Kapitel 15
Navigation, Datentransfer

Inhaltsverzeichnis

Navigieren zwischen Forms

1. Lernziele

In diesem Kapitel sollen folgende Lernziele erreicht werden:

- ✓ Die verschiedenen Verfahren aufführen, mit denen wir zwischen den Seiten einer Web-Anwendung navigieren können.
- ✓ Wissen, welche Navigationstechnik für die Anforderung am geeignetsten ist.
- ✓ Den Anzeigestatus einer Web Forms-Seite zwischen Anforderungen aufbewahren.
- ✓ Eine Seite in einem neuen Browser-Fenster anzeigen und dieses Fenster vom Servercode aus steuern.

2. Navigieren in HTML

In einer Web Forms-Anwendung werden *Hyperlinks* und die *Navigationsmethoden* verwendet, um mehrere Web Forms-Seiten zu einer zusammenhängenden Anwendung zusammenzufügen. ASP.NET stellt unterschiedliche Verfahren mit unterschiedlichen Zielen zur Verfügung. Diese sind:

Navigationsmethode	Verwenden, um ...
`Hyperlink-Steuerelement`	zu einer anderen Seite zu navigieren
`Response.Redirect`	vom Code aus zu einer anderen Seite zu navigieren. Dies entspricht dem Anklicken eines Hyperlinks. Funktioniert sowohl für `.aspx`- wie für `.html`-Seiten.
`Server.Transfer`	eine aktuelle Web Forms-Seite zu verfassen und eine andere Seite auszuführen. Diese Methode funktioniert nur, wenn die Zielseite eine Web Forms-Seite (`.aspx`) ist.
`Server.Execute`	eine andere Web Forms-Seite ausführen und die aktuelle Seite weiter anzeigen zu lassen. Der Inhalt beider Seiten wird kombiniert. Diese Methode funktioniert nur dann, wenn die Zielseite eine Web Forms-Seite (`.aspx`) ist.

2.1. Umleiten zu einer anderen Seite

Hyperlink-Serversteuerelemente reagieren auf Klickereignisse vom Anwender. Es wird die Seite angezeigt, die in der Eigenschaft `NavigateUrl` angegeben wurde. Das Hyperlink-Steuerelement stellt keine serverseitigen Benutzerereignisse bereit. Soll der Klick im Code abgefangen werden, sind das `LinkButton`- oder das `ImageButton`-Steuerelement zu verwenden.

Um von einem `LinkButton`- oder `ImageButton`-Serversteuerelement zu einer anderen Seite zu wechseln, verwende die Methode `Redirect` des `Response`-Objektes. Damit kann zu `.aspx`- und `.htm`-Dateien navigiert werden:

```
// Navigation mittels LinkButton mit der Redirect-Methode
```

```
private void lbStart_Click(object sender, System.EventArgs e)
{
      Response.Redirect("Transfer.aspx");
}
```

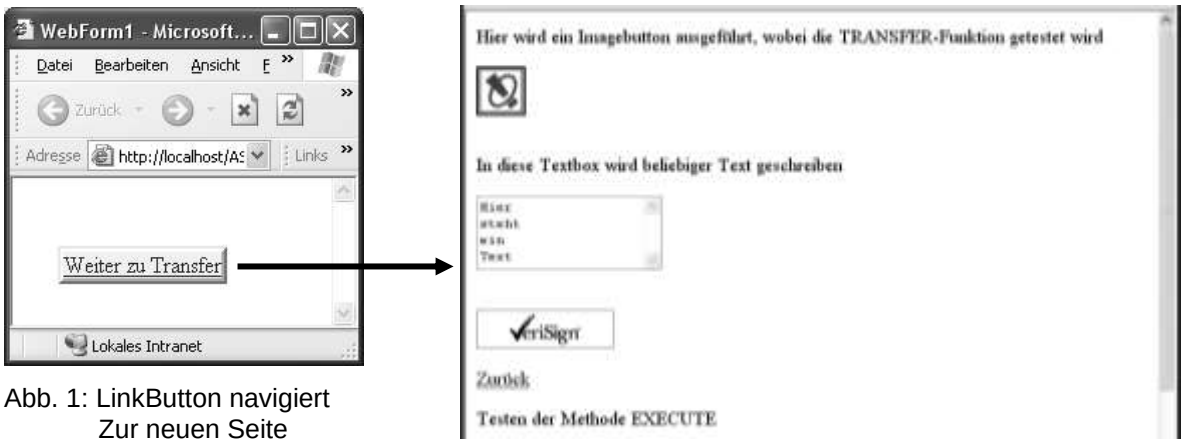

Abb. 1: LinkButton navigiert Zur neuen Seite **"Transfer.aspx"**

2.2. Umleiten zu einer anderen Seite und Übergabe von Literalen

Es besteht auch die Möglichkeit, Variablen von einer Seite (Default.aspx) auf eine andere Seite (Test.aspx) im Url zu übertragen.

Beispiel:

In diesem Beispiel liest die „Button-Click"-Methode die Zahl 3 aus der Textbox der Seite Default.aspx aus, verlinkt zur Seite Test.aspx und gibt dort den Wert 3 in der Textbox aus.

Die notwendigen Programmschritte dazu sind auf der Quellseite Default.aspx:

```
// Hole Wert aus der Textbox
int wert = Convert.ToInt32(tbxDefault.Text);
string info = "Wert erhalten";
// Erstelle den URL-String
String url = String.Format("Test.aspx?val1={0}&val2={1}", wert, info);
// Verlinke zur Zielseite
Response.Redirect(url);
```

Auf der Zielseite Test.aspx:

```
protected void Page_Load(object sender, EventArgs e)
{
    string sWert = Request["val1"];
    string sInfo = Request["val2"];
    tbxTest.Text = sWert;
    labTest.Text = sInfo;
}
```

Abb. 2: Response.Redirect mit Übergabe der Parameter in der URL.

3. Serverseitiges Cachen

3.1. Round Trip und Caching

Mit „**round trip**“ bezeichnet man den Vorgang, den eine clientseitige Aktion (z.B. *Button-Click*) auslöst. Diese Aktion ruft serverseitig die entsprechende .aspx-Seite auf, bearbeitet sie und sendet das Ergebnis in Form einer html-Seite wieder zurück. Wenn man Daten bei einem *round trip* zwischenspeichern möchte, kann man entscheiden, ob dies **serverseitig** oder **clientseitig** geschehen soll.

- **Serverseitiges Cachen von Daten:** Hier kann man aus *Session* state, *Application* state, und einem *cache* wählen.
- **Clientseitiges Cachen von Daten:** Hier kann man aus *View stat, localStorage* und einem *hidden field* wählen.

Das Zwischenspeichern von Daten auf dem Server verbraucht Serverressourcen. Das kann wiederum die Performance des Servers und Skalierbarkeit beeinträchtigen.

Cache kann diesem Problem begegnen, weil der Cache-Manager Daten verwirft, wenn der Server Speicher benötigt oder wenn die gecachten Daten nicht mehr aktuell sind (expire). Caching Daten benötigt jedoch eine Logik hinter der Seite, die entscheidet, wann Daten nicht mehr benötigt werden.

3.2. Session

Mit **Session** werden Informationen jeglicher Art serverseitig zwischengespeichert. D.h. es kann von einer einfachen Variablen bis hin zu einer Liste mit komplexen Daten (Klassen) gecacht werden.

Beispiel:

In diesem Beispiel soll aus einer .aspx-Datei (z.B. Default.aspx) der Wert 4711 aus einer Textbox ausgelesen und in einer *Session* gecacht werden. In der zweiten .aspx-Datei (z.B. Test.aspx) soll dieser Wert aus der *Session* ausgelesen und in einer Textbox angezeigt werden:

Implementierung „Cachen“:

```
Session["Wert"] = Convert.ToInt32(tbxSession.Text);
```

Implementierung „Wert aus Session holen“:

```
int wert = Convert.ToInt32(Session["Wert"]);
```

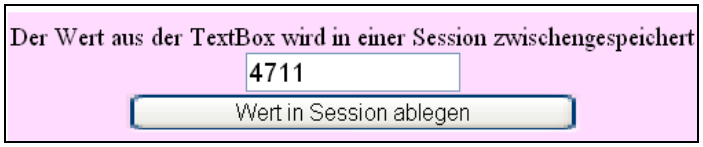

Abb. 3: Default.aspx: 4711 wird in Session gecacht

Abb. 4: Test.aspx: Ausgabe der gecachten Information

4. Clientseitiges Cachen

4.1. HiddenField: Übertragen von Daten aus JavaScript ←→ ASPX

Um Informationen von einer Funktion aus einer JS-Datei in eine Funktion von ASPX zu übertragen, bedienen wir uns des *HiddenFields*. *HiddenField* ist ein unsichtbares ASPX-Control,

das für die Speicherung von Informationen nützlich ist. Der Typ der Information ist in erste Linie String. Es können aber auch Felder und Objekte übermittelt werden, die aber in einem *JSON-Format* verwaltet werden müssen.

A) Übertragen eines *Wertes einfachen Datentyps*: Diese sind in Strings zu konvertieren und dann dem HiddenField zuzuweisen:

```
ASPX - write:  hdfNameID.Value = "Dr.Chess";
ASPX - read:  string name = hdfNameID.Value;
HTML – write:  var name = document.getElementById("hdfNameID");
               name.value = text;
HTML – read: var hdf = document.getElementById("hdfNameID");
             var text = hdf.value;
```

B) Übertragen eines *Feldes einfachen Datentyps – OHNE JSON-Funktionen*: Diese sind in Strings zu konvertieren und mit einem Trennzeichen zu versehen. Hier ist `artList[]` ein Feld von Artikeln und `hdfAllArts` das *HiddenField*, das die Artikel-Feld übernimmt. `AllArts` ist ein JS-Array, das die `Artikel.toString()` übernimmt:

```
ASPX – write: hdfAllArts.Value += artList[i].ToString() + '*';
ASPX – read: string[] strList = hdfAllArts.Value.Split('*');
HTML - write: hdfAllArts.value += AllArts[i] + '*';
HTML - read: AllArts = hdfAllArts.value.split('*');
             var art = AllArts[SelIdx];
```

C) Übertragen eines *Feldes einfachen Datentyps MIT JSON von ASPX → HTML:* Dazu benötigen wir zum Schreiben auf der ASPX-Seite die Funktion `JavaScriptSerializer().Serialize(HiddenField-Wert)`, auf der JS-Seite die Funktion `JSON.parse(Feld)`:

```
ASPX – write: hdfFld.Value = new JavaScriptSerializer().Serialize(iFld);
HTML - read: var iFld = JSON.parse(hdfFld.value);
```

Übertragen eines *Feldes einfachen Datentyps MIT JSON von HTML → ASPX:* Dazu benötigen wir zum Schreiben `JSON.stringify(Feld)` und `JavaScriptSerializer().Desrialize<Typ[]>(HiddenField-Wert)`.

```
HTML - write: hdfIntFld.value = JSON.stringify(ZahlenFld);
ASPX – read: int[] iFld = new JavaScriptSerializer().Deseria-
             lize<int[]>(hdfIntFld.Value);
```

D) Übertragen eines *Objektes mit JSON von ASPX → HTML:* Wir betrachten ein Auto-Objekt `a1`. Dazu benötigen wir zum Schreiben auf der ASPX-Seite die Funktion `JavaScriptSerializer().Serialize(obj)`, auf der JS-Seite die Funktion `JSON.parse(Object)` und den Konverter `Object.assign(new Auto(), geparster JSON-Strig)`:

```
ASPX – write: hdfAuto.Value = new JavaScriptSerializer().Serialize(a1);
HTML - read: var jsonAuto = JSON.parse(hdfAuto.value);
             var auto = Object.assign(new Auto(), jsonAuto);
```

Übertragen eines *Objektes mit JSON von HTML → ASPX:* Wir betrachten ein Auto-Objekt `a1`. Dazu benötigen wir einerseits die Funktion `stringify(…)` bzw. das `JavaScriptSerializer`-Objekt:

```
HTML - write: var a1 = new Auto("Merc", 12345.67, 220);
              hdfAuto.value = JSON.stringify(a1);
ASPX – read: Auto a1 = new JavaScriptSerializer().Deserialize<Auto>
             (hdfAuto.Value);
```

E) Übertragen eines *Feldes von Objekten mit JSON von ASPX → HTML:* Wir betrachten ein Feld von Auto-Objekten `autoFld`. Dazu benötigen wir auf der ASPX-Seite die `Serialize`-Funktion, auf der HTML-Seite die Funktion `JSON.parse(…)` und `Object.assign(…)`

```
ASPX – write: hdfAutoFld.Value = new JavaScriptSerializer().Seria-
                                 lize(autoFld);
HTML - read: var jsonAutoFld = JSON.parse(hdfAutos.value);
             var jsonAuto = Object.assign(new Auto(), jsonAutoFld[i]);
```

Übertragen eines *Feldes von Objekten mit JSON von HTML → ASPX:* Wir betrachten ein Feld von Auto-Objekten `autoFld`. Dazu benötigen wir einerseits die JS-Funktion `stringify(…)` bzw. das `JavaScriptSerializer`-Objekt:

```
HTML - write: hdfAuto.value = JSON.stringify(autoFld);
ASPX – read: Auto[] aFld = new JavaScriptSerializer().
                           Deserialize<Auto[]>(hdfAuto.Value);
```

4.2. localStorage: Übertragen von Daten aus JS ←→ JS

`localStorage`: Damit können Daten von einer HTML-Seite auf die andere übertragen werden. Achtung: Ist ein HTML5-Feature und daher nicht bei allen Bowsern enthalten und ausführbar.

A) Übertragen eines *Wertes einfachen Datentyps*: Dieser ist in einen Strings zu konvertieren und dann dem `localStorage` zuzuweisen:

`HTML – write: localStorage.setItem("daten", wert);` Die Information `wert` wird in den Speicher `localStorage` namens `"daten"` gespeichert.

`HTML – read: var value = localStorage.getItem("daten");` Die Information wird aus dem `localStorage` namens `"daten"` auf die Variable `value` ausgelesen.

B) Übertragen eines *Feldes einfachen Datentyps*: Es handelt sich dabei immer um string-Arrays. Die Elemente des Feldes werden durch ein Komma getrennt als String abgespeichert. Der Rückgabewert (HTML-read) ist ein string. Die Elemente des Feldes sind durch Komma getrennt:

```
HTML - write: localStorage.setItem("Artikel", AllArts);
HTML - read: var alleArtikelStr = localStorage.getItem("Artikel");
                          // alleArtikelStr ist ein String!
             var alleArtikelFld = alleArtikelStr.split(',');
                          // alleArtikelFld ist ein String-Feld!
  var art = alleArtikelFld[0]; // art enthält den 1. Artikel als String
```

C) Das Übertragen von *Objekten:* Hier sollte für das Objekt eine *„toString()"-Methode* implementiert werden und diese an *localStorage* übergeben werden! Dasselbe gilt für *Felder von Objekten.*

5. JavaScript Objekte:

JavaScript unterstützt die Programmierung von Objekten. Objekte werden aber nicht in Form von Klassen repräsentiert, sondern in Form von Funktionen. Ein JavaScript-Objekt hat Eigenschaften und Methoden. Beispielsweise hat ein String-Objekt die Eigenschaft **`length`** und die Methode **`toUpperCase()`:**

```
function beimLaden() {
    var string = "Hallo 5. Klasse";
    alert("Laenge von String: " + string.length);
    alert("UpperCase von String: " + string.toUpperCase());
}
```

JavaScript besitzt unter anderem die folgenden eingebauten Objekte (nähere Informationen unter http://de.selfhtml.org/javascript/objekte/index.htm):

- ✓ String
- ✓ Date (siehe Zufallszahlen)
- ✓ Array (siehe oben)
- ✓ Boolean
- ✓ Math (siehe oben)

Zum Erzeugen eigener Objekte gibt es zwei Möglichkeiten:

- ✓ Erzeugen einer direkten Instanz von `Object`
- ✓ Erzeugen eines *Templates*

5.1. Instanz von Template

Wir konzentrieren und auf Objekte von Templates. Ein Template ist nichts anderes als eine Funktion. In ihr definiert man die Membervariablen wie die Member-Funktionen.

Beispiel *Instanz als Template:* In dieser Anwendung wird das Objekt Toni Gigler, 10. 3. 2000 erstellt und über eine Funktion ausgegeben.

✓ Beim Laden wird das Objekt erstellt und der Objektstring für die Ausgabe bereitgestellt:

```
var Pers;
function beimLaden() {
   Pers = new Person("Toni", "Gigler", new Date("April 10, 1990"));
   Pers.showMe();
}
```

✓ Die Objektdefinition

```
function Person(vorname, nachname, alter) {
    this.vorname = vorname;
    this.nachname = nachname;
    this.alter = alter;
    this.PersString = "";
    this.showMe = OnShowMe; // Methoden sind JavaScript Funktionen
}
```

✓ Die Memberfunktion des Templates

```
function OnShowMe() {
    this.PersString = "Hi - I'm " + this.vorname + " " +
                      this.nachname + "\nBirthdate " + this.alter);
}
```

✓ Ausgabe der Person

```
function onGibToni() {
    alert(Pers.PersString);
}
```