

## 11 History-Management

In AJAX-Applikationen tritt für den Benutzer häufig das Problem auf, dass der *Zurück*-Button des Browsers nicht die letzte über AJAX aufgerufene Aktion rückgängig macht, sondern die letzte vollständig geladene Seite, die auch im Browsercache gespeichert ist, aufruft. Dieses Verhalten ist für den Benutzer nicht transparent. Andererseits soll nicht immer ein AJAX-Aufruf über den *Zurück*-Button rückgängig gemacht werden. Ein Beispiel dafür ist ein Eingabefeld, das über die Verwendung von AJAX eine Liste von Auswahlmöglichkeiten anzeigt und für diesen AJAX-Aufruf soll der *Zurück*-Button keine Auswirkung haben.

Was also tatsächlich für AJAX-Anwendungen benötigt wird, ist eine aktive Verwaltung der Historie der aufgerufenen Seiten. Das Google Web Toolkit hat dafür die `History`-Klasse eingeführt, die mit der Liste der aufgerufenen Seiten im Browser interagiert. Jedes Element auf dem Stapel der GWT-History repräsentiert damit eine einzelne Zeichenkette, die als Token benutzt wird. Um Änderungen am aktuellen History-Element durch den Benutzer abzufangen, gibt es das `HistoryListener`-Interface.

*History und  
HistoryListener*

Sehen wir uns zunächst die einzelnen Methoden der `History`-Klasse an:

**Tab. 11-1**  
Methoden der  
History-Klasse

Methoden	Beschreibung
addHistoryListener(HistoryListener)	fügt einen HistoryListener hinzu, der über Änderungen der Browserhistorie informiert wird
back()	entspricht dem Betätigen des <i>Back</i> -Buttons des Browsers
forward()	entspricht dem Betätigen des <i>Forward</i> -Buttons des Browsers
getToken()	ermittelt den aktuellen History-Eintrag
newItem(String)	fügt einen neuen Eintrag hinzu
removeHistoryListener(HistoryListener)	entfernt den HistoryListener

Die Schnittstelle HistoryListener enthält nur eine Methode:

**Tab. 11-2**  
Methoden der  
HistoryListener-  
Schnittstelle

Methoden	Beschreibung
onHistoryChanged(String historyToken)	wird aufgerufen, wenn die Buttons <i>Back</i> oder <i>Forward</i> betätigt werden oder die entsprechenden Methoden der History-Klasse aufgerufen werden

Sehen wir uns die Beispielapplikation aus der Dokumentation des GWT an:

**Listing 11.1**  
Beispielapplikation für  
die Historyklasse

```

1 public class HistoryExample implements EntryPoint,
    HistoryListener {
2
3     private Label lbl = new Label();
4
5     public void onModuleLoad() {
6         Hyperlink link0 = new Hyperlink("link to foo", "foo");
7         Hyperlink link1 = new Hyperlink("link to bar", "bar");
8         Hyperlink link2 = new Hyperlink("link to baz", "baz");
9
10        String initToken = History.getToken();
11        if (initToken.length() == 0){
12            initToken = "baz";
13        }

```

```

14
15     onHistoryChanged (initToken);
16
17     VerticalPanel panel = new VerticalPanel ();
18     panel.add(lbl);
19     panel.add(link0);
20     panel.add(link1);
21     panel.add(link2);
22     RootPanel.get().add(panel);
23
24     History.addHistoryListener(this);
25 }
26
27 public void onHistoryChanged(String historyToken) {
28     lbl.setText("The current history token is: " + historyToken)
29     ;
30 }

```

Die Methode `onModuleLoad` erstellt zunächst drei Hyperlinks. Danach wird in die Variable `initToken` der aktuelle Token der History geladen. Ist noch keiner vorhanden, da die Applikation gerade gestartet wurde, wird der Ausgangswert auf »baz« gesetzt, was dem Hyperlink `link2` entspricht. Als Nächstes wird die Funktion `onHistoryChanged` aufgerufen. Diese gibt im Label den aktuellen Token aus. Anschließend wird das Panel erzeugt und der `HistoryListener` zur eben erstellten Klasse hinzugefügt. Wird nun die History geändert, wird die Methode `onHistoryChanged` mit dem aktuellen Token aufgerufen und dadurch das Label verändert. Der Back- und Forward-Button erfüllt nun erwartungsgemäß seine Funktion und gibt die Historie in der richtigen Reihenfolge wieder.

Wichtig dabei ist allerdings, dass in der Hosted Website der *Hidden Frame* eingebaut wird, der die Aktionen steuert – ohne ihn funktioniert das History-Management nicht. Beim Aufbau einer Anwendung über die GWT-Bordmittel ist er aber ohnehin in der Hosted Website verankert:

```

1 <iframe id="__gwt_historyFrame" style="width:0; height:0; border
   :0"> </iframe>

```

Soviel zur Theorie der Verwaltung der Browser-Buttons – jetzt zur Umsetzung dieses Verhaltens in unserem Beispielprojekt.

### **Listing 11.2**

*Der Hidden Frame für  
das Hosted File*

## 11.1 History-Management im Beispiel

Im Use Case »Register« kann anhand des verwendeten Widgets `TabPanel` relativ einfach gezeigt werden, wie das History-Management mit Aktionen der *Back*- und *Forward*-Button umzugehen weiß.

Hier macht ein History-Management auch wirklich Sinn: Der Benutzer ist es ja gewohnt, die letzte Aktion mit dem *Back*-Button im Browser rückgängig zu machen. Hat er also als letzte Aktion ein anderes Tab gewählt, sollte er mit den Zurück-Button wieder auf das ursprüngliche Tab springen. Wir werden diese Funktionalität beispielhaft implementieren.

Zu Beginn registrieren wir die benötigten Listener – wir fügen der `ProfilUI`-Klasse einen `TabListener` und einen `HistoryListener` hinzu.

Unsere Implementierung des `TabListeners` benötigt zwei Methoden. Die Implementierung der `onTabSelected`-Methode sorgt dafür, dass der Index des ausgewählten Tabs in der History gespeichert wird. Dies wird durch den Aufruf der Funktion `newItem(String)` bewerkstelligt. Die `onBeforeTabSelected()`-Methode muss `true` zurückzugeben. Dieser Rückgabewert legt fest, dass die Auswahl des angesteuerten Tabs erlaubt ist – in unserem Use Case darf zu jedem Zeitpunkt von jedem Tab aus jedes andere Tab ausgewählt werden.

Die Methoden für den `TabListener` sehen wie folgt aus:

### Listing 11.3

Tablistener-Funktionen

```

1
2  public void onTabSelected(SourcesTabEvents sender, int index
3      ) {
4      History.newItem(String.valueOf(index));
5  }
6  public boolean onBeforeTabSelected(SourcesTabEvents sender,
7      int index) {
8      return true;
9  }
```

Zu beachten ist hier die Umwandlung des Index von einer Ganzzahl in eine Zeichenkette. Neue Einträge können in die History nur als Zeichenkette gespeichert werden.

Im nächsten Schritt wird der `HistoryListener` implementiert. Er sorgt dafür, dass bei einer Betätigung des *Back*- oder *Forward*-Buttons das richtige Tab angezeigt wird. Die `History`-Klasse übergibt dabei automatisch den logisch nächsten oder vorherigen Wert aus der internen Liste.

```
1     public void onHistoryChanged (String item){
2         Integer i = new Integer(item);
3         this.tb.selectTab(i.intValue());
4     }
```

**Listing 11.4**  
*HistoryListener-  
Funktionen*

Hier wird der umgekehrte Weg eingeschlagen – der Tab-Index wird aus der History ausgelesen, und der entsprechende Eintrag am Panel wird selektiert. Die `selectTab`-Funktion des `TabPanels` benötigt eine Ganzzahl – daher muss die Zeichenkette, die aus der History gelesen wird, in eine Ganzzahl umgewandelt werden.

Schließlich müssen beim Erstellen des Panels selbst einige Änderungen vorgenommen werden. Das `TabPanel` wird ein Instanzobjekt der Klasse und erhält einen Zeiger des `TabListeners` auf sich selbst. Der History wird ein neuer Listener hinzugefügt, der auf das `ProfilUI`-Objekt selbst zeigt. Im letzten Schritt muss der History mitgeteilt werden, dass das erste Element des Panels tatsächlich das erste angezeigte Element ist.

```
1     this.tb.addTabListener(this);
2     History.newItem(new String("0"));
3     History.addHistoryListener(this);
```

**Listing 11.5**  
*Änderungen in der  
ProfilUI*

Nun kann der Benutzer nach dem Start des Programms im Registrierungsvorgang mit den Back- und Forward-Buttons die einzelnen vorher ausgewählten Tabs durchwählen.