DWR (Direct Web Remoting)

Sang Shin
Michèle Garoche
http://www.javapassion.com
"Learn with Passion!"

Topics

- What is and Why DWR?
- Steps for building DWR-based AJAX application
- Registering callback functions
- Utility functions
- Engine functions
- Handling errors and warnings
- Security
- DWR and Web application frameworks

What is DWR?

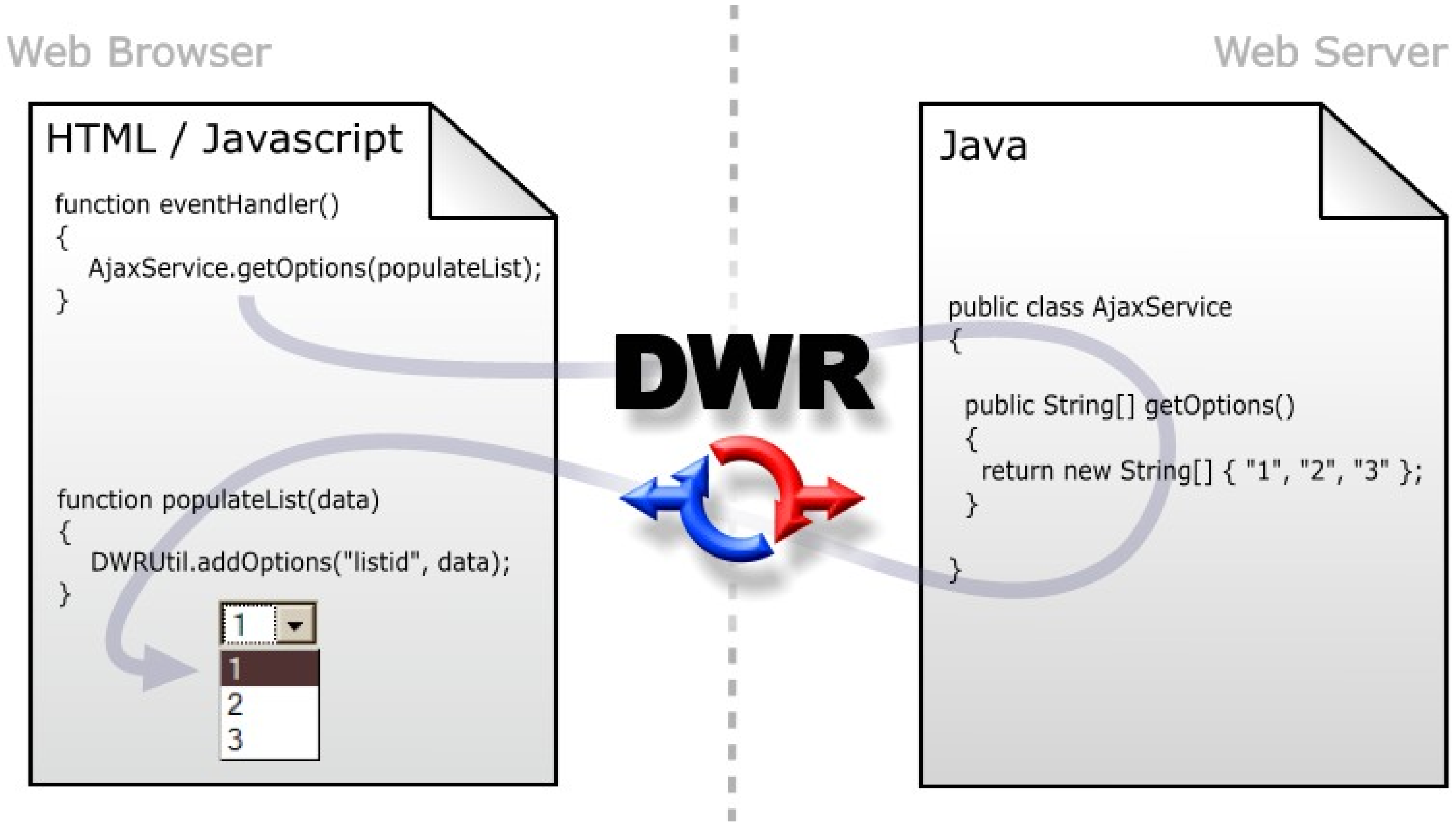
What is DWR?

- Is a Java and JavaScript open source library which allows you to write Ajax web applications
  - > Hides low-level *XMLHttpRequest* handling
- Specifically designed with Java technology in mind
  - > “Easy AJAX for Java”
- Allows JavaScript code in a browser to use Java methods running on a web server just as if they were in the browser
  - > Why it is called “Direct remoting”

Why DWR?

- Without DWR, you would have to create many Web application endpoints (servlets) that need to be address'able via URI's
- What happens if you have several methods in a class on the server that you want to invoke from the browser?
  - Each of these methods need to be addressable via URI whether you are using *XMLHttpRequest* directory or client-side only toolkit such as Dojo or Prototype
  - You would have to map parameters and return values to HTML input form parameters and responses yourself
- DWR comes with JavaScript utility functions

How DWR Works



```
function eventHandler()
{
    AjaxService.getOptions(populateList);
}

function populateList(data)
{
    DWRUtil.addOptions("listid", data);
}
```

```
public class AjaxService
{
  public String[] getOptions()
  {
    return new String[] { "1", "2", "3" };
  }
}
```

DWR Consists of Two Main Parts

- A DWR-runtime-provided Java Servlet running on the server that processes incoming DWR requests and sends responses back to the browser
  - > *uk.ltd.getahead.dwr.DWRServlet*
  - > This servlet delegates the call to the backend class you specify in the dwr.xml configuration file
- JavaScript running in the browser that sends requests and can dynamically update the webpage
  - > DWR handles *XMLHttpRequest* handling

source: http://getahead.ltd.uk/dwr

How Does DWR Work?

- DWR dynamically generates a matching client-side Javascript class from a backend Java class
  - Allows you then to write JavaScript code that looks like conventional RPC/RMI like code, which is much more intuitive than writing raw JavaScript code
- The generated JavaScript class handles remoting details between the browser and the backend server
  - Handles asynchronous communication via *XMLHttpRequest* - Invokes the callback function in the JavaScript
  - You provide the callback function as additional parameter
  - DWR converts all the parameters and return values between client side Javascript and backend Java

Steps for Building DWR-based AJAX Application

Steps to Follow

1. Copy dwr.jar file into the WEB-INF/lib directory of your web application
    - dwr.jar contains DWR runtime code including the DWR servlet
2. Edit web.xml in the WEB-INF directory
    - DWR servlet mapping needs to be specified
3. Create dwr.xml file in the WEB-INF directory
    - You specify which class and which methods of the backend service you want to expose
4. Write client-side JavaScript code, in which you invoke methods of remote Java class (or classes) in RPC/RMI-like syntax
5. Build, deploy, test the application

Step #1: Copy *dwr.jar* File in the WEB-INF/lib Directory

- dwr.jar contains DWR runtime code including the DWR servlet
- You can get dwr.jar file from http://getahead.ltd.uk/dwr/download
- The latest version is 2.0 (as of June 2007)

Step #2: Edit **web.xml** in the WEB-INF directory

```
<!-- Configure DWR for your Web application -->
<servlet>
  <servlet-name>dwr-invoker</servlet-name>
  <display-name>DWR Servlet</display-name>
  <servlet-class>uk.ltd.getahead.dwr.DWRServlet</servlet-class>
  <init-param>
     <param-name>debug</param-name>
     <param-value>true</param-value>
  </init-param>
</servlet>

<servlet-mapping>
  <servlet-name>dwr-invoker</servlet-name>
  <url-pattern>/dwr/*</url-pattern>
</servlet-mapping>
```

Step #3: Create dwr.xml file in the WEB-INF directory

- The dwr.xml config file defines what classes and what methods of those classes DWR can create and remote for use by client-side Javascript code
- Suppose I have a Java class called *mypackage.Chat* and I want to create a matching JavaScript class called *Chat*
  - mypackage.Chat Java class (server)
  - Chat JavaScript class (client)

Step #3: Create dwr.xml file in the WEB-INF directory

```
<!DOCTYPE dwr PUBLIC
    "-//GetAhead Limited//DTD Direct Web Remoting 1.0//EN"
    "http://www.getahead.ltd.uk/dwr/dwr10.dtd">

<dwr>
  <allow>
    <create creator="new" javascript="Chat">
      <param name="class" value="mypackage.Chat"/>
    </create>
    <convert converter="bean" match="mypackage.Message"/>
  </allow>
</dwr>
```

Step #4a: Write Client-side JavaScript code in which you invoke methods of a Java class

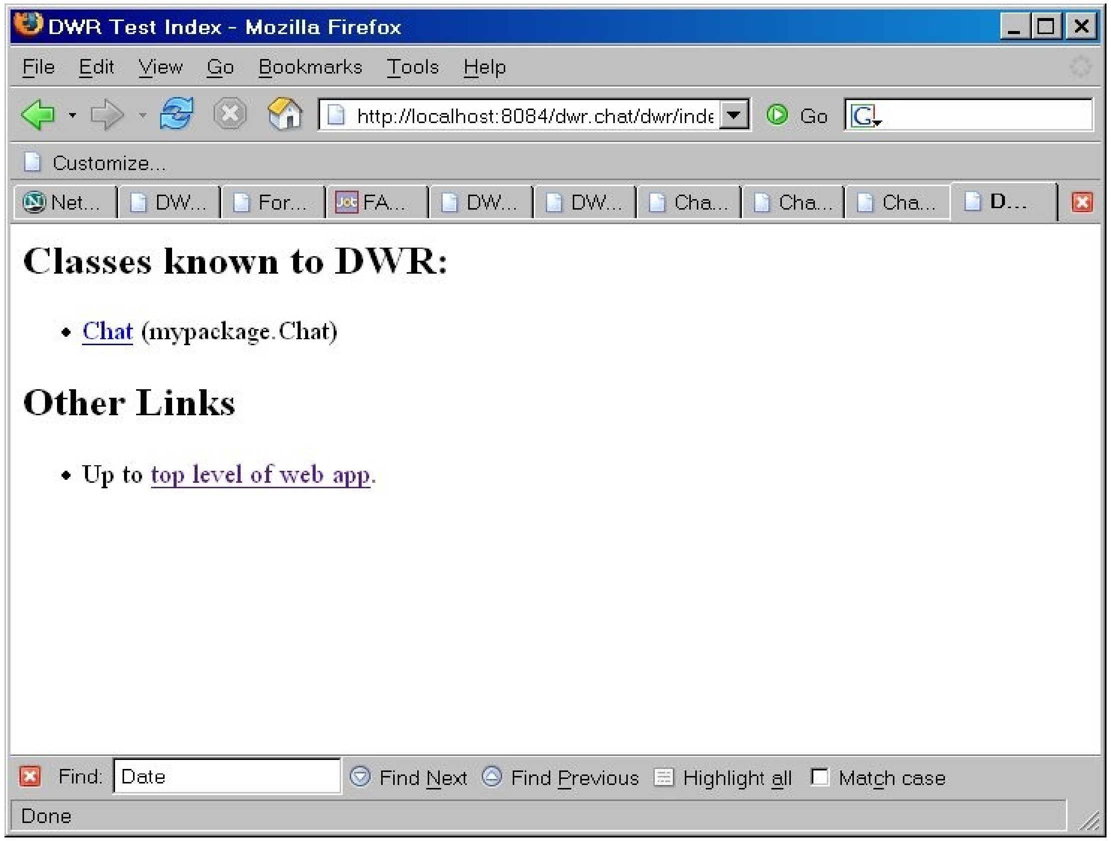
```
<!-- You have to include these two JavaScript files from DWR -->
<script type='text/javascript' src='dwr/engine.js'></script>
<script type='text/javascript' src='dwr/util.js'></script>

<!-- This JavaScript file is generated specifically for your application -->
<script type='text/javascript' src='dwr/interface/Chat.js'></script>
```

Step #4b: Write JavaScript client code in which you invoke methods of a Java class

```
<script type='text/javascript'>
        function sendMessage(){
           var text = DWRUtil.getValue("text");
           DWRUtil.setValue("text", "");

           // Invoke addMessage(text) method of the Chat class on
           // the server.  The gotMessages is a callback function.
           // Note the RPC/RMI like syntax.
           Chat.addMessage(gotMessages, text);
        }

        function checkMessages(){
           // Invoke getMessages() method of the Chat class on
           // the server.  The gotMessages is a callback function.
           Chat.getMessages(gotMessages);
        }

        ...
</script>
```

Step #5: Build, Deploy, & Test

- You can see the test page of your application
  - http://localhost:8084/<Your-Application-Context>/dwr


DWR Test Index - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://localhost:8084/dwr.chat/dwr/inde
Go
Customize...
Net... DW... For... FA... DW... DW... Cha... Cha... Cha... D...
Find: Date
Find Next
Find Previous
Highlight all
Match case
Done

Classes known to DWR:

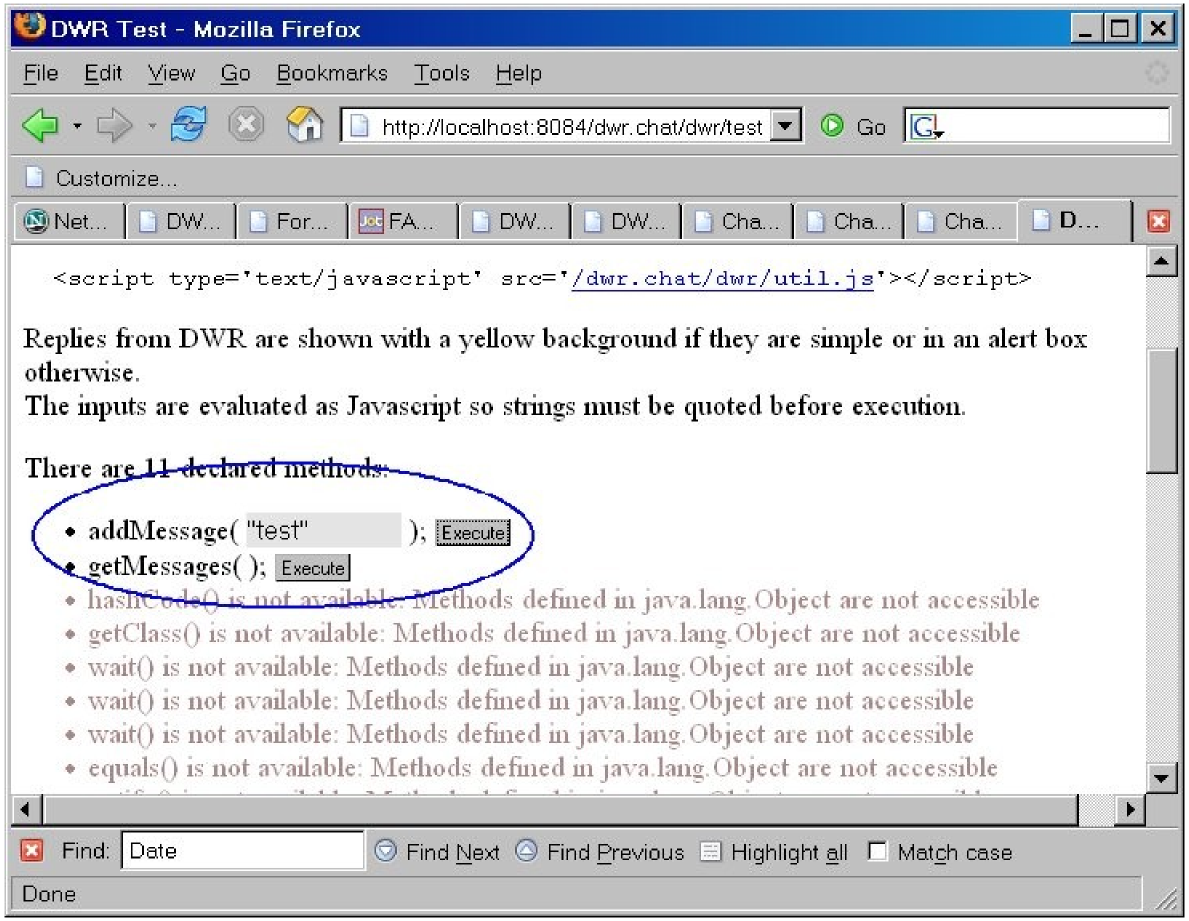
- Chat (mypackage.Chat)

Other Links

- Up to top level of web app.

Step #5: Build, Deploy, and Test

- You can actually test the interaction with the server


DWR Test - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://localhost:8084/dwr.chat/dwr/test
Go
Customize...
Net... DW... For... FA... DW... DW... Cha... Cha... Cha... D...
Find: Date
Find Next Find Previous Highlight all Match case
Done

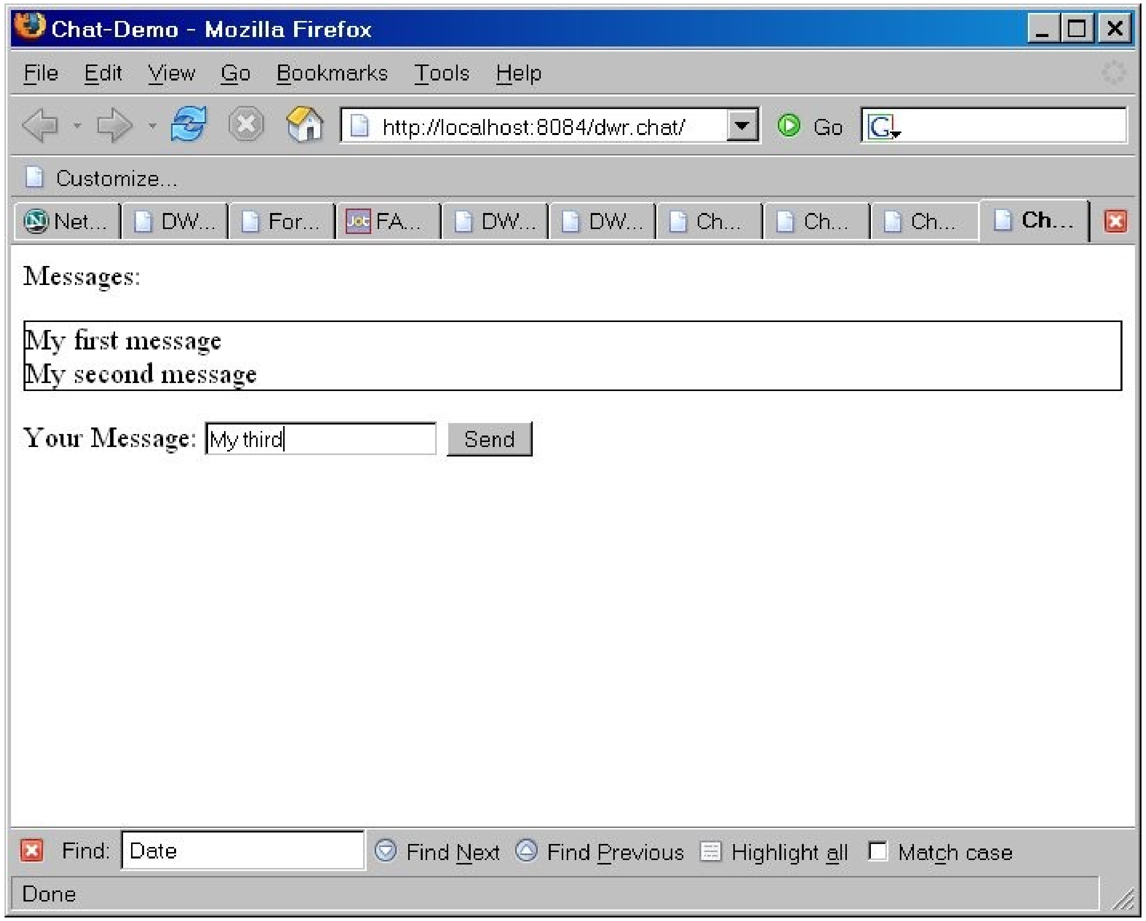
```
<script type='text/javascript' src='/dwr.chat/dwr/util.js'></script>
```

Replies from DWR are shown with a yellow background if they are simple or in an alert box otherwise.
The inputs are evaluated as Javascript so strings must be quoted before execution.

There are 11 declared methods:

- addMessage("test"); Execute
- getMessages( ); Execute
- hashCode() is not available: Methods defined in java.lang.Object are not accessible
- getClass() is not available: Methods defined in java.lang.Object are not accessible
- wait() is not available: Methods defined in java.lang.Object are not accessible
- wait() is not available: Methods defined in java.lang.Object are not accessible
- wait() is not available: Methods defined in java.lang.Object are not accessible
- equals() is not available: Methods defined in java.lang.Object are not accessible

Step #6: Run the Application


Chat-Demo - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://localhost:8084/dwr.chat/
Go
Customize...
Net... DW... For... FA... DW... DW... Ch... Ch... Ch... Ch...

Messages:

My first message
My second message

Your Message: My third Send

Find: Date Find Next Find Previous Highlight all Match case
Done

Registering Callback Function for AJAX-based Asynchronous Invocation

How DWR Handles Asynchronous AJAX-Call

- Calling JavaScript function at the client needs to be done asynchronously while calling a Java method (at the server) is synchronous
  - DWR handles this mismatch
- DWR provides a scheme for registering a callback function at the client
  - You pass the callback function as an additional parameter
  - The callback function is called when the data is returned from the server - this is AJAX behavior

Example 1: How Callback Function is Registered

- Suppose we have a Java method that looks like this:

```
// Server side Java code
public class MyJavaClass {
    public String getData(int index) { ... }
}
```

- We can use this from Javascript as follows:

```
// Callback function to be called
function handleGetData(str) {
  alert(str);
}
// The callback function is passed as an additional parameter
MyJavaScriptClass.getData(42, handleGetData);
```

Example 2: How Callback Function is Registered

- Suppose we have a Java method that looks like this:

```
// Server side Java code
public class MyJavaClass {
    public String getData(int index) { ... }
}
```

- Callback function can be in-lined

```
MyJavaScriptClass.getData(42,
                          function(str) { alert(str); });
```

Example 3: How Callback Function is Registered

- Suppose we have a Java method that looks like this:

```
// Server side Java code
public class MyRemoteJavaClass {
   public String getData(int index) { ... }
}
```

- You can use Meta-data object

```
MyRemoteJavaScriptClass.getData(42,
                                {callback:function(str)
                                        { alert(str); }});
```

Example 4: How Callback Function is Registered

- Suppose we have a Java method that looks like this:

```
// Server side Java code
public class MyRemoteJavaClass {
    public String getData(int index) { ... }
}
```

- You can specify timeout and error handler as well

```
MyRemoteJavaScriptClass.getData(42,
                      {callback:function(str) { alert(str); }}
                       timeout:5000,
                       errorHandler:function(message)
                                     { alert("Oops: " + message); }
);
```

Converters

Converters

- Converter marshals data between client and server
- Types of converters provided by DWR
  - Basic converters
  - Date converter
  - Bean and Object converters
  - Array converter
  - Collection converter
  - DOM Objects
- You can create your own converters
  - Rarely needed

Basic Converters

- Handles
  - boolean, byte, short, int, long, float, double, char, java.lang.Boolean, java.lang.Byte, java.lang.Short, java.lang.Integer, java.lang.Long, java.lang.Float, java.lang.Double, java.lang.Character, java.math.BigInteger, java.math.BigDecimal and java.lang.String
- No need to have a <convert ...> element in the <allow> section in *dwr.xml* to use them
  - They are enabled by default

Date Converter

- Marshalls between a Javascript *Date* and a *java.util.Date, java.sql.Date, java.sql.Times* or *java.sql.Timestamp*
- Is enabled by default
  - > Like Basic converters

Bean and Object Converters

- These are not automatically enabled
  - DWR makes sure that it has a permission before it touches any of your code
  - You have to specify your instruction in the dwr.xml
- Bean converter will convert POJOs into JavaScript associative arrays and back again
- Object converter is similar except that it work on object members directly rather than through getters and setters

Example: Bean Converter

- Enable the bean converter for a single class

```
<convert converter="bean"
            match="your.full.package.BeanName"/>
```

- Allow conversion of any class in the given package, or sub package

```
<convert converter="bean" match="your.full.package.*"/>
```

- Allow conversion of all Java Beans

```
<convert converter="bean" match="*"/>
```

Advanced Converters

- Declare new converters in the <init> element in *dwr.xml*
- Use $ for inner classes
- BeanConverter can restrict exported properties

Utility Functions

Utility Functions in util.js

- DWR comes with *util.js*
- The *util.js* contains a number of utility functions to help you update your web pages with JavaScript data
- You can use it outside of DWR because it does not depend on the rest of DWR to function

List of Utility Functions

- $(id)
- getValue, getValues, setValue, setValues
- addRows and removeAllRows
- addOptions and removeAllOptions
- getText
- onReturn
- selectRange
- toDescriptiveString
- useLoadingMessage

$(id)

- $(id) is the same thing as
  - *document.getElementById(id)* in DOM API
  - *dojo.byId(id)* in Dojo toolkit

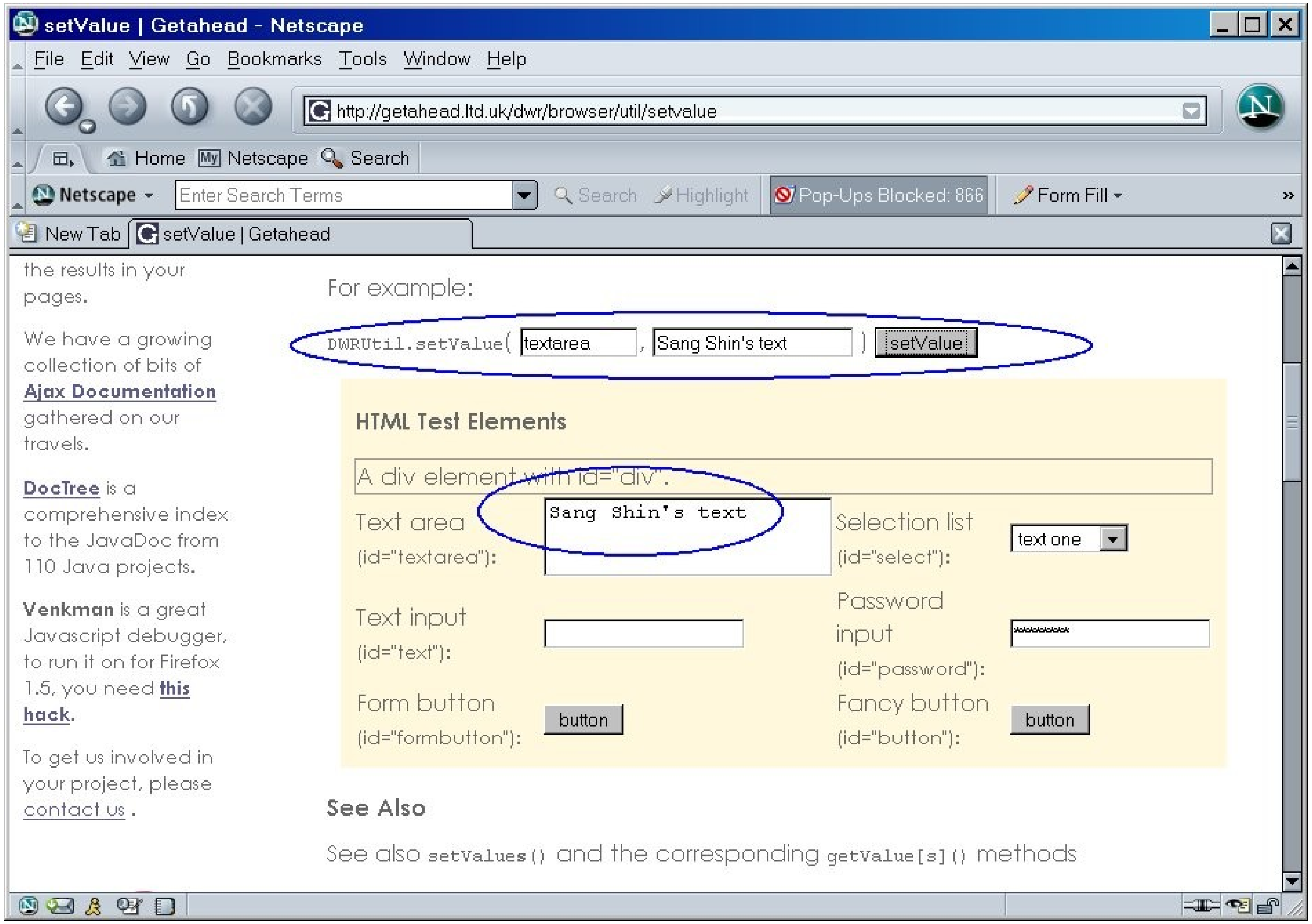
getValue, getValues

- DWRUtil.getValue(id);
- This gets the value(s) out of the HTML elements without you needing to worry about how a selection list differs from a div
- This method works for most HTML elements including selects (where the option with a matching value and not text is selected), input elements (including textarea's) div's and span's

setValue, setValues

- DWRUtil.setValue(id, value);
- This finds the element with the id specified in the first parameter and alters its contents to be the value in the second parameter.
- This method works for almost all HTML elements including selects (where the option with a matching value and not text is selected), input elements (including textarea's) div's and span's.

setValue, setValues


setValue | Getahead - Netscape
File Edit View Go Bookmarks Tools Window Help
http://getahead.ltd.uk/dwr/browser/util/setvalue
Home Netscape Search
Netscape Enter Search Terms Search Highlight Pop-Ups Blocked: 866 Form Fill
New Tab setValue | Getahead

the results in your pages.

We have a growing collection of bits of Ajax Documentation gathered on our travels.

DocTree is a comprehensive index to the JavaDoc from 110 Java projects.

Venkman is a great Javascript debugger, to run it on for Firefox 1.5, you need this hack.

To get us involved in your project, please contact us .

For example:

`DWRUtil.setValue(` textarea , Sang Shin's text `)` [setValue]

HTML Test Elements

A div element with id="div".

Text area (id="textarea"):	Sang Shin's text	Selection list (id="select"):	text one
Text input (id="text"):		Password input (id="password"):	********
Form button (id="formbutton"):	button	Fancy button (id="button"):	button

See Also

See also `setValues()` and the corresponding `getValue[s]()` methods

Manipulating Tables: addRows

- DWRUtil.addRows(id, array, cellfuncs, [options]);
  - Adds rows to a table element specified by id
- Parameters
  - id: The id of the table element (preferably a tbody element)
  - array: Array (or object from DWR 1.1) containing one entry for each row in the updated table
  - cellfuncs: An array of functions (one per column) for extracting cell data from the passed row data
  - options: An object containing various options

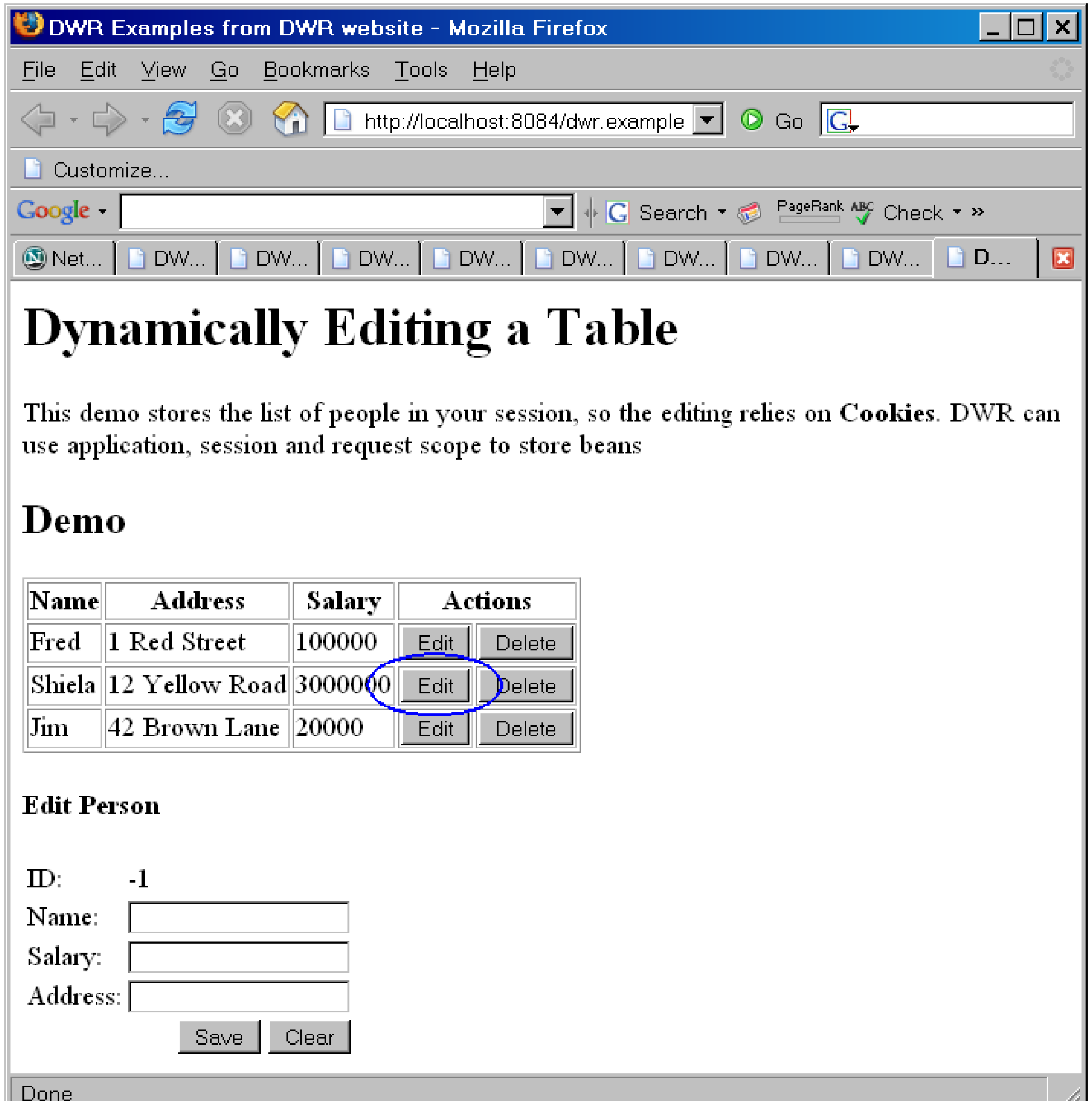
Manipulating Tables: removeAllRows(id);

- DWRUtil.removeAllRows(id);
  - > Removes all the rows in a table element specified by id
- Parameters
  - > id: The id of the table element (preferably a tbody element)

Exampe #1 : Manipulating Tables

```
<script type='text/javascript'>
        // Functions to be passed to DWRUtil.addRows
        var getName = function(person) { return person.name };
        var getDoB = function(person) { return person.address };
        var getSalary = function(person) { return person.salary };
        var getEdit = function(person) {
            return '<input type="button" value="Edit"
    onclick="readPerson('+person.id+')"/>';
        };
        var getDelete = function(person) {
            return '<input type="button" value="Delete"
    onclick="deletePerson('+person.id+', \''+person.name+'\')"/>';
        };

        // Callback function for getAllPeople method
        // The table is reconstructed
        function fillTable(people) {
            DWRUtil.removeAllRows("peoplebody");
            DWRUtil.addRows("peoplebody", people, [ getName, getDoB, getSalary,
    getEdit, getDelete ])
        }
</script>
```

Example #1: Manipulating Tables


DWR Examples from DWR website - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://localhost:8084/dwr.example
Go
Customize...
Google
Search
PageRank
Check
Net... DW... DW... DW... DW... DW... DW... DW... DW... D...
Done

Dynamically Editing a Table

This demo stores the list of people in your session, so the editing relies on **Cookies**. DWR can use application, session and request scope to store beans

Demo

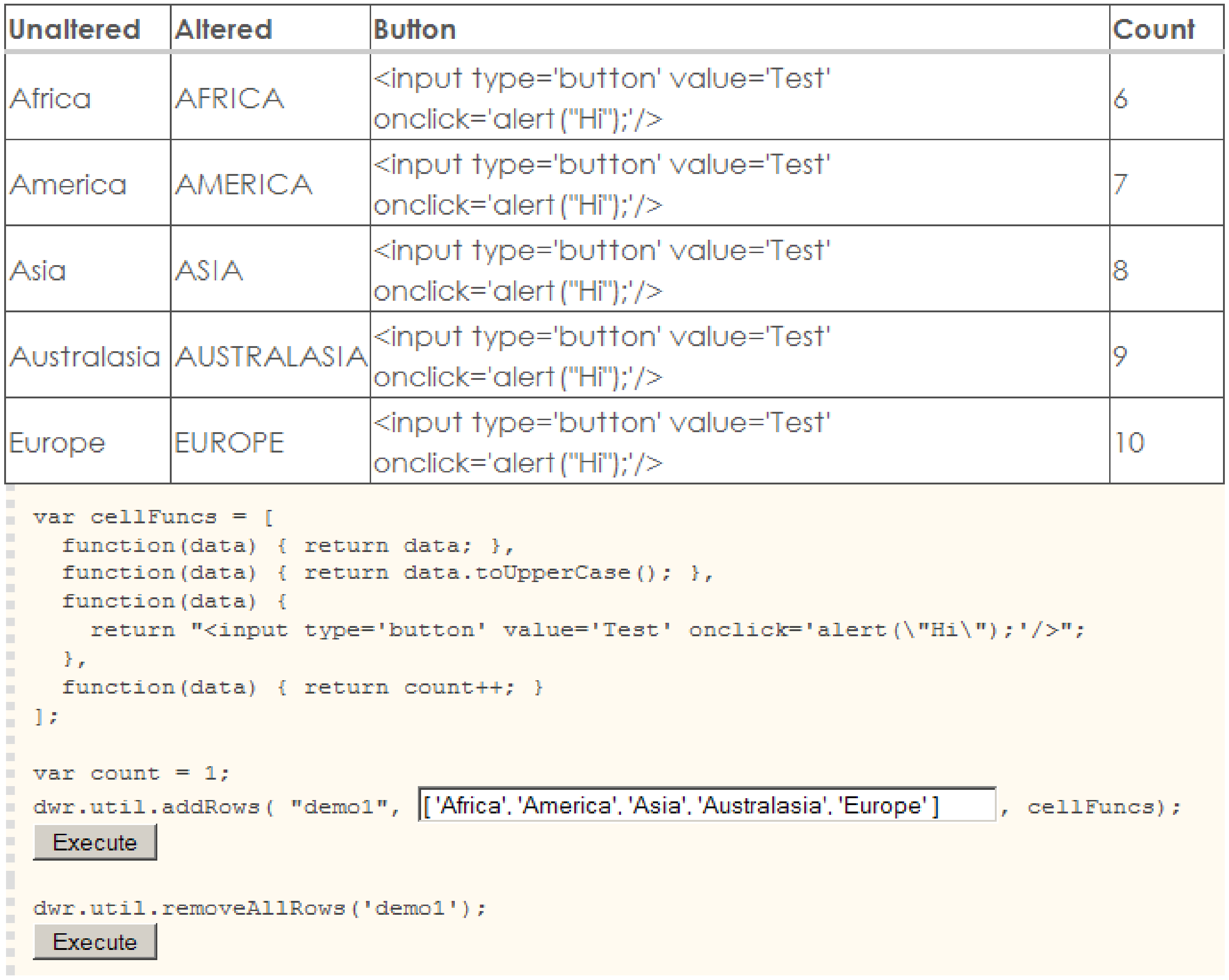
Name	Address	Salary	Actions	
Fred	1 Red Street	100000	Edit	Delete
Shiela	12 Yellow Road	3000000	Edit	Delete
Jim	42 Brown Lane	20000	Edit	Delete

Edit Person

ID:	-1
Name:	
Salary:	
Address:	

Save Clear

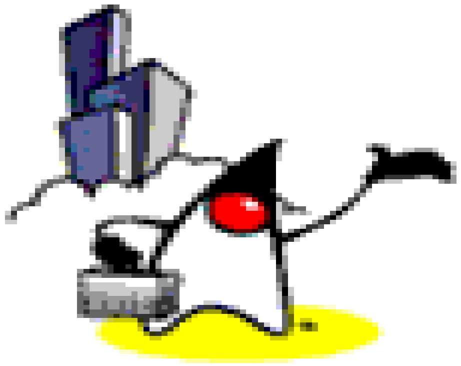
Example #2: Manipulating Tables



Unaltered	Altered	Button	Count
Africa	AFRICA	<input type='button' value='Test' onclick='alert("Hi");'/>	6
America	AMERICA	<input type='button' value='Test' onclick='alert("Hi");'/>	7
Asia	ASIA	<input type='button' value='Test' onclick='alert("Hi");'/>	8
Australasia	AUSTRALASIA	<input type='button' value='Test' onclick='alert("Hi");'/>	9
Europe	EUROPE	<input type='button' value='Test' onclick='alert("Hi");'/>	10

```
var cellFuncs = [
  function(data) { return data; },
  function(data) { return data.toUpperCase(); },
  function(data) {
    return "<input type='button' value='Test' onclick='alert(\"Hi\");'/>";
  },
  function(data) { return count++; }
];

var count = 1;
dwr.util.addRows( "demo1", ['Africa', 'America', 'Asia', 'Australasia', 'Europe'] , cellFuncs);
```

```
dwr.util.removeAllRows('demo1');
```



Demo: Utility Functions from http://getahead.ltd.uk/dwr/browser/util

Engine Functions

engine.js Functions

- *engine.js* is vital to DWR since it is used to marshal calls from the dynamically generated interface javascript function
- *engine.js* also contain set options methods
  - Options may be set globally (using a DWREngine.setX() function) or at a call or batch level (using call level meta data e.g { timeout:500, callback:myFunc })
  - A batch is several calls that are sent together.

Engine Options

- Robustness
  - errorHandler, warningHandler, timeout
- UI clues
  - preHook, postHook
- Remoting options
  - method, verb, async
- Call sequencing
  - ordered, callback
- Future
  - skipBatch, onBackButton, onForwardButton

Handling Errors and Warnings

Built-in Global Error Handlers

- Whenever there is some sort of failure, DWR calls an error or warning handler (depending on the severity of the error) and passes it the message
  - This method could be used to display error messages in an alert box or to the status bar
- DWR provides built-in global error handlers
  - *errorHandler* for errors
  - *warningHandler* for warnings
- You can set the global error handlers with your own
  - *DWREngine.setErrorHandler(youOwnErrorHandler);*
  - *DWREngine.setWarningHandler(youOwnWarningHandler);*

You Can Also Specify Handler In a Call

```
Remote.method(params, {
  callback:function(data) { ... },
  errorHandler:function(errorString, exception) { ... }
});
```

You Can Also Specify Handler In Batch Meta-data form

```
// Start the batch
DWREngine.beginBatch();

Remote.method(params, function(data) { ... });
// Other remote calls

DWREngine.endBatch({
  errorHandler:function(errorString, exception) { ... }
});
```

Setting Global Timeout

Setting Global Timeout

- *DWREngine.setTimeout()* function sets the timeout for all DWR calls
  - A value of 0 (the default) turns timeouts off
  - The units passed to setTimeout() are milli-seconds
  - If a call timeout happens, the appropriate error handler is called
- You can set the timeout on an individual call level

```
Remote.method(params, {
  callback:function(data) { alert("it worked"); },
  errorHandler:function(message) { alert("it broke"); },
  timeout:1000
});
```

How to Pass Servlet Objects as Parameters

Handling Servlet Objects (Implicit Objects)

- If a Java method has, a servlet object as a parameter, ignore it in the matching JavaScript method DWR will fill it in
  - HttpServletRequest
  - HttpServletResponse
  - HttpSession
  - ServletContext
  - ServletConfig

Handling Servlet Objects (Implicit Objects)

- For example if you have remoted a class like this:

```
public class Remote {
  public void method(int param,
                     ServletContext cx,
                       String s) { ... }
}
```

- Then you will be able to access it from Javascript just as though the ServletContext parameter was not there

```
Remote.method(42,             // int param
              "test",          // String s
              callback);       // Callback
```

Logging

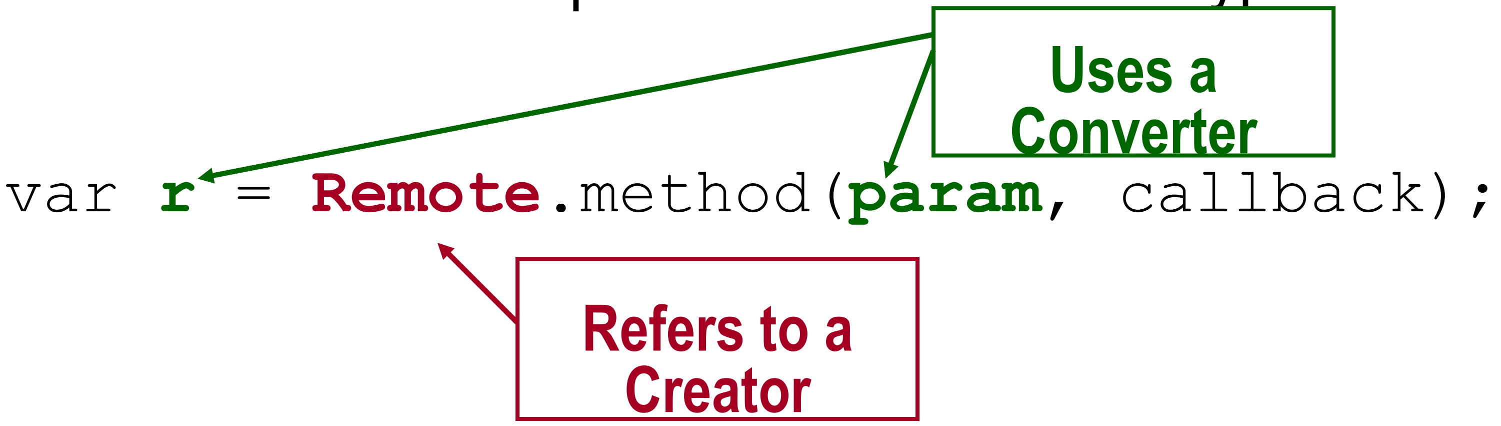
Logging

- DWR uses commons-logging if it is present
  - java.util.logging
  - log4j
- DWR uses HttpServlet.log() if commons-logging is not present

Creators

Creators and Converters

- Creators create objects that live on the server and have their methods remoted
- Converters marshal parameters and return types



```
var r = Remote.method(param, callback);
```

- Created object do things while Converted objects carry data

Advanced Creators

- Scope options
- Javascript names for session pre-population
- NullCreator for static methods
- Reloading the ScriptedCreator

DWR Security

Security

- DWR does not remote anything that you don't say it can via dwr.xml
- Audit
- Multiple dwr.xml Files
- Role based security
- Method level access control
- Risks

Multiple dwr.xml Files

- For separate J2EE security domains
- Or to separate components
- Configured in web.xml:

```
<init-param>
  <param-name>config*****</param-name>
  <param-value>WEB-INF/dwr.xml</param-value>
</init-param>
```

Signatures

- Sometimes introspection is not enough
- The <signature> element fixes the hole

```
<signatures>
  <![CDATA[
  import java.util.List;
  import com.example.Check;
  Check.setLotteryResults(List<Integer> nos);
  ]]>
</signatures>
```

DWR & Web Application Frameworks

DWR and Spring

- SpringCreator
- DwrController
- DwrSpringServlet
- SpringContainer
- beans.xml in place of dwr.xml

DWR and Other Libraries

- StrutsCreator
- JsfCreator and FacesExtensionFilter
- PageFlowCreator
- HibernateBeanConverter
- DOM, XOM, JDOM, DOM4J
- Rife

DWR 2.0

DWR 2.0

- DWR 1.1
- DWR 2.0
  - AjaxFilters
    - Security, Logging, Delay, Transactions
  - Spring Integration
    - No more dwr.xml, just use beans.xml
  - Reverse Ajax
    - Asynchronously push Javascript to the browser

Thank you!

Sang Shin
Michèle Garoche
http://www.javapassion.com
“Learn with Passion!”