Date: April 12, 2006 / Author: Ralf Eichinger

Ajax für Einsteiger

- Webanwendung alt
- Webanwendung neu
  - Kommunikation über XMLHttpRequest
  - Erzeugen des XMLHttpRequest-Objekts
  - Anfrage an Server (Request)
  - Antwort vom Server (Response)
  - Auswertung Antwortdaten in “responseText”
  - XML-Daten für einen einfachen RSS-Newsreader
  - Auswertung statischer Antwortdaten in “responseXML”
  - Auswertung dynamischer Antwortdaten in “responseXML”
  - Der Apache als Proxy
- Frameworks
- Fazit
- Quellen

Bis vor kurzem hätte wohl jeder auf die Frage “Was ist Ajax?” an ein Putzmittel gedacht. Im Umfeld von Webanwendungen sprechen jedoch viele schon vom Web der Zukunft - von Web 2.0.

Was hat es mit der Webtechnologie AJAX (Asynchronous JavaScript And XML) auf sich, dass sie solche Aufregung verursacht?

Webanwendung alt

Zunächst einmal muss man verstehen, wie Webanwendungen bisher funktionieren. Betrachten wir eine Webanwendung, die jeder kennt: den Google-Suchdienst.

Als Benutzer mit einem Webbrowser erfolgt die Nutzung in folgenden Schritten:

1. Der Benutzer ruft die Webadresse “http://www.google.de” auf.
2. Der Google-Server antwortet auf die Anfrage mit der Suchseite, die u.a. eine bunte Grafik, eine Fußzeile und das Suchfeld zur Eingabe von Stichwörtern enthält.
3. Der Benutzer gibt den Suchbegriff “Ajax” ein und schickt die Anfrage mit Klick auf den Knopf “Suchen” an den Google-Server.
4. Der Google-Server recherchiert in seiner Datenbank und antwortet auf die Anfrage mit einer Liste der Suchergebnisse und wiederum der Grafik, der Fußzeile, dem Suchfeld …

Zwei Dinge fallen bei diesem Beispielszenario auf:

- Webanwendungen funktionieren nach dem Prinzip “Anfrage (Request) - Antwort (Response) - Anfrage - Antwort - …”
- Der Google-Server schickt nicht nur die interessierende Suchergebnisliste, sondern zusätzlich einiges an “unnötigen” Ballast, d.h. Inhalten, die schon einmal übertragen wurden (Grafik, Fußzeile, Suchfeld)

Webanwendung neu

Genau dies ist der Ansatzpunkt von AJAX:

Der unveränderliche Inhalt einer Webseite soll nicht jedesmal neu übertragen werden, sondern nur die veränderlichen Nutzdaten.

Wie funktioniert das? Wie lassen sich Teile einer Webseite verändern ohne die Seite neu zu laden?

Die Antwort ist: mit DHTML (Dynamic HTML), das auf JavaScript und HTML basiert.

Mit Javascript wird die Anfrage an den Server gestellt, der dynamische Inhalt (z.B. Suchergebnisse) vom Server geholt und die Antwortdaten ausgewertet.

AJAX ist also keine neue Technologie, sondern “nur” die Weiterentwicklung bereits etablierter Technologien. Das “J”(avascript) in AJAX ist damit erklärt.

Aber wieso A wie “asynchron” und X wie “XML”?

“Asynchron” bedeutet laut Duden “nicht gleichzeitig”. Bei einem mit AJAX realisierten Google-Suchdienst würde dies bedeuten: Nach dem Senden der Anfrage an den Suchdienst wird nicht auf die Antwort gewartet (synchrones Verhalten), sondern das Programm ohne Verzögerung/Wartezeit fortgeführt und die Antwort dann ausgewertet, wenn sie (später) eintrifft (asynchrones Verhalten).

XML gilt unter AJAX als das bevorzugte Datenaustauschformat. XML-Antwortdaten lassen sich mit JavaScript einfach auswerten und in darstellbaren HTML-Inhalt umwandeln.

Der mittels JavaScript generierte HTML-Code kann an Positionen der HTML-Seite eingefügt werden, die durch CSS-IDs gekennzeichnet werden.

Zusammenfassend kann gesagt werden, daß AJAX auf den folgenden (altbekannten) Technologien basiert:

- JavaScript
- XML
- HTML
- CSS

Kommunikation über XMLHttpRequest

Doch wie kann man mit JavaScript eine Anfrage an einen Server schicken und die Antwort holen? Hier kommt eine neue Technologiekomponente ins Spiel, die in alten Browsern nicht vorhanden ist, AJAX jedoch erst möglich macht: XMLHttpRequest.

Folgende Browserversionen unterstützen diese Schnittstelle:

- Apple Safari (ab 1.2)
- Internet Explorer (ab 5.0)
- KDE Konqueror

- Mozilla Firefox (ab 1.0)
- Netscape (ab 7.1)
- Opera (ab 7.6)

Beim Internet Explorer handelt es sich dabei um ein ActiveX-Objekt. Bei allen anderen Browsern ist es direkt als XMLHttpRequest-Objekt verfügbar.

Mittels XMLHttpRequest können aus JavaScript heraus Anfragen (Requests) an einen Server geschickt und die Antwort-Daten (Response) empfangen werden. Die Antwort-Daten werden im XMLHttpRequest-Objekt als XML-DOM oder reiner Text gespeichert und können mit JavaScript ausgelesen und verarbeitet (und z.B. als HTML formatiert in die Seite eingefügt) werden.

Als XML-basierte Beispielanwendung bietet sich ein Newsreader an, der die Nachrichten im RSS-Format (einem XML-Format) von einem Nachrichtenserver holt und die Schlagzeilen als anklickbare Liste darstellt.

Dass der Zugriff auf Inhalte eines fremden Servers leider nicht so einfach ist, erklärt der Kasten “Fremdcontent nur indirekt”.

Fremdcontent nur indirekt

Der direkte Zugriff aus dem XMLHttpRequest-Objekt heraus auf fremde Server ist jedoch meist gesperrt. Es ist denselben Sicherheitseinstellungen des Browsers unterworfen wie jeder andere Javascript-Code auch. Um sicherzugehen, dass Ihre AJAX-Anwendung möglichst unabhängig von Sicherheitseinstellungen agieren kann, muß die Domain der Anfrage-URL dieselbe sein, wie die Domain, von der die AJAX-Seite stammt. Sie müssen also immer mit Ihrem eigenen Server kommunizieren.

Sie hindert natürlich niemand daran, dass der Server über ein CGI, Servlet, PHP-Skript etc. fremden Content besorgt und an Sie ausliefert...

Bevor wir uns an die Nachrichtenliste wagen, betrachten wir grundlegenden Beispiel-Code, wie in JavaScript mit dem XMLHttpRequest-Objekt gearbeitet wird.

Erzeugen des XMLHttpRequest-Objekts

Bei der Erzeugung des Objekts prüfen wir, ob es direkt (Nicht-IE-Browser) erzeugbar ist oder als ActiveX-Objekt erzeugt werden muß. Sollte beides nicht möglich sein (Browser, der die Schnittstelle nicht unterstützt), wird der auftretende Fehler abgefangen und sollte von Ihnen behandelt werden (z.B. Anzeige eines Fehler-Alerts).

```
<script type="text/javascript">
var xmlhttp;

try {
  if (window.XMLHttpRequest) {
    xmlhttp = new XMLHttpRequest();
  } else {
    xmlhttp = new ActiveXObject("Microsoft.XMLHTTP");
  }
} catch (e) {
  // Schnittstelle wird nicht unterstützt...
}
</script>
```

Bei allen Browsern (außer IE) kann das XMLHttpRequest-Objekt über “new XMLHttpRequest()” erzeugt werden. Beim Internet Explorer muß es als ActiveX-Objekt erzeugt werden. Der Parameter “Microsoft.XMLHTTP” funktioniert für alle IE-Versionen, die die Schnittstelle unterstützen, nutzt jedoch nicht die neuesten Versionen des Microsoft XML-Parsers (Msxml.dll). Die Parameter zur Nutzung der neueren Versionen lauten: “Msxml2.XMLHTTP.3.0”, “Msxml2.XMLHTTP.4.0”, “Msxml2.XMLHTTP.5.0”.

Unsere Funktion zur Erzeugung des XMLHttpRequest-Objekts wird daher um das unter [1] vorgeschlagene Verfahren erweitert:

```
<script type="text/javascript">
var xmlhttp = false;

try {
  if (window.XMLHttpRequest) {
```

```
    xmlhttp = new XMLHttpRequest();
  } else if (window.ActiveXObject) {
    for (var i=5; i; i--) {
      try {
        if (i == 2) {
          xmlhttp = new ActiveXObject("Microsoft.XMLHTTP");
        } else {
          xmlhttp = new ActiveXObject("Msxml2.XMLHTTP." + i + ".0");
        }
        break;
      } catch (e) {
        xmlhttp = false;
      }
    }
  }
} catch (e) {
  xmlhttp = false;
}
</script>
```

Methoden und Eigenschaften des XMLHttpRequest-Objekts

In den folgenden Tabellen sind alle Methoden und Eigenschaften des XMLHttpRequest-Objekts aufgelistet. Diese Tabellen sind den Artikeln unter [2] und [1] entnommen:

Methode	Kurzbeschreibung
abort()	Stoppt die aktuell laufende Server-Anfrage.
getAllResponseHeaders()	Gibt den kompletten vom Server gesendeten Headersatz (Feldnamen und Werte) als String zurück.
getResponseHeader("Headername")	Gibt den Wert eines einzelnen Headerfeldes als String zurück.
open("Methode", "URL"[, asyncFlag[, "Benutzername"[, "Passwort"]]])	Stellt eine Verbindung zum Server her. Parameter:

	• Methode: GET, POST, PUT, HEAD POST sollte verwendet werden, wenn die gesendeten Daten größer als 500 Bytes sind. HEAD wird verwendet, wenn nur Response-Header und keine Daten angefordert werden. Dieses kann z.B. verwendet werden, wenn für eine Datei auf dem Server das Datum der letzten Änderung (Last-Modified) abgefragt werden soll. • URL: Pfad (relativ oder absolut) + ggf. Anfrageparameter als Query-String • asyncFlag: true, false Asynchrone (true) oder synchrone (false) Datenübertragung • Benutzername, Passwort: Benutzername und Passwort für den Zugriff auf eine Ressource auf dem Server.
send(Anfrageparameter)	Sendet die Anfrage an den Server. Parameter: • Anfrageparameter: bei GET: null (Anfrageparameter bereits in URL als Query-String enthalten) bei POST: Anfrageparameter als Query-String oder DOM-Datenobjekt
setMimeType("Mime-Type")	Setzen des Mimetyps der angeforderten Daten (Response). (Wird vom Internet Explorer nicht unterstützt) Parameter: • Mime-Type: der Mimetyp als String, z.B. “text/xml”
setRequestHeader("Feldname", "Headerwert")	Setzt ein Name-/Wert-Paar im Request-Header, z.B. den Content-Type

Eigenschaft	Kurzbeschreibung

onreadystatechange	Name der Funktion (Event-Handler), die jedesmal aufgerufen wird, wenn sich der Verbindungsstatus (readyState) ändert.
readyState	Enthält den aktuellen Verbindungsstatus: Mögliche Werte: • 0 (uninitialized): Verbindung noch nicht aufgebaut (“open()” wurde noch nicht aufgerufen) • 1 (loading): Anfrage noch nicht gesendet (“send()” wurde noch nicht aufgerufen) • 2 (loaded): Anfrage gesendet, Antwort-Header und Antwort-Status kann abgefragt werden. • 3 (interactive): Übertragung der Daten vom Server findet statt, die eintreffenden Daten werden in “responseText” aufgesammelt. • 4 (complete): Die Kommunikation mit dem Server ist abgeschlossen. Alle Daten sind angekommen (bei status = 200), wenn kein Fehler aufgetreten ist (z.B. status = 404, Dokument nicht gefunden)
responseXML	Enthält die Antwortdaten im XML-Format (DOM-kompatibles Document-Objekt). Wenn die Daten als Plaintext gesendet worden sind, enthält responseXML null.
responseText	Enthält die Antwortdaten als Text.
status	Enthält den HTTP-Status der Verbindung als Zahl (z.B. 200 für "OK" oder 404 für "Not Found")
statusText	Enthält den HTTP-Status als Text-Meldung (z.B. "Not found")

Anfrage an Server (Request)

Nach der Erzeugung des XMLHttpRequest-Objekts kann eine Verbindung zum Server hergestellt und ein Request über das HTTP-Protokoll geschickt werden. Im einfachsten Fall handelt es sich um einen GET-Request:

```
xmlhttp.open("GET", url, true); // Verbindung herstellen
xmlhttp.onreadystatechange = handleResponse; // Handler registrieren
xmlhttp.send(null); // Request schicken
```

Die Parameter bei der Herstellung der Verbindung (Funktion “open”) sind:

1. Request-Methode: “GET”, “POST”, “PUT”, “HEAD”,
2. Abfrage-URL (enthält bei einem GET-Request die Abfrageparameter als Query-String), von der die Antwort erwartet wird,
3. true: die Kommunikation mit dem Server soll asynchron erfolgen, d.h. nach dem Wegschicken des Requests wird sofort der weitere Programmcode ausgeführt ohne die Antwort abzuwarten. Dadurch entsteht kein “Hängen” der Anwendung.

Bei asynchroner Verarbeitung (Standardfall) muß eine Funktion (sog. Event-Handler) angegeben werden, die die Antwortdaten beim Eintreffen der Antwort weiterverarbeitet. Dies geschieht über Zuweisung des Funktionsnamens zur Eigenschaft “onreadystatechange” des XMLHttpRequest-Objekts. Dies muß vor dem Senden des Requests geschehen.

Der Parameter der Funktion “send” ist nur bei einem POST-Request (mit den Abfrageparametern) zu füllen.

Antwort vom Server (Response)

Die Verarbeitung der Server-Antwort geschieht innerhalb der Handler-Funktion “handleResponse”. Diese Funktion wird bei jeder Änderung des Status der Kommunikation aufgerufen. Der Status kann vom Beginn bis zum Abschluß der Kommunikation mehrere Zustände annehmen.

Der Kommunikationsstatus kann über

```
xmlhttp.readyState
```

abgefragt werden und kann folgende Zahlenwerte annehmen:

0	UNINITIALIZED Verbindung wurde noch nicht über open() hergestellt
1	LOADING Request wurde noch nicht über send() abgeschickt

2	LOADED Request wurde über send() abgeschickt, Header und Status sind verfügbar
3	INTERACTIVE Antwort wird heruntergeladen, die Daten werden in responseText gespeichert
4	COMPLETED Alle Verarbeitungsschritte abgeschlossen. Daten oder Fehler können ausgewertet werden.

Sobald die Übertragung der Antwortdaten ohne Fehler abgeschlossen ist (readyState 4, HTTP-Status 200), kann auf diese über

```
xmlhttp.responseText
```

oder

```
xmlhttp.responseXML
```

zugegriffen werden. Die Text-Antwortdaten sind in “responseText”, XMLAntwortdaten in “responseXML” gespeichert.

“responseXML” ist dabei nur gefüllt, wenn die Daten im XML-Format geschickt wurden und der MIME-Typ “text/xml” im Header gesetzt wurde, ansonsten ist “responseXML” null.

Auswertung Antwortdaten in “responseText”

Handelt es sich bei der erwarteten Antwort nicht um XML, sondern z.B. um fertigen HTML-Code, der in die Webseite eingefügt werden soll, können die Antwortdaten direkt aus “responseText” verwendet werden.

Die “responseText”-Daten brauchen dazu nur in die “innerHTML”-Eigenschaft eines bestehenden HTML-Elements der Webseite geschrieben werden. Das HTML-Element muß dabei über eine eindeutige “id” ansprechbar sein:

```
function handleResponse() {
  if ((xmlhttp.readyState == 4) && (xmlhttp.status == 200)) {
    // xmlhttp.responseText object contains the response.
```

```
    document.getElementById("antwort").innerHTML = xmlhttp.responseText;
  }
}
```

In einem ersten Beispiel, wollen wir den Inhalt einer Datei “antwort.txt” in das HTML-Element mit der id “antwort” einfügen. Den Quellcode zur Erzeugung des XMLHttpRequest-Objekts kapseln wir in der Methode “getXMLHttpRequest()”, den Quellcode für die Anfrage an den Server in der Funktion “getResponse(url)” und die Auswertung der Antwort in der Funktion “handleResponse()”. Der Inhalt dieser Funktionen wird sich auch im weiteren Verlauf nicht mehr ändern. Den vollständigen Quellcode finden Sie im Kasten “Quellcode ‘Hallo Welt!’”.

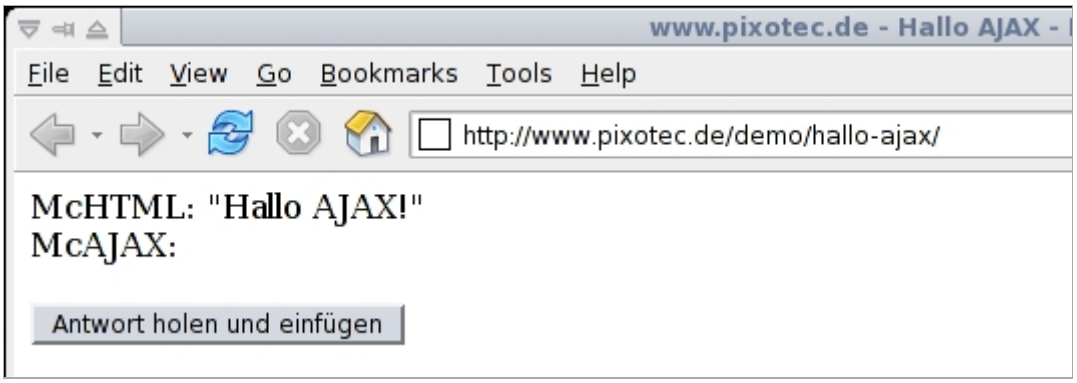

Bild: Die erste AJAX-Anwendung: “Hallo Welt!” vor dem Holen der Antwort

Quellcode “Hallo Welt!”

```
<html>
<head>
<title>www.pixotec.de - Hallo AJAX</title>
<script type="text/javascript">
var xmlhttp = null;

function getXMLHttpRequest() {
  if (xmlhttp == null) {
    try {
      if (window.XMLHttpRequest) {
        xmlhttp = new XMLHttpRequest();
      } else if (window.ActiveXObject) {
        for (var i=5; i; i--) {
          try {
            if (i == 2) {
              xmlhttp = new ActiveXObject("Microsoft.XMLHTTP");
```

```
            } else {
              xmlhttp = new ActiveXObject("Msxml2.XMLHTTP." + i + ".0");
            }
            break;
          } catch (e) {
            xmlhttp = null;
          }
        }
      }
    } catch (e) {
      xmlhttp = null;
    }
  }
  return xmlhttp;
}

function getResponse(url) {
  xmlhttpReq = getXMLHttpRequest();
  if (xmlhttpReq != null) {
    xmlhttpReq.open("GET", url, true);
    xmlhttpReq.onreadystatechange = handleResponse;
    xmlhttpReq.send(null);
    return true;
  } else {
    alert("AJAX wird von Ihrem Browser nicht unterstützt.");
    return false;
  }
}

function handleResponse() {
  if ((xmlhttp != null)
      && (xmlhttp.readyState == 4)
      && (xmlhttp.status == 200)) {
    document.getElementById("antwort").innerHTML = xmlhttp.responseText;
  }
}
</script>
</head>

<body>
McHTML: "Hallo AJAX!"<br>
```

```
McAJAX: <span id="antwort"></span><br>
<br>
<input type="button" onClick="getResponse('/demo/hallo-ajax/antwort.txt'
       value="Antwort holen und einfügen" />
</body>
</html>
```

Beim Klick auf den Button “Antwort holen und einfügen”, wird der Inhalt der Datei “antwort.txt” vom Server geholt und innerhalb des HTML-Elements `<span id="antwort">` eingefügt.

Das Beispiel können Sie live testen unter http://www.pixotec.de/demo/hallo-ajax/.

Dieses Beispiel dient als Basis für unseren RSS-Newsreader. Der Unterschied besteht in der Auswertung der Daten.

XML-Daten für einen einfachen RSS-Newsreader

Daten im XML-Format bieten mehr Möglichkeiten als unstrukturierte Textdaten. Durch standardisierte Zugriffsmethoden des W3C Document Object Model (DOM) lassen sich XML-Antwortdaten einfach auswerten. Die Antwort vom Server ist als DOM-Datenmodell in der Eigenschaft “responseXML” des XMLHttpRequest-Objekts gespeichert.

Im Falle unseres Nachrichten-Lesers, sind dies XML-Daten im RSS-Nachrichtenformat. Es handelt sich dabei um den News-Überblick der Zeitschrift “PC Magazin”, abrufbar unter der URL:

https://www.pc-magazin.de/rss/alle.xml (aktualisiert am 28.02.2022)

Auswertung statischer Antwortdaten in “responseXML”

Für den ersten Prototypen empfiehlt es sich mit einer statischen XML-Datei zu arbeiten, die im gleichen Verzeichnis liegt wie die HTML-Seite mit dem AJAX-Code für den Newsreader. Wir haben über die oben genannte URL von der PC-Magazin-

Webseite eine Liste der aktuellen News heruntergeladen und neben “index.html” unter “rss.xml” abgespeichert.

Der Inhalt der XML-Datei “rss.xml” ist folgendermaßen strukturiert:

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<rss version="0.92">
  <channel>
    <title>Aktuelle www.pc-magazin.de-News</title>
    ...
    <lastBuildDate>Sat, 08 Apr 2006 20:27:15 GMT</lastBuildData>
    ...
    <item>
      <title>MIT-Forscher bauen Akkus aus Viren</title>
      ...
      <link>http://www.pc-magazin.de/common/nws/einemeldung.php?id=44580</
    </item>
    <item>
      ...
    </item>
    ...
  </channel>
</rss>
```

Das gesamte XML-Dokument enthält noch einige andere Informationselemente, die wir jedoch in unserem einfachen Newsreader nicht auswerten wollen. Der Newsreader soll eine einfache Liste der Schlagzeilen darstellen, von denen jede mit dem Volltext der Nachricht verlinkt ist.

Nehmen wir den Quellcode des “Hallo Welt!”-Beispiels als Grundlage. Es ändern sich folgende Abschnitte:

- Der Titel des Dokuments
- Die Abfragemaske (body-Abschnitt) und der darin enthaltene “getResponse(url)”-Aufruf.
- Die Auswertungsfunktion “handleResponse()”, die nun nicht mehr Text-Inhalte aus “responseText”, sondern XML-Daten aus “responseXML” auswerten soll.

Der neue Quellcode ist im Kasten “Quellcode statischer Newsreader” aufgelistet. Abschnitte, die sich nicht geändert haben, sind mit “…” gekürzt.

Quellcode statischer Newsreader

```
<html>
<head>
<title>www.pixotec.de - pixoFeed (statisch)</title>
<script type="text/javascript">
...
function getXMLHttpRequest() { ... }

function getResponse(url) { ... }

function handleResponse() {
  if ((xmlhttp != null)
      && (xmlhttp.readyState == 4)
      && (xmlhttp.status == 200)) {
    var newsliste = document.getElementById("newsliste");
    newsliste.innerHTML = ""; // vorherige Liste löschen

    // Titel des News-Channel
    var channel =
    xmlhttp.responseXML.getElementsByTagName("channel")\[0\];

    var titleNewsChannel = channel.getElementsByTagName("title");
    var channelTitle = document.createTextNode(titleNewsChannel\[0\].fir
    newsliste.appendChild(channelTitle);

    var date = channel.getElementsByTagName("lastBuildDate");
    var dateText = document.createTextNode(" (" + date\[0\].firstChild.r
    newsliste.appendChild(dateText);

    // Liste aller Schlagzeilen
    var ul = document.createElement("ul");
    newsliste.appendChild(ul);

    var items = channel.getElementsByTagName("item");
    for (var i=0; i<items.length; i++) {
      var item = items\[i\];
```

```
      var li = document.createElement("li");
      var itemTitle = item.getElementsByTagName("title")\[0\].firstChild
      var itemLink = item.getElementsByTagName("link")\[0\].firstChild.r
      var a = document.createElement("a");

      var href = document.createAttribute("href");
      href.nodeValue = itemLink;
      a.setAttributeNode(href);

      var target = document.createAttribute("target");
      target.nodeValue = "_blank";
      a.setAttributeNode(target);

      a.appendChild(document.createTextNode(itemTitle));
      li.appendChild(a);
      ul.appendChild(li);
    }
  }
}
</script>
</head>
<body bgcolor="#FFFFFF">

  <div align="center">
    <table border="2" bgcolor="\#AAAAAA">
      <tr>
        <th>pixoFeed - Newreader</th>
      </tr>
      <tr>
        <td align="center">
          <input type="button" onClick="getResponse('rss.xml');" value="
        </td>
      </tr>
      <tr>
        <td bgcolor="#FFFFFF">
          <span id="newsliste"></span>
        </td>
      </tr>
    </table>
  </div>
</body>
```

```
</html>
```

Die Antwort wird nun aus der Datei “rss.xml” geholt, die Daten aus dem DOM-Modell der Antwort ausgelesen, HTML erzeugt und das erzeugte HTML in das HTML-Element `<span id="newsliste">` eingefügt.

Die Nachrichten-Schlagzeilen lassen sich nun anklicken und der Volltext wird in einem neuen Fenster geöffnet.

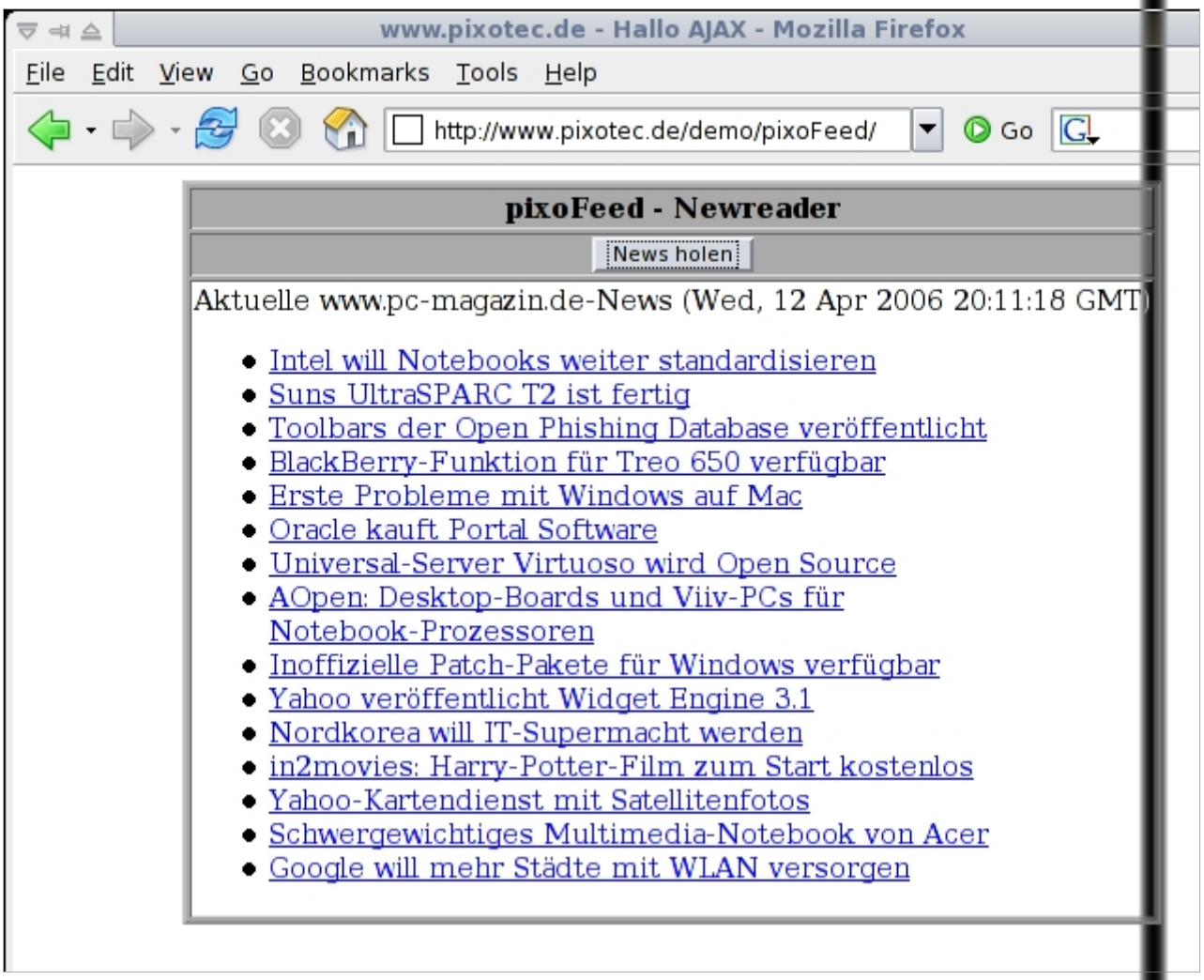

Bild: Der fertige AJAX-Newsreader nach dem Holen der Daten

Zur Auswertung der DOM-Daten wird dabei auf folgende Funktionen zurückgegriffen:

- .getElementById(“id”):
  gibt das DOM-Element mit der angegebenen eindeutigen id zurück
- .getElementsByTagName(“tagname”):
  gibt alle DOM-Elemente mit dem angegebenen Tag-Namen als Array zurück
- .firstChild:

gibt das erste Kindelement eines DOM-Elements zurück (z.B. den darin enthaltenen Text-Knoten)

- .nodeValue:
  gibt den enthaltenen Wert eines Knotens zurück (z.B. den Text)
- .appendChild(element):
  fügt ein Node-Element als Kindknoten ein (um z.B. verschachtelte HTML-Strukturen zu erzeugen)
- .setAttributeNode(attribut):
  fügt einem Node-Element ein Attribut hinzu\
- document.createTextNode(“value”):
  erzeugt einen neuen Text-Knoten mit dem Text-Inhalt “value”
- document.createElement(“elementname”):
  erzeugt ein neues DOM-Element mit dem Tag-Namen “elementname”
- document.createAttribute(“attributename”):
  erzeugt ein neues Element-Attribut mit dem Namen “attributname”

Eine detailliertere Beschreibung der Javascript/DOM-Funktionen sind unter [3] zu finden.

Auswertung dynamischer Antwortdaten in “responseXML”

Der arbeitsintensivste Teil (Auswertung der XML-Daten) ist nun fertig. Auf die Dauer werden die immer selben Nachrichten natürlich langweilig. Um immer die aktuellsten Nachrichten zu holen wäre der direkte Zugriff auf die RSS-URL des “PC Magazin”-Servers notwendig. Das lassen die Sicherheitseinstellungen der meisten Browser jedoch nicht zu. Um die Daten über den eigenen Server zu holen, könnte man evtl. auf PHP, CGI, Servlet oder eine andere serverseitige Technik zurückgreifen. In unserem Anwendungsfall ist dies jedoch weit einfacher möglich.

Der Apache als Proxy

Der Apache-Webserver bietet die Möglichkeit, als Proxy zu fungieren. Anfragen an

eine URL auf einem anderen Server können somit über den Apache umgeleitet werden. Aus Sicht des AJAX-Programms scheinen die Antwortdaten direkt vom eigenen Apache-Server zu kommen. Es kommt zu keinem Sicherheitskonflikt mehr.

Um den Apache für die Umleitung zu konfigurieren, sind folgende Einträge in der Konfigurationsdatei `httpd.conf` nötig:

```
LoadModule proxy_module modules/mod_proxy.so

<IfModule mod_proxy.c>
  ProxyPass /pcmagazin/ http://www.pc-magazin.de/
</IfModule>
```

Das Modul `mod_proxy` leitet somit alle Anfragen an “/pcmagazin/” an den Server “http://www.pc-magazin.de/” weiter.

Die URL zum Holen der dynamisch erzeugten RSS-Datei lautet somit:

`/pcmagazin/rss/alle.xml`

Der Quellcode unseres dynamischen Newsreaders ist somit vollständig. Der Quellcode des statischen Newsreaders muß für die dynamische Variante nur minimal geändert werden. Es ändert sich (außer dem Titel des Dokuments) nur die Anfrage-URL:

```
<input type="button"
       onClick="getResponse('/pcmagazin/rss/alle.xml');"
       value="News holen" />
```

Die fertige Anwendung können Sie live testen unter “http://www.pixotec.de/demo/pixoFeed/”.

Frameworks

Schon in unseren einfachen Beispielen wird deutlich, daß sich einige Funktionen nicht ändern und daher in ein eigenes “Framework” (im Sinne von “Funktions-Bibliothek”) auslagern lassen. Der Aufbau eines eigenen Frameworks mag in einigen Fällen Sinn machen, es sind jedoch schon umfangreiche OpenSource-Frameworks

verfügbar.

Dazu gehören u.a.:

- Rico (JavaScript for Rich Internet Applications): http://openrico.org
- Prototype (JavaScript Framework): http://prototype.conio.net
- script.aculo.us (Erweiterung von Prototype): http://script.aculo.us
- Dojo (the browser toolkit): http://dojotoolkit.org
- Sajax (Simple AJAX Toolkit): http://www.modernmethod.com/sajax

Fazit

Durch die Verwendung von AJAX werden Webanwendungen kleiner, schneller und benutzerfreundlicher. Da es sich um eine rein browserseitige Technologie (JavaScript) handelt, ist keine zusätzliche Software auf dem Server notwendig. Durch Verwendung von Standardtechniken ist auch kein Plugin nötig.

Neben der flüssigeren Seitenaufbereitung bieten AJAX-Anwendungen weitere Verbesserungen hinsichtlich der Benutzerfreundlichkeit: Mit AJAX lassen sich Funktionalitäten umsetzen, die der Benutzer von “normalen” Desktop-Anwendungen her gewohnt ist wie Kontext-Menüs und Drag&Drop.

Programmierer müssen sich daran gewöhnen, neben serverseitiger verstärkt auch clientseitige Programmierung einzusetzen. Ohne JavaScript-Kenntnisse ist AJAX-Programmierung nicht möglich.

Durch den Einsatz vorhandener Frameworks wird die tägliche Programmierung jedoch stark vereinfacht.

Komplexe AJAX-Anwendung sind bereits im Einsatz und lassen viele auf einen zweiten Internet-Hype unter dem Titel “Web 2.0” hoffen. Verpassen Sie nicht den

Anschluß!

Einige AJAX-Anwendungen

- Google Gmail (http://gmail.google.com)
- Web-Groupware “Zimbra”: Mail-Client ähnlich Outlook (http://ww.zimbra.com)
- Online-Textverarbeitung “Writely” (http://www.writely.com)
- Online-Tabellenkalkulation “NumSum” (http://www.numsum.com)
- Online-Desktop “Windows Live” (http://www.live.com)
- Online-Desktop “Netvibes” (http://www.netvibes.com)
- Online Desktop “Protopage” (http://www.protopage.com)
- Aufgaben und Terminverwaltung “Backpack” (http://www.backpackit.com)
- Social Bookmarking “del.icio.us” (http://del.icio.us)

Quellen

1. AJAX - Asynchronous Javascript and XML, ajax.get-the-code.de ↩ ↩2
2. Dynamic HTML and XML: The XMLHttpRequest Object, http://developer.apple.com/internet/webcontent/xmlhttpreq.html, Apple Developer Connection ↩
3. SelfHTML 8.1.1, http://de.selfhtml.org, Kapitel “JavaScript/DOM” ↩

Tags: topics javascript