IBM Software

ConnectED2015

IBM Software

ConnectED2015

BP108: Be Open - Use WebServices and REST in XPages

Bernd Hort, assono GmbH

Agenda

- Introduction / Motivation
- Consume Web Services from Java
- Consume RESTful Web Services from Java
- Consume Web Services from JavaScript
- Consume RESTful Web Services from JavaScript
- dojo Controls
- Questions and Answers

INTRODUCTION / MOTIVATION

Bernd Hort
assono GmbH

- IBM® Notes® Developer/Designer since Notes 3.3
- OpenNTF Contributer: assono Framework 2
  - An OOP LotusScript Framework
- Lately specialized in XPages development
- Working for assono GmbH, Germany
- Blog http://blog.assono.de
- Twitter @BerndHort

Remote Systems

Language

Internet

Standard

Be Open

Independent

Social

API

Application
Programming
Interface

CONSUME WEB SERVICES FROM JAVA

Web Services in a Nutshell

- Communication between two maschines
  - Web Service Consumer and Web Service Provider
- Over a netwok
  - Mostly using HTTP
- Data encoded as XML
- Messages transfered in SOAP envelops
- Specified by the World Wide Web Consortium (W3C)
- Definition of a Web Service in WSDL
  - Web Service Description Language

Consume Web Services in Java

- Java EE 6 has built-in classes to consume Web Services:
  Java API for XML Web Services (JAX-WS)
- The Apache project CXF provides a tool to generate Java files from a WSDL file
  wsdl2java
- The generated Java classes maps through Java Annotations the Web Service actions to Java methods using
  Java Architecture for XML Binding - JAXB
- XPages runtime builds on Java EE 6

Apache CXF – wsdl2java

- Download CXF from the Apache project site
- Download the wsdl file from the remote system or access it directly from the remote system
- Call the wsdl2java.bat file with the following parameters

`-client`	To generate a sample Java class calling the Web Service
`-exsh true`	Enables or disables processing of implicit SOAP headers (i.e. SOAP headers defined in the wsdl:binding but not wsdl:portType section.)
`-frontend jaxws21`	Currently only supported frontend JAX-WS 2.1
`-verbose`	Displays comments during the code generation process.
`-d {output directory}`	The path for the generated Java classes.
`-p {package name}`	Java package name for the generated Java classes
`{wsdl location}`	The file path or URI for the WSDL file

Modification of java.policy file needed!

- JAX-WS uses Java Reflection
- A modification of the java.policy file on the IBM Domino Server is needed to allow a different class loader
  - {Domino Program Path}/jvm/lib/security/java.policy
- The following lines have to be added

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

Basic Authentication

- Basic Authentication on the HTTP level is widely used for securing Web Services
- JAX-WS supports 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 file protected with Basic Authentication

- At runtime the WSDL definition is needed for the “Service Endpoint” a.k.a. the Web Service URL
- Sometimes the WSDL itself is protected with Basic Authentication
  - Workaround is a Notes application with anonymous access as a WSDL proxy
  - Form with content type “text/xml”

```
<?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>
```

CONSUME RESTful WEB SERVICES FROM JAVA

REST in a Nutshell

- Representational state transfer (REST) is an architectural style bases on web standards for accessing resources
- RESTful Web Services implements access and modification of resources
- Mostly using HTTP
- Resources may have different representations, e.g. text, xml, json etc.
  - The rest client can ask for specific representation via the HTTP protocol (content negotiation).
- No standard!
- The WSDL equivalent Web Application Description Language (WADL) is mostly optional
  - Hopefully the documentation is good ;-)

REST in a Nutshell – Nouns and Verbs

- Resources are identified via a URI - Nouns
- Actions on resources uses the HTTP methods - Verbs

Resource	***Get***	***Post***	***Put***	***Delete***
Collections of resources http://yourserver/path/sessions	**Lists** all resources	**Create** a new entry in the collection.	**Replace** the entire collection with another collection.	**Delete** the entire collection
Single resource http://yourserver/path/sessions/BP206	**Retrieve** the details of a given resource	*Normally not implemented*	**Replace** the addressed member of the collection, or if it doesn't exist, **create** it.	**Delete** the specified resource

Also known as create, read, update and delete (CRUD)

Consume RESTful Web Services in Java

- Java EE 7 has built-in classes to consume RESTful Web Services: Java API for RESTful Services (JAX-RS)
- XPages runtime builds on Java EE 6 😟
- Sun / Oracles reference implementation Jersey provides all needed Java classes
  - Jersey version 1.71.1 is the last stable version supporting Java 6 and JAX-RS 1.1

Jersey

Consume RESTful Web Services in Java - XML

- If the RESTful Web Service supports XML, Jersey just uses Java Architecture for XML Binding - JAXB to map the XML to Java classes
- At runtime Java classes provided with Java Annotations and the same property names are mapped to the data from Web Services

Example RESTful Web Services in Java

- Representation data as JSON

```
[{
  sessionId: "BP108",
  title: "Be Open - Use Web Services and REST in XPages Applications",
  description: "...",
  speakers: [{
         name: "Bernd Hort",
         company: "assono GmbH"}]
 },
 {
  sessionId: "Chalk405",
  title: "XPages and Java: Share Your Experiences",
  description: "...",
  speakers: [{
         name: "Bernd Hort",
         company: "assono GmbH"}]
 }]
```

Example RESTful Web Services in Java

- Java Class

```
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
  // […]
}
```

Example RESTful Web Services in Java

- Java Class for accessing the Web Service

```java
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();
  }
}
```

Consume RESTful Web Services in Java - JSON

- Instead of using JAXB to map XML to Java classes use a JSON to Java mapper
- Jersey 1.71.1 comes with Jackson 1.9.2 from Codehouse.org
  - Recently moved to https://github.com/FasterXML/jackson

Example RESTful Web Services in Java

- Java Class for accessing the Web Service

```
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();
  }
}
```

Modification of java.policy file needed!

- Jersy uses a different class loader
- So another modification of the java.policy file on the IBM Domino Server is needed to allow a different class loader
  - {Domino Program Path}/jvm/lib/security/java.policy
- The following lines needs to be added

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

CONSUME WEB SERVICES FROM JAVASCRIPT

Consume Web Services from JavaScript

- Historically is consuming a Web Services the base for all AJAX style techniques
  - The JavaScript class XMLHttpRequest handles the communication with a server from within a web site
- Astonishingly the support for calling Web Services is still mostly hand coding
  - No efforts are taken because most servers offers REST nowadays
- The SOAP Envelope must be built and send to a server using XMLHttpRequest
- The returning XML has to be parsed

```
<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>
```

How to get a SOAP Envelope?

- A good way to get the SOAP Envelope needed for a specific Web Service is
  using a test tool
- Point the test tool to the WSDL to let it generate the test cases
- Invoke the Web Service with sample data and copy the SOAP Envelope
- A good choice is SoapUI
- Or use the Web Service Explorer from Eclipse
  - Open the Java EE perspective
  - Menu Run\Launch the Web Service Explorer
  - Change to the WSDL page
  - Fill in the sample data
  - Click on “Source”

dojo Support

- dojo may not have direct support for consuming SOAP Web Services
- But dojo has a lot of helpers to offer
  - the dojo base object
    - dojo.formToJson : Convert a DOM Form to JSON.
    - dojo.formToObject : Convert a DOM Form to a JavaScript object.
    - dojo.formToQuery : Convert a DOM Form to a query string.
    - dojo.objectToQuery : Convert a JavaScript object to a query string.
    - dojo.queryToObject : Convert a query string to a JavaScript Object
  - dojo.xhr offers AJAX I/O transports and utility methods
    - dojo.xhrDelete : Use HTTP DELETE method to make an xhr call.
    - dojo.xhrGet : Use HTTP GET method to make an xhr call.
    - dojo.xhrPost : Use HTTP POST method to make an xhr call.
    - dojo.xhrPut : Use HTTP PUT method to make an xhr call.

dojo Support

- dojox.rpc communicate via Remote Procedure Calls (RPC) with Backend Servers
- dojox.rpc.Service is the foundation of most dojox.RPC transportation
- A Service Mapping Description (SMD) is a JSON representation describing web services and it is used by dojox.rpc.Service.
  - Dojo comes with predefined SMDs for accessing
    - Google
    - Twitter
    - Yahoo!
    - Wikipedia

CONSUME RESTful WEB SERVICES FROM JAVASCRIPT

Consume RESTful Web Services from JavaScript

- Either do it the hard way and use the JavaScript class XMLHttpRequest directly
- Or use dojo
  - dojo.xhrDelete : Use HTTP DELETE method to make an xhr call.
  - dojo.xhrGet : Use HTTP GET method to make an xhr call.
  - dojo.xhrPost : Use HTTP POST method to make an xhr call.
  - dojo.xhrPut : Use HTTP PUT method to make an xhr call.

dojo.xhrGet

- dojo.xhrGet expects a parameter object with the following properties

Parameter	***Description***
url	The URL to request data from.
handleAs	The expected format of the data. The currently supported options are: ▪text (default) ▪json ▪json-comment-optional ▪json-comment-filtered ▪javascript ▪xml

dojo.xhrGet – Optional Parameters

Parameter	*Description*
sync	Boolean value whether to block the browser while the request is running. Default is false.
preventCache	Boolean value – if set to yes dojo adds a unique query parameter to each request to prevent the browser cache
content	A JavaScript object with name/string value pairs. dojo will encode the values and append it to the query. E.g. ?key1=value1&key2=value2&key3=value3
headers	A JavaScript object of name/string value pairs. The headers are send as part of the request.
timeout	Number of milliseconds to wait until timing out the request. Default is ‘0’, which means infinite (no timeout).
user	For Basic web authentication the username.
password	For Basic web authentication the password.

dojo.xhrGet – Parameters for Handling the Data

Parameter	*Description*
load	A JavaScript function invoked when the data is returned from the server. The data will be passed as the first parameter of the function. The format of the data is controlled by the previously mentioned handleAs parameter.
error	A JavaScript function called in case of an error. The first parameter passed to the error function is a JavaScript Error object indicating what the failure was.
handle	A JavaScript function called regardless if the request was sucessfull or not. The first parameter passed to this callback is the response (or error) and the second parameter is the IO args object, from which you can get the status code and determine success or failure.

dojo.xhrGet - Example

```
var xhrArgs = {
    url: "http://www.example.com/someservice/sessions/BP108",
    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.xhrPost

- dojo.xhrPost accepts the same parameter as dojo.xhrGet with the following addition

Parameter	***Description***
content	A JavaScript object of name/string value pairs. xhrPost will convert this into proper POST format and send it with the post data. Note that this parameter is handled differently from dojo.xhrGet, which encodes it as a query string in url.
form	For posting FORM data, you can provide either the DOM node of your form or the ID of the form. xhrPost will convert this into proper POST format and send it with the post data. If a url is not set in the args to dojo.xhrPost, then it tries to extract the url from the form 'action' attribute.
postData	A string of data you wish to send as the post body. dojo.xhrPost (and dojo.rawXhrPost), do not do any processing of this It is merely passed through as the POST body.

Only one of these parameters can be used in a request.

dojo.xhrPut

- dojo.xhrPut accepts the same parameter as dojo.xhrGet with the following addition

Parameter	***Description***
content	A JavaScript object of name/string value pairs. xhrPost will convert this into proper POST format and send it with the post data. Note that this parameter is handled differently from dojo.xhrGet, which encodes it as a query string in url.
form	For posting FORM data, you can provide either the DOM node of your form or the ID of the form. xhrPost will convert this into proper POST format and send it with the post data. If a url is not set in the args to dojo.xhrPost, then it tries to extract the url from the form 'action' attribute.
postData	A string of data you wish to send as the post body. dojo.xhrPost (and dojo.rawXhrPost), do not do any processing of this It is merely passed through as the POST body.

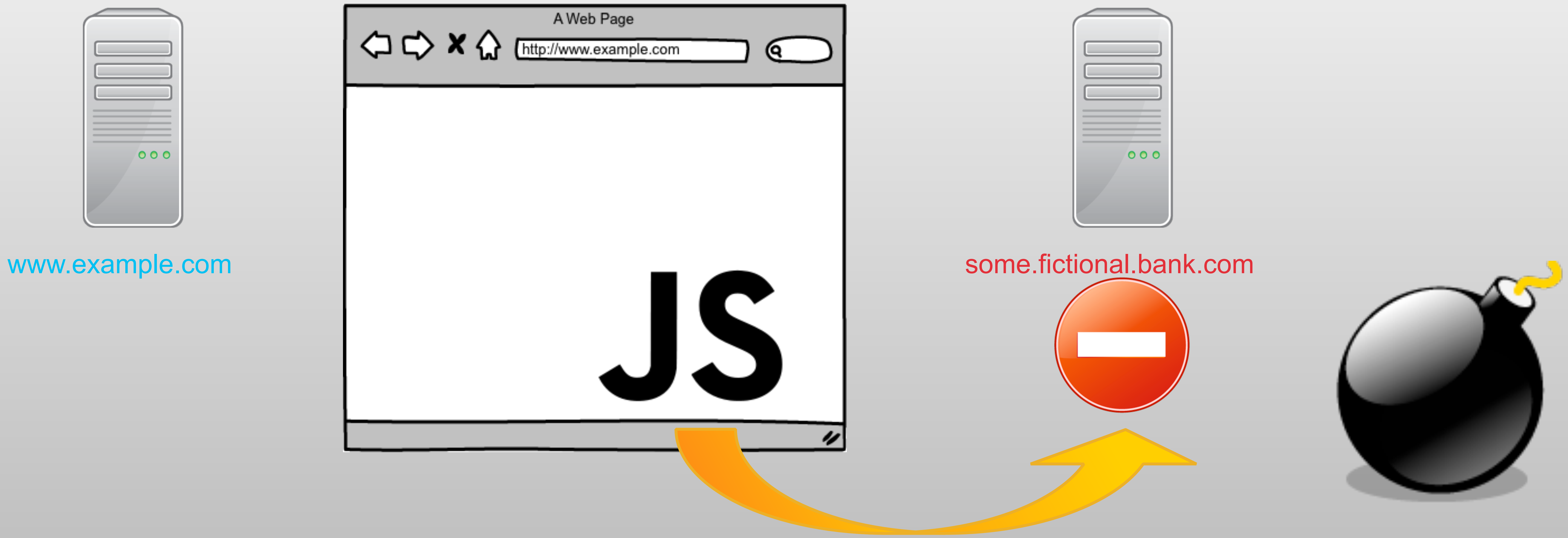
Only one of these parameters can be used in a request.

dojo.xhrDelete

- dojo.xhrDelete accepts the same parameter as dojo.xhrGet.
- There is no **content** parameter in dojo.xhrDelete because the URI (URL + query params) defines what to delete.

Same Origin Policy - SOP

- For security reasons a script runing in a browser can only access resources from the same server (origin)
- This limitation also applies for Web Services calls



JSONP – JSON with padding

A solution with side effects

- The Same Origin Policy does not apply to JavaScript tags
  - The JavaScript source could be referenced from any server
- JSONP uses the URI of an embedded JavaScript tag to send a request
- The requested data is returned as a function call to an existing JavaScript function
- This technique is limited to GET requests
- Potential security issue: Whatever data is returned from the "other" server is executed as a JavaScript function in the context of the current browser session
- dojo supports JSONP with the dojo.io.script package

JSONP – JSON with padding

A solution with side effects

JSON-Data to be returned

```
{
  sessionId: "BP108",
  title: "Be Open - Use Web Services and REST in XPages Applications",
  description: "...",
  speaker: {
             name: "Bernd Hort",
             company: "assono GmbH"}
 }
```

Call to get data

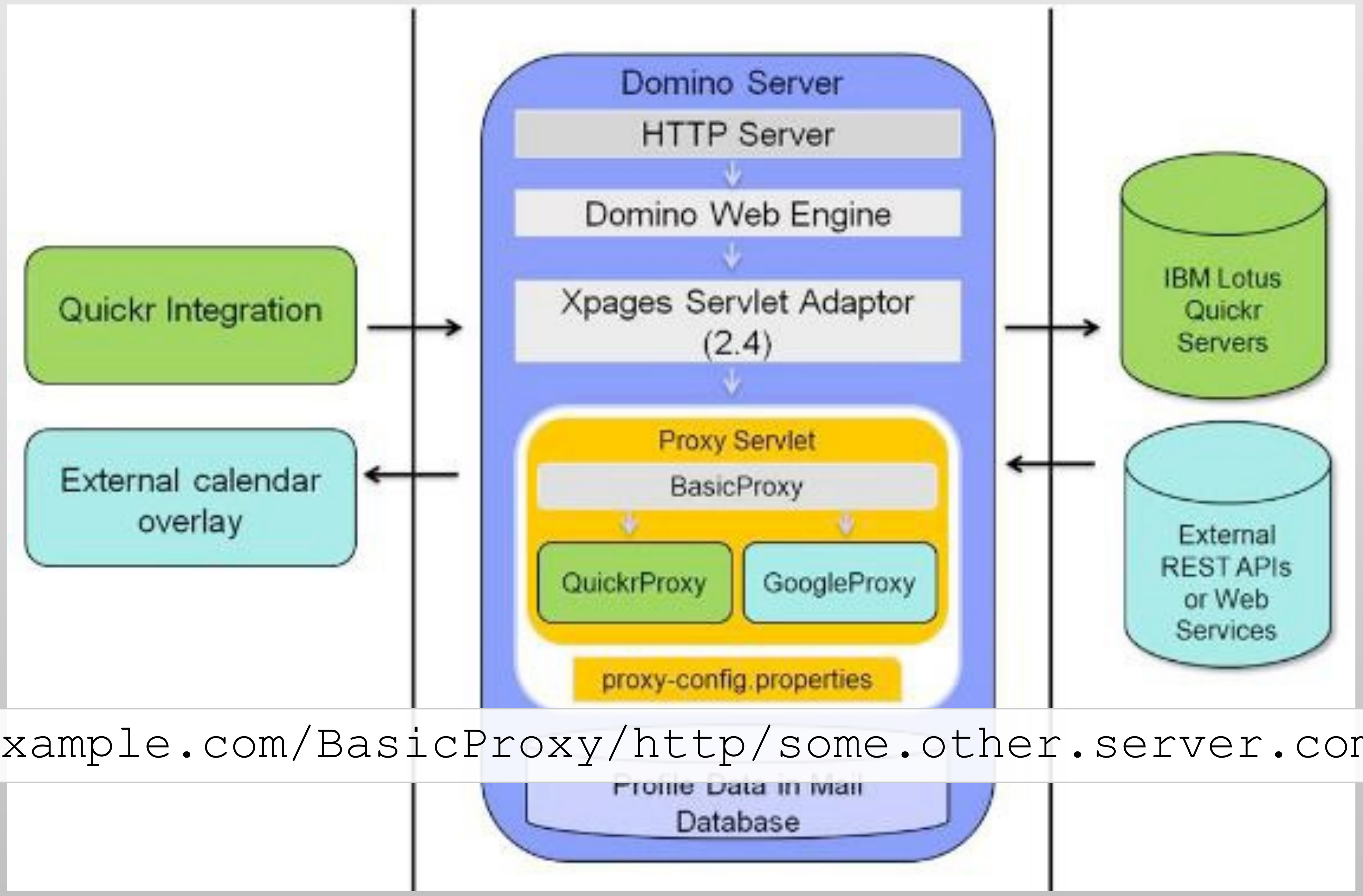
```
<script type="application/javascript"
src="http://{someserver}/Sessions/BP108?jsonp=updateSessionDetails">
</script>
```

Returned JavaScript

```
updateSessionDetails({
  sessionId: "BP108",  title: "Be Open - Use Web Services and REST in XPages
  Applications",  description: "...",   speaker: {name: "Bernd Hort",
  company: "assono GmbH"}
 })
```

IBM® iNotes® Proxy Servlet

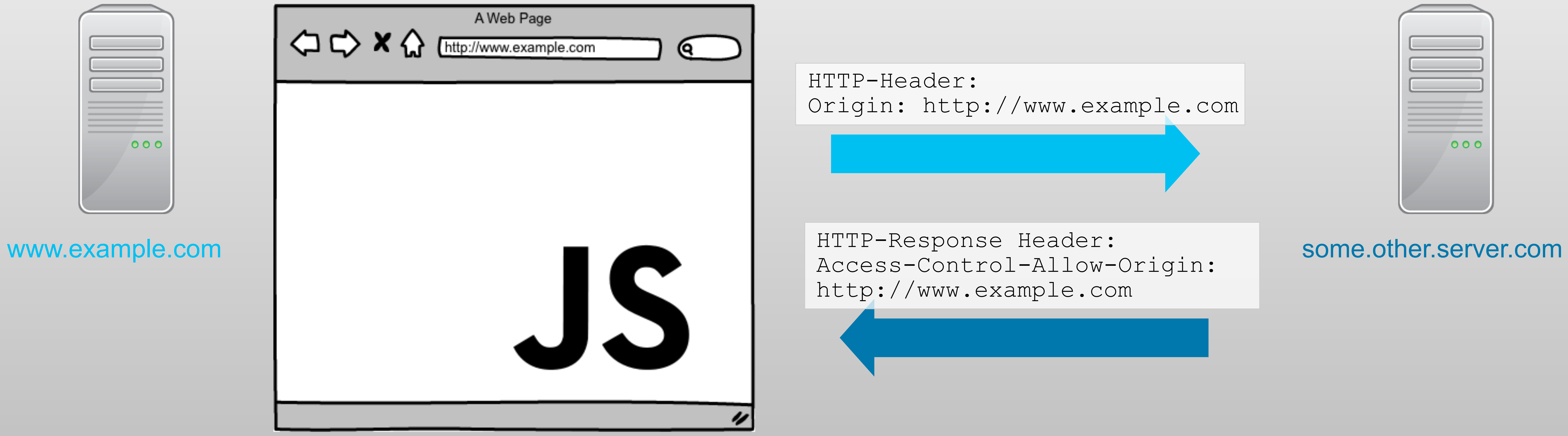
- In IBM® Lotus® iNotes® 8.5 a proxy servlet was introduced to overcome the SOP
- A request going through the proxy will have the same origin seen from the browsers
- Details can be found in New features in IBM Lotus iNotes 8.5: Full mode



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

Cross-origin resource sharing – CORS

- With Cross-origin resource sharing – CORS the origin is added to the header of the request
- The requesting server returns in the response header if he accepts the origin domain



```
HTTP-Header:
Origin: http://www.example.com
```

```
HTTP-Response Header:
Access-Control-Allow-Origin:
http://www.example.com
```

Cross-origin resource sharing – CORS

- CORS works on all HTTP actions: GET, POST, PUT, DELETE …
- CORS is supported in all modern browsers (from Wikipedia)
  - Gecko 1.9.1 (Firefox 3.5, SeaMonkey 2.0, Camino 2.1) and above.
  - WebKit (Initial revision uncertain, Safari 4 and above, Google Chrome 3 and above, possibly earlier)
  - MSHTML/Trident 6.0 (Internet Explorer 10) has native support. MSHTML/Trident 4.0 & 5.0 (Internet Explorer 8 & 9) provides partial support via the XDomainRequest object.
  - Presto-based browsers (Opera) implement CORS as of Opera 12.00 and Opera Mobile 12, but not Opera Mini.

dojo CONTROLS

dojo Store API

- The dojo Store API is an abstract API for handling data based on [HTML5/W3C's IndexedDB object store API]

Method	*Description*
get(id)	Retrieves an object by its identifier, returning the object.
query(query, options)	Queries the store using the provided query. Options include • start - Starting offset • count - Number of objects to return • sort - Follows the Dojo Data sort definition • queryOptions - Follows the Dojo Data queryOptions definition
put(object, options)	Saves the given object. options.id (optional) indicates the identifier.
add(object, options)	Create a new object. options.id (optional) indicates the identifier.
remove(id)	Delete the object by id..

dojo Store API (continue)

Method	*Description*
getIdentity(object)	Returns an object's identity
queryEngine(query, options)	This takes a query and query options and returns a function that can execute the provided query on a JavaScript array. The queryEngine may be replace to provide more sophisticated querying capabilities. The returned query function may have a "matches" property that can be used to determine if an object matches the query.
transaction()	Starts a transaction and returns a transaction object. The transaction object should include: commit() - Commits all the changes that took place during the transaction. abort() - Aborts all the changes that took place during the transaction. Note that a store user might not call transaction() prior to using put, delete, etc. in which case these operations effectively could be thought of as "auto-commit" style actions.
getChildren(object, options)	Returns the children of an object. The options parameter may include the same properties as query options
getMetadata(object)	Returns any metadata about the object. This may include attribution, cache directives, history, or version information. (addresses #3126, #3127)

dojo.store.JsonRest

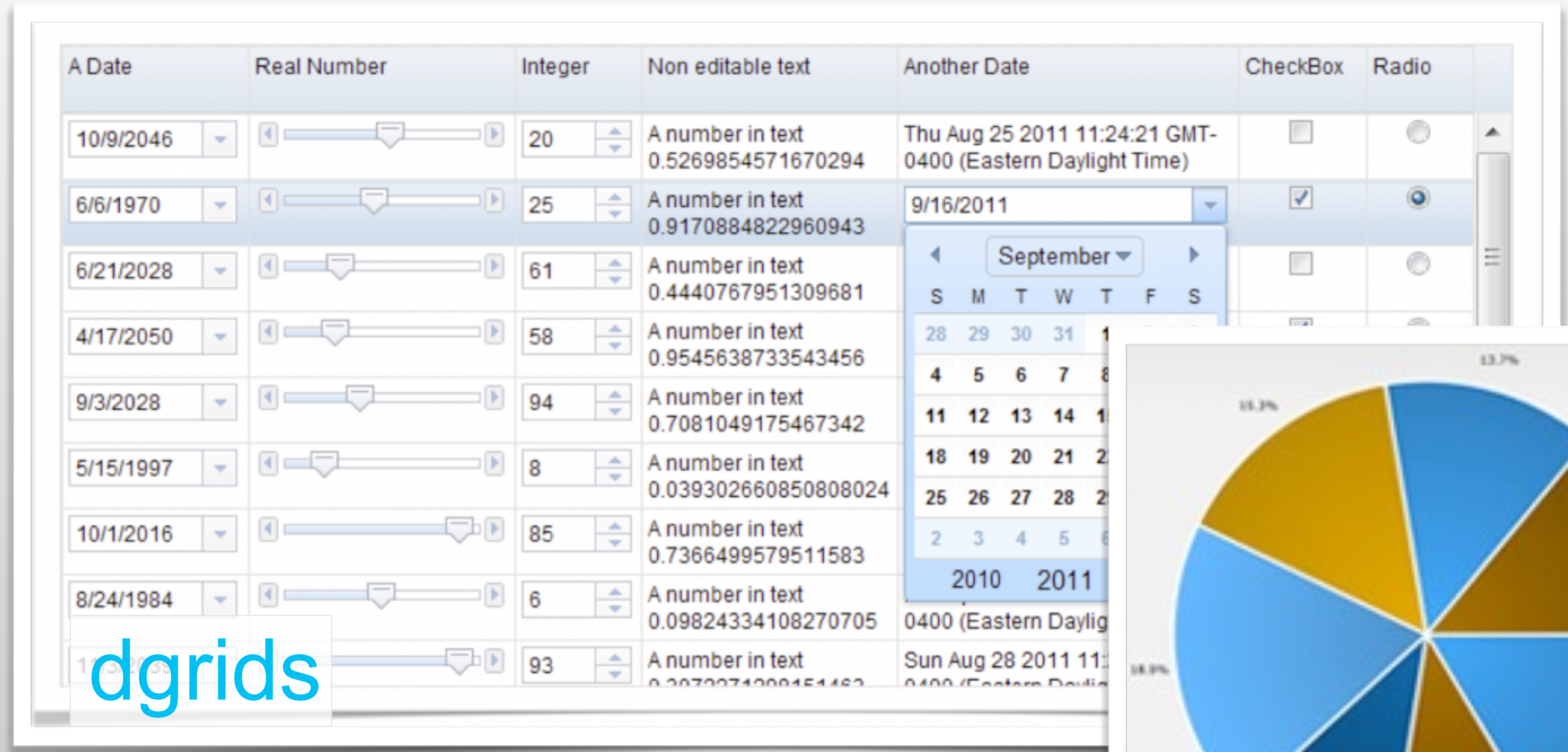
- The dojo.store.JsonRest handles the full HTTP/REST communication with a server based on the HTTP standards using the full range of GET, PUT, POST, and DELETE methods
- It implements all dojo Store API
- A JsonRest store can be instantiated with a URL

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

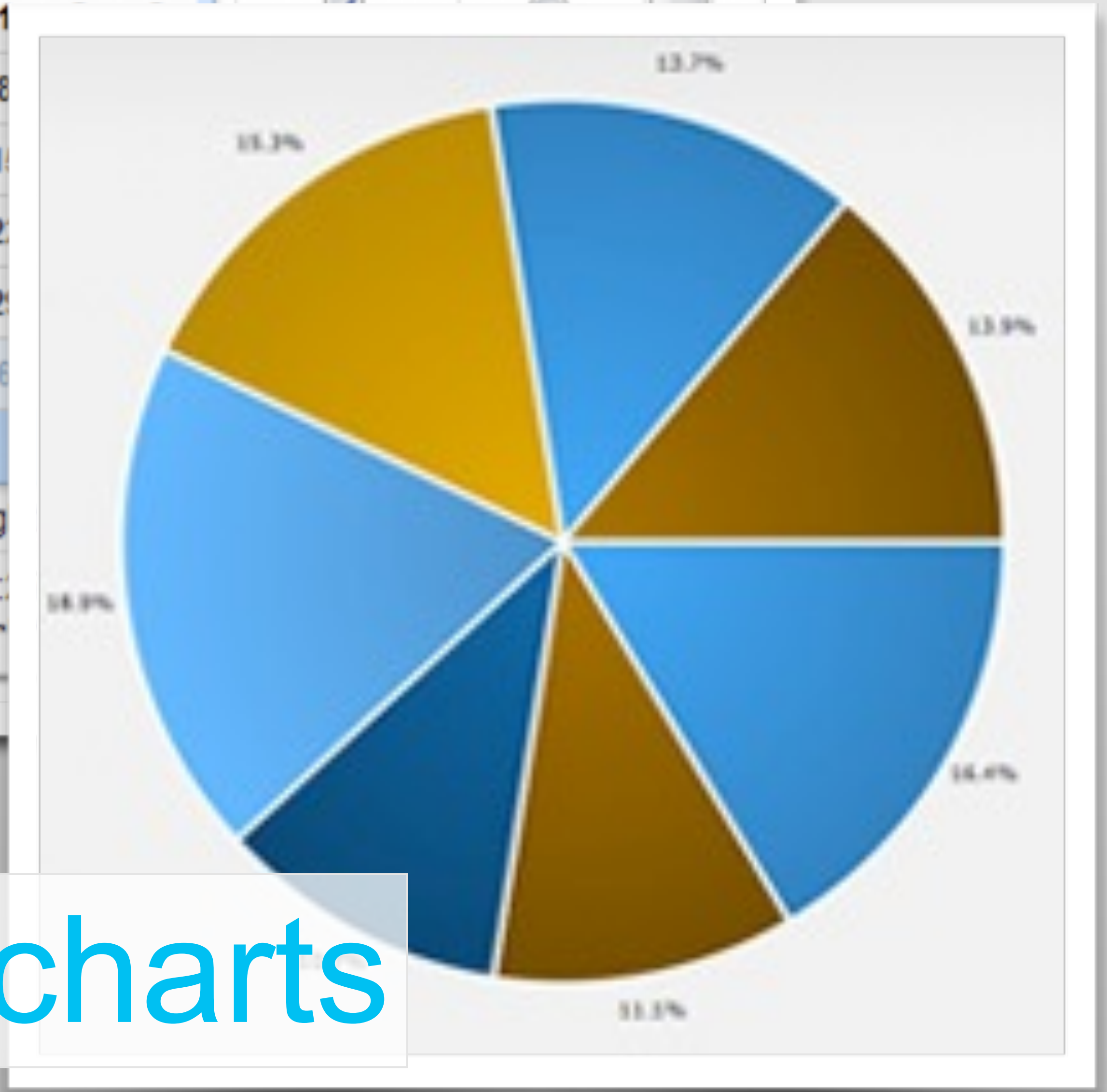
- Paging is supported
- Sorting is supportedt

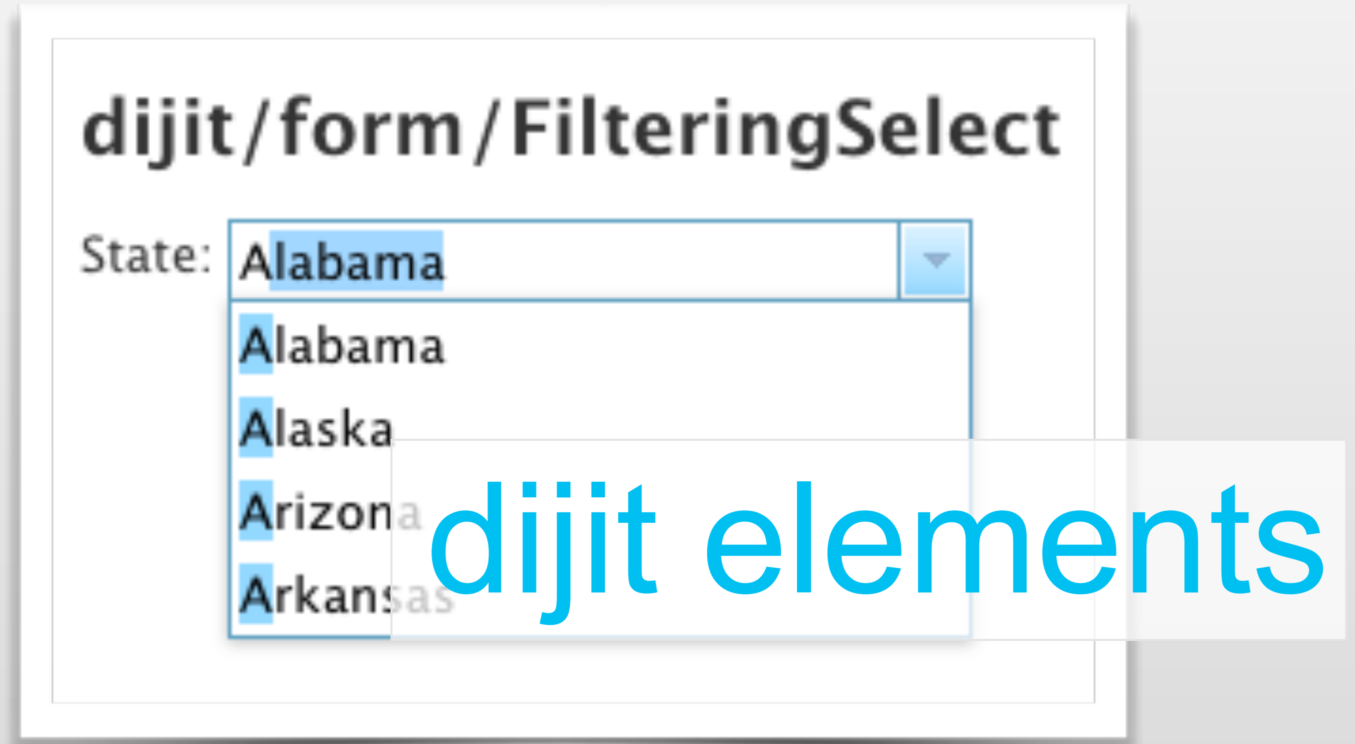
Use of dojo Stores

- dojo Data Stores can be used as data sources for



A Date	Real Number	Integer	Non editable text	Another Date	CheckBox	Radio
10/9/2046		20	A number in text 0.5269854571670294	Thu Aug 25 2011 11:24:21 GMT-0400 (Eastern Daylight Time)	☐	○
6/6/1970		25	A number in text 0.9170884822960943	9/16/2011	✓	●
6/21/2028		61	A number in text 0.4440767951309681		☐	○
4/17/2050		58	A number in text 0.9545638733543456			
9/3/2028		94	A number in text 0.7081049175467342			
5/15/1997		8	A number in text 0.039302660850808024			
10/1/2016		85	A number in text 0.7366499579511583			
8/24/1984		6	A number in text 0.09824334108270705	0400 (Eastern Daylig		
		93	A number in text	Sun Aug 28 2011 11:		





dijit/form/FilteringSelect

State: Alabama

XPages Extension Library

- The XPages Extension Library has components that provide or uses RESTful Web Services
- Components that provide RESTful Web Services
  - Domino Data Services
  - RESTService control
  - Custom Database Servlet
  - Custom Wink Service
- Controls that uses RESTful Web Services
  - Dojo Data Grid

Thank You!

- Get the latest version of this presentation and the sample database from http://www.assono.de/blog/d6plinks/ibmconnected2015-bp108

Engage Online

- **SocialBiz User Group** socialbizug.org
  - Join the epicenter of Notes and Collaboration user groups
- **Social Business Insights blog** ibm.com/blogs/socialbusiness
  - Read and engage with our bloggers
- **Follow us on Twitter**
  - @IBMConnect and @IBMSocialBiz
- **LinkedIn** http://bit.ly/SBComm
  - Participate in the IBM Social Business group on LinkedIn
- **Facebook** https://www.facebook.com/IBMConnected
  - Like IBM Social Business on Facebook

Notices and Disclaimers