FREE eBook

LEARNING dojo

Free unaffiliated eBook created from
Stack Overflow contributors.

#dojo

Table of Contents

About

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: dojo

It is an unofficial and free dojo ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official dojo.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Chapter 1: Getting started with dojo

Remarks

This section provides an overview of what dojo is, and why a developer might want to use it.

It should also mention any large subjects within dojo, and link out to the related topics. Since the Documentation for dojo is new, you may need to create initial versions of those related topics.

Versions

Version	Release Date
0.4.4	2006-11-05
1.0.3	2007-11-05
1.1.2	2008-03-26
1.2.4	2008-10-02
1.3.3	2009-03-26
1.4.6	2009-12-07
1.5.4	2010-07-22
1.6.3	2011-03-15
1.7.10	2011-10-27
1.8.12	2012-08-15
1.9.9	2013-05-01
1.10.6	2014-06-13
1.11.2	2016-06-09
1.12.1	2016-12-21

Examples

Installation or Setup

Use Dojo from CDN

Load Dojo through `<script>` tags in your HTML page pointing to Google CDN.

Example:

```
<script src="//ajax.googleapis.com/ajax/libs/dojo/1.11.2/dojo/dojo.js"></script>
```

Install Dojo with Bower

Type the following command in your project directory:

```
bower install dojo/dojo dojo/dijit dojo/dojox dojo/util
```

Bower installs to a bower_components sub-directory by default, but if you'd like to install to the current directory instead add a `.bowerrc` with the following:

```
{
    "directory": "."
}
```

Use dojo themes from CDN

Dojo provides us various themes like tundra, claro etc.

Load themes using `link` tag in your HTML page pointing to Google CDN.

Sample page

This example is a sample page that shows how to use **Dojo** to display a **"Hello world"** text inside `<h1>` tag.

```
<!DOCTYPE html>
<html>

  <head>
    <meta charset="utf-8">
    <title>Dojo sample</title>
    <script src="//ajax.googleapis.com/ajax/libs/dojo/1.12.1/dojo/dojo.js" data-dojo-
config="async: true"></script>
  </head>

  <body>
    <h1 id="Hello"></h1>


    <script>
      require([
        'dojo/dom'
      ], function(dom) {

        dom.byId('Hello').innerHTML = 'Hello world';
      });

    </script>
```

```
  </body>

</html>
```

Read Getting started with dojo online: https://riptutorial.com/dojo/topic/3540/getting-started-with-dojo

Chapter 2: Configuring Dojo with dojoConfig

Remarks

The **dojoConfig** object (formerly **djConfig**) allows you to set options and default behavior for various aspects of the dojo toolkit. Examples will explain what's possible and how you can put dojoConfig to use in your code.

Note that dojoConfig is defined in a script block before dojo.js is loaded. This is of paramount importance—if reversed, the configuration properties will be ignored.

Examples

Load DojoConfig

In Below sample we are creating one global javascript object `dojoConfig` which will contain all the configuration values.

Note: dojoConfig is defined in a script block before dojo.js is loaded. This is of paramount importance—if reversed, the configuration properties will be ignored.

```
<script>
    dojoConfig= {
        has: {
            "dojo-firebug": true
        },
        parseOnLoad: false,
        foo: "bar",
        async: true
    };
</script>
<script src="//ajax.googleapis.com/ajax/libs/dojo/1.10.4/dojo/dojo.js"></script>

<script>
// Require the registry, parser, Dialog, and wait for domReady
require(["dijit/registry", "dojo/parser", "dojo/json", "dojo/_base/config", "dijit/Dialog",
"dojo/domReady!"]
, function(registry, parser, JSON, config) {
    // Explicitly parse the page
    parser.parse();
    // Find the dialog
    var dialog = registry.byId("dialog");
    // Set the content equal to what dojo.config is
    dialog.set("content", "<pre>" + JSON.stringify(config, null, "\t") + "```");
    // Show the dialog
    dialog.show();
});
</script>

<!-- and later in the page -->
<div id="dialog" data-dojo-type="dijit/Dialog" data-dojo-props="title: 'dojoConfig /
dojo/_base/config'"></div>
```

Loader Configuration

Dojo received a new loader in Dojo 1.7 to accommodate for the toolkit's new AMD module format. This new loader added a few new configuration options that are crucial to defining packages, maps, and more. For details on the loader, see the Advanced AMD Usage tutorial. Important loader configuration parameters include:

baseUrl: The base URL prepended to a module identifier when converting it to a path or URL.

```
baseUrl: "/js"
```

packages: An array of objects which provide the package name and location:

```
packages: [{
    name: "myapp",
    location: "/js/myapp"
}]
```

map: Allows you to map paths in module identifiers to different paths:

```
map: {
    dijit16: {
        dojo: "dojo16"
    }
}
```

paths: a map of module id fragments to file paths:

```
var dojoConfig = {
    packages: [
        "package1",
        "package2"
    ],
    paths: {
        package1: "../lib/package1",
        package2: "/js/package2"
    }
};

    // ...is equivalent to:
var dojoConfig = {
    packages: [
        { name: "package1", location: "../lib/package1" },
        { name: "package2", location: "/js/package2" }
    ]
};
```

async: Defines if Dojo core should be loaded asynchronously. Values can be true, false or legacyAsync, which puts the loader permanently in legacy cross-domain mode.

```
async: true
```

parseOnLoad: If true, parses the page with dojo/parser when the DOM and all initial dependencies (including those in the dojoConfig.deps array) have loaded.

```
parseOnLoad: true

It is recommended that parseOnLoad be left at false (it defaults to false, so you can simply
omit this property), and that developers explicitly require dojo/parser and call
parser.parse().
```

deps: An array of resource paths which should load immediately once Dojo has loaded:

```
deps: ["dojo/parser"]
```

callback: The callback to execute once deps have been retrieved:

```
callback: function(parser) {
    // Use the resources provided here
}
```

waitSeconds: Amount of time to wait before signaling load timeout for a module; defaults to 0 (wait forever):

```
waitSeconds: 5
```

cacheBust: If true, appends the time as a querystring to each module URL to avoid module caching:

```
cacheBust: true
```

Now let's create a simple demo that puts the basic parameters to use. One very common scenario is using Dojo Toolkit from CDN with local modules.

Let's say we use Google CDN with modules in the `/documentation/tutorials/1.10/dojo_config/demo` space:

```
<!-- Configure Dojo first -->
<script>
    dojoConfig = {
        has: {
            "dojo-firebug": true,
            "dojo-debug-messages": true
        },
        // Don't attempt to parse the page for widgets
        parseOnLoad: false,
        packages: [
            // Any references to a "demo" resource should load modules locally, *not* from CDN
            {
                name: "demo",
                location: "/documentation/tutorials/1.10/dojo_config/demo"
            }
        ],
        // Timeout after 10 seconds
```

```
        waitSeconds: 10,
        map: {
            // Instead of having to type "dojo/domReady!", we just want "ready!" instead
            "*": {
                ready: "dojo/domReady"
            }
        },
        // Get "fresh" resources
        cacheBust: true
    };
</script>

<!-- Load Dojo, Dijit, and DojoX resources from Google CDN -->
<script src="//ajax.googleapis.com/ajax/libs/dojo/1.10.4/dojo/dojo.js"></script>

<!-- Load a "demo" module -->

<script>
    require(["demo/AuthoredDialog", "dojo/parser", "ready!"], function(AuthoredDialog, parser)
{
        // Parse the page
        parser.parse();

        // Do something with demo/AuthoredDialog...
    });
</script>
```

By using the packages configuration, we've made all references to `demo/*` point to our local `/documentation/tutorials/1.10/dojo_config/demo/` directory, while allowing any references to `dojo`, `dijit`, and `dojox` to come from Google CDN. Had the demo package not been defined, the request for `demo/AuthoredDialog` would have gone to `//ajax.googleapis.com/ajax/libs/dojo/1.10.4/dojo/demo/AuthoredDialog.js`. We also used alias, by associating ready with `dojo/domReady`.

Read Configuring Dojo with dojoConfig online: https://riptutorial.com/dojo/topic/5699/configuring-dojo-with-dojoconfig

Chapter 3: DOM Manipulation

Introduction

Dojo provides different functions that allows you to manipulate DOM elements such as creation, placement and destruction.

Parameters

Argument	Type
node	DomNode or String

Examples

dom-construct

This module can be used to :

- Create a new element.
- Delete an element from HTML document.
- Place element in HTML document.

Iinitialisation

To be able to use the `dom-construct` module we need to load it as fallow :

```
require(["dojo/dom-construct"], function(domConstruct){
    // Write code here
});
```

---

create()

This function can be used to create an element and add it in a specific position. It also allows you to set attributes and content.

Usage

```
var node = domConstruct.create("div", { style: { color: "red" }}, "someId", "first");
```

---

destroy()

This function allows you to delete an element including it's children and content from the document.

Usage

```
domConstruct.destroy("someId");
```

---

place()

This function can be used to place nodes in a particular position in an HTML document

Usage

```
domConstruct.place("someNode", "refNode", "after");
```

---

empty()

This function can be used to delete content and all its children of a DOM element

Usage

```
domConstruct.empty("someId");
```

dom-class

This module provides function that allows you to manipulate CSS classes of DOM elements.

Initialization

To be able to use the dom-class module we need to load it as fallow :

```
require(["dojo/dom-class"], function(domClass){
    // Write code here
});
```

---

contains()

This function checks if a node contains a specific class

Usage

```
if (domClass.contains("someId", "className")){
    // do something if it contains
}
```

---

add()

This function allows you to add CSS Classes to a DOM node without duplication.

Usage

```
domClass.add("someId", "className");
```

---

remove()

This function allows you to remove CSS Classes from a DOM node.

Usage

```
domClass.remove("someId", "className");
```

---

replace()

This function allows you to remove classes and replace it with other classes.

Usage

```
domClass.replace("someId", "addedClassName", "removedClassName");
```

---

toggle()

This function allows you to remove a class if it exist, or add it if it doesn't exist.

Usage

```
domClass.toggle("someId", "className");
```

Read DOM Manipulation online: https://riptutorial.com/dojo/topic/10572/dom-manipulation

Chapter 4: Hello Dojo

Introduction

A simple Hello Dojo like Hello World in other programming languages. Dojo is simple to install/configure and use. You can download the latest version of Dojo from http://dojotoolkit.org/download/ or you can use what is called CDN's in your project. I prefer to download and use it. The latest version at the time of this topic is 1.12.1 and work on Dojo 2.0 is in progress.

Examples

Hello World

Create an HTML file like below. In the script tag, either you can use CDN link or from your local like

```
<script type="text/javascript" src="dojo/dojo/dojo.js"
        data-dojo-config="async:true">

<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <title>Tutorial: Hello Dojo!</title>
</head>
<body>
    <h1 id="greeting">Hello</h1>
    <!-- load Dojo -->
    <script src="//ajax.googleapis.com/ajax/libs/dojo/1.10.4/dojo/dojo.js"
            data-dojo-config="async: true"></script>
</body>
</html>
```

That's it. When you run this HTML, you should see Hello. Yes, we are yet to use any Dojo specific code in this example. But this shows how you can create a simple page using Dojo. Next, we will see how we can apply Dojo to this simple page.

Dojo AMD

In the next example, let's use Dojo features and understand what AMD (Asynchronous Module Definition) means.

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <title>Tutorial: Hello Dojo!</title>
</head>
```

```
<body>
    <h1 id="greeting">Hello</h1>
    <!-- load Dojo -->
    <script src="//ajax.googleapis.com/ajax/libs/dojo/1.10.4/dojo/dojo.js"
            data-dojo-config="async: true"></script>

    <script>
        require([
            'dojo/dom',
            'dojo/dom-construct'
        ], function (dom, domConstruct) {
            var greetingNode = dom.byId('greeting');
            domConstruct.place('<em> Dojo!</em>', greetingNode);
        });
    </script>
</body>
</html>
```

The difference from previous example is that, there is an extra script tag and within that, there are few dojo featues being used. Let's see each one of them

require In layman term, require is similar to import or using statements in other languages where you import some OOTB libraries (in dojo you call them modules). Existing modules are loaded using the keyword 'require' and new modules are created using the keyword 'define'. We will learn more about modules in the later section. For this example, we have used two OOTB modules 'dojo/dom' and 'dojo/dom-construct'. dojo/dom (dom) is the core DOM (document object model) which can be used to get a node from html. For javascript developers, it is similar to document.getElementById('') in-fact, internally dojo uses the same method. dojo/dom-construct(domConstruct) is used to create nodes like div, li, ul etc. It's DOM construction API and it can also be used to insert a node into DOM at any position. Let's say, you have a div 'abc' and want to create another div 'xyz' and place it after 'abc'. You can accomplish that like

```
domConstruct.create("div", { id:"xyz",innerHTML: "<p>new DIV</p>" });
domConstruct.place(dojo.byId("xyz"), dojo.byId("abc"), "after");
```

Coming back to our example, we have

```
require([
            'dojo/dom',
            'dojo/dom-construct'
        ], function (dom, domConstruct) {
            var greetingNode = dom.byId('greeting');
        domConstruct.place('<em> Dojo!</em>', greetingNode);
        });
```

within function, you see dom and domConstruct. This is how we reffer to dojo/dom and dojo/dom-construct. You can use whatever the naming convention you want like

```
require([
                'dojo/dom',
                'dojo/dom-construct'
            ], function (hi, bye) {
                var greetingNode = hi.byId('greeting');
```

```
            bye.place('<em> Dojo!</em>', greetingNode);
        });
```

But it's a good practice to you have meaningful names like for dojo/dom use dom and for dojo/dom-construct, user domConstruct.

Now within the function, we have

```
var greetingNode = hi.byId('greeting');
```

What this does is that it searches the a dom (div in this case) with id='greeting'. variable greetingNode, will have the actual dom node. Then we have,

```
domConstruct.place('<em> Dojo!</em>', greetingNode);
```

So here, we are appending Dojo! to the node greetingNode. This is like Hello+Dojo! and the output will be Hello Dojo!

So with this, we learned

1. How to use dojo features
2. How to use OOTB modules
3. How to manipulate dom

Read Hello Dojo online: https://riptutorial.com/dojo/topic/10941/hello-dojo

Credits

S. No	Chapters	Contributors
1	Getting started with dojo	Ayushi Jha, Chiller, Community, GibboK, Himanshu, Vikash Pandey
2	Configuring Dojo with dojoConfig	Vikash Pandey
3	DOM Manipulation	Chiller
4	Hello Dojo	Manjunatha Muniyappa