



Introduction to Dojo Toolkit (with focus on dojo.io.bind & dojo.connect.event)

Sang Shin
Java Technology Evangelist
Sun Microsystems, Inc.





Disclaimer & Acknowledgments

- Even though Sang Shin is a full-time employee of Sun Microsystems, the contents here are created as his own personal endeavor and thus does not necessarily reflect any official stance of Sun Microsystems on any particular technology
- Many slides are created from the contents posted in dojotoolkit.org website

Topics

- What is and Why Dojo Toolkit?
- Dojo Toolkit Layered Architecture & Package System
- Remoting via [dojo.io.bind](http://dojo.io/bind)
- Dojo DOM Manipulation
- Backward/Forward buttons, Bookmarking
- Dojo Event System
 - > Overview
 - > DOM events
 - > Chaining function calls
 - > AOP event model
 - > Publish and subscribe
- Dojo 0.9 Release

3

This is the list of topics we are going to talk about in this session. First, I will talk about what is and why Dojo toolkit. Second, I will briefly talk about various Dojo toolkit packages.

Next, I will talk about how to do the remoting by the use of `dojo.io.bin()` call. By the way, by remoting, I am talking about how to perform XMLHttpRequest operation.

Then I will talk about how to leverage Dojo event system. As you will see later on, Dojo has very comprehensive and flexible event system.

Topics Covered in Advanced Dojo Presentation

- Creation of Dojo Widgets
- Dojo Drag and Drop
- Dojo Animation
- Dojo Storage
- Performance tuning



What is and Why Dojo Toolkit?

Now let's talk about what is and Why Dojo toolkit.

What is Dojo Toolkit?

- Open Source DHTML toolkit written in JavaScript
 - > It is a set of JavaScript libraries
- Aims to solve some long-standing historical problems with DHTML
 - > Browser incompatibility
- Allows you to easily build dynamic capabilities into web pages
 - > Widgets
 - > Animations
- Server technology agnostic

source: dojotoolkit.org

6

Dojo toolkit is open source DHTML toolkit written in JavaScript. In other words, it is a set of JavaScript libraries.

Dojo toolkit aims to solve some long-standing historical problems with DHTML such as browser incompatibility.

Dojo toolkit also allows you easily add dynamic capabilities into the web pages by the usage of pre-built widgets and animations.

Because it is client-side technology, it can work with any server side technology.

Why Dojo Toolkit?

- You can use Dojo to make your web applications more useable, responsive, and functional
 - > Supports AJAX
- Dojo provides lots of plumbing facilities
 - > Hides XMLHttpRequest processing
 - > Handles browser incompatibilities
- Strong developer community

source: dojotoolkit.org

7

You probably got the idea already why Dojo toolkit is useful. Just to repeat one more time, you can use Dojo toolkit to make your web applications to be more usable, responsive, and functional. Using Dojo toolkit is one of the most effective means of building AJAX-based Web applications.

Dojo provides lots of plumbing facilities, for example, it hides low-level XMLHttpRequest processing. It also handles browser incompatibilities.

Dojo enjoys a very strong user and developer community right now.

Features of Dojo Toolkit

- Powerful AJAX-based I/O abstraction (remoting)
- Graceful degradation
- Backward, forward, bookmarking support
- Aspect-oriented event system
- Markup-based UI construction through widgets
- Widget prototyping
- Animation
- Lots of useful libraries

8

This slide lists the features of Dojo toolkit. First, it provides powerful AJAX-based IO abstraction. What this means is that it handles asynchronous communication through the usage of XMLHttpRequest . Again this is called remoting.

It provides hooks for backward, forward, bookmarking support. Its event system is pretty powerful, for example, it lets you to invoke your event handler before or after an event occurs. This is called “aspect-oriented” event system.

It does provide markup based UI constructions through widgets. You can also build and reuse widgets that you built or built by someone else.

It does support animation. And again, Dojo toolkit comes with lots of useful JavaScript libraries.



Dojo Architecture

Dojo toolkit is made of a set of layered libraries



Dojo toolkit's layered libraries

- Widget
 - > Contains a set of APIs and functionality for defining and instantiating reusable components
- Application support libraries
 - > The bulk of Dojo toolkit functionality
 - > io.* - remoting capability with Ajax
- Language libraries
 - > Improve and simplify the lives of JavaScript developers
- Packaging system
 - > Allows you to customize the distribution of Dojo for your application



Dojo Toolkit Package System

Now let's talk about Dojo toolkit libraries.

Dojo Toolkit Package System

- Dojo libraries are organized in packages just like Java libraries are
- You import only the packages you need
 - > `dojo.require("dojo.event.*");`
 - > `dojo.require("dojo.dnd.*");`
- The `require()` will go out and dynamically retrieve the JavaScript code and load them up in the page
- You can write your own packages

13

Dojo libes are organized in packages just like Java packages. Just like Java libraries, you only need to import only the packages you need.

Dojo Toolkit Libraries

- [dojo.io](#): AJAX-based communication with the server
- [dojo.event](#): unified event system for DOM and programmatic events
- [dojo.lang](#): utility routines to make JavaScript easier to use.
- [dojo.string](#): String manipulation routines
- [dojo.dom](#): DOM manipulation routines
- [dojo.style](#): CSS Style manipulation routines

14

This slide and the next slide shows the list of general purpose Dojo toolkit libraries. Among these, “dojo.io” and “dojo.event” libraries are most basic and most important and those are two that we will focus today.

Dojo Toolkit Libraries

- dojo.html: HTML specific operations
- dojo.reflect: Reflection API
- dojo.date: Date manipulation
- dojo.logging.Logger: Logging library
- dojo.profile: JS Profiler
- dojo.regexp: Regular Expression generators
- [dojo.dad: Drag and Drop](#)

Dojo Toolkit Libraries

- `dojo.collections`: Collection data structures
- `dojo.crypto`: Cryptographic API
- `dojo.reflection`: Reflection routines
- `dojo.math`: Mathematic routines
- `dojo.storage`: [Storage routines](#)
- `dojo.uri`: URL handling routines
- `dojo.widget`: [Widget framework](#)



`dojo.io.bind()`

So let's learn about `dojo.io.bind()`.

dojo.io.bind()

- The `dojo.io.*` namespace contains generic APIs for doing network I/O
 - > `dojo.io.bind()` hides low-level `XMLHttpRequest` operations
- Also handles
 - > back-button interception
 - > transparent form submission
 - > advanced error handling

18

The `dojo.io` package contains generic APIs for doing network I/O. Again, it hides low-level `XMLHttpRequest` operations. And in your JavaScript code, `dojo.io.bind()` is how you make AJAX request.

By using `dojo.io.bind()` call, you can also intercept the back-button and provide desired behavior. It also supports transparent form submission. And you can also hook your error handling code.

dojo.io.bind() Syntax

```
// Make a request that returns raw text from a URL
dojo.io.bind({
  // URL to make a request to
  url: "http://foo.bar.com/something",

  // Callback function to execute upon successful response
  load: function(type, data, evt){ /*do something w/ the data */ },

  // Type of data that is returned
  mimetype: "text/plain"

  // More options
});
```

source: dojotoolkit.org

19

So this shows an example of how `dojo.io.bind()` call is used in your JavaScript code.

There are three important things you have to specify when you use `dojo.io.bind()`. The `url` specifies the destination to which the call is intended. The `load` specifies a callback function to be invoked upon a successful response from the server. The `mimetype` indicates the type of data that is returned from the server. In this example, you are asking to receive plain string.

Example #1: Simple Request

```
// Make a request that returns raw text from a URL.
dojo.io.bind({
  url: "http://foo.bar.com/sampleData.txt",
  load: function(type, data, evt){ /*do something w/ the data */ },
  mimetype: "text/plain"
});
```

source: dojotoolkit.org

20

From here, we are going to see several examples on how `dojo.io.bind()` call is used.

This is the same example we saw in the previous slide.

Example #2: Simple Request

```
// Create an argument first
var bindArgs = {
  url: "http://foo.bar.com/sampleData.txt",
  load: function(type, data, evt){ /*do something w/ the data */ },
  mimetype: "text/plain"
};

// dispatch the request
dojo.io.bind(bindArgs);
```

source: dojotoolkit.org

21

This is the same example except that you create a "bindArgs" variable, which is then passed as a parameter to the `dojo.io.bind()` method.

Example #2: Error Handling

```
// Make a request that returns raw text from a URL
// with error handling
dojo.io.bind({
  url: "http://foo.bar.com/sampleData.txt",
  load: function(type, data, evt){ /*do something w/ the data */ },
  error: function(type, error){ /*do something w/ the error*/ },
  mimetype: "text/plain"
});
```

source: dojotoolkit.org

22

This example shows how you can register an error handler

Example #3: Same as Example #2

```
// Handle error condition using type
dojo.io.bind({
  url: "http://foo.bar.com/sampleData.txt",
  handle: function(type, data, evt){
    if(type == "load"){
      // do something with the data object
    }else if(type == "error"){
      // here, "data" is our error object
      // respond to the error here
    }else{
      // other types of events might get passed, handle them here
    }
  },
  mimetype: "text/plain"
});
```

source: dojotoolkit.org

23

It's possible to also register just a single handler that will figure out what kind of event got passed and react accordingly instead of registering separate load and error handlers:

mimetype - Specifies the response format (Client side interpretation)

- text/plain
 - > response is in string format – you will have to parse
- text/javascript
 - > response is in JavaScript format
- text/xml
 - > response is in XML format
- text/json
 - > response is in JSON format

Example #4: Dynamic Content Loading (Getting JavaScript string)

// Request a JavaScript literal string and then evaluate it.

```
dojo.io.bind({  
  url: "http://foo.bar.com/sampleData.js",  
  load: function(type, evaldObj){ /* do something */ },  
  mimetype: "text/javascript"  
});
```

source: dojotoolkit.org

25

One common idiom for dynamic content loading is (for performance reasons) to request a JavaScript literal string and then evaluate it. That's also baked into bind, just provide a different expected response type with the mimetype argument.

Example #5: Explicitly Specify XMLHTTP Transport

```
// If you want to be DARN SURE you're using the XMLHTTP
// transport, specify XMLHTTP Transport.
dojo.io.bind({
  url: "http://foo.bar.com/sampleData.js",
  load: function(type, evaldObj){ /* do something */ },
  mimetype: "text/plain", // get plain text, don't eval()
  transport: "XMLHTTPTransport"
});
```

source: dojotoolkit.org

26

If you want to make sure you are using XMLHTTP transport, you can specify it. In general, this is not needed.

Example #6: Submission of forms via a request

```
// Being a jack-of-all-trades, bind() also supports the submission
// of forms via a request
dojo.io.bind({
  url: "http://foo.bar.com/processForm.cgi",
  load: function(type, evaldObj){ /* do something */ },
  formNode: document.getElementById("formToSubmit")
});
```

source: dojotoolkit.org

27

Being a jack-of-all-trades, bind() also supports the submission of forms via a request (with the single caveat that it won't do file upload over XMLHttpRequest):



Demo: Remoting with `dojo.io.bind()`

28

Now, I am going to give a quick demo on how to use `dojo.io.bind()`. This is something you are going to try yourself during the hands-on lab.



`dojo.io.bind()`: Back/Forward Buttons & Bookmarking

Let's start with the overview.

Example #6: Back Button

```
var sampleFormNode = document.getElementById("formToSubmit");

dojo.io.bind({
  uri: "http://foo.bar.com/sampleData.js",
  load: function(type, evalObj){
    // hide the form when we hear back that it submitted successfully
    sampleFormNode.style.display = "none";
  },
  backButton: function(){
    // ...and then when the user hits "back", re-show the form
    sampleFormNode.style.display = "";
  },
  formNode: sampleFormNode
});
```

source: dojotoolkit.org

30

Under Dojo, catching the back button simply requires registering a callback that will fire when the user hits the back button. The example in the slide the above form submitting example that does a rollback of the form-hide behavior when the back button is clicked:

That's it. Just provide that extra `backButton` argument and your users can now hit the back button and count on your application doing something smarter than destroying their work (with a little help from you, that is).

Example #7: Forward Button

```
var sampleFormNode = document.getElementById("formToSubmit");

dojo.io.bind({
  url: "http://foo.bar.com/sampleData.js",
  load: function(type, evaldObj){
    // hide the form when we hear back that it submitted successfully
    sampleFormNode.style.display = "none";
  },
  backButton: function(){
    // ...and then when the user hits "back", re-show the form
    sampleFormNode.style.display = "";
  },
  forwardButton: function(){
    // and if they hit "forward" before making another request, this
    // happens:
    sampleFormNode.style.display = "none"; // we don't re-submit
  },
  formNode: sampleFormNode
});
```

source: dojotoolkit.org

31

What happens if they hit the forward button after a user hits the backward button? Without any intervention on your part, nothing. But, as I'm sure you've come to expect by now, Dojo gives you a hook to handle that situation as well.

Note that forward button triggers are only fired when the user is linearly progressing between the forward and back buttons. If the user takes an action after going back-back-forward that would fire a new bind() request, the next "forward" will have no callback fired from it since a new branch in the history tree has been created and the old one is likely invalid.

Example #8: Bookmarking

- Simply pass in the value "true" to the flag "changeURL"
- Your top-level page will now have a new timestamp hash value appended to the end

```
dojo.io.bind({  
  url: "http://foo.bar.com/sampleData.txt",  
  load: function(type, data, evt){ /*do something w/ the data */ },  
  changeURL: true,  
  mimetype: "text/plain"  
});
```

source: dojotoolkit.org

32

Bookmarkability is another can of worms for dynamic web applications. With that in mind, `dojo.io.bind()` gives you one last little bit of control for making things bookmarkable. Simply pass in the value "true" to the flag "changeURL". Here's the example:

```
dojo.io.bind({  
  url: "http://foo.bar.com/sampleData.txt",  
  load: function(type, data, evt){ /*do something w/ the data */ },  
  changeURL: true,  
  mimetype: "text/plain"  
});
```

Example #9: Bookmarking

- Alternately, it's possible to include your own hash value by providing a string instead of "true" to changeURL
- <http://foo.bar.com/howdy.php#foo,bar,baz>

```
dojo.io.bind({
  url: "http://foo.bar.com/sampleData.txt",
  load: function(type, data, evt){ /*do something w/ the data */ },
  changeURL: "foo,bar,baz",
  mimetype: "text/plain"
});
```

source: dojotoolkit.org

33

Alternately, it's possible to include your own hash value by providing a string instead of "true" to changeURL:

```
dojo.io.bind({
  url: "http://foo.bar.com/sampleData.txt",
  load: function(type, data, evt){ /*do something w/ the data */ },
  changeURL: "foo,bar,baz",
  mimetype: "text/plain"
});
```

If the URL of the application had previously been:

<http://foo.bar.com/howdy.php>

The user's address bar would now show:

<http://foo.bar.com/howdy.php#foo,bar,baz>



dojo.io.bind(): Transport

Let's start with the overview.

Transports

- `dojo.io.bind` and related functions can communicate with the server using various methods, called transports
 - > XMLHttpRequest
 - > IFrame I/O
 - > ScriptSrcIO
- Each has certain limitations, so you should pick the transport that works correctly for your situation
- The default transport is XMLHttpRequest

source: dojotoolkit.org

35

`dojo.io.bind` and related functions can communicate with the server using various methods, called transports. Each has certain limitations, so you should pick the transport that works correctly for your situation.

The default transport is XMLHttpRequest.

XMLHttp Transport

- It works well in most cases, but it cannot transfer files, cannot work across domains (ie, cannot connect to another site than the current page), and doesn't work with the file:// protocol

```
dojo.require("dojo.io.");

function mySubmit() {
  dojo.io.bind ({
    url: 'server.cfm',
    handler: callBack,
    formNode: dojo.byId('myForm')
  });
}

function callBack(type, data, evt) {
  dojo.byId('result').innerHTML = data;
}
```

source: dojotoolkit.org

36

The XMLHttpRequest transport is the default transport.

It works well in most cases, but it cannot transfer files, cannot work across domains (ie, cannot connect to another site than the current page), and doesn't work with the file:// protocol.

IFrame I/O Transport

- The IFrame I/O transport is useful because it can upload files to the server

```
dojo.require("dojo.io.*");  
dojo.require("dojo.io.IframeIO");
```

```
function mySubmit() {  
  dojo.io.bind ({  
    url: 'server.cfm',  
    transport: "IframeTransport",  
    handler: callBack,  
    formNode: dojo.byId('myForm')  
  });  
}
```

```
function callBack(type, data, evt) {  
  dojo.byId('result').innerHTML = data;  
}
```

source: dojotoolkit.org

37

The IFrame I/O transport is useful because it can upload files to the server.

ScriptSrcIO Transport

- Due to security restrictions, XMLHttpRequest cannot load data from another domain
- The ScriptSrc transport is useful for doing this
- Yahoo's RPC service is implemented using ScriptSrc

```

dojo.require("dojo.io.*");
dojo.require("dojo.io.ScriptSrcIO");

dojo.io.bind({
  uri: "http://example.com/json.php",
  transport: "ScriptSrcTransport",
  jsonParamName: "callback",
  mimetype: "text/json",
  content: { ... }
});

```

source: dojotoolkit.org

38

Due to security restrictions, XMLHttpRequest cannot load data from another domain. The ScriptSrcIO? transport is useful for doing this. Yahoo's RPC service is implemented using ScriptSrcIO?.

To use ScriptSrcIO, use the following require statements:

```

* dojo.require("dojo.io.*");
* dojo.require("dojo.io.ScriptSrcIO");

```

and use the normal dojo.io.bind() method.

To force a ScriptSrcTransport request, use transport: "ScriptSrcTransport" in the keyword arguments to dojo.io.bind().



DOM Manipulation

Dojo toolkit also provides various DOM manipulation APIs.

Dojo DOM Manipulation Routines

- `dojo.byId("someid")`
 - > Same as `document.getElementById("someid");`
- `dojo.dom.isNode(node)`
- `dojo.dom.tagName(node)`
- `dojo.dom.firstElement(node)`
- `dojo.dom.lastElement(node)`
- `dojo.dom.nextElement(node)`
- `dojo.dom.prevElement(node)`

source: j2ee blueprints

40

This slide shows the list of DOM manipulation routines. The `dojo.byId("someid")` works the same as `document.getElementById("someid")` DOM API.

Dojo DOM Manipulation Routines

- `ddojo.dom.moveChildren (srcNode, destNode, trim)`
- `dojo.dom.copyChildren (srcNode, destNode, trim)`
- `dojo.dom.removeChildren(node)`
- `dojo.dom.replaceChildren(node, newChild)`
- `dojo.dom.removeNode(node)`
- `dojo.dom.getAncestors(node, filterFunction, returnFirstHit)`
- `dojo.dom.getAncestorsByTag(node, tag)`
- `dojo.dom.innerHTML(node)`

Dojo DOM Manipulation Routines

- `dojo.dom.createDocumentFromText(str, mimetype):`
- `dojo.dom.insertAfter(node, reference, force):`
- `dojo.dom.insertAtPosition(node, reference, position)`
- `dojo.dom.textContent(node)`
 - > Gets the text-only serialization of a node's children
- `dojo.dom.textContent(node, text)`
 - > Sets the text-only serialization of a node's children



Dojo Event System

Now let's talk about Dojo Event system, which is another important features of Dojo toolkit.

Dojo Event System Topics

- Overview
- Handling DOM events
- Chaining function calls
- Before advice (Aspected-oriented event system)
- Disconnecting event handlers
- Resources

These are the topics we are going to talk about regarding Dojo toolkit event system. First, I will give you a quick overview on the event system. Then I will spend the rest of the presentation explaining various ways of how event system works.



Dojo Event System: Overview

Let's start with the overview.

JavaScript Event Handling

- Events are essential in JavaScript components because
 - > as they drive the user interface
 - > result in AJAX requests
 - > allow JavaScript components to interact with each other

Events are essential in JavaScript components as they drive the user interface, result in AJAX requests, and allow JavaScript components to interact with each other.

Issues of JavaScript Event Handling

- As the number JavaScript components in a page increases, the component code can tend to become more tightly coupled, thus less maintainable
- Attaching multiple event handlers to a node is hard
 - > The previously attached event handler is overwritten by a new one
- Cross browser event handling code is difficult to write from scratch
 - > There are various ways in JavaScript of handling events and each browser has its own quirks and issues

47

Cross browser event handling code is difficult to write from scratch as there are various ways in JavaScript of handling events and each browser has its own quirks and issues.

As the number JavaScript components in a page increases, the component code can tend to become more tightly coupled. This is not an effective way of developing user interfaces as the resulting code becomes less re-usable, difficult to manage, and maintain.

There is a need for a way to effectively allow JavaScript components to not be as "tightly coupled" and be capable of being used with many other components with a minimum set of glue code.

Also attaching multiple event handlers to a node is hard. As the example code in the slide shows, the previously attached event handler is clobbered when you attach a new one,

Why Dojo Event System?

- It abstracts the JavaScript event system
 - > Lets you register to "hear" about any action through a uniform and simple to use API - `dojo.event.connect()`
- Treat any function call as an event that can be listened to
 - > Handles more than simple DOM events
- It provides advanced event handling schemes
 - > Aspect oriented - your event handler can be called "before" or "after"
- Less unobtrusive
 - > The setting of event handlers on DOM Nodes happen without explicit `on*` attributes in markup

48

So what are the reasons for using Dojo event system? It addresses the issues mentioned in the previous slide.

First, it abstracts the JavaScript event system. It let's you register or attach an event handler through a uniform and simple API called `dojo.event.connect()`.

Unlike the DOM events that web programmers normally associate with the word "event", Dojo takes a broad view of events. The tools in `dojo.event.*` allow developers to treat any function call (DOM event or otherwise) as an event that can be listened to

Dojo event system provides aspect oriented event system. What this means is that you can attach an event handler not only after but also before an event is fired.

Usage of `dojo.event.connect()` is less unobtrusive since the setting of event handlers on DOM nodes are done explicitly.

`dojo.event.connect(srcObj, "srcFunc", "targetFunc")`

- Provides a **uniform API for event handling**
 - > Abstracts browser differences
- Takes care of the details of attaching more than one event handler (multiple event handlers) to a single event type
- Prevents memory leaks that appear on some browsers

49

Now let's talk about the `dojo.event.connect()` call. It provides a uniform API for event handling. It also abstracts browser differences. It does prevent memory leaks that appear in some browsers. And as you will see later on, it takes care of the details of attaching more than one event handler to a single event.

As for the syntax of the `dojo.event.connect()` method, we will learn them in the rest of the presentation by looking at the example code.



Dojo Event System: Handling DOM Events

So let's learn how to do use Dojo event system by looking at the examples.

Example #1 - DOM event Using Named Event Handler

```
window.onload = function () {  
  var link = document.getElementById("mylink");  
  
  // "myHandler" event handler is registered for the  
  // "onclick" event of "mylink" node.  
  dojo.event.connect(link, "onclick", myHandler);  
}  
  
// Define an event handler named as "myHandler"  
function myHandler(evt) {  
  alert("myHandler: invoked - this is my named event handler");  
}  
</script>  
<a href="#" id="mylink">Click Me</a>
```

51

This is the similar example as the previous slide with more context.

Above the "onclick" property of link element is mapped to the event handler function myHandler. If there was an existing handler the Dojo handler will be called after the existing handler is called.

Example #2 Using Unnamed (Anonymous) Event Handler

```
window.onload = function () {
  var link = document.getElementById("mylink");
  // connect link element 'onclick' property to an anonymous function
  dojo.event.connect(link, "onclick", function(evt) {
    var srcElement;
    if (evt.target) {
      srcElement = evt.target;
    } else if (evt.srcElement) {
      srcElement = evt.srcElement;
    }
    if (srcElement) {
      alert("dojo.event.connect event: " + srcElement.getAttribute("id"));
    }
  });
}
</script>
<a href="#" id="mylink">Click Me</a>
```

52

This is the similar example with more context. This example also shows how to use the event parameter.

Example #3: Attaching a method of an object as an event handler

```
// Identify the node for which you want register event handler
var handlerNode = document.getElementById("handler");

// This connect() call ensures that when handlerNode.onclick()
// is called, object.handler() will be called with the same arguments.
dojo.event.connect(handlerNode, "onclick", object, "handler");
```

53

This example code shows how to attach a method of an object as an event handler to a DOM node. If there is already an event handler attached for the onclick event, it will be called first and the handler method of the object will be invoked.

Example #4: Registration of Multiple Handlers

```
var handlerNode = document.getElementById("handler");  
  
// Connect also transparently handles multiple listeners.  
// They are called in the order they are registered.  
// This would kick off two separate actions from a single onclick event:  
dojo.event.connect(handlerNode, "onclick", object, "handler");  
dojo.event.connect(handlerNode, "onclick", object, "handler2");
```

54

This example shows how to attach multiple event handlers are registering. They will be called in the order they are registered.



Dojo Event System: Chaining Function Calls

So far, we have seen how to attach or register event handlers to a DOM node.

Attaching function of an object to another function

- Used when you have a need to invoke another function **when a function is invoked**
 - > The source of the event is a function call not DOM event
- Use the same `dojo.event.connect()`
 - > `dojo.event.connect(srcObj, "srcFunc", targetObj, "targetFunc");`

56

Using Dojo event system, you can attach a function of an object to another function. This could be used when you have a need to invoke another function when a function is invoked.

The syntax of the call is you specify the source object and source function and target object and target function. So let's see an example of this.

Example #5a: Attaching a function of an object to another function

// Define a simple object with a couple of methods

```
var exampleObj = {  
  counter: 0,  
  foo: function(){  
    this.counter++;  
    alert("foo: counter = " + this.counter);  
  },  
  bar: function(){  
    this.counter++;  
    alert("bar: counter = " + this.counter);  
  }  
};
```

// I want exampleObj.bar to get called whenever exampleObj.foo
// is called. How can I do this?

57

Let's say that there are foo and bar functions. What I want to do is to invoke "bar" function whenever "foo" function is called.

Example #5b: Attaching a function of an object to another function

```
// We can set this up the same way that we do with DOM events.  
// Now calling foo() will also call bar(), thereby incrementing the  
// counter twice and alerting "foo" and then "bar".  
dojo.event.connect(exampleObj, "foo", exampleObj, "bar");
```

58

This is how you do it. Since we use the same object, the source object and target object are the same. And the source function is "foo" and the target function is "bar". So calling "foo" will also call "bar".



Dojo Event System: Aspect-Oriented ("before advice" and "after advice")

Now let's talk about aspect-oriented event handling. Basically, Dojo event system lets you call an event handler "before" or "after" an event is fired. And they are called "before advice" and "after advice".

Before Advice

- `dojo.event.connect()` can be used to call listeners (event handlers) before the source function is called
 - > In Aspect Oriented Programming terminology, this is called "before advice"
- Example: "bar" gets called before "foo" when `exampleObj.foo` is called:
 - > `dojo.event.connect("before", exampleObj, "foo", exampleObj, "bar");`
- "after advice" is a default

60

By using "before advice", you can call event handlers before the event occurs or source function is called. Now how do you do it? Basically, you provide "before" flag as the first parameter of the `dojo.event.connect()` call.

If you don't provide it, "after" is assumed as a default and that is what we've seen so far.

Example #6: Attaching a function of an object to another function - “before”

```
var exampleObj = {  
  counter: 0,  
  foo: function(){  
    this.counter++;  
    alert("foo: counter = " + this.counter);  
  },  
  bar: function(){  
    this.counter++;  
    alert("bar: counter = " + this.counter);  
  }  
};
```

```
// "bar" function will be called before "foo"  
dojo.event.connect("before", exampleObj, "foo", exampleObj, "bar");
```

source: j2ee blueprints

61

So here is the example in which you want to call “bar” first before “foo”.

Event Handler Wrapper

- In some case you may want to modify the behavior or arguments of an event handler without modifying the source code of the JavaScript component you are using
 - > Adding "before" or "after" an event handler listeners may not be enough
- In Dojo, this is accomplished using `dojo.io.connect` with "around" as the first argument
 - > `dojo.io.connect("around", ...);`
- It behaves like Servlet Filter scheme
 - > Intercept requests and modify the behavior

62

In some cases, you may want to modify behavior or arguments of an event handler without modifying the source code of the JavaScript component you are using. And "before" and "after" event handlers might do any good.

In Dojo event system, this can be accomplished using "around" flag as the first argument. In a sense, it works like servlet filter, which lets you intercept the incoming request and modify it before it passes it to the destination.

In a sense, this behaves like servlet filter scheme, in which all incoming HTTP requests are intercepted and can be modified before they are passed to the servlet.

Example #7: Event Handler Wrapper

```

// custom event handler wrapper
function customLoadHandler(invocation) {
    alert("custom menu name = " + invocation.args[0].name);
    // update the name property of the argument
    invocation.args[0].name = "Custom Menu";
    //call the default event handler
    invocation.proceed();
}

function ImageScroller() {
    this.load = function (args) {
        alert("default menu name=" + args.name);
    }
}

var is = new ImageScroller();
dojo.event.connect("around", is, "load", this, "customLoadHandler");
is.load({name: "My Menu", items: ['File', 'Save']});
// alerts "custom menu name=My Menu"
// alerts "default menu name=Custom Menu"

```

63

In the example above shows how you can write custom code to intercept the call to the public function load on ImageScroller and wrap it with the function customLoadHandler. The customLoadHandler function in this example manipulates the arguments that are presented to the load function. Note that customLoadHandler function is passed a single argument by the Dojo runtime (invocation in the example above) with two properties: "args" which is an array of the arguments passed to the target event handler and "proceed" which is a function that will call the target event handler.

Publish/Subscribe Event Handling

- Used when you have a need to communicate events anonymously between components
- Allows customizing the component to allow the topic name to be passed in as an initialization parameter to make the component more flexible



Demo: Dojo Event System

65

So let me do some demos on how to use `dojo.event.connect()` for the various event handling's we've talked about so far.



Dojo Event System: Publish & Subscribe

What is Publish and Subscribe?

- In addition to the simple event system created by `dojo.connect`, dojo offers support for anonymous publication and subscription of objects, via *dojo.publish* and *dojo.subscribe*
 - > These methods allow a function to broadcast objects to any other function that has subscribed.
 - > This is dojo's topic system, and it makes it very easy to allow separate components to communicate without explicit knowledge of one another's internals.

Three functions

- `dojo.publish`
 - > Calls any functions that are connected to the topic via `dojo.subscribe`, passing to those subscribed functions arguments that are published
- `dojo.subscribe`
 - > Connect a junction to a topic
- `dojo.unsubscribe`
 - > Will cause a previously subscribed function to no longer be called when `dojo.publish` is called in the future

Three functions: Syntax

- `dojo.publish`
 - > `dojo.publish`(Topic Name [string], Arguments to Pass to Subscribed Function [array])
- `dojo.subscribe`
 - > `handle = dojo.subscribe`(Topic Name [string], Context of Linked Method [string or null], Linked Method [string or function])
- `dojo.unsubscribe`
 - > `dojo.unsubscribe`(Handle [handle object])

Example Code for Reference

```
function globalGuy(arg) { console.debug("Global Guy  
fired with arg " + arg); }  
var someObject = {  
  bar: function(first, second) { console.debug("Bar  
fired with first of "+first+" and second of "+second);  
return 7; },  
}
```

Subscribing and Publishing

- Subscribing

```
topics = [];  
topics[0] = dojo.subscribe("globalEvents", null, globalGuy);  
topics[1] = dojo.subscribe("fullNames", "someObject", bar);
```

- Publishing

```
dojo.publish("globalEvents", ["data from an interesting source"]);  
dojo.publish("fullNames", ["Alex", "Russell"]);
```



Dojo Event System: Disconnecting Event Handlers

So far, we learned how to attach or register event handlers. Now let's talk a bit on how to disconnect event handlers.

Disconnecting Event Handler

- Use `dojo.event.disconnect()` for events
 - > Must pass exactly the same arguments as were passed to `dojo.event.connect()`
- `dojo.event.topic.unsubscribe()` for topics
 - > Must pass exactly the same arguments as were passed to `dojo.event.topic.subscribe()`

73

Basically you use `dojo.event.disconnect()` all for disconnecting event handlers. One caveat is that you have to pass exactly the same arguments as were passed to `dojo.event.connect()` call.

Same is true for publish/subscribe event handling.



Dojo Event System: Resources

Resources on Dojo Event System

- “Events for JavaScript Components” article written by Greg Murray
 - > https://bpcatalog.dev.java.net/source/browse/*checkout*/bpcatalog/ee5/docs/ajax/handling-js-events.html
- “Dojo Event System” home page
 - > <http://dojotoolkit.org/book/dojo-book-0-9/part-3-programmatic-dijit-and-dojo/event-system>
- “Dojo Event Examples” Wiki
 - > <http://dojo.jot.com/EventExamples>

75

This is the list of useful resources for learning Dojo toolkit's event system.



Dojo Toolkit V0.9

Dojo Toolkit 0.9

- Major change from 0.4.3
- **Not backward compatible** with 0.4.3
 - > The amount of code needed to maintain backwards compatibility with 0.4.3 would have bloated the size of 0.9 release defeating one of the major points of such a drastic reorganization of the Dojo code base, footprint on the wire.
- `dojo.io.bind()` has been replaced with more specific functions that give a better indication about which IO transport is being used
 - > `dojo.xhrGet(..)`
 - > `dojo.xhrPost(..)`

77

This is the list of useful resources for learning Dojo toolkit's event system.



Questions?

Sang Shin
Java Technology Architect
Sun Microsystems, Inc.
sang.shin@sun.com
www.javapassion.com

Welcome to the "Introduction to Dojo toolkit" presentation. My name is Sang Shin from Sun Microsystems.