Artykuł pochodzi ze wpisu o Dojo Toolkit z bloga autora.

Dojo jest potężnym frameworkiem JavaScript, posiadającym nie tylko wsparcie dla mechanizmów JS, ale także zestaw customizowlanych widgetów, skórki, narzędzia do kompresji , wiele łat na JavaScript sprawiających, że jest on "normalniejszy" (w znaczeniu języków podobnych do Javy) czy specjalny mechanizm ładowania plików z odpowiednimi klasami. Tak w dojo występują klasy , o czym kiedyś jeszcze napiszę, a póki co odsyłam do dokumentacji :).

Szybki start

Aby zacząć z dojo należy po prostu załączyć odpowiedni plik z kodem JavaScript i wykorzystywać API frameworka.

{codecitation class="brush: html"}<html> <head> <title>Quick start dojo!</title> <script src="http://ajax.googleapis.com/ajax/libs/dojo/1.6/dojo/dojo.xd.js"></script> <meta http-equiv="content-type" content="text/html;charset=utf-8" /> <script> dojo.addOnLoad(function() { alert("dojo działa!"); }); </script> </head> <body> Szkielet strony wykorzystującej dojo </body></html>{/codecitation}

demo online

Powyższy kod chyba najprostszym kodem dojo, który będzie działał :). Załączamy plik frameworka, a a następnie przypisujemy funkcje obsługi zdarzenia `load' (załadowanie strony strony). Oczywiście, to nadal nie wszystko, na co pozwala dojo.

Operacje na drzewie DOM

Opiszę dwa główne sposoby dostępu do węzłów dom z wykorzystaniem dojo.

dojo.byId

Metoda `dojo.byId' jest skrótem dla 'document.getElementById'. Jako parametr podaje się wartość atrybuty `id' elementu DOM. W wyniku otrzymujemy obiekt węzła DOM.

{codecitation class="brush: javascript"}dojo.addOnLoad(function() { var domElemDojo = dojo.byId('example'), domElemTrad = document.getElementById('example'); alert(domElemDojo === domElemTrad); // true });{/codecitation}

demo online

dojo.query

Do metody `dojo.query' przekazujemy odpowiedni selektor (zgodny składniowo z tym stosowanym w jQuery). W wyniku otrzymujemy **tablicę** obiektów.

{codecitation class="brush: javascript"}dojo.addOnLoad(function() { // odczyta wszystkie p z dokumentu var pElems = dojo.query('p'), // odczyta wszystkie a zawarte w dowolnym znaczniku p, // który znajduje się w elemencie o id = "przyklad" sofisticated = dojo.query('#przyklad p a'); alert(pElems.length); alert(sofisticated.length); });{/codecitation}

demo online

Skoro już wiemy jak łapać odpowiednie węzły, czas spróbować z nimi coś zrobić. Najpierw zmienimy styl odpowiedniego węzła:

{codecitation class="brush: javascript"}dojo.addOnLoad(function() { // można najpierw złapać odpowiedni węzeł var elem = dojo.byId('example'); // i przypisać do niego style, przekazując obiekt dojo.style(elem, {'border': '1px solid red'}); // albo podać id węzła i następne dwa parametry do metody dojo.style('example2', 'border', '1px solid blue'); // lub id węzła i obiekt ze stylami dojo.style('example3', {'border' : '1px solid green', 'color' : 'blue'}); });{/codecitation}

demo online

W przypadku korzystania z `dojo.query' musimy przeiterować po wszystkich elementach:

{codecitation class="brush: javascript"}dojo.addOnLoad(function() { var elems = dojo.query('p'); dojo.forEach(elems, function(elem) { dojo.style(elem, {'border' : '1px solid red'}); }); });{/codecitation}

demo online

Warto przeczytać:

- http://dojotoolkit.org/reference-guide/dojo/style.html
- http://dojotoolkit.org/reference-guide/dojo/byId.html
- http://dojotoolkit.org/reference-guide/dojo/query.html

Animacje

Każdy szanujący się framework JS posiada przyjazne programiście funkcje animujące elementy strony. Nie inaczej jest z dojo. Poniżej pokażę tylko jeden przykład, po więcej zapraszam do dokumentacji i dojo campus .

{codecitation class="brush: javascript"}var fadeInObj, fadeOutObj; dojo.addOnLoad(function() { // przypisanie obiektów musi nastąpić dopiero po wczytaniu strony // stąd ten kod w `addOnLoad' fadeOutObj = dojo.fadeOut({ node: "example" }); fadeInObj = dojo.fadeIn({ node: "example" }); }); function fadeOut() { fadeOutObj.play(); } function fadeIn() { fadeInObj.play(); }{/codecitation}

demo online

Póki co przykład nie zapiera dechu w piersiach. Z pewnością w tym momencie jquerowe fade'y są przyjaźniejsze. Co jednak powiecie na możliwość zdefiniowania specjalnych funkcji, które będą wywoływane w zależności od aktualnego stanu animacji? Istnieje obsługiwanych "zdarzeń":

- beforeBegin - synchronicznie, przed rozpoczęciem animacji
- onBegin - asynchronicznie, na rozpoczęcie animacji
- onEnd - synchronicznie na zakończenie animacji
- onPlay - synchronicznie, gdy animacja jest odtwarzana
- onAnimate - uruchamiana na każdym kroku animacji

Przykładowe zastosowanie tych zdarzeń:

{codecitation class="brush: javascript"}dojo.addOnLoad(function() { fadeOutObj = dojo.fadeOut({ node: "example", duration : 8000, beforeBegin : function(node) { node.innerHTML = "Znikam!"; }, onBegin : function(value) { this.node.InnerHTML = "No to zaczynam znikać"; }, onEnd : function(node) { node.innerHTML = ""; }, onPlay : function(value) { this.node.innerHTML = "Coraz mniej mnie!"; }, onAnimate : function(value) { this.node.innerHTML += "."; } }); fadeInObj = dojo.fadeIn({ node: "example" }); });{/codecitation}

demo online

Już lepiej , prawda :)

Po więcej przykładów i opcji odsyłam do dokumentacji . Warto zapoznać się także z takimi opcjami jak odpowiednie funkcje zanikania:
http://docs.dojocampus.org/dojo/fx/easing
.

Warto przeczytać:

- http://dojotoolkit.org/reference-guide/dojo/fadeIn.html
- http://dojotoolkit.org/reference-guide/dojo/fadeOut.html
- http://dojotoolkit.org/documentation/tutorials/1.6/effects/

Obsługa zdarzeń - connect i disconnect

Dojo posiada dwie bardzo przydatne metody pozwalające m. in. na przypisywanie funkcji obsługi zdarzeń.

dojo.connect

Do "związania" funkcji ze zdarzeniem służy metoda `dojo.connect':

{codecitation class="brush: javascript"}dojo.addOnLoad(function() {

dojo.connect(dojo.byId('example'), 'onclick', function() { alert('Kliknięto'); }); });{/codecitation}

demo online

Ogólnie nie wygląda trudno, prawda :). Starczy po prostu jako pierwszy atrybut podać obiekt (tu obiekt węzła DOM), jako drugi podać metodę, na jaką ma zostać wywołana funkcja obsługi zdarzenia (przekazywana jako parametr 3). Metoda `dojo.connect' zwraca odpowiedni handler, za pomocą którego można w późniejszym czasie "rozwiązać" połączenie.

dojo.disconnect

Działa przeciwnie do metody `dojo.connect'. Pozwala wyłączyć funkcję obsługi zdarzenia.

{codecitation class="brush: javascript"}dojo.addOnLoad(function() { var handler = dojo.connect(dojo.byId('example'), 'onclick', function() { alert('Kliknięto, następne kliknięcie nic nie da'); dojo.disconnect(handler); }); });{/codecitation}

demo online

Inne zastosowania

Metoda `dojo.connect' ma prócz wyżej pokazanych sporo ciekawych zastosowań. Można np. powiązać wywołanie metody dowolnego obiektu z funkcją. W ten sposób tworzy się jakby nowe "zdarzenie":

{codecitation class="brush: javascript"}var myObject = { a : function() { alert('myObject.a'); } }; dojo.connect(myObject, 'a', function() { alert('Wywołano funckję powiązaną z metodą `myObject.a`'); }); myObject.a();{/codecitation}

demo online

Wydaje Ci się ciekawe? :) To co powiesz na możliwość wiązania w ten sposób metod dwóch obiektów?

{codecitation class="brush: javascript"}var MyObject = function() { this.b = 'MyObject'; this.a = function() { alert('Wywołano metodę `myObject.a`, wartość `b` = ' + this.b); } }; var YourObject = function() { this.b = 'YourObject'; this.x = function() { alert('Wywołano metodę `yourObject.a`, wartość `b` = ' + this.b); } }; var obj1 = new

MyObject(), obj2 = new YourObject(); dojo.connect(obj1, 'a', obj2, 'x'); // #obj1.a();{/codecitation}

demo online

W wyniku otrzymamy dwa komunikaty:

{codecitation class="brush: text"}Wywołano metodę `myObject.a`, wartość `b` = MyObject Wywołano metodę `yourObject.a`, wartość `b` = YourObject{/codecitation}

To nadal nie wszystki możliwości. Stosując powyższy kod, z jedną zmianą [w linii "#"]:

{codecitation class="brush: javascript"}dojo.connect(obj1, 'a', obj1, obj2.x); //#{/codecitation}

demo online

Po tej zmianie otrzymujemy:

{codecitation class="brush: text"}Wywołano metodę `myObject.a`, wartość `b` = MyObject Wywołano metodę `yourObject.a`, wartość `b` = MyObject{/codecitation}

Ma to związek z możliwością sterowania kontekstem w JavaScript. Szerzej o tym możesz przeczytać w "JavaScript na poważnie" , rozdziały 1.4.7 i 1.4.8. Prócz `connect'/`disconnect' w dojo istnieje także mechanizm publisher / subscriber.

Warto przeczytać:

- http://dojotoolkit.org/reference-guide/dojo/connect.html
- http://dojotoolkit.org/reference-guide/dojo/disconnect.html
- http://dojotoolkit.org/reference-guide/dojo/publish.html
- http://dojotoolkit.org/reference-guide/dojo/subscribe.html

Dijit

Dijit jest biblioteką komponentów UI dojo. Aby skorzystać z jakichkolwiek widgetów

wykorzystujemy przestrzeń nazw `dijit' [nie jak wcześniej dojo].

Dijit posiada 2 główne sposoby na umieszczenie elementu na stronie - programistycznie i deklaratywnie (w kodzie HTML z niestandardowymi atrybutami znaczników).

{codecitation class="brush: html"}<html> <head> <title>dijit - button deklaratywnie</title> <meta http-equiv="content-type" content="text/html;charset=utf-8" /> <script src="http://ajax.googleapis.com/ajax/libs/dojo/1.6/dojo/dojo.xd.js" djConfig="parseOnLoad: true"></script> <script type="text/javascript"> dojo.require("dijit.form.Button"); </script> <link rel="stylesheet" type="text/css" href="http://ajax.googleapis.com/ajax/libs/dojo/1.6/dijit/themes/claro/claro.css" /> </head> <body> <button dojoType="dijit.form.Button" type="button">Kliknij mnie!</button> </body></html>{/codecitation}

demo online

Jak już wcześniej powiedziałem, można także osiągnąć ten sam efekt programistycznie:

{codecitation class="brush: html"}<html> <head> <title>dijit - button programistycznie</title> <meta http-equiv="content-type" content="text/html;charset=utf-8" /> <script src="http://ajax.googleapis.com/ajax/libs/dojo/1.6/dojo/dojo.xd.js" djConfig="parseOnLoad: true"></script> <script type="text/javascript"> dojo.require("dijit.form.Button"); dojo.addOnLoad(function() { var button = new dijit.form.Button({ label: "Kliknij mnie!!", onClick: function() { dojo.byId("example").innerHTML += "Kliknąłeś!"; } }, "myButton"); }); </script> <link rel="stylesheet" type="text/css" href="http://ajax.googleapis.com/ajax/libs/dojo/1.6/dijit/themes/claro/claro.css" /> </head> <body> <button id="myButton" type="button"></button> </body></html>{/codecitation}

demo online

W kodzie programistycznym, także dopisałem obsługę zdarzenia `onClick'. Uwaga, zdarzenie `onClick' dijitowego widgetu nie jest tym samym, co zdarzenie `onclick' węzła DOM.

W przypadku stosowania deklaratywnej metody można także przypisywać funkcje obsługi zdarzeń:

{codecitation class="brush: html"}... <button dojoType="dijit.form.Button" type="button">Kliknij mnie! <script type="dojo/method" event="onClick" args="evt"> dojo.byId("example").innerHTML += "Kliknąłeś! "; </script> </button> ...{/codecitation}

demo online

Istnieje jeszcze kilka sposobów na powiązanie zdarzenia z funkcją obsługi. Jednym z nich jest wykorzystanie metody ` dijit.byId '. Zwraca ona obiekt widgetu o zadanym id.

Oczywiście pokazany tu przycisk jest wierzchołkiem góry lodowej olbrzymiej liczby gotowych komponentów w bibliotece dijit. Zachęcam do poszukania w bardzo przyjaznej dokumentacji.

Warto przeczytać:

- http://dojotoolkit.org/reference-guide/dijit/index.html
- http://dojotoolkit.org/api/1.6/dijit.byId
- http://dojotoolkit.org/reference-guide/dijit/form/Button.html

Warto przeczytać

- http://dojotoolkit.org/reference-guide/quickstart/
- http://dojotoolkit.org/api/