Stefan Fischerländer

»Crawl me, Baby!

Mit dem Boom des Web 2.0 gewann auch die lange Zeit verpönte Programmiersprache Javascript als wichtigstes Element der Technologie „Ajax" wieder an Bedeutung. Doch sind Ajax-Websites konzeptbedingt für Suchmaschinen unzugänglich. Nun hat Google ein Verfahren entwickelt, um dieses Problem zu lösen. Lernen Sie, wie Sie eine bestehende Website „ajaxifizieren" und welche Fallstricke dabei lauern.

Die im Webbrowser integrierte Sprache Javascript ermöglicht seit ihrem Erscheinen 1995 spannende Interaktionsmöglichkeiten, allerdings war der Nutzen dieser Möglichkeiten lange Zeit recht gering. Denn es gab – von üblen Basteleien mit LiveConnect einmal abgesehen – keine Möglichkeit, Nutzereingaben an den Webserver zurückzusenden und somit das im Browser ausgeführte Javascript-Programm mit neuen Daten zu versorgen. Dass ausgerechnet Microsoft diesen Mangel mit der Einführung der XMLHttpRequest-API behob und so Konkurrenzprodukte wie Google Docs erst möglich machte, ist eine Ironie der Webgeschichte.

Der XMLHttpRequest* erlaubt dem Browser, eine Anfrage an den Webserver zu senden und in der darauf folgenden Antwort Daten zu erhalten, ohne dass dadurch die aktuelle Seite verlassen würde. Der Nutzer bekommt von dieser im Hintergrund stattfindenden Kommunikation nichts mit: Weder ändert sich die im Browser angezeigte URL noch wird der Seiteninhalt neu aufgebaut – ja, er kann sogar die Seite weiternutzen und etwa Eingaben in ein Formular tippen. Allerdings kann der Browser mithilfe von Javascript den angezeigten HTML-Code ändern und so die Daten, entsprechend aufbereitet, anzeigen. Wenn Sie sich darunter immer noch nichts vorstellen können, dann sollten Sie jetzt Google Maps aufrufen und mit der Karte spielen. Jedes neu angezeigte Kartenelement wird im Hintergrund per XMLHttpRequest nachgeladen und dann mithilfe von Javascript im Browser dargestellt. Als ab 2004 die XMLHttpRequest-API in den drei wichtigsten Browser-Engines (Internet Explorer, Gecko/Mozilla und Safari) zur Verfügung stand, nutzten immer mehr Webentwickler diese Möglichkeiten und die eingesetzten Technologien wurden zu einem neuen Akronym zusammengesetzt, das bald große Karriere machen sollte: AJAX.

Asynchrones Javascript

Diese Abkürzung steht für Asynchronous Javascript And XML. Asynchron bedeutet, dass die Übertragung im Hintergrund stattfindet und die Seite währenddessen wie gewohnt weiter benutzt werden kann. Javascript ist die Programmiersprache, die die Abläufe im Browser steuert, und XML* das (zumindest früher häufig) eingesetzte Datenformat für die Übertragung. Heute übernimmt oftmals das einfacher zu verarbeitende JSON diese Rolle. Das daraus abgeleitete Akronym AJAJ konnte sich aber bislang nicht durchsetzen.

> Ajax bringt vor allem eines: Geschwindigkeit!

Welchen Vorteil bietet nun der Einsatz von Ajax? Das lässt sich auf einen Punkt bringen: Geschwindigkeit. Mit Ajax müssen immer nur die Daten übertragen werden, die im Moment gebraucht werden. Verschiebt der Nutzer in Google Maps die Karte ein kleines Stück, so werden nur die Kartenbestandteile vom Browser angefordert, die nun neu zu sehen sind. Alles andere, etwa

* siehe Glossar Seite 96-98

DER AUTOR

Stefan Fischerländer arbeitet seit zehn Jahren als SEO-Consultant und ist Mitinhaber der SEO-Agentur Gipfelstolz in Passau.

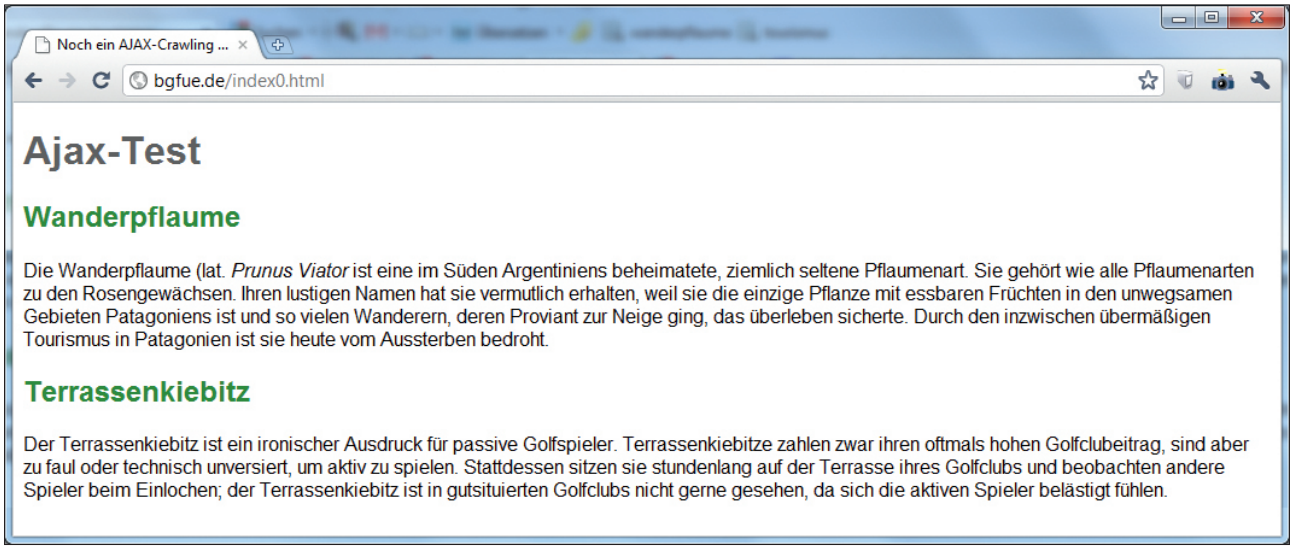

Abb. 1: Ausgangssituation: Standard-HTML (index0.html)

der ganze HTML-Rahmen inklusive eingebetteter Grafiken, muss dank Ajax nicht neu übertragen werden. Zudem bleiben während der Übertragung die bereits vorhandenen Kartenelemente sichtbar und der Nutzer kann dort wie gewohnt weiterarbeiten.

Allerdings wird dieser Vorteil durch ein großes Problem erkauft. Suchmaschinen wissen mit Ajax-Seiten nicht viel anzufangen. Schließlich wird ein – oftmals entscheidender – Teil der angezeigten Inhalte erst nach einer Nutzeraktion vom Webserver geladen; somit können Google und Co. nur die Inhalte indexieren, die beim ersten Aufruf bereits dargestellt werden. Zudem lassen sich Ajax-Seiten nicht richtig bookmarken, da der konkrete Zustand der Seite nicht in der URL kodiert ist. Was das heißt, können Sie wieder an Google Maps recht gut erkennen: Wählen Sie einen beliebigen Kartenausschnitt und bookmarken Sie diesen. Wenn Sie nun diesen Bookmark wieder aufrufen, erhalten Sie die Google-Maps-Startseite, Ihr mühsam eingestellter Ausschnitt ist verloren.

Der Zustand einer Ajax-Seite kann aber sehr wohl in der URL mitgeführt werden, ohne eine neue Seite aufzurufen. Der Trick dazu liegt in der Verwendung von URL-Fragmenten. Das sind jene Teile am Ende einer Internetadresse, die mit einem Doppelkreuz beginnen und ursprünglich einen Anker innerhalb der aktuellen Seite angeben, zu dem der Browser dann automatisch scrollt: `http:/www.example.com/langeseite.html#ziemlich_unten`

URL-Fragmente werden entsprechend der Definition von URLs im W3C-Dokument RFC 3986 (*http://einfach.st/ieft*) bei der Auflösung von URL nicht berücksichtigt. Einfacher formuliert heißt das, dass ein URL-Fragment niemals an den Webserver gesendet, sondern nur innerhalb des Browsers zur Anzeige des bereits geladenen Inhalts eingesetzt wird.

Für die Navigation innerhalb einer Ajax-Anwendung lässt sich das nun wunderbar ausnutzen. Der Zustand der Anwendung wird einfach in Parametern gespeichert, die nach dem Doppelkreuz notiert sind. Die Javascript-Funktionen der Seite können mittels `location.hash` auf das jeweils gültige Fragement, somit also auf die Parameter zugreifen, während der Browser selber keine Aktion ausführt. Genau mit dieser Methode sind seit einiger Zeit die neuen URLs von twitter.com gebaut. So ist unter `http://twitter.com/#!/gip-felstolz` nun das Twitterprofil meiner Agentur zu erreichen. Die vom Webserver abgerufene URL lautet `http://twitter.com/`, und nur das Fragment `#!/gipfelstolz` sagt dem Server, welches Profil er darstellen soll.

Crash-Kurs

Damit haben wir das Bookmarking-Problem zwar gelöst, doch für Suchmaschinen sind die Unterseiten immer noch unsichtbar. Für die weitere Behandlung dieses Themas ist nun etwas Technik unabdingbar, weshalb wir versuchen wollen, eine kleine Anwendung (nun ja, eigentlich nur eine einzelne HTML-Seite) Schritt für Schritt auf Ajax umzustellen.

In der Ausgangssituation haben wir eine einzelne Webseite, die zwei Miniartikel untereinander dargestellt enthält. Diese Artikel wollen wir künftig erst beim Klick auf einen entsprechenden Link vom Webserver per XMLHttpRequest nachladen. (Natürlich würde diese zwergenhafte Beispielseite auch ohne Ajax extrem schnell geladen – aber stellen Sie sich vor, Google Maps müsste alle Kartenausschnitte in allen Vergrößerungsstufen beim Aufruf von *maps.google.de* auf einmal herunterladen. Daran wird hoffentlich deutlich, warum wir diese künstlich wirkende Ajaxifizierung vornehmen.)

Die einzelnen Schritte unserer Umstellung stehen als Dateien zum Download unter *http://einfach.st/ajaxscript* zur Verfügung, sodass Sie die Details jeweils genau nachvollziehen können. Zusätzlich finden Sie die vorgestellten Beispiele online unter der Webadresse

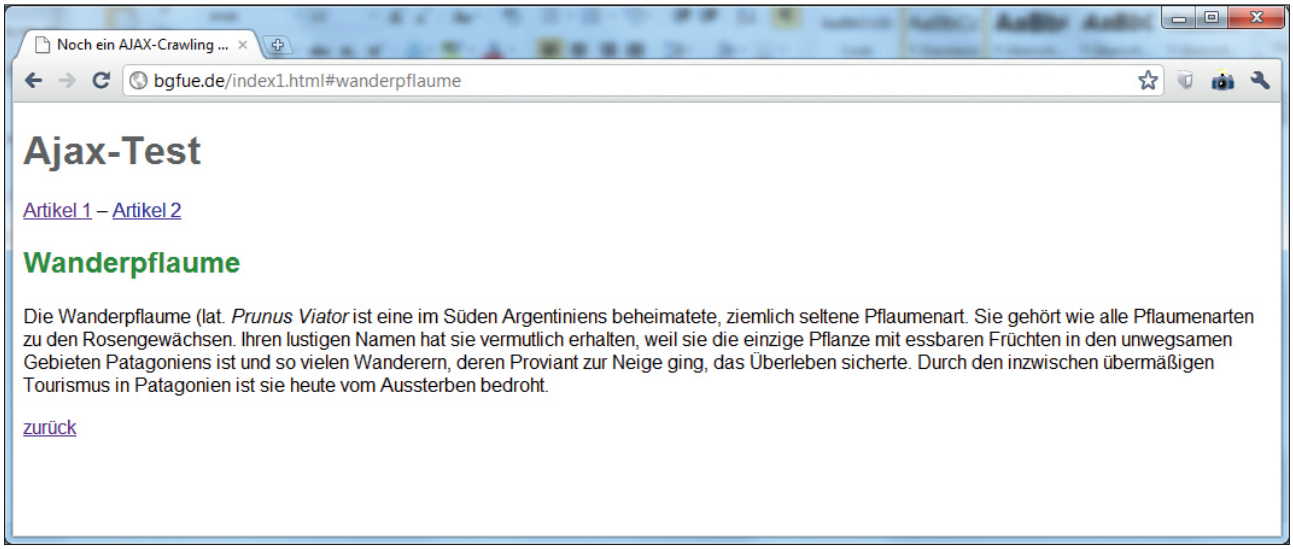

Abb. 2: Zwischenschritt: Seite mit Ajax (index1.html)

bgfue.de. In der Praxis wird heute jeder ernsthafte Webentwickler ein Ajax-Framework wie z. B. jQuery einsetzen. Doch so handlich diese Frameworks für denjenigen sind, der damit umzugehen weiß, so sehr verstellen sie auch die Sicht auf das, was sich im Hintergrund zwischen Browser und Server abspielt.

Ich möchte deshalb in unserem Beispiel kein mächtiges Framework einsetzen, sondern auf eine minimale Javascript-Library zurückgreifen, in der ich die wichtigsten Funktionen gekapselt habe. Letztlich stellt diese nano-ajax-lib.js genannte Bibliothek nur eine einzige Funktion zur Verfügung: `ajax(url, callback)`. Der Funktion `ajax()` wird die im Hintergrund aufzurufende URL übergeben sowie der Name einer Callback-Funktion, die während der Übertragung mehrmals aufgerufen wird und im Browser die nötigen Aktionen (z. B. die Darstellung der erhaltenen Daten) übernimmt. Der im Rahmen dieses Artikels gezeigte Code einschließlich der kleinen Javascript-Bibliothek ist nicht für den produktiven Einsatz gedacht, sondern soll ausschließlich die Vorgänge beim Ajax-Crawling verdeutlichen helfen.

Zunächst löschen wir also die beiden Mustertexte über die „Wanderpflaume“ und den „Terrassenkiebitz“ aus der HTML-Datei und speichern sie jeweils in eigenen Dateien ab, die wir `wanderpflaume.inc.html` sowie `terrassenkiebitz.inc.html` nennen. In die HTML-Datei integrieren wir die bereits erwähnte Mini-Ajax-Bibliothek:

```
<script type="text/javascript" src="nano-ajax-lib.js"></script>
```

Damit sind die Vorarbeiten erledigt, nun kommt die eigentliche Ajax-Funktionalität. Dazu setzen wir einen neuen -Bereich in den HTML-Code und geben diesem eine eindeutige ID, in unserem Beispiel:

```
<div id="ajax_content">
```

Innerhalb dieses Bereichs wird unsere Callback-Funktion die jeweiligen Artikelinhalte darstellen. Damit der Nutzer aber die Artikel anfordern kann, setzen wir noch zwei Links, die jeweils einen Artikel via Ajax anfordern:

```
<a href="#wanderpflaume" onclick="ajax('wanderpflaume.inc.html', callback);">
<a href="#terrassenkiebitz" onclick="ajax('terrassenkiebitz.inc.html', callback);">
```

Jetzt fehlt uns nur noch die Callback-Funktion. Die XMLHttpRequest-API ruft diese Callback-Funktion immer dann auf, wenn sich der Übertragungszustand ändert. Für uns ist nur der Zustand 4 interessant, der das Ende der Kommunikation anzeigt. Unsere Callback-Funktion muss also nur den Zustand („readyState“) des XMLHttpRequest-Objekts überprüfen. Sobald die Übertragung komplett ist, wird der Inhalt der vom Webserver per Ajax abgerufenen Daten mit Javascript in das vorher beschriebene HTML-Element mit der ID *ajax_content* eingefügt.

```
function callback(xhr) {
    if (xhr.readyState == 4) {
     document.getElementById("ajax_content").innerHTML = xhr.responseText;
    }
}
```

Das war’s schon. Unsere Seite lädt nun per Ajax den jeweils angeklickten Artikel nach, wovon der Nutzer nichts mitbekommt, da die Übertragung komplett im Hintergrund abläuft. Nur wer auf die Adresszeile schaut, sieht dort eine kleine Änderung. Statt `index1.html` steht dort nun `index1.html#wanderpflaume`. Da sich in der URL dabei aber

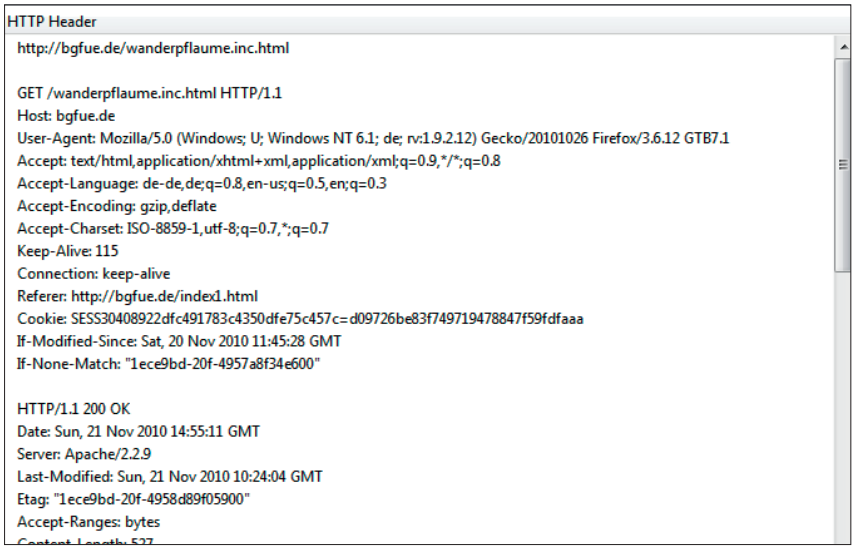

Abb. 3: Die Ausgabe der FireFox Extension *Live HTTP Headers*

nur das Fragment ändert, wird – entsprechend der oben genannten URL-Definition – kein neuer Request ausgelöst. Um zu überprüfen, ob denn im Hintergrund tatsächlich eine Datenübertragung stattgefunden hat, können wir das Firefox-Plugin *Live HTTP Headers* nutzen. Dies zeigt beim Klick auf den Artikellink tatsächlich einen Aufruf der Datei `wanderpflaume.inc.html` an.

Die Beispielanwendung nutzt nun Ajax, aber Google kann mit der jetzigen Version nichts anfangen. Ebenso funktionieren die im Browser angezeigten URLs nicht. Rufen Sie dazu die Datei `index1.html` auf und fügen Sie das Fragment `#wanderpflaume` von Hand hinzu. Es passiert: nichts! Das sollte uns nicht überraschen, da eine Änderung im Fragment keinerlei Aktionen des Browsers auslöst. Dazu müssen wir schon, wie weiter oben angedeutet, mit der Javascript-Eigenschaft `location.hash` nachhelfen.

Googles Erkennungszeichen

Um diese Ajax-Anwendung in den Google-Index zu bekommen, müssen wir sie noch weiter umbauen. Allerdings verlassen wir dazu in gewisser Weise die offizielle W3-Basis. Die nun vorgestellte Methode, Ajax crawlbar zu machen, basiert auf einer von Google ersonnenen Vorgehensweise, deren genaues Funktionieren Google unter (*http://einfach.st/goo5*) dokumentiert hat. Die eingesetzten Techniken sind zwar allesamt W3-kompatibel, allerdings erhält die URL eine besondere Formatierung, die zumindest derzeit nur Google versteht – für alle anderen Suchmaschinen wie Bing oder Yahoo sind die Ajax-Inhalte weiterhin unsichtbar.

Das Problem des Ajax-Crawling ist ja, dass der Inhalt einer Seite über die im Fragment enthaltenen Parameter gesteuert wird, das Fragment aber laut Definition nicht zwischen Webserver und Webclient (Browser, Suchmaschinenrobot) ausgetauscht werden soll. Um also W3-konform zu bleiben, muss das Fragment (und damit die Parameter) anders übergeben werden, nämlich als gewöhnliche GET-Parameter mit einem speziellen Parameternamen. Dieser Parametername lautet `_escaped_fragment_`. Aus einer Ajax-URL wie index1.html#wanderpflaume wird damit also: `index1.html?_escaped_fragment_=wanderpflaume`.

Eine Ajax-Website, die von Google gecrawlt werden will, muss also in der Lage sein, für URLs mit dem `_escaped_fragment_`-Parameter den gleichen Inhalt auszuliefern, wie er dem Nutzer in seinem Ajax-fähigen Browser angezeigt wird. Diesen Inhalt bezeichnet Google als HTML-Snapshot.

Damit wird klar, dass eine Webanwendung speziell auf das Ajax-Crawling vorbereitet werden muss, schließlich kann eine normale Site nichts mit diesen `_escaped_fragment_`-URLs anfangen. Wie weiß nun aber Google, wann eine Ajax-Anwendung dieses googlespezifische Crawlingverhalten beherrscht? Dazu hat sich Google ein Erkennungszeichen ausgedacht: Sobald in einem URL-Fragment unmittelbar nach dem Doppelkreuz ein Ausrufezeichen kommt, signalisiert der Webserver dem Googlebot: Hallo, ich bin Ajax-crawlbar!

Jetzt kommen aber plötzlich für eine Seite zwei URLs ins Spiel, die schönen Ajax-URLs (in unserem Beispiel wäre das etwa `index1.html#!wanderpflaume`) und die daraus abgeleiteten hässlichen URLs, die der Googlebot zum Crawling benutzt: `index1.html?_escaped_fragment_=wanderpflaume`. Trifft der Googlebot beim Crawling nun auf URLs, die ein `#!`-Fragment enthalten, ersetzt er `#!` mit `?_escaped_fragment_=` und fordert dann diese URL an. Aber keine Angst, im Google-Index landen nur die schönen Ajax-URLs, die Adressen mit dem Fragezeichen bleiben für den Nutzer unsichtbar. (Ich benutze hier die Bezeichnungen *schöne* und *hässliche* URL, so wie das Google in der englischen Dokumentation macht. Die in der deutschen Version benutzten Ausdrücke *ordentliche* und *falsche* URL halte ich für missverständlich.)

Somit wissen wir nun, wie Ajax-Crawling funktioniert. Die nötigen Schritte kurz zusammengefasst:

1. Wir ändern unsere Ajax-URLs von `#` in `#!`. Damit sagt die Website Google, dass sie das Ajax-Crawling-Protokoll versteht.
2. Wir programmieren die Website nun so, dass alle URLs die korrekten Inhalte (HTML-Snapshots) wiedergeben, wenn man `#!` durch `?_escaped_fragment_=` ersetzt.
3. Wir sorgen dafür, dass ein direkter Aufruf einer schönen Ajax-URL (also mit `#!`) den korrekten Inhalt anzeigt.

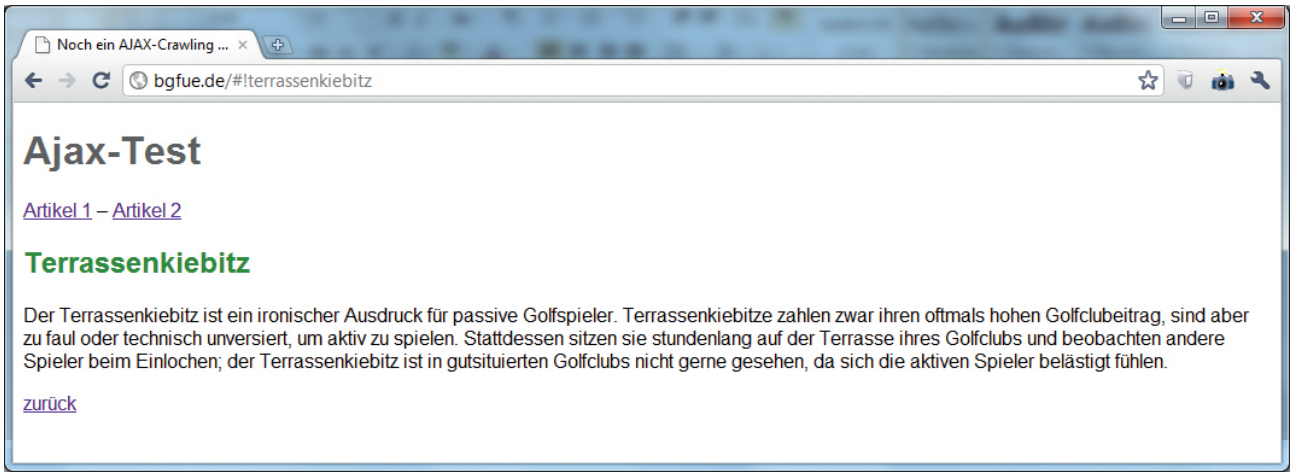

Abb. 4: Endzustand: Seite mit Ajax, von Google crawlbar (index.php)

Natürlich kommen wir beim Schritt 2 nur mit purem HTML nicht weiter. Unser Server muss in der Lage sein, auf den Parameter `_escaped_fragment_` zu antworten. Deshalb bauen wir unser Beispiel nun so um, dass es mittels PHP* in der Lage ist, das Ajax-Crawling-Protokoll zu verstehen.

Beginnen wir mit den Punkten 1 und 3, da die schnell umzusetzen sind. Wir ändern lediglich die Links zu den Ajax-Aufrufen in `href="#!wanderpflaume"` bzw. `href="#!terrassenkiebitz"` um und schon ist der erste Punkt erfüllt.

Für Punkt 3 der Liste bauen wir in den HTML-Kopf diesen Javascript-Code ein:

```
if( location.hash ) {
   if( location.hash == "#!wanderpflaume" )
      ajax('wanderpflaume.inc.html', callback);
   if( location.hash == "#!terrassenkiebitz" )
      ajax('terrassenkiebitz.inc.html', callback);
}
```

Der Code sorgt dafür, dass beim direkten Aufruf einer schönen URL der entsprechende Ajax-Aufruf ausgeführt und somit die Seite im Browser so dargestellt wird, als hätte der Nutzer den passenden Link angeklickt.

Die wichtigste Arbeit steht uns aber noch bevor: Wir müssen die HTML-Snapshots erzeugen. Das erledigen wir in diesem einfachen Beispiel gleich im Rahmen unserer einen Seite, die wir jetzt `index.php` nennen. Im HTML-Bereich `ajax_content` bringen wir einige simple PHP-Zeilen unter, die abhängig vom Wert des Parameters `_escaped_fragment_` den jeweils richtigen Inhalt ausgeben. Damit erzeugt unsere Seite nun per PHP die korrekten HTML-Snapshots für den Googlebot.

```
<div id="ajax_content"><?php
if( $_GET['_escaped_fragment_'] == 'wanderpflaume' )
   include('wanderpflaume.inc.html');
elseif( $_GET['_escaped_fragment_'] == 'terrassen-
kiebitz' )
   include('terrassenkiebitz.inc.html');
else
   include('standard.inc.html');
?></div>
```

Am besten, Sie probieren die beiden URL-Versionen gleich mal aus und überzeugen sich selbst davon, dass nun sowohl die schönen Ajax-URLs als auch die hässlichen mit dem Fragezeichen jeweils den gleichen Inhalt zurückliefern:

```
http://bgfue.de/#!wanderpflaume
http://bgfue.de/?_escaped_fragment_=wanderpflaume
```

Unsere Beispielseite sollte nun also für Google trotz Ajax crawlbar sein. Also laden wir den Googlebot ein, unsere Anwendung zu besuchen. Das geht derzeit über Twitter am schnellsten – wie schnell, zeigen die (hier gekürzt dargestellten) Logdaten des Webservers:

```
66.249.66.117 13:08:55 /robots.txt Mozilla/5.0 (com-
patible; Googlebot/2.1;
+http://www.google.com/bot.html)
66.249.66.117 13:08:55 / Mozilla/5.0 (compatible;
Googlebot/2.1; +http://www.google.com/bot.html)
66.249.66.117 13:09:24 /?_escaped_fragment_=wander-
pflaume Mozilla/5.0 (compatible; Googlebot/2.1;
+http://www.google.com/bot.html)
66.249.66.117 13:09:28 /?_escaped_fragment_=terras-
senkiebitz      Mozilla/5.0 (compatible; Google-
bot/2.1; +http://www.google.com/bot.html)
```

Kaum ist der Tweet (um 13:08 Uhr) abgesetzt, kommt auch schon der Googlebot vorbei. Und wie deutlich zu sehen ist, ruft er vereinbarungsgemäß die hässlichen URLs auf. Was aber passiert im Google-Index? Suchen Sie doch einfach nach

* siehe Glossar Seite 96-98

Abb. 5: Das Suchergebnis mit schöner Ajax-URL

wanderpflaume – Sie sollten dann in etwa das Ergebnis bekommen wie in Abb. 5 zu sehen ist.

Google hat also die hässliche URL vom Webserver abgerufen, aber den Inhalt unter der schönen Ajax-URL indexiert. Hier wird nochmals deutlich, warum wir per location.hash dafür sorgen müssen, dass ein direkter Aufruf der Ajax-URL den richtigen Inhalt anzeigt. Würden wir Schritt 3 unserer Änderungsliste nicht berücksichtigen, erhielte jeder von Google kommende Besucher nur die Startseite präsentiert. Das wäre nicht besonders schön.

Cloaking-API

Falls jemand jetzt große, leuchtende Augen bekommt und sofort an böse Tricks denkt: Ja, damit bietet Google quasi eine offizielle Cloaking*-Schnittstelle! Cloaking heißt ja, dem Googlebot etwas anderes zu zeigen als dem Browser. Und dank Ajax-Crawling nimmt der Googlebot den Inhalt der hässlichen Fragezeichen-URL und ordnet ihm die schöne Ajax-URL zu, die dann auch in den Suchergebnissen angezeigt wird. Natürlich lässt sich das für üble Manipulationen ausnutzen, weshalb Google auch deutlich darauf hinweist, dass doch die Inhalte beider URLs gefälligst übereinzustimmen haben. Ob diese Überprüfungen lediglich bei Verdacht manuell geschehen oder automatisiert, kann ich derzeit nicht beantworten. Im Rahmen meiner Tests konnte ich keinen Zugriff feststellen, der darauf schließen ließe, dass Google hier einen automatischen Inhaltsabgleich durchführen würde – was ja auch technisch nicht so trivial ist, denn sonst hätte Google ja das Ajax-Crawling erst gar nicht erfinden müssen.

Umgekehrt bergen die beiden URLs – einmal schön, einmal hässlich – eine andere Gefahr: Duplicate Content*. Sollte die Snapshot-Generierung Fehler enthalten, läuft die Anwendung für alle Nutzer ohne Probleme weiter; nur der Googlebot bekommt immer den gleichen Inhalt gezeigt. Damit ergibt sich ein gefährliches Einfallstor für unbemerkte Duplicate-Content-Erzeugung. Sie sollten deshalb die Snapshot-Generierung im Rahmen der Qualitätssicherung Ihrer Website genauso überwachen, wie Sie das mit Ihrer gewöhnlichen Webanwendung tun. Wenn alles korrekt umgesetzt ist, dürften auch nur die schönen URLs in den Suchmaschinen zu finden sein. Passen Sie aber auf, dass Sie nirgendwo versehentlich die hässlichen URLs verlinken! Denn dann werden diese URLs wie normale URLs gecrawlt und von den Suchmaschinen aufgenommen. Deshalb dürfen Sie auch in einer XML-Sitemap nur die schönen Ajax-URLs angeben, ansonsten landen beide URL-Varianten im Index.

Einen Sonderfall der URL-Behandlung gibt es noch: Die Startseite einer Webanwendung hat ja kein URL-Fragment, also lässt sich dort kein Hinweis unterbringen, dass die Seite Ajax-crawlbar ist. Für diesen Sonderfall hat sich Google einen neuen Metatag einfallen lassen:

```
<meta name="fragment" con-
tent="!">
```

Ist auf einer Seite dieser Code angegeben, weiß der Googlebot, dass diese Seite Ajax-crawlbar ist und crawlt stattdessen die URL `example.com/?_esca-ped_fragment_=` – er behandelt diese URL so, als hätte sie ein leeres Ajax-Crawling-Fragment.

Übrigens funktioniert das Ajax-Crawling offenbar nur, wenn Google der Ansicht ist, dass die gesamte Website – oder zumindest ein großer Teil davon – per Ajax realisiert ist. Zwar lässt sich dazu nichts in der Dokumentation finden, aber meine Tests zeigten, dass Google der Implementation von Ajax-Crawling auf nur einer einzelnen Unterseite einer größeren Webanwendung nicht vertraute. Trotz mehrerer direkter Links schafften es diese URLs auch nach mehreren Wochen nicht in den Index. Womöglich ist das ein kleiner Schutzmechanismus von Google, um simple Cloaking-Tricks an unauffälligen Stellen einer Website zu verhindern.

Early Adaptor: Twitter

Google hat dieses Ajax-Crawling-Protokoll bereits vor einem guten Jahr veröffentlicht, die Resonanz darauf ist aber noch recht zurückhaltend. Jedenfalls zeigt ein Blick in die Suchergebnislisten bislang nur wenige Early Adaptors dieser Technologie. Allerdings setzt mit Twitter bereits eine der bekanntesten Websites diese Methode ein, auch wenn sich das nicht unmittelbar in den Trefferlisten niederschlägt. In Google haben die Twitter-URLs nach wie vor die Form `twitter.com/gipfelstolz`. Das überrascht, denn mit einem halbwegs modernen Browser nutzt Twitter Ajax-URLs der Art `twitter.com/#!/gipfelstolz` und signalisiert somit dem Googlebot, dass der Webserver Ajax-Crawling unterstützt.

Woher stammen also die „klassi-

* siehe Glossar Seite 96-98

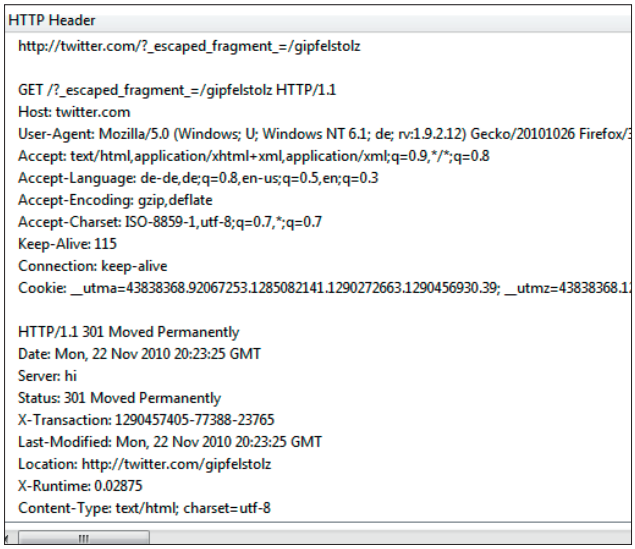

Abb. 6: HTTP Header der URL: twitter.com/?_escaped_fragment_=/gipfelstolz

schen“ URLs im Google-Index? Das lässt sich mit der bereits erwähnten Firefox-Erweiterung Live HTTP Headers leicht überprüfen. Dazu spielen wir einfach selber Googlebot und wandeln die schöne URL in eine hässliche URL um. Diese hässliche URL – im obigen Beispiel also `twitter.com/?_escaped_fragment_=/gipfelstolz` – geben wir in die Adresszeile der Firefox ein und beobachten, was passiert.

Wie wir in der Abbildung oben sehen können, beherrscht Twitter.com durchaus Ajax-Crawling, aber anstatt einen HTML-Snapshot zu generieren, antwortet der Server mit einer Weiterleitung auf die „klassischen“ URLs. Der Vorteil dieser Methode ist, dass sich der Websitebetreiber die Entwicklung eines Moduls zur Snapshot-Generierung spart und die sowieso als Fallback für alte Browser nötigen klassischen URLs dazu einsetzt.

Ajax geblitzt

Einen Einsatzzweck von Ajax habe ich bisher noch nicht erwähnt: Ajax wird heute gerne in Kombination mit Flash genutzt. So lässt sich die oben beschriebene Methode, Zustände in URL-Fragmenten zu speichern, auch einsetzen, um beliebige Zustände einer Flash-Anwendung direkt anlinken oder als Bookmark speichern zu können. Da ja auch eine Flash-Anwendung eine Fallback-Lösung für alte Browser, moderne Tablet-Computer und Suchmaschinen haben sollte, liegt der Gedanke nahe, Ajax-Crawling genau dafür zu nutzen.

Der Webserver zeigt also dem Googlebot die schönen Ajax-Links, die dieser wie nun schon mehrmals beschrieben, intern umwandelt und crawlt. Dabei wird dem Googlebot der sowieso als Fallback für Nicht-Flash-Benutzer vorhandene HTML-Inhalt präsentiert. Das hat auch bisher schon prima funktioniert, allerdings landen diese HTML-Ersatzinhalte unter ihrer eigenen URL in den Google-Suchergebnislisten. Ein von Google kommender Besucher muss also erst unschön auf die Ajax-Version weitergeleitet werden. Dieser Schritt fällt jetzt weg, denn dank Ajax-Crawling zeigt Google gleich

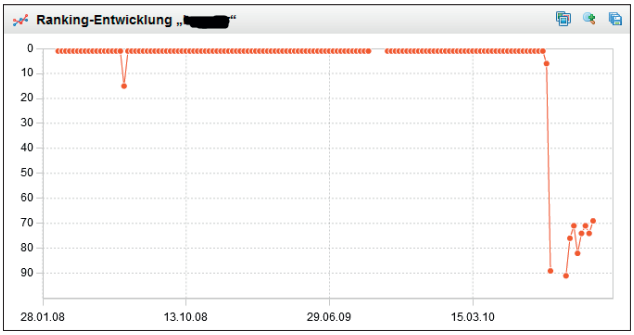

Abb. 7: Rankingentwicklung nach Umstellung auf Ajax-Crawling.

die korrekten, schönen URLs als Suchergebnis an.

Das klingt in der Theorie recht einfach. Allerdings sind Flash-Sites häufig verbunden mit anderen Techniken, die für Suchmaschinen problematisch sind. So setzen etwa Autohersteller oder Modemarken mit internationaler Ausrichtung gerne Flash auf ihren Websites ein. Um die Nutzer dann auf die richtigen Unterbereiche zu lenken, wird in einem solchen Fall entweder der User-Agent des Browsers oder die lokale Zuordnung der IP-Adresse ausgewertet. Ein derartiges Setup zu analysieren und dabei zu gewährleisten, dass der Googlebot den richtigen Weg durch die Site findet, ist nun nicht mehr trivial, denn wir müssen schöne Ajax-URLs, hässliche URLs mit _escaped_fragment_ sowie nach wie vor erreichbare Backfall-URLs mit normalem HTML-Inhalt unterscheiden. Hinzu kommt dann noch die Vervielfältigung dieser URLs für die einzelnen Märkte. Soll eine derartige Website also die drei wichtigsten deutschsprachigen Märkte Deutschland, Österreich und Schweiz ansprechen, so haben wir dreimal drei, also neun URLs mit identischem Inhalt zu verwalten.

Wie leicht sich der Googlebot im Gestrüpp dieser URLs verheddert, zeigt der Screenshot oben mit der Rankingentwicklung einer Website, die seit der Umstellung auf Ajax-Crawling schlagartig für ihren eigenen Markennamen von Position 1 auf etwa 70 abgestürzt ist. Ganz offensichtlich liegt dieser Absturz aber nicht am Ajax-Crawling, sondern daran, dass Google wegen des parallel dazu eingesetzten IP-Geotargetings die Startseite nicht mehr indexiert.
Um derartige Nachteile zu vermeiden, bietet sich eine Strategie an: Walk slowly! Es mag verführerisch sein, den großen Wurf machen zu wollen, aber es ist für Ihre Website meist besser, die Änderungen Schritt für Schritt umzusetzen. Zumindest lässt sich dann bei Problemen leicht erkennen, wo der Fehler zu suchen ist.

Fazit

Mit dem Ajax-Crawling hat Google einen eleganten Mechanismus geschaffen, um Ajax-gestützte Websites indexierbar zu machen. Dass Ajax-Crawling funktioniert, sehen wir an unserer kleinen Beispielanwendung ebenso wie am großen Vorbild twitter.com. Allerdings erkaufen wir uns diese Möglichkeit mit einer Beschränkung auf die Plattform Google und mit einer wachsenden Komplexität der Website.¶