assono

AJAX in Domino-Web-Anwendungen — der nächste Schritt

28. Februar 2007

Thomas Bahn
assono GmbH
tbahn@assono.de
http://www.assono.de
+49/4307/900-401

assono

Agenda

- Organisatorisches
- Wie kann man den Domino-Server per AJAX aufrufen?
  - Agenten
  - Servlets (intern/extern)
  - Web Services
  - ReadViewEntries-URL-Befehl
  - Ansichten
  - Dokumente und Masken
  - Seiten
  - SaveDocument-URL-Befehl
  - CreateDocument-URL-Befehl
  - ReadViewEntries-URL-Befehl mit OutputFormat=JSON

assono

Agenda

- Welcher Datenformate gibt es für die Antwort?
  - „Einfacher" Text
    - einfacher Werte
    - Listen
    - CSV
    - Tabelle mit festen Spaltebreiten
  - HTML
  - XML
    - DXL
    - SOAP
  - JSON
  - JavaScript

assono

Agenda

- Demos
  1. Abhängige Auswahllisten nachladen
  2. Test auf Eindeutigkeit
  3. Fortschrittsanzeige
  4. Bearbeitbare Ansichten
  5. JavaScript-Fehler auf Server protokollieren
  6. Ansicht auf neue Dokumente überwachen

assono

Organisatorisches

Aufrufe

Formate

Demos

assono

Organisatorisches

- Wer bin ich?
- Thomas Bahn, IT-Berater, Dipl.-Math., 36 Jahre, verheiratet
- Mitgründer und -inhaber der assono GmbH
- seit 1997 entwickle ich mit Java und RDBMS, z. B. Oracle (OCP)
- seit 1999 mit IBM Lotus Notes/Domino
  (IBM Certified Application Developer - Lotus Notes/Domino R4 - 7)
- Mein Schwerpunkt liegt auf Anwendungen mit Schnittstellen zu anderen Systemen, z. B. RDBMS, SAP R/3, und interaktiven Web-Anwendungen
- auch Administrator
  (IBM Certified System Administrator – Lotus Notes/Domino 6 - 7)

assono

Organisatorisches

- bitte zum Schluss Bewertungsbögen ausfüllen
- Zwischenfragen erwünscht!
- bitte Handys aus- oder stumm schalten
- Der Vortrag ist sehr lang. So ich notfalls länger machen oder Details überspringen?

assono

Organisatorisches

- Vorweg noch ein paar Fragen:
  - Wer hat schon Domino-Web-Anwendungen mit AJAX entwickelt?
  - Wurde ein Framework benutzt, und wenn ja, welches?
  - Wird die Anwendung schon produktiv eingesetzt?
  - Wer hat mit AJAX, aber nicht unter Domino entwickelt?
  - Wer hat Domino-Web-Anwendungen ohne AJAX entwickelt?
  - Wer kennt AJAX, aber ohne Praxis (oder nur ausprobiert)?
  - Wer war letztes Jahr bei meinem Vortrag
    - auf dem EntwicklerCamp 2006?
    - auf der Herbst-DNUG?

assono

Organisatorisches

Aufrufe

Formate

Demos

assono

Wie kann man den Domino-Server per AJAX aufrufen?

- Agenten
- Servlets
  - intern (Domino Servlet Engine)
  - extern (anderer Servlet-Container)
- Web Services
- ReadViewEntries-URL-Befehl
- Ansichten
- Dokumente und Masken
- Seiten
- SaveDocument-URL-Befehl
- CreateDocument-URL-Befehl
- ReadViewEntries-URL-Befehl mit OutputFormat=JSON

assono

Agenten

- weit verbreiteter Ansatz, sehr flexibel
- können Daten erzeugen, lesen, ändern und löschen
- können auch auf andere (Nicht-Domino-)Systeme zugreifen, auf einzelne oder mehrere Dokumente
- können Aufruf-Parameter nutzen
- können mit GET und POST aufgerufen werden
- können beliebige Datenformate zurückgeben
- erfordern LotusScript- oder Java-Kenntnisse
- erfordern Programmierung (Entwicklungskosten)
- erfordern Zugriff auf die Gestaltung der Datenbank (nicht möglich bei versteckter Gestaltung)
- sind relativ langsam (Server) wegen Overhead (Initialisierung, JVM!)
- Einstellung im Server-Dokument: Web-Agenten parallel ausführen

assono

Agenten (forts.)

- Beispiel-Aufruf (GET):

```
var agentURL = 'http://server/db.nsf/AgentName?OpenAgent';
var parameters = 'Parameter1=Hallo+Welt&Parameter2';
request.open('GET', agentURL+escape('&'+parameters), true);
```

- Beispiel-Aufruf (POST):

```
var agentURL = 'http://server/db.nsf/AgentName?OpenAgent';
var parameters = 'Parameter1=Hallo+Welt&Parameter2';
request.open('POST', agentURL, true);
...
request.send(parameters);
```

assono

Agenten (forts.)

- Beispiel-Agent:

```
Sub Initialize
   Dim currentTimestamp As NotesDateTime

   Set currentTimestamp = New NotesDateTime("")
   currentTimestamp.SetNow

   Print currentTimestamp.GMTTime
End Sub
```


assono

Agenten (forts.)

- Ergebnis:

```
<html>
<head>
<META HTTP-EQUIV="Content-Type" CONTENT="text/html;
   charset=US-ASCII">
</head>
<body text="#000000">
07.01.2007 04:25:39 PM GMT
</body>
</html>
```

assono

Agenten (forts.)

- Beispiel-Agent mit Content-Type:

```
Sub Initialize
   Dim currentTimestamp As NotesDateTime

   Set currentTimestamp = New NotesDateTime("")
   currentTimestamp.SetNow

   Print "Content-Type: text/plain"
   Print
   Print currentTimestamp.GMTTime
End Sub
```


assono

Agenten (forts.)

- Ergebnis:

24.02.2007 08:21:38 PM GMT(0x0a)

- Print hängt immer Linefeed (ASCII-Code 10) an

assono

Servlets

- sehr flexibel
- können Daten erzeugen, lesen, ändern und löschen
- können auch auf andere (Nicht-Domino-)Systeme zugreifen
- können Aufruf-Parameter nutzen
- können mit GET und POST aufgerufen werden
- können beliebige Datenformate zurückgeben
- erfordern Java-Kenntnisse
- können fertige Java-Komponenten und –Bibliotheken nutzen
- erfordern Programmierung (Entwicklungskosten)
- erfordern keinen Zugriff auf die Gestaltung der Datenbank
- sind schneller wegen einmaliger Initialisierung, skalieren besser
- müssen synchron mit Web-Anwendung aktualisiert werden, keine automatische Verteilung oder Replikation

assono

Interne Servlets

- So nenne ich Servlets, die in der Domino Servlet Engine laufen
- Egal welche Domino-Version: Die JVM ist immer „veraltet".

- Domino Servlet Engine muss diese aktiviert sein.
- Jedes Servlet muss in die servlet.properties eingetragen werden.
- Für das Ausliefern sind Schreibrechte auf Dateisystem des Domino-Servers notwendig (konfiguriertes Servlet-Verzeichnis).
- Der HTTP-Task muss bei jeder Aktualisierung neu gestartet werden.

- Mehr zur Domino-Entwicklung mit Servlets unter:

http://www-128.ibm.com/developerworks/lotus/library/ls-servlets/index.html

assono

Interne Servlets (forts.)

- Beispiel: InternalServlet
- In servlet.properties:

```
servlet.InternalServlet.code = de.assono.InternalServlet
servlets.startup = InternalServlet
```

- Aufruf:
  http://server/servlet/InternalServlet

assono

Interne Servlets (forts.)

```
public class InternalServlet extends HttpServlet {
   protected void doGet(HttpServletRequest request,
         HttpServletResponse response) throws ServletException, IOException {

         response.setContentType("text/plain");
         PrintWriter writer = response.getWriter();
         try {
                  NotesThread.sinitThread();
                  Session session = NotesFactory.createSession();
                  DateTime dt = session.createDateTime(new Date());
                  writer.print(dt.getGMTTime());
                  dt.recycle();
                  session.recycle();
         } catch (NotesException ne) {
                  throw new ServletException(…);
         } finally {
                  NotesThread.stermThread();
         }
   }
}
```

assono

Externe Servlets

- So nenne ich alle Servlets, die in einem anderen Servlet-Container laufen, z. B. Apache Tomcat, Apache Geronimo, IBM WebSphere, … (beliebiger Java Enterprise Edition (JEE) Server)
- Jede Java-Version ist möglich.
- Dadurch sind können auch Komponten und Bibliotheken genutzt werden, die eine aktuelle JVM voraussetzen.
- Der andere Server muss in der gleichen Domäne laufen (Sicherheitsbeschränkung von XMLHttpRequest)
- Er muss zusätzlich installiert und gewartet werden (Lizenz- und Wartungskosten).
- Aktualisierungen von Web-Anwendungen und Servlets müssen synchron durchgeführt werden und sind daher komplex und aufwendig zu automatisieren.
- DIIOP-Task muss laufen

assono

Externe Servlets (forts.)

- Beispiel: ExternalServlet
- Aufruf:
  http://server:8080/path/ExternalServlet

assono

Externe Servlets (forts.)

```
public class ExternalServlet extends HttpServlet implements Servlet{
   private Session session = null;

   public void init() throws ServletException {
      super.init();
      try {
         this.session = NotesFactory.createSession("127.0.0.1");
      } catch (NotesException ne) {…}
   }
   public void destroy() {
      super.destroy();
      try {
         if (this.session != null) this.session.recycle();
      } catch (NotesException ne) {…}
   }
   …
```


assono

Externe Servlets (forts.)

```
   …
   protected void doGet(HttpServletRequest request,
   HttpServletResponse response) throws ServletException,
   IOException {
      response.setContentType("text/plain");
      PrintWriter writer = response.getWriter();
      try {
         DateTime dt = session.createDateTime(new Date());
         writer.print(dt.getGMTTime());
         dt.recycle();
      } catch (NotesException ne) {
         throw new ServletException(…);
      }
   }
}
```

assono

Web Services

- sehr flexibel, aber komplex
- können Daten erzeugen, lesen, ändern und löschen
- können auch auf andere (Nicht-Domino-)Systeme zugreifen, zum Beispiel SAP oder .net-Anwendungen
- vorhandene Web Services können genutzt werden, wenn sie in der gleichen Domäne liegen (Sicherheitsbeschränkung)
- erfordern weiteren Server (außer Domino 7 Web Services)
- Dieser muss zusätzlich installiert und gewartet werden (Lizenz- und Wartungskosten, außer Domino 7 Web Services).

assono

Web Services (forts.)

- können Aufruf-Parameter nutzen
- können (eigentlich) nur mit POST aufgerufen werden
- für Aufrufe und Antworten nur Simple Object Access Protocol (SOAP) möglich
- relativ langsam, da SOAP-Nachrichten erstellt und Antworten geparst werden müssen
- erfordern Kenntnisse in Java (oder Domino 7 und LotusScript) und Web Services
- erfordern Programmierung (Entwicklungskosten)
- erfordern keinen Zugriff auf die Gestaltung der Datenbank
- Aktualisierungen von Web-Anwendungen und Web Services müssen synchron durchgeführt werden und sind daher komplex und aufwendig zu automatisieren (außer Domino 7 Web Services).

assono

ReadViewEntries-URL-Befehl

- weit verbreiteter Ansatz
- eingeführt in Domino 5.0.2 für View-Applet
- sehr schnell und skalierbar, da kein interpretierter Code aufgerufen wird und Anfragen gecacht werden können
- sicher, da die normalen Zugriffsbeschränkungen angewendet werden
- kann Daten nur lesen
- kann nur Standard-Parameter nutzen: Start, Count, StartKey, UntilKey, RestrictToCategory usw.
- kann nur mit GET aufgerufen werden
- erfordert keine (serverseitige) Programmierung
- erfordert keinen Zugriff auf die Gestaltung der Datenbank, wenn es eine Ansicht mit den benötigten Informationen schon gibt
- Rückgaben sind immer in DXL und müssen im Browser geparst werden (automatisch).

assono

ReadViewEntries-URL-Befehl (forts.)

- Beispiel
  http://server/path/db.nsf/CarModels?ReadViewEntries

- Ergebnis

```
<?xml version="1.0" encoding="UTF-8"?>
<viewentries toplevelentries="58">
<viewentry position="1" unid="…" noteid="4DF2" siblings="58">
<entrydata columnnumber="0" name="Manufacturer">
<text>Acura</text>
</entrydata>
<entrydata columnnumber="1" name="Models">
<textlist><text>MDX</text><text>NSX</text>…<text>TSX</text></textlist>
</entrydata>
</viewentry>
...
```

assono

ReadViewEntries-URL-Befehl (forts.)

- Navigation mit DOM-Operationen in JavaScript möglich

- Beispiel

```
var xmlRoot = request.responseXML;
var viewEntriesFirstNode =
   xmlRoot.getElementsByTagName('viewentries')[0];
var rowsCount =
   viewEntriesFirstNode.getAttribute('toplevelentries');
alert("Number of rows in view: " + rowsCount);
```

- „Trick": Count=0: Minimale Datentransfer und die Anzahl (toplevelentries) ist enthalten

assono

Ansichten

- Ansichten, auch solche mit $$ViewTemplate-Maske, können mit OpenView-URL-Befehlen gelesen werden; eingebettete Ansichten mit OpenPage, OpenForm oder ReadForm.
- sehr schnell, skalierbar, Caching möglich
- können Daten nur lesen
- können nur mit GET aufgerufen werden
- Ansichten können nur Standard-Parameter nutzen: Start, Count, StartKey, UntilKey, RestrictToCategory usw.;
  eingebettete Ansichten/Ansichten mit $$ViewTemplate-Maske können auch weitere Parameter nutzen (werden in der Maske bzw. Seite verwendet)
- flexibler als ReadViewEntries-URL-Befehl:

assono

Ansichten (forts.)

- Rückgabe normalerweise HTML-Seite (mit <table>, <tr> usw.); um nicht parsen zu müssen, „Ansichtsinhalt als HTML behandeln" aktivieren und in Masken mit Durchgangs-HTML benutzen
- bei Ansichten jedes Format ohne Kopf- und Fußzeilen möglich, bei eingebetten Ansichten/mit $$ViewTemplate-Maske fast jedes Format
- dadurch sehr kleine Antworten = geringe Netzlast möglich
- erfordern keine serverseitige Programmierung (wohl aber entsprechende Gestaltungselemente)
- erfordern Zugriff auf die Gestaltung der Datenbank (nicht möglich bei versteckter Gestaltung)

assono

Ansichten (forts.)

- **Achtung**:
  Die Anzahl der Zeilen, die im Web für eine Ansicht zurückgegeben werden, wird ohne Count-Parameter bestimmt durch die Einstellung „Default lines per view page" (Vorgabe 30), und ist nach oben begrenzt durch „Maximum lines per view page" (Vorgabe: 1.000; Server-Dokument – Internet Protocols – Domino Web Engine – Conversion/Display)

assono

Ansichten (forts.)

- Beispiel ViewDemoPage.js/ViewDemoView
- Aufruf
  http://server/pfad/db.nsf/ViewDemoPage.js?OpenPage&Count=500
- Gestaltung der Seite ViewDemoPage.js:

ew) - Ansicht | ViewDemoPage.js - Seite

```
var manufacturerModelsArray = new Array();

function addManufacturer(manufacturer, modelsArray) {
        manufacturerModelsArray[manufacturer] = modelsArray;
}

// hier beginnt die eingebettete Ansicht:
addManufacturer("Acura", nev "MDX", "NSX", "RL", "RSX", "TL", "TSX"
addManufacturer("Alfa Romet "Alfa 145", "Alfa 146", "Alfa 147", "Alfa 155", "Alfa 156
addManufacturer("Alpina", net "B10", "B12", "B3", "B5", "B6", "B7", "B8", "D 10", "Ro
addManufacturer("Aston Marti "AR1", "DB", "DB7", "DB9", "Lagonda", "V8", "Vanqu
```


assono

Ansichten (forts.)

- Das Ergebnis kann einfach „ausgeführt" werden:

```
eval(request.responseText);
var bmws = markenModelsArray["BMW"];
alert(bmws.join(","));
```

assono

Dokumente und Masken

- Dokumente werden mit OpenDocument-URL-Befehlen geöffnet, Masken mit OpenForm und ReadForm.
- sehr schnell, Caching möglich
- können Daten nur lesen
- können Aufruf-Parameter nutzen
- können nur mit GET aufgerufen werden
- Rückgabe normalerweise HTML-Seite; um nicht parsen zu müssen, Durchgangs-HTML benutzen; so aber jedes Format möglich
- erfordern keine serverseitige Programmierung (wohl aber entsprechende Gestaltungselemente)
- erfordern Zugriff auf die Gestaltung der Datenbank (nicht möglich bei versteckter Gestaltung)

assono

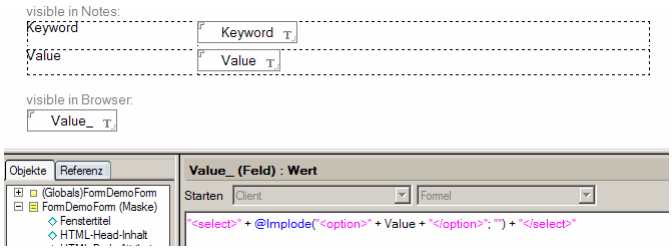
Dokumente und Masken (forts.)

- Verschiedenste Aufrufvarianten:
  http://server/path/db.nsf/maskenname?OpenForm
  http://server/path/db.nsf/maskenname?ReadForm
  http://server/path/db.nsf/ansichtenname/schlüssel?OpenDocument
  http://server/path/db.nsf/0/UNID?OpenDocument
  und viele weitere
- Beispiel (FormDemoView) und FormDemo-Maske
- Aufruf
  http://server/path/db.nsf/FormDemoView/Weekdays

assono

Dokumente und Masken (forts.)

- Maske



assono

Dokumente und Masken (forts.)

- Ergebnis

```
<select>
<option>Sunday</option>
<option>Monday</option>
<option>Tuesday</option>
<option>Wednesday</option>
<option>Thursday</option>
<option>Friday</option>
<option>Saturday</option>
</select>
```

assono

Dokumente und Masken (forts.)

- kann genutzt werden, um zum Beispiel zu füllen

- Eine HTML-Seite enthalte

```
<span id="aSelect"></span>
```

- dann kann man den „Inhalt" mit dem innerHTML-Attribut ersetzen

```
var spanElement = $('aSelect');
spanElement.innerHtml = request.responseText;
```


assono

Seiten

- Seiten werden mit OpenPage-URL-Befehlen geöffnet.
- sehr schnell
- können Daten nur lesen
- können Aufruf-Parameter nutzen (@UrlQueryString)
- können nur mit GET aufgerufen werden
- Dynamik über Berechneten Text, also nur Formelsprache möglich
- können beliebige Datenformate zurückgeben
- erfordern keine serverseitige Programmierung (wohl aber entsprechende Gestaltungselemente)
- erfordern Zugriff auf die Gestaltung der Datenbank (nicht möglich bei versteckter Gestaltung)

assono

Seiten (forts.)

- Beispiel PageDemoPage
- Berechneter Text:

```
@Text(@Elements(@DbColumn("Notes" : "NoCache"; "";
   "JSErrorLogs"; 1)))
```

- Aufruf
  http://server/pfad/db.nsf/PageDemoPage?OpenPage
- gibt aktuelle Anzahl von JSErrorLog-Dokumenten als String zurück

assono

SaveDocument-URL-Befehl

- Huch, was ist denn das? ☺
- SaveDocument-URL-Befehle werden vom Domino-Web-Server selbst genutzt, um vorhandene Dokumente zu ändern, die mit einer EditDocument-URL im Browser geöffnet wurden, z. B. von einer Aktion mit @Command([EditDocument]).
- Ist die Datenbank-Option „JavaScript beim Erstellen von Seiten verwenden" ausgeschaltet, steht im action-Attribut des form-Tags eine SaveDocument-URL.

assono

SaveDocument-URL-Befehl (forts.)

- Beispiel

```
<form method="post"
    action="/db.nsf/68b8d2c1b1decde7c125711500442475/
    86c96937a3c280a6c1257115004447f5?SaveDocument"
    name="_CarModels">
```

- Die erste lange Zeichenkette ist die UNID einer Ansicht, die zweite die UNID eines Dokuments.
- Wie üblich funktionieren aber auch die „0-Ansicht“, Name, NoteID, Suchschlüssel usw.

assono

SaveDocument-URL-Befehl (forts.)

- sehr schnell
- nur vorhandene Dokumente können geändert werden
- kann Aufruf-Parameter nutzen
- kann nur mit POST aufgerufen werden
- neue und geänderte Items des Dokuments als Name=Wert-Paare, getrennt mit & im Körper des POST-Requests
- Kein escape() für die Parameter verwenden!
- erfordert keine serverseitige Programmierung
- erfordert keinen Zugriff auf die Gestaltung der Datenbank, wenn geeignete Maske vorliegt
- nutzt Web-Agenten, berechnete Felder, Validierungen usw.
- Rückgabe vorgegebenes, sprachabhängiges HTML, wenn es kein $$Return-Feld gibt

assono

SaveDocument-URL-Befehl (forts.)

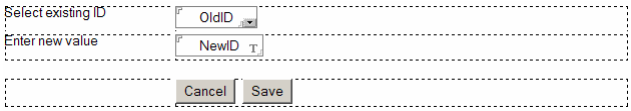
- Aufruf
  http://server/path/db.nsf/Ansichtsname/NoteID?SaveDocument
  http://server/path/db.nsf/0/UNID?SaveDocument

- Reihenfolge
  - Dokument wird lokalisiert und die Items geändert
  - WebQueryOpen-Agent
  - Maske wird neu berechnet:
    - berechnete Felder
    - Eingabeumsetzungen
    - Eingabevalidierungen(!)
  - WebQuerySave-Agent
  - Dokument wird gespeichert

assono

SaveDocument-URL-Befehl (forts.)

- Beispiel SaveDocumentDemo-Maske



- OldID: Dialogliste vorhandener IDs
- NewID: Textfeld für neue ID
- Save: Ändert ID im gewählten Customer-Dokument

assono

SaveDocument-URL-Befehl (forts.)

```
var oldIDSelect = $('OldID');
var oldID = oldIDSelect[oldIDSelect.selectedIndex].value;
var newID = $('NewID').value;

var url = '/' + $('WebDBName_').value + '/CustomersByID/' +
   OldID + '?SaveDocument';
var postArguments = 'ID=' + newID;
callServer('POST', url, true, 5000, idChanged, undefined,
   undefined, 'application/x-www-form-urlencoded', undefined,
   postArguments);

function idChanged(request) {…}
```


assono

CreateDocument-URL-Befehl

- CreateDocument-URL-Befehle werden vom Domino-Web-Server selbst genutzt, um neue Dokumente zu erstellen.
- Ist die Datenbank-Option „JavaScript beim Erstellen von Seiten verwenden" ausgeschaltet, steht in Seiten, die mit einer OpenForm-URL geöffnet wurden (also z.B. nach @Command([Compose],…) in Web-Anwendungen), im action-Attribut des form-Tags eine CreateDocument-URL
- Beispiel

```
<form method="post"
   action="/db.nsf/b679ef8824d78f39c12571150043d487?CreateDocument"
   name="_CarModels">
```

- Die lange Zeichenkette ist die UNID der Maske.
- Der Maskenname funktioniert aber auch.

assono

CreateDocument-URL-Befehl (forts.)

- sehr schnell
- kann nur Dokumente erstellen
- kann Aufruf-Parameter nutzen
- kann nur mit POST aufgerufen werden
- neue Items des Dokuments als Name=Wert-Paare, getrennt mit & im Körper des POST-Requests
- Kein escape() für die Parameter verwenden!
- erfordert keine serverseitige Programmierung
- erfordert keinen Zugriff auf die Gestaltung der Datenbank, wenn geeignete Maske vorliegt
- nutzt Web-Agenten, berechnete Felder, Validierungen usw.
- Rückgabe vorgegebenes, sprachabhängiges HTML, wenn es kein $$Return-Feld gibt

assono

CreateDocument-URL-Befehl (forts.)

- Aufruf
  http://server/path/db.nsf/Maskenname?CreateDocument
- Reihenfolge
  - Dokument wird erstellt und die Items gesetzt
  - WebQueryOpen-Agent
  - Maske wird neu berechnet:
    - berechnete Felder
    - Eingabeumsetzungen
    - Eingabevalidierungen(!)
  - WebQuerySave-Agent
  - Dokument wird gespeichert

assono

CreateDocument-URL-Befehl (forts.)

- Beispiel CreateDocumentDemo-Maske

- Create: erstellt neues Dokument mit den eingetragenen Werten

assono

CreateDocument-URL-Befehl (forts.)

```
var url = '/' + $('WebDBName_').value +
   '/JSErrorLog?CreateDocument';
var postArguments = 'Error=' + $('Error').value +
   '&Message=' + $('Message').value + '&Location=' +
   $('Location').value;
callServer('POST', url, true, 5000, errorLogged, undefined,
   undefined, 'application/x-www-form-urlencoded', undefined,
   postArguments);

function errorLogged(request) {…}
```

- Anwendung: Umfragen

assono

ReadViewEntries-URL-Befehl mit OutputFormat=JSON

- In der Version 7.0.2 wurde klammheimlich ein neuer Format-Parameter für die ReadViewEntries-, ReadViewDesign- und ReadEntries-URL-Befehle eingeführt.
- Wird an eine solche URL der Parameter OutputFormat=JSON angehängt, wird das Ergebnis nicht in DXL, sondern als JSON zurückgegeben.
- Vorteil: leichtere Navigation
- Vorteil: schnelleres Parsen
- undokumentiert, Format kann sich in Domino 8 (leicht) ändern
- ansonsten alles wie bei ReadViewEntries gesagt

assono

ReadViewEntries-URL-Befehl mit OutputFormat=JSON (forts.)

- ReadViewEntries-Beispiel war Liste der CarModels
- Ergebnis mit OutputFormat=JSON:


```
{"@toplevelentries": 58,
viewentry: [
{"@position": '1',
"@unid": '86C96937A3C280A6C1257115004447F5',
"@noteid": '4EBE',
"@siblings": 58,
entrydata: [
{"@columnnumber": 0, "@name": 'Manufacturer', text: {0: 'Acura'}},
{"@columnnumber": 1, "@name": 'Models',
   textlist: {text: [{0: 'MDX'}, {0: 'NSX'},…,{0: 'TSX'}]}}
]},
```

assono

ReadViewEntries-URL-Befehl mit OutputFormat=JSON (forts.)

- Rückgabe nicht wesentlich kürzer als DXL
- einfacher nutzbar durch JavaScript-Interpreter des Browsers
- einfache Navigation durch das Ergebnis

```
var response = eval('(' + request.responseText + ')');
var rowsCount = response['@toplevelentries'];
alert("Number of rows in view: " + rowsCount);
```


assono

Die „richtige" Technik finden

- Stell dir die folgenden Fragen:
  - Musst du Daten lesen, erzeugen oder ändern?
  - Ist die Gestaltung der Datenbank offen, darfst du sie ändern?
  - Kannst du JavaScript, LotusScript und/oder Java?
    Was davon am besten?
  - Kennst du XML (DXL, SOAP) und/oder JSON?
  - Wie viele Benutzer werden die Web-Anwendung gleichzeitig benutzen?

assono

Wie kann man den Domino-Server per AJAX aufrufen?

- Agenten
- Servlets
  - intern (Domino Servlet Engine)
  - extern (anderer Servlet-Container)
- Web Services
- ReadViewEntries-URL-Befehl
- Ansichten
- Dokumente und Masken
- Seiten
- SaveDocument-URL-Befehl
- CreateDocument-URL-Befehl
- ReadViewEntries-URL-Befehl mit OutputFormat=JSON

assono

Organisatorisches

Aufrufe

Formate

Demos

assono

Welcher Datenformate gibt es für die Antwort?

- „Einfacher" Text
  - einfacher Werte
  - Listen
  - CSV
  - Tabelle mit festen Spaltebreiten
- HTML
- XML
  - DXL
  - SOAP
- JSON
- JavaScript

assono

„Einfacher" Text

- Einfacher Text ist das älteste Datenaustausch-Format
- schnell und effizient zu implementieren
- „Vertrag" über Format und Inhalt
- Man kann einfache Werte „einfach so" als String zurückgeben.
- Für Listen von Werten definiert man ein Trennzeichen.
- Für Tabellen entweder Trennzeichen oder feste Spaltenbreiten.
- Bekanntes Format: CSV
- effizient, da kaum Overhead = geringe Netzwerkbelastung
- zerbrechlich, da keine Strukturinformationen
- Empfänger hängt vom Sender ab
- Änderungen müssen synchron implementiert werden

assono

HTML

- HTML wird „einfach so" vom Domino-Web-Server erstellt
- Nutzung als Daten sehr aufwendig, da zu parsen (insbesondere der „Domino-Dialekt" von HTML)
- Aber: Nutzung als Teil einer Seite einfach
- Das innerHTML-Attribut eines vorhandenen Elements überschreiben.
- Beispiel: siehe Dokumente und Masken-Abschnitt
- robust, wenn es nicht interpretiert wird, sonst…
- Empfänger hängt vom Sender ab

assono

XML

- sehr verbreitetes Datenaustausch-Format
- vor allem nützlich, wenn Sender und Empfänger keinen Kontakt zueinander haben
- zum Beispiel: Sender (Server) ist Standardprogramm (etwa SAP), Empfänger (Browser) ist eine selbstentwickelte Web-Anwendung
- Die Hauptstärke von XML ist, dass es zusammen mit den Daten auch eine Strukturbeschreibung enthält.
- Dadurch ist XML relativ robust; neue Elemente und Attribute können vom Empfänger ignoriert werden, so dass Änderungen am „Kontrakt" asynchron (erst Sender, Empfänger irgendwann später) durchgeführt werden können.
- Die Syntax des „Strukturkontrakts" kann zum Beispiel mittels XML Schema, RelaxNG oder Document Type Definition (DTD) beschrieben und ihre Einhaltung automatisiert geprüft werden.

assono

XML (forts.)

- weiterer Vorteil: durch die weite Verbreitung von XML gibt es viele Werkzeuge und für fast alle Sprachen auch interne Unterstützung
- Beispiel: freie JavaScript-Bibliothek Sarissa unterstützt bei Browser-unabhängiger XML-Verarbeitung (http://sarissa.sourceforge.net)
- insbesondere das XMLHttpRequest-Objekt unterstützt XML: Eine Serverantwort in XML wird automatisch geparst und in dem responseXML-Attribut zur Verfügung gestellt.
- im Browser dann Navigation über DOM-Objekte möglich
- Ein Spezialfall ist SOAP, ein XML-Datenformat, dass für Web Service-Anfragen und –Antworten genutzt wird
- Ein weiterer Spezialfall ist die Domino XML Language (DXL), die mit Domino 6 eingeführt wurde und z. B. vom ReadViewEntries-URL-Befehl erzeugt wird.

assono

XML (forts.)

- Der Vorteil von XML ist auch sein Nachteil: Durch die Strukturinformationen werden mehr Daten übertragen, was eine höhere Netzwerkbelastung bedeutet.
- Neben den dadurch höheren Übertragungszeiten muss das relativ komplexe Datenformat auf dem Server erzeugt und beim Empfänger geparst werden, was wiederum die Performance negativ beeinflusst.
- Empfänger hängt vom Sender ab

assono

JSON

- JSON = JavaScript Object Notation
- sprachunabhängiger Standart
- von http://json.org:

„**JSON** (JavaScript Object Notation) is a lightweight data-interchange format. It is easy for humans to read and write. It is easy for machines to parse and generate. It is based on a subset of the JavaScript Programming Language,[…]. JSON is a text format that is completely language independent but uses conventions that are familiar to programmers of the C-family of languages, […]. These properties make JSON an ideal data-interchange language."

assono

JSON (forts.)

- JSON ist die Notation von Objekt- und Array-Literalen in JavaScript
- Objekte sind kommaseparierte Listen von Name : Wert-Paaren in geschweiften Klammern: {}
- Mögliche Werte sind Strings, Zahlen, Objekte, Arrays oder eine der Konstanten: true, false, null.
- Arrays sind kommaseparierte Listen von Werten in eckigen Klammern: []
- Namen sind Strings in doppelten Anführungsstrichen: ""

assono

JSON (forts.)

- Beispiel:

```
{  "Name" : "Thomas Bahn",
   "Adresse" : {"Ort" : "Raisdorf", "PLZ" : "24223"},
   "Telefonnummern" : ["04307/900-401", "0173/9357916"]
}
```

- Nutzung

```
var person = eval('(' + request.responseText + ')');
alert(person.Name);
alert(person.Adresse.Ort);
alert(person.Telefonnummern[1]);
```


assono

JSON (forts.)

- sehr leichte Navigation durch .-Notation oder Array-Notation
- durch die Strukturinformationen ähnlich robust
- kein automatische Prüfung gegen „Strukturkontrakt“ (entsprechend XML Schema o .ä. bei XML)
- eingebauter „Parser“ in JavaScript: eval()
- dadurch wachsende Popularität im AJAX-Umfeld

assono

JavaScript

- statt JSON kann man auch vollständiges JavaScript als Datenaustausch-Format benutzen
- wie JSON kann JavaScript mit eval() geparst und ausgeführt werden
- Beispiel: siehe Ansichten-Abschnitt
- JavaScript ist „anders" als die anderen Formate
  - aktiv
  - Sender kann vom Empfänger abhängen
  - **sehr** flexibel
  - gefährlich, da Script ungeprüft ausgeführt wird; vertrauenswürdige Datenquelle wichtig

assono

Welcher Datenformate gibt es für die Antwort?

- „Einfacher" Text
  - einfacher Werte
  - Listen
  - CSV
  - Tabelle mit festen Spaltebreiten
- HTML
- XML
  - DXL
  - SOAP
- JSON
- JavaScript

assono

Organisatorisches

Aufrufe

Formate

Demos

assono

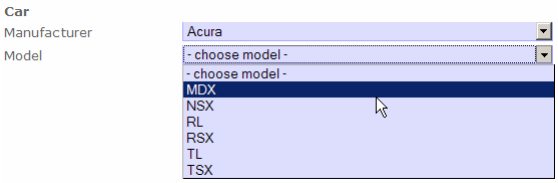
Demos

1. Abhängige Auswahllisten nachladen
2. Test auf Eindeutigkeit
3. Fortschrittsanzeige
4. Bearbeitbare Ansichten
5. JavaScript-Fehler auf Server protokollieren
6. Ansicht auf neue Dokumente überwachen

assono

Abhängige Auswahllisten nachladen

- Nach Auswahl der Marke die passenden Modelle in zweite Auswahlliste eintragen
- Zu viele Daten, um alle Modelle für alle Marken im Voraus zu laden



assono

Abhängige Auswahllisten nachladen (forts.)

- Manufacturer.onchange="getModels();":

```
var manufacturerSelect = $('Manufacturer');
var currentManufacturer =
   manufacturerSelect[manufacturerSelect.selectedIndex].value;

var url = '/' + $('WebDBName_').value +
   '/CarModels?ReadViewEntries&count=1&startKey=' +
   escape(currentManufacturer);
callServer('GET', url, true, 0,
   createCallFunctionWithXMLHandler('setModelSelection'));
```

assono

Abhängige Auswahllisten nachladen (forts.)

- Rückgabe

```
<?xml version="1.0" encoding="UTF-8"?>
<viewentries toplevelentries="58">
<viewentry position="1" unid="…" noteid="95E" siblings="58">
<entrydata columnnumber="0" name="Manufacturer">
   <text>Acura</text>
</entrydata>
<entrydata columnnumber="1" name="Models">
   <textlist><text>MDX</text><text>NSX</text><text>RL</text>
   <text>RSX</text><text>TL</text><text>TSX</text></textlist>
</entrydata>
</viewentry>
</viewentries>
```


assono

Abhängige Auswahllisten nachladen (forts.)

- Aufruf mit ReadViewEntries
  - Count=1: Rückgabe auf einen Datensatz begrenzen
  - StartKey=*{Manufacturer}*: genau den Datensatz zum ausgewählten Hersteller
- Rückgabe: DXL fest, da ReadViewEntries
- Vorhandene Ansicht genutzt
- sparsam: nur einen Datensatz, aber viel Strukturdaten

assono

Test auf Eindeutigkeit

- Nach Eingabe der ID ihre Eindeutigkeit überprüfen

- ID.onchange:

```
var url = '/' + $('WebDBName_').value +
   '/CustomersByID?ReadViewEntries&OutputFormat=JSON&' +
   'Count=1&StartKey=' + escape($('ID').value);
callServer('GET', url, true, 0, checkIDUniqueness);
```

assono

Test auf Eindeutigkeit (forts.)

- Rückgabe

```
{  "@toplevelentries": 3,
   viewentry: [{   "@position": '3',
                   "@unid": 'BA2ACEE913DB51B6C12572620072596C',
                   "@noteid": '9E6',
                   "@siblings": 3,
                   entrydata: [{   "@columnnumber": 0,
                                   "@name": 'ID',
                                   text: {0: 'tbahn'}
                   }]
   }]
}
```

assono

Test auf Eindeutigkeit (forts.)

```
function checkIDUniqueness(request) {
   var response = eval('(' + request.responseText + ')');
   if (response.viewentry) {
        if (response.viewentry[0].entrydata[0].text['0'] ==
                                              $('ID').value) {
              showIDNotUnique();
        } else {
              showIDUnique();
        }
   } else {
        showIDUnique();
   }
}
```


assono

Test auf Eindeutigkeit (forts.)

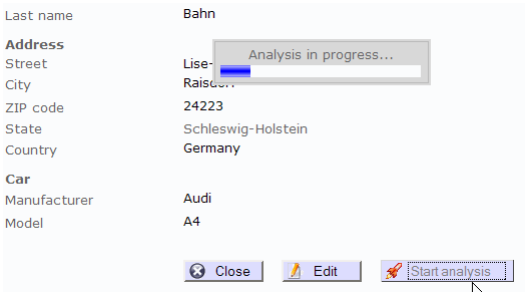
- Aufruf mit ReadViewEntries
  - OutputFormat=JSON: Rückgabe als JSON
  - Count=1: Rückgabe auf einen Datensatz begrenzen
  - StartKey=*{ID}*: wenn ein Customer mit der ID existiert, wird dessen Datensatz zurückgegeben, sonst der mit der nächst größeren ID; gibt es keinen solchen Customer, wird kein Datensatz zurückgegeben:

    {"@toplevelentries": 3}
- Rückgabe: JSON fest, da ReadViewEntries mit OutputFormat=JSON
- leichte Navigation im Ergebnis
- vorhandene Ansicht genutzt
- sparsam: nur einen Datensatz, aber viel Strukturdaten

assono

Fortschrittsanzeige

- Den Fortschritt einer lang laufenden Aktion als wachsenden Balken anzeigen
- Analysieren startet lang laufenden Prozess
- window.setInterval um regelmäßig Fortschrittsanzeige zu aktualisieren



assono

Fortschrittsanzeige (forts.)

```
function startAnalysis() {
   idValue = $('IDSpan').firstChild.nodeValue;
   var url = '/' + $('WebDBName_').value +
       '/StartAnalysis?OpenAgent&ID=' + escape(idValue);
   callServer('GET', url, true, 60000, analysisCompleted,
       analysisError, analysisTimedOut);

   prepareAnalysis();

   currentTimer = window.setTimeout('getProgress();', 500)
}
```

assono

Fortschrittsanzeige (forts.)

- `StartAnalysis`-Agent:
  - Erstellt ein `CustomerAnalysis`-Dokument
  - In einer Schleife mit Verzögerung wird langsam das `Progress`-Item dieses Dokuments erhöht
  - Ist `Progress` irgendwann 100, wird ein zufälliges Ergebnis zurückgegeben
- Ergebnis des `StartAnalysis`-Agenten z. B.: „good customer"
- Der Agent soll ein langlaufenden Prozess simulieren, dessen fortschreiten anhand eines Wertes in einem Dokument „beobachtet" werden kann.

assono

Fortschrittsanzeige (forts.)

```
function getProgress() {
   idValue = $('IDSpan').firstChild.nodeValue;
   var url = '/' + $('WebDBName_').value +
        '/GetAnalysisProgress?OpenPage&ID=' + escape(idValue);
   callServer('GET', url, true, 1000,
        createCallFunctionWithTextHandler('displayProgress'),
        undefined, undefined, 'text/plain',
        [['Cache-Control', 'no-cache']]);

   currentTimer = window.setTimeout('getProgress();', 500);
}
```

assono

Fortschrittsanzeige (forts.)

- GetAnalysisProgress-Seite enthält berechneten Text: Das Ergebnis eines @DbLookups
- Die Rückgaben sind Zahlen in Form von Strings (einfacher Text)
- kein Overhead!
- lohnt sich, da der AJAX-Call zweimal je Sekunde aufgerufen wird

Analysis - Agent | GetAnalysisProgress - Seite

<Berechneter Wert>

Berechneter Text : Wert

Starten | Client | Formel

```
@SetHTTPHeader("Cache-Control"; "no-cache");

_id := @UrlQueryString("ID");

@If(
    _id = "";
    @Return("Error: Mandatory parameter \"ID\" omitted in URL");
    "";
);

@IfError(@DbLookup("Notes" : "NoCache"; ""; "CustomerAnalyses"; _id; 2); "0")
```


assono

Bearbeitbare Ansichten

- Daten direkt **in der Ansicht** bearbeiten
- Nach dem Anklicken Text durch Textfeld ersetzen
- Nach Tab oder Return Änderungen sofort speichern

Customers assono

ID	Sal.	First name	Last name	Street	City	ZIP	State	Country
rzufall	Mr.	Rainer	Zufall	Bergstr. 101	Kiel	24103		Germany
tbahn	Dr.	Thommy	Bahn	Lise-Meitner-Str. 1-7	Raisdorf	24223	Schleswig-Holstein	Germany
lbahn	Mrs.	Lydia	Bahn	Lise-Meitner-Str. 1-7	Raisdorf	24223	Schleswig-Holstein	Germany

assono

Bearbeitbare Ansichten (forts.)

- jede bearbeitbare Spalte mit Formel entsprechend dieser:

```
_itemName := "Street";
"<td class=\"EditableCell\" id=\"" + _itemName + "." +
    @Text(@DocumentUniqueID) + "\">" + @GetField(_itemName) +
    "</td>"
```

- $$ViewTemplate for Customers.onload = prepareEditableCells();
- erzeugt dynamisch Aufrufe von startEditingCell(), saveEditedCell() und stopEditingCell() auf allen markierten Spalten

assono

Bearbeitbare Ansichten (forts.)

```
function saveEditedCell() {
   var cellForDisplay = …;
   var cellForEditing = …;
   var cellValueAfterEditing = …;
   function valueStored(request) {...}

   var url = '/' + $('WebDBName_').value + '/0/' +
       escape(docUNIDCurrentlyEdited) + '?SaveDocument';
   var postArguments = itemNameCurrentlyEdited + '=' +
       cellValueAfterEditing;
   callServer('POST', url, true, 2000, valueStored,
       undefined, undefined, 'application/x-www-form-
       urlencoded',undefined, postArguments);
}
```

assono

Bearbeitbare Ansichten (forts.)

- gespeichertes Dokument soll geändert werden
- hier wird SaveDocument-URL-Befehl genutzt

- im Browser sehr einfach zu implementieren
- serverseitig nichts zu tun
- aber $$Return mit sinnvoller Rückgabe wäre hilfreich für Fehlerprüfung

assono

JavaScript-Fehler auf Server protokollieren

- Tritt auf der Seite ein JavaScript-Fehler auf, wird er zum Server gesendet und dort protokolliert

```
function trapError(message, url, line) {
   if (SHOW_ERRORS) {
        alert(message);
   }
   var error = new Error(message);
   error.location = url + ', line:' + line;
   logJSError(error);
}

window.onerror = trapError;
```

assono

JavaScript-Fehler auf Server protokollieren (forts.)

```
function logJSError(jsError) {
   var url = '/' + $('WebDBName_').value +
       '/JSErrorLog?CreateDocument';
   var postArguments = 'Error=' + escape(jsError.name)
         + '&Message=' + escape(jsError.message)
         + '&Location=' + escape(jsError.location)
         + '&Referer=' + escape(location.href)
         + …
         + '&Cookie=' + escape(document.cookie);
   callServer('POST', url, true, 5000,
       createCallFunctionWithTextHandler('errorLogged'),
       undefined, undefined, 'application/x-www-form-
       urlencoded', undefined, postArguments)
}
```


assono

JavaScript-Fehler auf Server protokollieren (forts.)

- neues JSErrorLog-Dokumente soll erstellt werden
- dazu wird CreateDocument-URL-Befehl genutzt
- neue Items werden übergeben, weitere in berechneten Feldern in der Maske berechnet

- im Browser sehr einfach zu implementieren
- serverseitig nichts zu tun
- aber $$Return mit sinnvoller Rückgabe wäre hilfreich für Fehlerprüfung

assono

Ansicht auf neue Dokumente überwachen

- Es soll eine Ansicht überwacht werden.
- Wenn dort neue Dokumente erscheinen, soll der aktuelle Benutzer benachrichtigt werden.

- Anfangs wird die Anzahl der Dokumente in der Ansicht ermittelt:

```
function getViewDocsCount() {
  var url = viewURL +
      '?ReadViewEntries&Count=0&OutputFormat=JSON';
  callServer('GET', url, true, 0, setInitialDocsCount,
      undefined, undefined, 'text/plain',
      [['Cache-Control',  'no-cache']]);
}
```


assono

Ansicht auf neue Dokumente überwachen (forts.)

- Dann werden regelmäßig Anfragen gesendet:

```
function getNewViewDocs() {
  var url = viewURL + '?ReadViewEntries&Start=' +
      (viewDocsCount + 1) + '&OutputFormat=JSON&PreFormat';
  callServer('GET', url, true, 60000, checkView, undefined,
  undefined, 'text/plain', [['Cache-Control', 'no-cache']]);
}
```

- Trick: Durch `Start=(viewDocsCount + 1)` werden nur **neue** Dokumente zurückgegeben.
  Beispiel: Es gab 3 Dokumente, dann werden nur das 4. und alle weiteren Dokumente zurückgegeben.
- Die Option `PreFormat` sorgt dafür, dass Zeit-/Datumswerte als String zurückgegeben werden.

assono

Ansicht auf neue Dokumente überwachen (forts.)

- Die folgende Funktion wird für jeden Datensatz in der Rückgabe einzeln aufgerufen und gibt ein paar Details zur neuen Fehlerprotokoll aus:

```
function doOnEachNewLogEntry(logEntry) {
   var timestamp = getColumnByName(logEntry, 'Timestamp').text['0'];
   var error = getColumnByName(logEntry, 'Error').text['0'];
   var message = getColumnByName(logEntry, 'Message').text['0'];
   var username = getColumnByName(logEntry, '$UserName').text['0'];
   alert('New JS log entry on server:\nTimestamp:\t' + timestamp +
        '\nError:\t\t     ' + error + '\nMessage:\t ' + message +
        '\nUser name:\t' + username);
}
```

- dabei ist `logEntry = response.viewentry[i]`

assono

Ansicht auf neue Dokumente überwachen (forts.)

- und `getColumnByName` eine kleine Hilfsfunktion, um per Name und nicht per Index auf den Spaltenwert zuzugreifen:

```
function getColumnByName(viewEntry, columnName) {
   var entryData = viewEntry.entrydata;
   for (i = 0; i < entryData.length; ++i) {
        var currentEntry = entryData[i];
        if (currentEntry['@name'] == columnName) {
               return currentEntry;
        }
   }
   return undefined;
}
```

assono

Ansicht auf neue Dokumente überwachen (forts.)

- Um die Ansicht erstmalig und regelmäßig abzufragen, wird wieder ein ReadViewEntries-URL-Befehl mit OutputFormat=JSON verwendet.
- Durch den `Start=(bisherigeAnzahl + 1)`-Trick werden nur dann Datensätze übertragen, wenn es neue gibt, sonst nur der Minimalsatz mit der Gesamtanzahl (wie bei `Count=0`).
- effizienter Aufruf und Datentransport
- keine serverseitige Programmierung notwendig
- leichte Navigation in der Rückgabe

assono

Demos

1. Abhängige Auswahllisten nachladen
2. Test auf Eindeutigkeit
3. Fortschrittsanzeige
4. Bearbeitbare Ansichten
5. JavaScript-Fehler auf Server protokollieren
6. Ansicht auf neue Dokumente überwachen

assono

Noch mehr ins Detail…

- In diesem Vortrag konnte ich viele Themen nur anreißen.
- Mehr ins Details gehen meine beiden Artikel in „The View"
- Ausgabe March/April 2007: „Enhance Web application performance and flexibility with these 10 AJAX/Domino techniques"
- Ausgabe May/June 2007: „Advanced examples of using AJAX"

assono

Zum guten Schluss

- Fragen?
  - jetzt stellen oder später
    - E-Mail: tbahn@assono.de
    - Telefon: 04307 900-401
- Folien und Beispiele unter http://www.assono.de/blog.nsf/d6plinks/EntwicklerCamp2007
- In eigener Sache – wir suchen Verstärkung:

  IT-Berater (m/w)

  Details unter http://www.assono.de/jobs.html