Bilder schneller im Browser laden

Im Klartext

Christian Kirsch

Meist laden Webseiten Bilder, indem sie vom Server eine Datei anfordern. Eine Alternative kommt jedoch ohne diesen Request aus und ist deshalb in bestimmten Situationen schneller.

Seit Anbeginn des Web gibt es das *img*-Element, mit dem Webseiten Bilder vom Server anfordern und darstellen. In der Regel trägt man dazu im *src*-Attribut die URL des Bildes ein:

```
<img src="http://example.com/images/image.png">
```

Dadurch setzt der Browser einen Request ab, den der Server mit dem Inhalt der angeforderten URL beantwortet, also den Bilddaten. Bei jedem dieser Requests entsteht Overhead unter anderem durch den Aufbau der Verbindung und den Festplattenzugriff des Servers.

Eine andere Methode vermeidet diesen zusätzlichen Aufwand, indem sie die Daten direkt in das *img*-Element einbettet:

```
<img src="...">
```

Dazu dient ein URI mit *data:* als Pseudoprotokoll, gefolgt vom MIME-Typ des Bildes und *base64* als Codierung der anschließenden Bilddaten. Obwohl es sich hierbei um einen URI handelt, ist häufig von „Data-URL“ die Rede. *base64* gibt an, dass die Bilddaten nicht binär vorliegen, sondern in Zahlen, Buchstaben und zwei Sonderzeichen aus dem ASCII-Zeichensatz umgesetzt sind.

Zwar kann man mit einem Data-URI statische Bilddaten in HTML-Seiten unterbringen, analog per CSS-Regel, etwa für Hintergrundbilder. Doch das bringt auch Nachteile mit sich. So sind die Base64-codierten Daten rund ein Drittel größer als das Original, und der Browser speichert Data-URIs nicht in seinem Cache. Nützlicher ist die Technik im Zusammenhang mit Webseiten, die wechselnde Bilder anzeigen, bei denen also der Cache kaum oder gar nicht zum Tragen kommt.

Als hilfreich erweisen sich Data-URLs, will man viele *img*-Elemente auf einer Seite bereitstellen, deren Funktionieren nicht von der Sichtbarkeit der Bilder abhängt. In diesem Fall bietet es sich an, sie asynchron per Ajax zu laden oder in einem PHP-Skript bereitzustellen, sodass der Anwender nicht auf das Laden warten muss. Das ist etwa für Thumbnails sinnvoll, die erst beim Überfahren eines Bereichs mit der Maus erscheinen sollen.

Listing: Traditionelle *img*-Elemente erzeugen

```
var ids = [1532,  1806,  1946,  2250, ... ];

var base_url = 'http://localhost/~ck/ix-test/
                                        Bilder/';
window.onload = fill_table;

function fill_table() {
  var tbl = document.getElementById('tabelle');
  var trs = tbl.getElementsByTagName('tr');
  var counter = 0;
  for (var i = 0; i <= 6; i++) {
    var tr = trs[i];
    var td = tr.getElementsByTagName('td');
    for (var j = 0; j <= 3; j++) {
      var img = document.createElement('img');
      img.src = base_url + ids[counter++] + '.jpg';
      td[j].appendChild(img);
    }
  }
}
```

Die für Data-URIs benötigten Base64-codierten Bilddaten können aus einer Datenbank oder aus einer Datei kommen. Für das Umwandeln in den Klartext eignen sich Datenbankfunktionen (etwa *TO_BASE64* in MySQL oder *encode* in PostgreSQL) oder Betriebssystemtools wie *base64* oder Routinen der eingesetzten Skriptsprache, etwa das Perl-Modul *MIME::Base64* und PHPs *base64_encode*.

PHP-Code, der eine Bilddatei als Data-URI einbettet, könnte etwa so aussehen:

```
$bild_datei = 'bild.jpg';
$bild_base64 = base64_encode ↵
                    (file_get_contents($bild));
// URI-Format erzeugen
$uri = 'data: '. mime_content_type($bild) .
  ';base64,' . $bild_base64;
echo '<img src="', $src, '">';
```

Zwar liest dieses Fragment das Bild aus dem Dateisystem, was der Webserver genauso machen würde. Doch entfällt hier der Overhead für die HTTP-Anfragen und -Antworten des klassischen *<img src="http://pfad.zum.bild">*. Mit Perl lassen sich die Daten für eine Ajax-Anwendung aus einer MySQL-Datenbank lesen und im JSON-Format etwa so ausgeben:

```
my $query = 'SELECT bild_id as id, breite, ↵
                                        hoehe,'
    . ' to_base64(minibild) as thumbnail  '
    .' FROM bilder';
my $sth = $dbh->prepare($query);
$sth->execute();
my $result = $sth->fetchall_hashref('id');
print encode_json $result;
```

Der korrespondierende Ajax-Handler könnte die bei ihm eintreffenden JSON-Daten *bild* beispielsweise folgendermaßen weiterverarbeiten:

```
img = document.createElement("img");
img.src = ""
        + bild.thumbnail;
```

Geht es darum, bei einem Mouse-over-Event jeweils nur ein Bild anzuzeigen, bietet sich ein einzelnes *img*-Element an, dessen *src*-Attribut man im Event-Handler mit den aktuellen Daten versieht.

Ein Minibeispiel illustriert die Zeitunterschiede. Es zeigt 28 Bilder in einer Tabelle aus sieben Zeilen und vier Spalten an, wozu es das klassische Verfahren benutzt (siehe Listing) und zum Vergleich Data-URIs, deren Daten ein Ajax-Aufruf aus einer Datenbank holt. Im ersten Fall übertragen 30 HTTP-Requests 119 KByte, im zweiten holen nur drei 159 KByte. Die Anzahl der Requests bleibt in jedem Fall gleich, da im herkömmlichen Verfahren der Browser für jedes Bild nachfragen muss, ob es sich auf dem Server geändert hat oder er die Daten aus seinem Cache holen kann.

Wie die Tabelle zeigt, beeinflusst die Wahl des Browsers die Ladezeiten deutlich stärker als die Entscheidung für oder gegen Data-URIs. (tiw)

Ladezeiten für 28 Bilder

Browser	Chrome 51	Firefox 47
klassisch (ohne Cache)	325 ms	552 ms
klassisch (mit Cache)	227 ms	311 ms
Data-URI (ohne Cache)	185 ms	358 ms
Data-URI (mit Cache)	150 ms	300 ms

Angegeben ist die Zeit für das Ausführen des Onload-Handlers (Firefox) beziehungsweise die bis zum letzten UpdateLayerTree (Chrome).

Christian Kirsch

war *iX*-Redakteur und ist freier IT-Journalist.