XMLHttpRequest Objekt

```
<?php
require_once('counter.inc');

$cnt=read_cnt();
$ts = time();
$ts_str = date("[d.m.Y H:i] ", $ts);
echo $ts_str,$cnt;
?>
$
```

Bewertung von Ajax

3 Bewertung von Ajax

-> Diskussion

4 AJAX Frameworks

-> RICO Framework