Dojo Basics 1 (dojo base and core)

Topics

- What is and why Dojo toolkit?
- Dojo architecture
- Loading and configurating Dojo toolkit
- DOM manipulation via *dojo.byId* & more
- Document lifecycle interception via *dojo.addOnload*
- DOM query via *dojo.query*
- Array manipulation via *dojo.NoteList* & *dojo.forEach*
- Event handling via *dojo.connect*

Topics (Continued)

- Event propagation via *dojo.subscribe* and *dojo.publish*
- *OOP support via dojo.declare, dojo.mixin, dojo.extend*

Dojo Learning Resources You Should know upfront

- http://docs.dojocampus.org/
  - Official Dojo documentation
  - Search for anything you want to know
- http://dojocampus.org/explorer/
  - Dojo Feature Explorer

- Both come with lots of ready-to-run examples and their code
- We are going to use these two resources for most of our demos

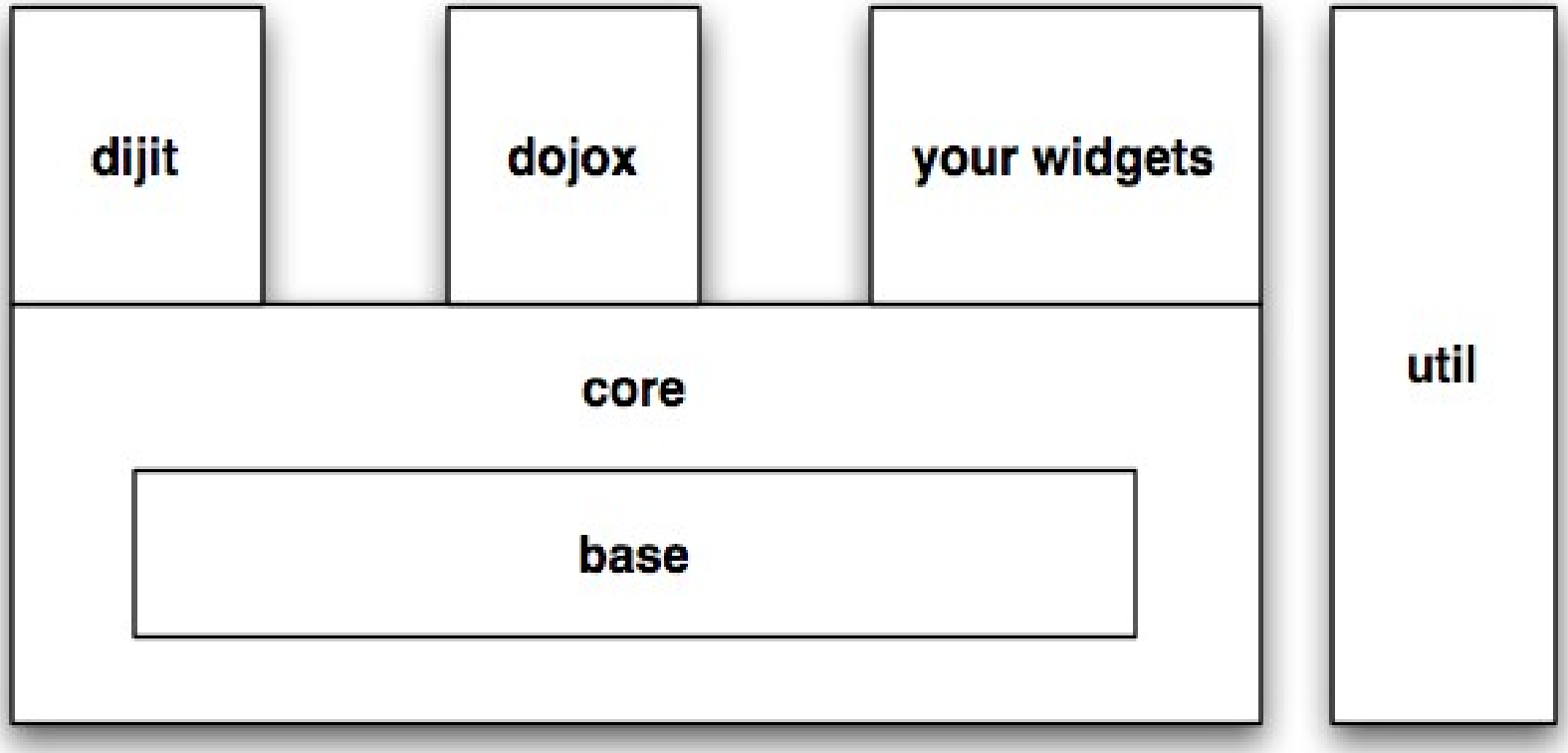
What is and Why Dojo Toolkit?

What is and Why Dojo Toolkit?

- Unified JavaScript Toolkit
  - Comprehensive feature set
  - Rich Widget set
- High quality, high performance, proven
- Easy to learn
- Use only pieces you want (efficient)
  - *dojo.js* is 28k
- Well-structured
  - core+widgets+extras
- Active developer community
- Open-source

Dojo Architecture

Dojo Toolkit Architecture



Dojo Toolkit: 3 Main Parts

- dojo (base + core)
  - Browser normalization, package,
  - DOM access and manipulation, Firebug Lite debugging, Events,
  - Data access, Drag and drop, Ajax, JSON, and more
- dijit
  - Widgets
- dojox
  - Grid, Charting, CometD, Offline/Google Gears, 2D/3D GFX, many more

Loading & Configuring Dojo Toolkit

Two Options

- Option #1: Load it from CDN (Content Delivery Network)
- Option #2: Load it from locally installed version

Load it from CDN

- From Google

```
<script
src="http://ajax.googleapis.com/ajax/libs/dojo/1.5/dojo/do
jo.xd.js">
</script>
```

- From AOL

```
<script
      src="http://o.aolcdn.com/dojo/1.5/dojo/dojo.xd.js">
</script>
```

- CDN (Content Delivery Network)
  - Geographic edge caching
  - gzip-compressed

Option #2: Load it from locally installed version

- You have to download it first from http://dojotoolkit.org/downloads

- Two options
  - Install it docroot directory of your web server (one time installation)
  - Install it as part of each Web application

```
<script type="text/javascript"
    src="/path/dojo.js" >
</script>
```

djConfig

- Dojo allows developers to override certain global settings
  - Whether to trigger automatic parsing of dijit objects when a page is loaded
  - Whether to enable Firebug-Lite
  - Setting specific i18n or localization

Example: djConfig

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
    <meta http-equiv="Content-Type" content="text/html;
          charset=UTF-8">
    <title>Dojo djConfig Tutorial&lt;/title>
    <script type="text/javascript"
          src="http://CDNpath/dojo/dojo.xd.js"
          djConfig="parseOnLoad: true,
                    isDebug: true",
                    locale: 'en-us',
                    extraLocale: ['ja-jp','zh-cn']>
    </script>
</head>
<body>
    <p>...</p>
</body>
</html>
```

DOM Manipulation via dojo.byId & more

dojo.byId

- Same as "document.getElementById”

```
// fetch a node whose id="someNode"
var node = dojo.byId("someNode");

// set a node to display "Hello World"
dojo.byId("someNode").innerHTML = "Hello World";

// If you pass dojo.byId a string, and no domNode
// is found to match, ''undefined'' or the null
// object is returned (depending on the browser),
// which is adequate truthiness to use conditionally:
var node = dojo.byId("notExistent");
if(node){
  node.innerHTML = "I was found!";
}else{
  console.log("no node with id='fooBar' found!");
}
```

Other DOM Manipulation Examples

```
// Set a node to have a new id
dojo.attr(someNode, "id", "newId");

// Create a <div> with content:
var n = dojo.create("div", { innerHTML: "<p>hi</p>" });

// add someId to the node someOtherId as the first-child
dojo.place("someId", "someOtherId", "first");
```

Document Lifecycle Management via dojo.addOnLoad

dojo.addOnLoad

- Passing addOnLoad a function will register the function to run when the DOM is ready
- dojo.addOnLoad(...) defers script execution until all the HTML and modules are loaded

```
function setAfrobeat(){
   document.musicPrefs.other.value="Afrobeat";
}
dojo.addOnLoad(setAfrobeat);

// Same as above. When the function is small,
// you may prefer to write it inline:
dojo.addOnLoad(
  function(){
    document.musicPrefs.other.value="Afrobeat";
  }
);
```

DOM Query via dojo.query()

dojo.query

- *dojo.query()* returns a list of DOM nodes based on a CSS selector
  - Returns *dojo.NodeList* object
- CSS selector provides comprehensive set of selection criteria

A bad solution: using the DOM API (Not using dojo.query)

```
<!-- To select HTML elements in JavaScript, you can use
the browser's native DOM API, but they're verbose and hard
to work with...not to mention slow. -->

<script type="text/javascript">

  // list every node with the class "progressIndicator":
  var list = [];
  var nodes = document.getElementsByTagName("*");

  // iterate over every node in the document....SLOOOW
  for(var x = 0; x < nodes.length; x++){
      // only nodes with the class "progressIndicator":
      if(nodes[x].className == "progressIndicator"){
          // add to array:
          list.push(nodes[x]);
      }
  }
  console.dir(list);
</script>
```

Better and faster: dojo.query

```
<!-- dojo.query gives us a more compact way to do it, and
it's often faster, particularly as we ask for more
sophisticated kinds of relationships. -->

<script type="text/javascript">
  // Find all nodes with the class "progressIndicator":
  dojo.query(".progressIndicator");

  // Search in the subtree of the an existing node
  var myList = dojo.byId('myList');
  var items = dojo.query('li', myList);
</script>
```

Example Queries with CSS Selector

```
// all <h3> elements
dojo.query('h3')
// all <h3> elements which are first-child of their
// parent node
dojo.query('h3:first-child')
// a node with id="main"
dojo.query('#main')
// all <h3> elements within a node with id="main"
dojo.query('#main h3')
// a <div> with an id="main"
dojo.query('div#main')
// all <h3> elements within a div with id="main"
dojo.query('div#main h3')
// all <h3> elements that are immediate children of a
// <div>, within node with id="main"
dojo.query('#main div > h3')
// all nodes with class="foo"
dojo.query('.foo')
// all nodes with classes "foo" and "bar"
dojo.query('.foo.bar')
```

Array manipulation via dojo.NodeList & dojo.forEach()

dojo.NodeList

- Subclass of Array
- Adds syntactic sugar for
  - Chaining
  - Common iteration operations
  - Animation
  - Node manipulation
- Returns as a result of dojo.query()

Example: dojo.NodeList

```
// Set onclick handler for all lists (ordered
// and unordered)
dojo.query('ol, ul').onclick(function(event){});

// Iterator over items in list (ordered
// and unordered)
dojo.query('ol li, ul li').forEach(function(){});

// Empty the snippet
dojo.query('#snippet').empty();
```

Array Manipulation

- Dojo comes with a bunch of useful methods to deal with arrays, a few more than you get from your browser by default.
- Examples
  - dojo.forEach - most useful
  - dojo.indexOf
  - dojo.lastIndexOf
  - dojo.filter
  - dojo.map
  - dojo.some
  - dojo.every

dojo.forEach()

```
<script type="text/javascript">
    dojo.require("dijit.form.Button");
    var arrFruit = ["apples", "kiwis", "pineapples"];

    function populateData() {
        dojo.forEach(arrFruit, function(item, i) {
            var li = dojo.doc.createElement("li");
            li.innerHTML = i + 1 + ". " + item;
            dojo.byId("forEach-items").appendChild(li);
        });
    }
</script>
```

dojo.forEach() with dojo.query()

```
// The returned object of a dojo.query() call is an
// instance of dojo.NodeList, a subclass of Array
// with many convenience methods added for making
// DOM manipulation and event handling easier.

dojo.addOnLoad(function(){

  // For every element class "blueButton" assigned,
  // invoke a function.
  dojo.query(".blueButton")
      .forEach(function(node, index, arr){
                  console.debug(node.innerHTML);
              }
  );
});
```

dojo.some() - tell you whether an array has some of the asked values

```
<script type="text/javascript">
    // this Button is just to make the demo look nicer:
    dojo.require("dijit.form.Button");

    var arrIndxSome = [200000, 500000, 350000,
                       1000000, 75, 3];

    function testIndxSome() {
        if (dojo.some(arrIndxSome, function(item) {
            return item >= 1000000
        })) {
            result = 'yes, there are';
        } else {
            result = 'no, there aren no such items';
        }
        dojo.place("<p>The answer is: " + result + "</p>",
                   "result6", "after");
    }
</script>
```

Event handling via dojo.connect()

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

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

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.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

dojo.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

Dojo Event System:
Handling DOM Events

Example #1 - DOM event Using Named Event Handler

```
window.onload = function () {
  var link = dojo.byId("mylink");

  // “myHandler” event handler is registered for the
  // "onclick" event of “mylink” node.
  dojo.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>
```

Example #2 Using Unnamed (Anonymous) Event Handler

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

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

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

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

Example #4: Registration of Multiple Handlers

```
var handlerNode = dojo.byId("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.connect(handlerNode, "onclick", object, "handler");
dojo.connect(handlerNode, "onclick", object, "handler2");
```

Dojo Event System:
Chaining Function Calls

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.connect()
  - dojo.connect(srcObj, "srcFunc", targetObj, "targetFunc");

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?
```

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.connect(exampleObj, "foo", exampleObj, "bar");
```

Publish & Subscribe via dojo.publish() & dojo.subscribe()

What is Publish and Subscribe?

- Used for anonymous publication and subscription of topics (channels)
- Allows separate components to communicate one another

Three functions

- dojo.publish
  - Used by the publisher of the event to publish an event to the topic (channel)
- dojo.subscribe
  - Used by the subscriber of the event to subscribe a topic (channel)
- dojo.unsubscribe
  - Used by the subscriber to unsubscribe previously subscribed topic (channel)

dojo.subscribe & dojo.publish

```
// Subscribe a channel - function gets called
// with data when an event is published
dojo.subscribe("/foo/bar/baz", function(data){
    console.log("i got", data);
 });

// Publish an event to the channel with some data
dojo.publish("/foo/bar/baz", [{ some:"object data" }]);


// The channel name can be any string you choose:
dojo.subscribe("foo-bar", function(data){ /* handle */ });
dojo.subscribe("bar", function(data){ /* handle */ });
dojo.subscribe("/foo/bar", function(data){ /* handle*/ });
```

OOP Support via dojo.declare(), dojo.mixin(), dojo.extend()

dojo.declare()

- Javascript doesn't have a Class system with built-in inheritance like Java
- *dojo.declare()* simulates the Java Class system
- Syntax
  - dojo.declare('className', superClass, property map)

Example: dojo.declare()

```
// Declare a simple class named my.Thinger, not based
// on any parent, and provide a single property named
// constructor. The constructor function is run once
// for each mixed Class.
dojo.declare("my.Thinger", null, {
  constructor: function(/* Object */args){
    // mixes the variable count into the properties
    // of the instance, making it available as a
    // member of the instance
    dojo.safeMixin(this, args);
  }
});

// You could then create a Thinger
var thing = new my.Thinger({ count:100 });
console.log(thing.count);
```

Example: dojo.declare() - Inheritance

```
// Create a new class derived from my.Thinger.
// First, the constructor of my.Thinger is called,
// mixing in the args parameter. Then, we're using
// the reserved word this to access instance
// properties, creating a new instance property
// total based on some simple code.
dojo.declare("my.OtherThinger", [my.Thinger], {
  divisor: 5,
  constructor: function(args){
    console.log('OtherThinger constructor called');
    this.total = this.count / this.divisor;
  }
});
var thing = new my.OtherThinger({ count:50 });
console.log(thing.total); // 10
```

dojo.mixin()

- dojo.mixin() is a simple utility function for mixing objects together.
- Mixin combines two objects from right to left, overwriting the left-most object, and returning the newly mixed object for use.
- Dojo mixin is very similar to dojo.extend() but only works on objects, whereas extend explicitly extends an object.prototype.

Example: dojo.mixin()

```
var a = { b:"c", d:"e" };
dojo.mixin(a, { d:"f", g:"h" });
console.log(a); // b:c, d:f, g:h
```

dojo.extend()

- dojo.extend() works much like dojo.mixin(), though works directly on an object's prototype.
- Following the same pattern as mixin, dojo.extend() mixes members from the right-most object into the first object, modifying the object directly.
- We can use extend to extend functionality into existing classes

Example: dojo.extend()

```
// After the extend, any new instances of a TitlePane
// will have the 'randomAttribute' member mixed into
// the instance. dojo.extend affects all future
// instances of a Class (or rather, any object with
// a .prototype).
dojo.require("dijit.TitlePane");
dojo.extend(dijit.TitlePane, {
    randomAttribute:"value"
});


// Now randomAttribute is recognized
<div dojoType="dijit.TitlePane"
     randomAttribute="newValue">
</div>
```

Thank you!

Check JavaPassion.com Codecamps!
http://www.javapassion.com/codecamps
"Learn with Passion!"