Firefox: Javascript-Reset und HTML-Reset können unterschiedliche Ergebnisse liefern

Das Problem tauchte im Zusammenhang mit der Reset-Funktion

```
<%= javascript_tag "form.reset()"%>
<%= javascript_tag "$('formID').reset()"%>
<%= javascript_tag "document.reset()"%>
```

auf: Wurden Textfelder geändert und mit dem Javascript-Reset bereinigt, so tauchte deren Wert nach einem Blättervorgang und sogar nach der Speicherung wie aus dem Nichts wieder auf, obwohl – nach der Speicherung – sogar der Quelltext die richtigen Inhalte angab. Wurde die Bereinigung dagegen über das HTML-Reset

```
<input type=reset>
```

ausgeführt, verhielt sich Firefox korrekt in Bezug auf die Textfelder.

Hinsichtlich der Ajax-kontrollierten Felder wie den Listen oder Drag & Drop verhielt es sich dagegen umgekehrt.

4.1.11.2 Verbleibende Grenzen gegenüber internen GUIs

Was deshalb trotz Ajax und allem Rails-Komfort gerade für Programmierer, die bisher nur mit internen Oberflächen arbeiteten, bedauernswert bleiben dürfte, ist die Tatsache, dass die grundsätzliche Zerlegung von Programm und Darstellung einem völlig freien Datenaustausch zwischen beiden natürlich im Wege steht. Während der Rails-Precompiler große Freiheiten hinsichtlich der Übergabe von Ruby-Variablen an den Browser gewährt, ist bei der Entgegennahme von HTML-Werten selbstverständlich eine entsprechende Transaktion des Browsers erforderlich. Dabei ist immer zu berücksichtigen, dass das Ganze über Internet und möglicherweise sogar über langsame Leitungen erfolgen könnte, sodass diese Vorgänge soweit als möglich „komprimiert“ erfolgen sollten – das war schließlich mit ein Grund für den Siegeszug der Ajax-Technologie, die nur noch „Teile“ des darzustellenden Ganzen bearbeitet. Das erlaubt es ja auch, „Leitung zu sparen“ bei dem erhöhten Verkehrsaufkommen, das steigender Bedienungskomfort so mit sich bringt.

Warum überhaupt Werte des Browsers interessant sein können? Beispielsweise für die Frage, welches Listenelement von den Anwendern ausgewählt wurde

oder auch, auf welcher Blätter-Instanz die Benutzer gerade Halt gemacht haben. Schließlich erlaubt ein Browser häufig, dass derselbe Vorgang beliebig oft durchgeführt wird: Die Anwender können beispielsweise über Blättern oder Links beliebig oft ein und denselben Satz zur Verarbeitung auswählen – und dann zwischen den einzelnen aufgeblätterten Datensätzen hin- und herhüpfen. Das verarbeitende „ferne" Programm kann hier unmöglich die Kontrolle darüber behalten, welcher Satz den nun tatsächlich derjenige ist, den die Anwender gerade angesprungen haben. Es muss also schon prüfen, was genau die Benutzer gerade im Griff haben – und dafür braucht es ein HTML-Element, das mit den bearbeiteten Sätzen eindeutig korreliert, mit dem also gewährleistet werden kann, dass ein Satz auch eindeutig identifiziert werden kann.

Deshalb ist im Zusammenhang mit Ajax Javascript eigentlich immer die erste Wahl für eine weitergehende Bearbeitung solcher Browser-Instanzen, da es sich „innerhalb" der betrachteten Instanzen bewegt, während die „entfernten" Ruby-Programme prinzipiell das Problem haben, die jeweilige Instanz (oder zumindest deren Werte) korrekt bestimmen zu müssen. Dass dies nicht immer ganz leicht ist, zeigt der Fall, dass die Anzeige nach dem Droppen sich als nicht sehr dauerhaft beim Blättern erweist. So ist dieser Mangel sogar bei der Demonstration der Funktion des script.aculo.us-Frameworks (noch) vorzufinden.

> Das Wort zum Abschluss:
> script.aculo.us weist diesen Fehler nicht mehr auf – Open Source funktioniert, nicht wahr?

Es drohen freilich Missverständnisse und Dateninkonsistenz, wenn die Anwender etwas anderes als das Programm sehen. Dabei ist es unbeachtlich, dass einerseits das Blättern im Browser von Javascript (bisher) noch nicht abfragbar ist (es gibt aktuell noch kein Event „OnBack" oder „OnForward") und andererseits die von Ajax bearbeitete Variable sich einfach so „ändern" kann aufgrund eines solchen Vorganges – das Programm hat irgendwie Klarheit zu schaffen, auch wenn sich ein veränderter Ajax-Wert nicht wirklich per Javascript festnageln lässt. Während die vom Browser eingeladenen Seiten in seiner Historie gespeichert werden, sodass sie per Seiten-Quelltext angesehen und per Blättern angesprungen werden können, sind die „Zwischenzustände",

die nur über Javascript erzeugt werden, nur der Javascript-Engine selbst bekannt: Sie kennt die unterschiedlichen Variablenwerte auf den diversen Zwischenzuständen genau, bietet jedoch (aktuell) keine Möglichkeit, die einzelnen Zwischenzustände gezielt anzusprechen. Damit ist auch nicht in Erfahrung zu bringen, ob und wie ein Element durch Javascript abgeändert wurde. Zwar erzeugt der Browser beim Blättern wieder einen wohldefinierten, „greifbaren" Zustand, es ist jedoch der Ausgangszustand der letzten vollständigen HTML-Ausgabe, nicht der zeitlich jüngste durch Ajax manipulierte Zwischenzustand.

Dass Javascript die Kontrolle nicht verliert und die diversen Verarbeitungsschritte genau auseinander halten kann, lässt sich natürlich ausnutzen. Werden die maßgeblichen Werte im HTML-Code des aktuellen Blattes gespeichert und später per Javascript abgefragt, besteht wenigstens die Sicherheit, die Daten genauso erfahren zu können, wie sie auch der Browser den Anwendern anzeigt. Hierbei ist freilich zu beachten, dass der Zugriff von Rails, mit dem HTML-Werte manipuliert werden, ausschließlich über die „render"-Funktionalität erfolgt. Wird also, wie in Schritt 4, ein Update des Ajax-Vorgangs auf ein anderes, größeres DOM-Element ausgeführt als „gerendert" wird, so kann per Rails trotzdem nur auf das letztere Element zugegriffen werden.

Cachen im Zusammenhang mit dem Browser ist mit Vorsicht zu genießen

Merke: „Cachen", hier im Sinne von programmseitigem Vermerken von Feldinhalten, ist also auch in diesem Zusammenhang eine Sache, die in jedem einzelnen Fall auf Korrektheit überprüft werden muss – sicher der Grund, warum Rails seine mit dem Browser kommunizierenden Klassen ständig neu initialisiert.

Anwendungsbeispiel: Der dritte Schritt und der vierte Schritt

4.1.11.3 Wie es nicht gemacht werden soll

Da Ajax eine elementgenaue Bearbeitung ermöglicht, verlangt es auch eine elementgenaue Präzision in der Verarbeitung. Partials sind hier ein sehr gutes Mittel, diese Präzision zu erreichen, doch trotz ihrer Einfachheit erlauben auch sie Fehler.