IBM Software

ConnectED2015

BP108: Be Open - Use WebServices and REST in XPages

Bernd Hort, assono GmbH

IBM

NOTES & DOMINO ENTWICKLERCAMP

assono

XPages - WebServices und REST

EntwicklerCamp
04. März 2015

Innovative Software-Lösungen.

www.assono.de

NOTES & DOMINO ENTWICKLERCAMP

assono

Bernd Hort

Diplom-Informatiker, Universität Hamburg

seit 1995 entwickle ich Lotus Notes Anwendungen

IBM Certified Application Developer,
System Administrator & Instructor

Sprecher auf diversen Konferenzen &
Lotusphere 2008/IBM Connect 2014/IBM ConnectED 2015

bhort@assono.de
http://www.assono.de/blog
040/73 44 28-315

assono
IT-Consulting & Solutions

Agenda

- Einführung / Motivation
- Web Services in Java konsumieren
- RESTful Web Services in Java konsumieren
- Web Services in JavaScript konsumieren
- RESTful Web Services in JavaScript konsumieren
- dojo Controls
- Fragen und Antworten

NOTES & DOMINO ENTWICKLERCAMP

assono

Agenda

- Einführung / Motivation
- Web Services in Java konsumieren
- RESTful Web Services in Java konsumieren
- Web Services in JavaScript konsumieren
- RESTful Web Services in JavaScript konsumieren
- dojo Controls
- Fragen und Antworten

Remote Systems

Language Independent

Internet Social

Standard

Be Open

API Application Programming Interface

Agenda

- Einführung / Motivation
- Web Services in Java konsumieren
- RESTful Web Services in Java konsumieren
- Web Services in JavaScript konsumieren
- RESTful Web Services in JavaScript konsumieren
- dojo Controls
- Fragen und Antworten

Web Services in a Nutshell

- Kommunikation zwischen zwei Maschinen
  - Web Service Consumer
  - Web Service Provider
- Über eine Netzwerkverbindung
  - Meistens mittels des HTTP-Protokolls
- Daten werden in XML kodiert
  - Nachrichten werden SOAP Envelops übermittelt
- Spezifiziert durch das World Wide Web Consortium (W3C)
- Beschreibung eines Web Service in WSDL
  - Web Service Description Language

Web Services in Java konsumieren

- Java EE 6 beinhaltet Klassen zum Konsumieren von Web Services:
  Java API for XML Web Services (JAX-WS)

- Das Apache Projekt CXF liefert ein Werkzeug zur Generierung von Java Klassen anhand einer WSDL Datei
  wsdl2java

- Die generierten Java Klassen ordnen durch Java Annotationen den Web Service Aktionen den Java Methoden zu mit Hilfe der Java Architecture for XML Binding - JAXB

- XPages Runtime basiert auf Java EE 6

Apache CXF – wsdl2java

- Download CXF von der Apache Projekt Seite
- Download der WSDL Datei von dem Remote System oder direkter Zugriff auf das Remote System
- Aufruf wsdl2java.bat mit den folgenden Parametern

-client	Um eine Beispiel Java Klasse für die Verwendung des Web Services zu generieren
-exsh true	Aktiviert oder deaktiviert die Verarbeitung von expliziten SOAP Headers (SOAP headers, die in dem wsdl:binding aber nicht im wsdl:portType definiert sind.)
-frontend jaxws21	Aktuell nur frontend JAX-WS 2.1 unterstützt
-verbose	Ausgabe von ausführlicheren Kommentaren bei der Generierung
-d {Ausgabepfad}	Der Dateipfad für die generierten Java Klassen
-p {Packagename}	Der Java Packagename für die generierten Java Klassen
{WSDL Pfad}	Der Dateipfad oder die URI der WSDL Datei

Anpassungen an der java.policy Datei notwendig!

- JAX-WS verwendet Java Reflection
- Ein Anpassung an der java.policy Datei auf dem IBM Domino Server ist notwendig, um die Verwendung eines anderen Class Loaders zuzulassen
  - {Domino Program Path}/jvm/lib/security/java.policy
- Die folgenden Zeilen müssen hinzugefügt werden

```
grant {
  permission java.lang.RuntimePermission
   "setContextClassLoader";
  permission java.lang.reflect.ReflectPermission
   "suppressAccessChecks";
};
```

Basic Authentication

- Basic Authentication auf der HTTP Ebene wird häufig für die Absicherung von Web Services verwendet

- JAX-WS unterstützt Basic Authentication

```
import javax.xml.namespace.QName;
import javax.xml.ws.BindingProvider;
import javax.xml.ws.Service;

protected static final QName SERVICE_NAME = new
QName(„{WebServiceTargetNamespace}", „{WebServiceName}");


Service service = {YourWebServiceImplementationClass}Service.create(wsdlURL,
SERVICE_NAME);
port = service.getPort({YourWebServiceClass}.class);


Map<String, Object> ctx = ((BindingProvider) port).getRequestContext();
ctx.put(BindingProvider.USERNAME_PROPERTY, username);
ctx.put(BindingProvider.PASSWORD_PROPERTY, password);
```

WSDL Zugrif bei Verwendung Basic Authentication

- Zur Laufzeit wird die WSDL Definition benötigt für den “Service Endpoint” bzw. die Web Service URL
- Problematisch wenn die WSDL selber durch Basic Authentication geschützt ist
- Als Workaround kann eine Notes Anwendung mit Anonymous Zugriff als WSDL Proxy fungieren
- Maske mit Content Type “text/xml”

Beispiel einer WSDL-Datei

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions xmlns:soap="http://schemas.xmlsoap.org/wsdl/
soap/"[…]>
  <types>[…]</types>
  <message>[…]</message>
  <portType>[…]</portType>
  <binding>[…]</binding>
  <service name=„{WebServiceImplementation}Service">
    <port name=„{WebService}Port"
      <soap:address location=„{URL of Web Service}"></
soap:address>
    </port>
  </service>
</definitions>
```

Agenda

- Einführung / Motivation
- Web Services in Java konsumieren
- RESTful Web Services in Java konsumieren
- Web Services in JavaScript konsumieren
- RESTful Web Services in JavaScript konsumieren
- dojo Controls
- Fragen und Antworten

REST in a Nutshell

- Representational state transfer (REST) ist ein Programmierstil für den Zugriff auf Ressourcen basierend auf Web Standards
- RESTful Web Services implementieren den Zugriff und die Veränderung von solchen Ressourcen
- Meistens mittels des HTTP-Protokolls
- Ressourcen können unterschiedliche Repräsentationen haben, z.B. text, xml, json etc.
  - Der Rest Client kann über das HTTP Protokoll (content negotiation) nach einer bestimmten Repräsentation fragen.
- Kein Standard!

WADL

- Das WSDL Equivalent
  Web Application Description Language (WADL)
  ist optional

- Aufruf, Parameter und Rückgabewerte müssen der Dokumentation entnommen werden

- Hoffentlich ist die Dokumentation gut ;-)

NOTES & DOMINO ENTWICKLERCAMP

assono

REST in a Nutshell – Nomen und Verben

- Ressourcen werden über eine URI identifiziert - Nomen
- Aktionen auf den Ressourcen werden durch HTTP Methoden bestimmt - Verben

Ressource	Get	Post	Put	Delete
Collection von Ressourcen http://yourserver/path/sessions	**Listed** alle Ressourcen auf	**Erzeugt** ein neuen Eintrag in der Collection	**Tauscht** die gesamte Collection durch eine andere Collection aus.	**Löscht** die gesamte Collection
Einzelne Ressource http://yourserver/path/sessions/BP206	**Zeigt** die Details einer bestimmten Ressource an	Normalerweise nicht implementiert	**Akualisiert** den adressierte Eintrag in der Collection, oder **erzeugt** ihn, wenn er nicht existiert.	**Löscht** die spezifisierte Ressource

Auch bekannt als create, read, update and delete (CRUD)

RESTful Web Services in Java konsumieren

- Java EE 7 beinhaltet Klassen zum Konsumieren von RESTful Web Services:
  Java API for RESTful Services (JAX-RS)
- XPages Runtime basiert auf Java EE 6 😟
- Sun / Oracles Reference Implementierung
  Jersey stellte alle notwendigen Java Klassen zur Verfügung
  - Jersey version 1.17.1 ist die letzte stabile Version die Java 6 und JAX-RS 1.1 unterstützt

Jersey

RESTful Web Services in Java konsumieren - XML

- Wenn der RESTful Web Service XML unterstützt, verwendet Jersey einfach die Java Architecture for XML Binding - JAXB um die XML-Elemente Java-Klassen zuzuweisen

- Zur Laufzeit wird entweder nach gleichen Property Namen oder nach zugehörigen Java Annotations gesucht

Jersey

Beispiel RESTful Web Services in Java

- Repräsentation der Daten als JSON

```
[{
  sessionId: "BP206",
  title: "Be Open - Use Web Services and REST
          in XPages Applications",
  description: "...",
  speakers: [{
          name: "Bernd Hort",
          company: "assono GmbH"}]
},
{
 sessionId: "BOF201",
 title: "XPages and Java: Share Your Experiences",
 description: "...",
 speakers: [{
          name: "Bernd Hort",
          company: "assono GmbH"}]
}]
```

Beispiel RESTful Web Services in Java

- Java Klasse

```
import javax.xml.bind.annotation.XmlRootElement;

@XmlRootElement
public class Speaker {


  private String name;
  private String company;

  // Getter and setter methods
  // […]
}
```

```
@XmlRootElement
public class Session {


  private String sessionId;
  private String title;
  private String description;
  private Speaker[] speaker;

  // Getter and setter methods
  // […]
}
```

Beispiel RESTful Web Services in Java

- Java Klasse für Verwendung des Web Services

```
import java.net.URI;

import javax.ws.rs.core.MediaType;
import javax.ws.rs.core.UriBuilder;

import com.sun.jersey.api.client.Client;
import com.sun.jersey.api.client.WebResource;
import com.sun.jersey.api.client.config.ClientConfig;
import com.sun.jersey.api.client.config.DefaultClientConfig;

public class RESTClient {
  public static Session[] getSessions() {
    ClientConfig config = new DefaultClientConfig();
    Client client = Client.create(config);
    WebResource service = client.resource(getBaseURI());

    return
service.path("sessions").accept(MediaType.TEXT_XML).get(Session[].class);
  }

  private static URI getBaseURI() {
    return UriBuilder.fromUri("http://{server}/{applicationpath}").build();
  }
}
```

RESTful Web Services in Java konsumieren - JSON

- Statt JAXB zu verwenden, um XML auf Java Klassen zu „mappen“ wird ein „JSON to Java Mapper“ benötigt
- Jersey 1.17.1 beinhaltet Jackson 1.9.2 von Codehouse.org
- Letztens umgezogen zu https://github.com/FasterXML/jackson

Jersey

Beispiel RESTful Web Services in Java

- Java Klasse für Verwendung des Web Services

```
import [...]

import org.codehaus.jackson.map.ObjectMapper;

public class RESTClient {
  public static Session[] getSessions() {
    ClientConfig config = new DefaultClientConfig();
    Client client = Client.create(config);
    WebResource service = client.resource(getBaseURI());

    String json =
      service.path("Sessions").accept(MediaType.APPLICATION_JSON).get(String.class);


    ObjectMapper mapper = new ObjectMapper();
    sessions = mapper.readValue(json, Session[].class);

    return sessions;
  }

  private static URI getBaseURI() {
    return UriBuilder.fromUri("http://{server}/{applicationpath}").build();
  }
}
```

Anpassung der java.policy Datei notwendig!

- Jersy verwendet einen anderen Class Loader
- Also benötigen wir ein weitere Anpassung der java.policy Datei auf dem IBM Domino Server
  - {Domino Program Path}/jvm/lib/security/java.policy
- Die folgende Zeile muss hinzugefügt werden

```
grant {
  permission java.lang.RuntimePermission
   "setContextClassLoader";
  permission java.lang.RuntimePermission "getClassLoader";
  permission java.lang.reflect.ReflectPermission
   "suppressAccessChecks";
};
```

Agenda

- Einführung / Motivation
- Web Services in Java konsumieren
- RESTful Web Services in Java konsumieren
- Web Services in JavaScript konsumieren
- RESTful Web Services in JavaScript konsumieren
- dojo Controls
- Fragen und Antworten

Web Services in JavaScript konsumieren

- Historisch gesehen ist das Konsumieren eines Web Services die Basis für alle AJAX Requests
  - Die JavaScript Klasse XMLHttpRequest kümmert sich um die Kommunikation mit einem Server von einer Web Seite aus
- Erstaunlicherweise gibt es kaum Unterstützung für den Aufruf von Web Services
  - Keine Veränderung in Sicht, weil die meisten Server inzwischen REST Schnittstellen anbieten
- Der SOAP Envelope muss manuell zusammengestellt werden und via XMLHttpRequest zum Server geschickt werden
- Das XML-Antwort muss geparst werden

```
<script type="text/javascript">
    function soap() {
      var xmlhttp = new XMLHttpRequest();
      xmlhttp.open('POST', 'https://somesoapurl.com/', true);

      // build SOAP request
      var sr =
        '<?xml version="1.0" encoding="utf-8"?>' +
        '<soapenv:Envelope ' +
          'xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" ' +
          'xmlns:api="http://{demo web service}" ' +
          'xmlns:xsd="http://www.w3.org/2001/XMLSchema" ' +
          'xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">' +
          '<soapenv:Body>' +
            '<api:some_api_call soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">' +
              '<username xsi:type="xsd:string">login_username</username>' +
              '<password xsi:type="xsd:string">password</password>' +
            '</api:some_api_call>' +
          '</soapenv:Body>' +
        '</soapenv:Envelope>';

      xmlhttp.onreadystatechange = function () {
        if (xmlhttp.readyState == 4) {
          if (xmlhttp.status == 200) {
            alert('done - use firebug to see response');
          }
        }
      }
      // Send the POST request
      xmlhttp.setRequestHeader('Content-Type', 'text/xml');
      xmlhttp.send(sr);
      // send request
      // ...
    }
</script>
```

Wie erstellt man einen SOAP Envelope?

- Ein guter Weg einen SOAP Envelope zu erstellen ist die Verwendung eines Testwerkzeuges.
- Im Testwerkzeug mittels der WSDL Test Cases generieren
- Aufruf des Web Services mit Beispieldaten und dann den SOAP Envelope kopieren
- Eine gute Wahl ist SoapUI

- Oder der Web Service Explorer von Eclipse
  - Java EE Perspective öffnen
  - Menü Run\Launch the Web Service Explorer
  - Zur WSDL Seite wechseln
  - Beispieldaten eingeben
  - Auf “Source” klicken

dojo Support

- dojo hat keine direkte Unterstützung für SOAP Web Services
- Allerdings hat dojo eine Menge Hilfsmethoden
  - Das dojo base Objekt
    - dojo.formToJson: Konvertiert eine DOM Form nach JSON.
    - dojo.formToObject: Konvertiert eine DOM Form in ein JavaScript Objekt
    - dojo.formToQuery: Konvertiert eine DOM Form in einen Query String
    - dojo.objectToQuery: Konvertiert ein JavaScript Objekt in einen Query String
    - dojo.queryToObject : Konvertiert ein Query String in ein JavaScript Objekt

dojo Support

- dojo.xhr behandelt AJAX I/O Transporte und hat Helfer Methoden
  - dojo.xhrDelete: Benutzt die HTTP DELETE Methode für einen xhr Aufruf
  - dojo.xhrGet: Benutzt die HTTP GET Methode für einen xhr Aufruf
  - dojo.xhrPost: Benutzt die HTTP POST Methode für einen xhr Aufruf
  - dojo.xhrPut : Benutzt die HTTP PUT Methode für einen xhr Aufruf

dojo Support

- dojox.rpc kommuniziert mittels Remote Procedure Calls (RPC) mit Backend Servern
- dojox.rpc.Service ist die Basis für dojox.RPC
- Eine Service Mapping Description (SMD) ist eine JSON Representation einer Beschreibung eines Web Service Aufrufes und wird von dojox.rpc.Service benutzt.
  - Dojo wird mit vordefinierten SMDs ausgeliefert für
    - Google
    - Twitter
    - Yahoo!
    - Wikipedia

dojoX

Agenda

- Einführung / Motivation
- Web Services in Java konsumieren
- RESTful Web Services in Java konsumieren
- Web Services in JavaScript konsumieren
- RESTful Web Services in JavaScript konsumieren
- dojo Controls
- Fragen und Antworten

RESTful Web Services in JavaScript konsumieren

- Entweder man geht den umständlichen Weg und verwendet die JavaScript Klasse XMLHttpRequest direkt
- Oder man verwendet dojo
  - dojo.xhrDelete: Benutzt die HTTP DELETE Methode für einen xhr Aufruf
  - dojo.xhrGet: Benutzt die HTTP GET Methode für einen xhr Aufruf
  - dojo.xhrPost: Benutzt die HTTP POST Methode für einen xhr Aufruf
  - dojo.xhrPut : Benutzt die HTTP PUT Methode für einen xhr Aufruf

dojo.xhrGet

- dojo.xhrGet erwartet ein Parameter Objekt mit den folgenden Eigenschaften

Parameter	Beschreibung
url	Die URL für die Abfrage der Daten
handleAs	Das erwartete Format der Daten. Aktuell werden die folgenden Optionen unterstützt: ▪text (default) ▪json ▪json-comment-optional ▪json-comment-filtered ▪javascript ▪xml

dojo.xhrGet – Optionale Parameter

Parameter	Beschreibung
sync	Boolean – bei `true` wird der Browser während des Requests blockiert. Default ist false.
preventCache	Boolean – bei `true` hängt dojo einen sich bei jedem Request ändernden Parameter an die Anfrage um das Cachen im Browser zu verhindern
content	Ein JavaScript Objekt mit Name-Wert-Paaren – dojo wird die Werte codieren und an die Query anhängen z.B. ?key1=value1&key2=value2&key3=value3
headers	Ein JavaScript Objekt mit Name-Wert-Paaren – die Werte werden als Header beim Request mitgesendet.
timeout	Milliseconds bevor ein Timeout des Requests ausgelöst wird Default ist ‘0’, was gleichzusetzen ist mit unendlich (kein Timeout)
user	Username für Basic Web Authentication
password	Passwort für Basic Web Authentication

dojo

dojo.xhrGet – Parameter für die Verarbeitung der Daten

Parameter	Beschreibung
load	Eine JavaScript Funktion, die aufgerufen wird, wenn die Daten vom Server eingetroffen sind. Die Daten werden als erster Parameter an die Funktion übergeben. Das Format der Daten wird durch den Parameter handleAs bestimmt.
error	Eine JavaScript Funktion, die im Fehlerfall aufgerufen wird. Der erste Parameter beinhaltet ein JavaScript Fehler Objekt mit der Art des Fehlers.
handle	Eine JavaScript Funktion, die unabhängig von Erfolg oder Fehler aufgerufen wird. Der erste Parameter enthält die Rückantwort bzw. den Fehler. Der zweite Parameter ist das IO Objekt, welches unter anderem den Status beinhaltet.

dojo

dojo.xhrGet - Beispiel

```
var xhrArgs = {
    url: "http://www.example.com/someservice/sessions/BP206",
    handleAs: "json",
    load: function(data){
      dojo.byId("#{id:titleSpan1}").innerHTML = data.title;
    },
    error: function(error){
      dojo.byId("#{id:titleSpan1}").innerHTML =
      "An unexpected error occurred: " + error;
    }
  };
dojo.xhrGet(xhrArgs);
```

dojo

dojo.xhrPost

- dojo.xhrPost akzeptiert die gleichen Parameter wie dojo.xhrGet mit den folgenden Erweiterungen

Parameter	Description
content	Ein JavaScript Objekt mit Name-Wert-Paaren – xhrPost konvertiert das Objekt in das POST Format und sendet es mit dem Post Body. Dieser Parameter wird anders gehandhabt als in dojo.xhrGet, wo der Inhalt als Query String in der URL übertragen wird.
form	Um die Werte eines HTML Formulars zu übertragen, kann entweder der DOM Knoten des Formular oder die ID übergeben werden. xhrPost konvertiert dieses in das POST Format und sendet es mit dem Post Body. Wenn die URL nicht als Parameter übergeben wurde, wird versucht sie aus dem „Action“ Attribute des Formulars auszulesen.
postData	Ein String der unverändert als Post Body übertragen wird. dojo.xhrPost (und dojo.rawXhrPost) nehmen keine Konvertierung vor.

Nur einer dieser Parameter kann in einem Aufruf verwendet werden!

dojo

dojo.xhrPut

- dojo.xhrPut akzeptiert die gleichen Parameter wie dojo.xhrGet mit den folgenden Erweiterungen

Parameter	Description
content	Ein JavaScript Objekt mit Name-Wert-Paaren – xhrPost konvertiert das Objekt in das PUT Format und sendet es mit dem Put Body. Dieser Parameter wird anders gehandhabt als in dojo.xhrGet, wo der Inhalt als Query String in der URL übertragen wird.
form	Um die Werte eines HTML Formulars zu übertragen, kann entweder der DOM Knoten des Formular oder die ID übergeben werden. xhrPut konvertiert dieses in das PUT Format und sendet es mit dem Put Body. Wenn die URL nicht als Parameter übergeben wurde, wird versucht sie aus dem „Action“ Attribute des Formulars auszulesen.
postData	Ein String der unverändert als Post Body übertragen wird. dojo.xhrPut (und dojo.rawXhrPut) nehmen keine Konvertierung vor.

Nur einer dieser Parameter kann in einem Aufruf verwendet werden!

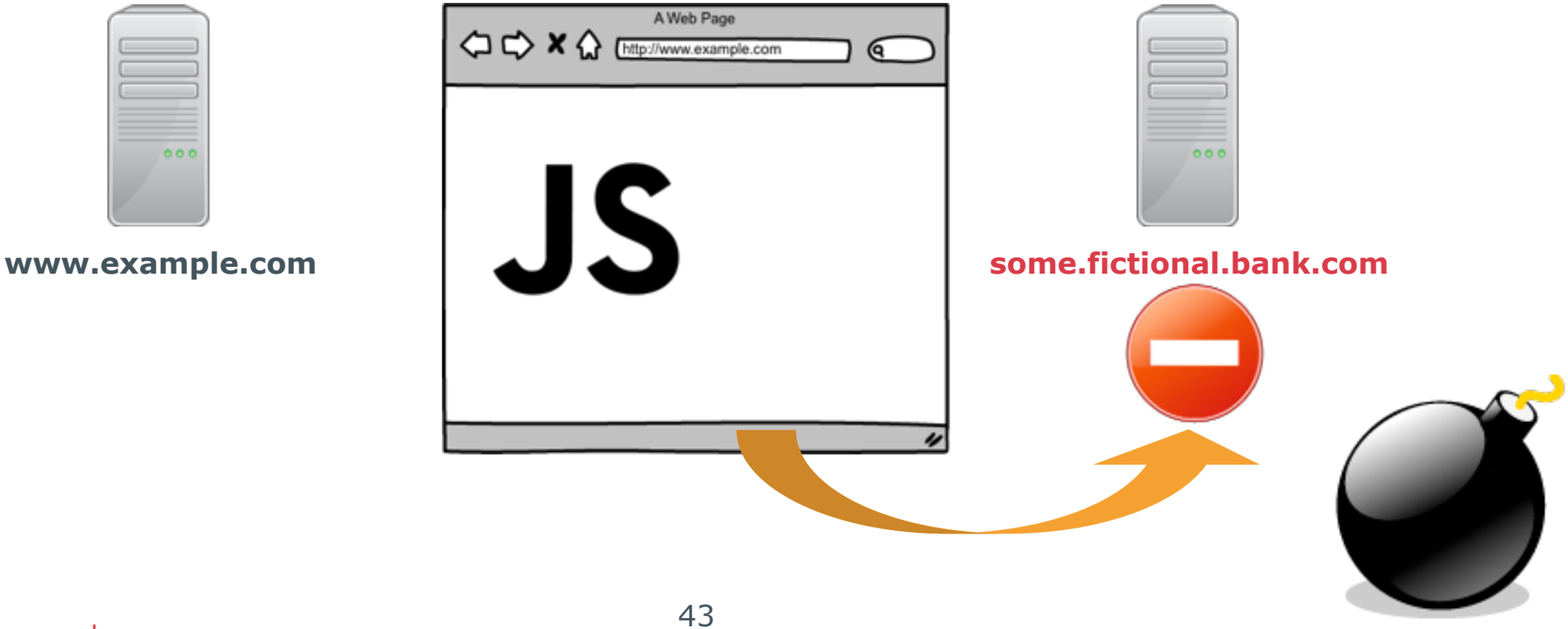
dojo

dojo.xhrDelete

- dojo.xhrDelete akzeptiert die gleichen Parameter wie dojo.xhrGet.
- Es gibt keinen content Parameter in dojo.xhrDelete, weil die URI (URL + Query Parameter) definiert was gelöscht werden soll.

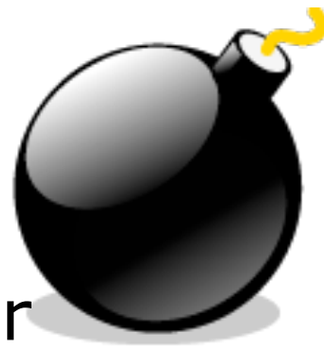
Same Origin Policy - SOP

- Aus Sicherheitsgründen darf ein Script im Browser nur auf Ressourcen zugreifen, die vom gleichem Server stammen – Same Origin.
- Diese Beschränkung gilt auch für Web Services Aufrufe



JSONP – JSON with padding

- Die Same Origin Policy gilt nicht für JavaScript tags
  - Die JavaScript Sourcen können von jedem Server geladen werden
- JSONP benutzt die URI eines eingebetteten JavaScript Tags, um einen Request zu senden
- Die Daten werden als Funktionsaufruf zu einer existierenden JavaScript Funktion zurückgeliefert
- Die Technik funktioniert nur mit GET Requests
- Potenzielles Sicherheitsrisiko: Welche Daten auch immer von dem „anderen“ Server zurückgeliefert werden, werden als JavaScript Funktion im Kontext des Browsers ausgeführt



- dojo unterstützt JSONP mit dem dojo.io.script Package

JSONP – JSON with padding

JSON-Data als Rückgabewert

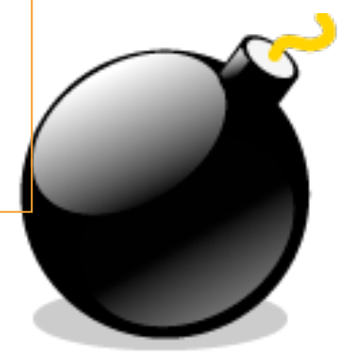
```
{
  sessionId: "BP206",
  title: "Be Open - Use Web Services and REST in XPages Applications",
  description: "...",
  speaker: {
            name: "Bernd Hort",
            company: "assono GmbH"}
 }
```

Aufruf, um die Daten zu bekommen

```
<script type="application/javascript"
src="http://{someserver}/Sessions/BP206?jsonp=updateSessionDetails">
</script>
```

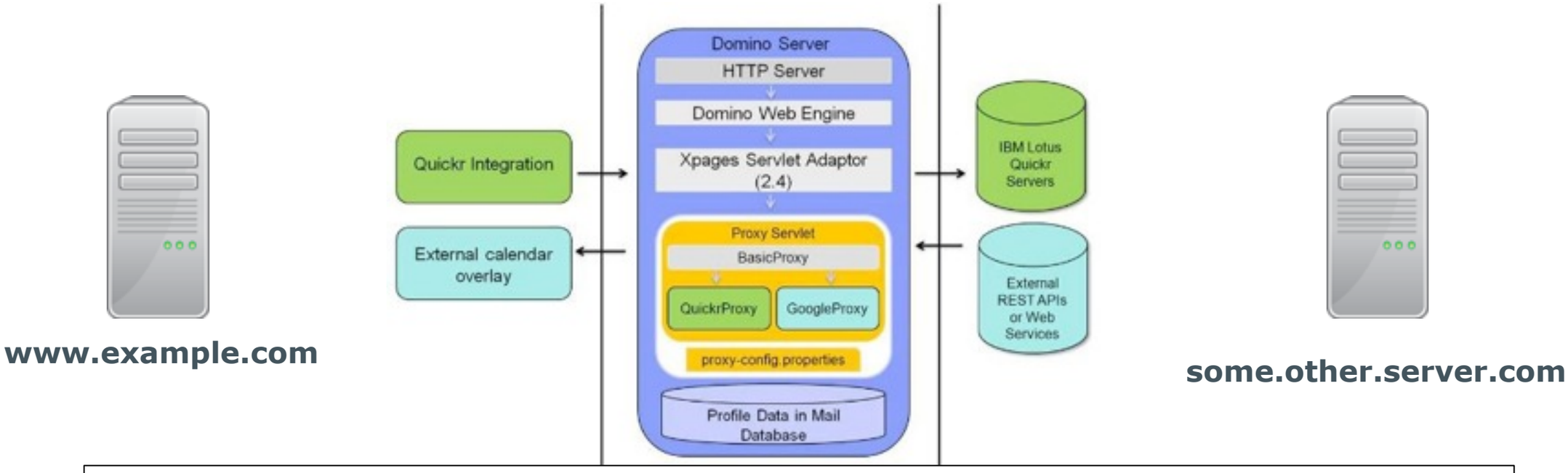
Zurückgeliefertes JavaScript

```
updateSessionDetails({
  sessionId: "BP206",  title: "Be Open - Use Web Services and REST in
XPages
  Applications",  description: "...",   speaker: {name: "Bernd Hort",
  company: "assono GmbH"}
 })
```



IBM® iNotes® Proxy Servlet

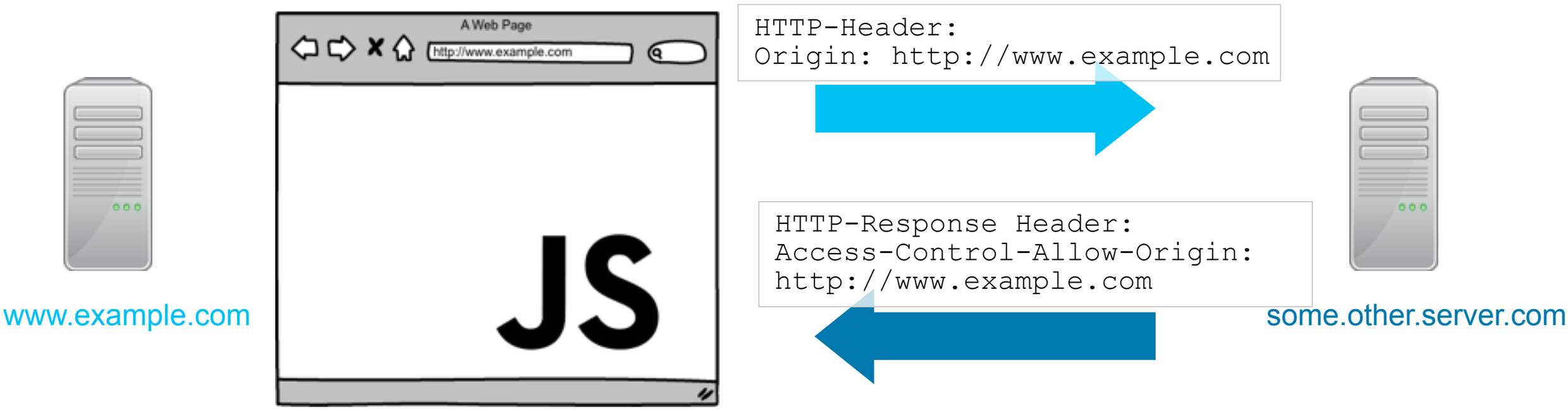
- In IBM® Lotus® iNotes® 8.5 wurde ein Proxy Servlet eingeführt, um das SOP Problem zu umgehen.
- Ein Request, welches durch den Proxy geht, hat aus Sicht des Browsers den gleichen Ursprung wie der Rest der Seite
- Details werden beschrieben in
  New features in IBM Lotus iNotes 8.5: Full mode



http://www.example.com/BasicProxy/http/some.other.server.com?param1=value1

Cross-origin resource sharing – CORS

- Beim Cross-origin resource sharing – CORS wird der Ursprung des Aufrufes mit im Header übertragen
- Der angefragte Server übergibt im Response Header, ob er die „Origin Domain“ akzeptiert



Cross-origin resource sharing – CORS

- CORS funktioniert mit allen HTTP Methoden:
  GET, POST, PUT, DELETE …
- CORS wird in allen modernen Browsern unterstützt (from Wikipedia)
  - Gecko 1.9.1 (Firefox 3.5, SeaMonkey 2.0, Camino 2.1) und höhere.
  - WebKit (Erste Version ungewiss, Safari 4 und höher, Google Chrome 3 und höher)
  - MSHTML/Trident 6.0 (Internet Explorer 10) hat native Unterstützung. MSHTML/Trident 4.0 & 5.0 (Internet Explorer 8 & 9) unterstützt es zum Teil mit dem XDomainRequest Objekt.
  - Presto-based Browsers (Opera) unterstützen CORS ab Opera 12.00 und Opera Mobile 12, aber nicht Opera Mini.

NOTES & DOMINO ENTWICKLERCAMP

assono

Agenda

- Einführung / Motivation
- Web Services in Java konsumieren
- RESTful Web Services in Java konsumieren
- Web Services in JavaScript konsumieren
- RESTful Web Services in JavaScript konsumieren
- dojo Controls
- Fragen und Antworten

dojo Store API

- Die dojo Store API ist eine abstrakte API für das Handling von Daten basierend auf der HTML5/W3C's IndexedDB object store API

Methode	Beschreibung
get(id)	Liefert ein Objekt anhand seiner Id zurück
query(query, options)	Durchsucht den Store anhand einer Suchabfrage Optionen beinhalten ▪start – Start Offset ▪count – Anzahl der zurückzugebenden Objekte ▪sort – Sortierung anhand der Dojo Data Sortier Definition ▪queryOptions – Abfrage Optionen anhand der Dojo Data queryOptions Definition
put(object, options)	Speicher das übergebene Objekt. options.id (optional) spezifiziert die Id.
add(object, options)	Erzeugt ein neues Objekt. options.id (optional) spezifiziert die Id.
remove(id)	Löscht ein Objekt anhand seiner Id.

dojo Store API (continue)

Methode	Beschreibung
getIdentity(object)	Liefert die Id eines Objektes zurück
queryEngine(query, options)	Erstellt anhand der Suchabfrage inkl. den Optionen ein Suchobjekt, welches auf ein JavaScript Array angewandt werden kann. Das zurückgelieferte Objekt hat „eventuell“ eine “matches” Methode, mit der überprüft werden kann, ob ein Objekt die Suchkriterien erfüllt.
transaction()	Startet eine Transaktion und erzeugt ein Transaktionsobjekt. Das Transaktionsobjekt sollte die folgenden Methoden beinhalten: commit() – Commited alle während der Transaktion durchgeführten Änderungen. abort() – Setzt alle Änderungen während der Transaktion wieder zurück. Ein Anwender des Stores kann eventuell die Methoden put, delete, etc. aufrufen, ohne ein Transaktionsobjekt zu erzeugen. In dem Fall werden die Aktionen direkt als quasi “auto-commit” ausgeführt.
getChildren(object, options)	Liefert alle Kindobjekte eines Objektes zurück. Das Optionsobjekt unterstützt alle Eigenschaften, die auch das Query-Options-Objekt unterstützt.
getMetadata(object)	Liefert alle verfügbaren Metadaten über das Objekt zurück. Das kann Attribute, Cache Anweisungen, Historien und Versions Informationen beinhalten.

dojo.store.JsonRest

- Der dojo.store.JsonRest übernimmt die gesamte HTTP/REST Kommunikation mit einem Server basierend auf den HTTP Standards inklusive der GET, PUT, POST, und DELETE Methoden

- Er implementiert die dojo Store API

- Ein JsonRest Store kann mit einer URL instanziert werden

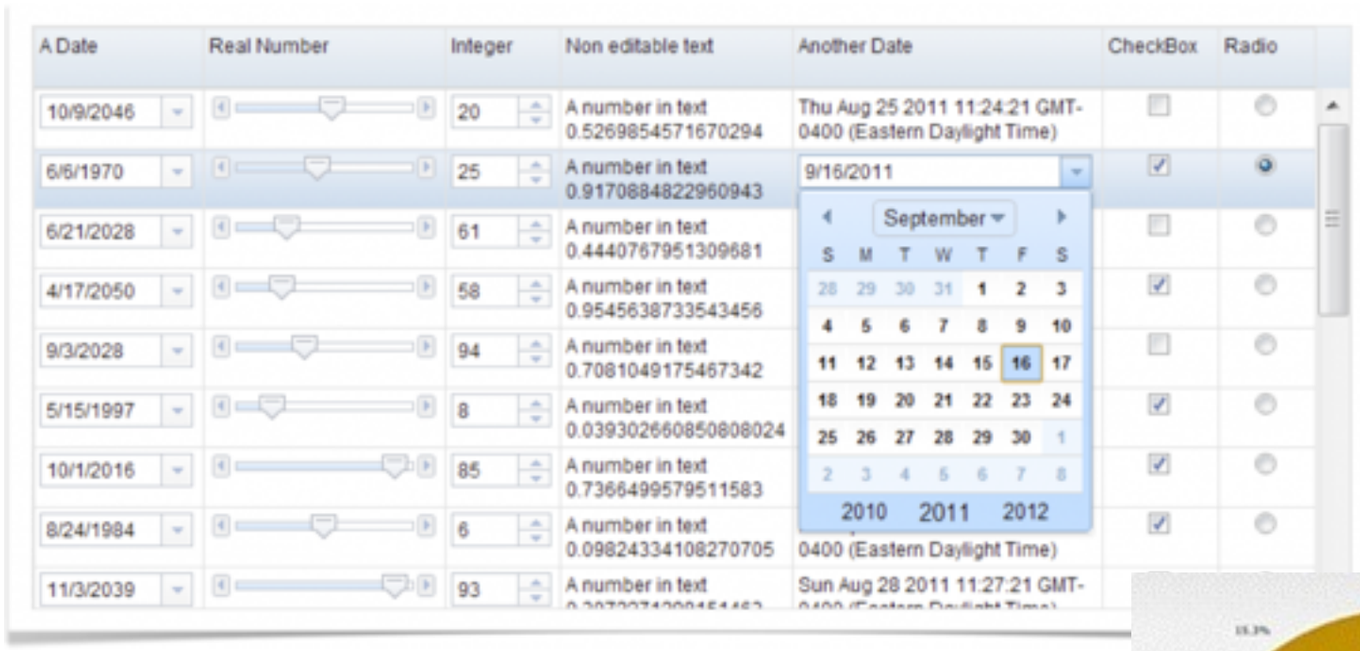
```
var store = new dojo.store.JsonRest({
    target:"/Sessions/",
    idProperty:"sessionId"}
);
```

- Paging wird unterstützt

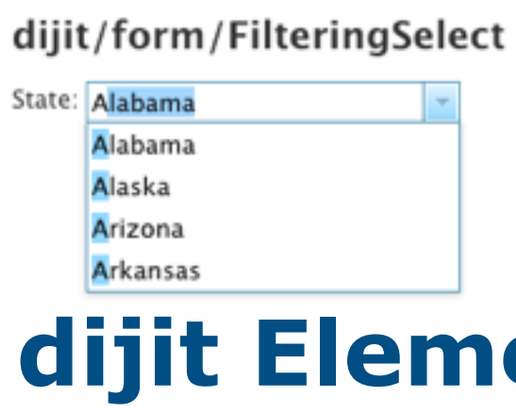
- Sortierung wird unterstützt

Verwendung von dojo Stores

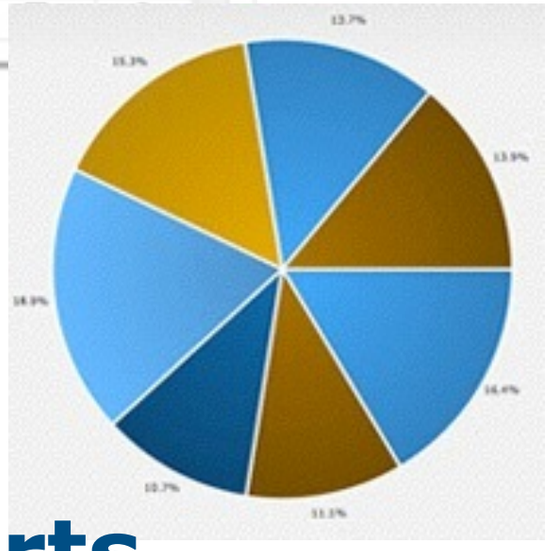
- dojo Data Stores kann verwendet werden für



dgrids



dijit Elemente

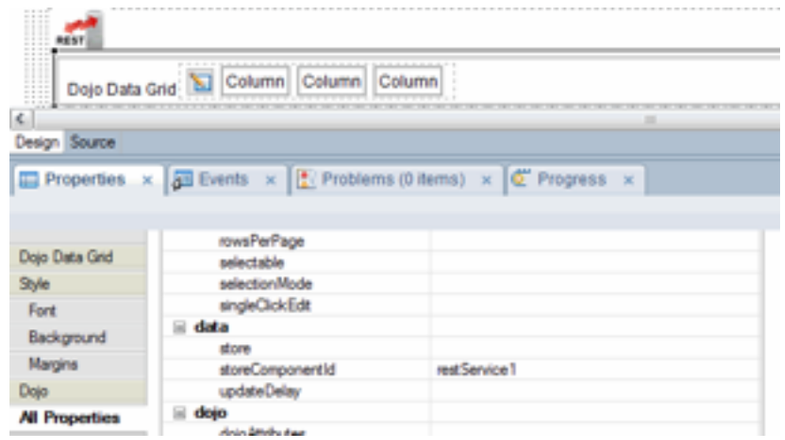


charts

dojo

XPages Extension Library

- Die XPages Extension Library hat Komponenten die RESTful Web Services entweder zur Verfügung stellen oder konsumieren
- Komponenten die RESTful Web Services stellen sind
  - Domino Data Services
  - RESTService control
  - Custom Database Servlet
  - Custom Wink Service



- Controls die RESTful Web Services konsumieren sind
  - Dojo Data Grid

openntf

Fragen?

jetzt stellen – oder später:

bhort@assono.de

http://www.assono.de/blog

040/73 44 28-315

assono
IT-Consulting & Solutions

Folien & Beispieldatenbank unter
http://www.assono.de/blog/d6plinks/EntwicklerCamp-2015-XPages-WebServices