Easy Two-Way Data Binding in JavaScript

Dec 2nd, 2012

Two-way data binding refers to the ability to bind changes to an object's properties to changes in the UI, and viceversa. In other words, if we have a `user` object with a `name` property, whenever we assign a new value to `user.name` the UI should show the new name. In the same way, if the UI includes an input field for the user's name, entering a value should cause the `name` property of the `user` object to be changed accordingly.

Many popular client-side JavaScript frameworks like **Ember.js**, **Angular.js** or **KnockoutJS** advertise two-way data binding among their top features. This doesn't mean that it is too hard to implement it from scratch, nor that adopting one of those frameworks is the only option when this kind of functionality is needed. The underlying idea is in fact quite basic, and can be condensed into a 3-point action plan:

1. We need a way to specify which UI elements are bound to which properties
2. We need to monitor changes on the properties and on the UI elements
3. We need to propagate any change to all bound objects and elements

While there are multiple ways to achieve this, a simple and efficient approach makes use of the *PubSub* pattern. The idea is simple: we can use custom `data` attributes to specify bindings in the HTML code. All JavaScript objects and DOM elements that are bound together will "subscribe" to a *PubSub* object. Anytime a change is detected on either the JavaScript object or on an HTML input element, we proxy the event to the *PubSub*, which in turn broadcasts and propagates the change on all the other bound objects and elements.

A simple implementation using jQuery

It is quite straightforward to implement what discussed using **jQuery**, as the popular library lets us easily subscribe and publish DOM events, as well as custom ones:

```
function DataBinder( object_id ) {
  // Use a jQuery object as simple PubSub
  var pubSub = jQuery({});

  // We expect a `data` element specifying the binding
  // in the form: data-bind-<object_id>="<property_name>"
  var data_attr = "bind-" + object_id,
      message = object_id + ":change";

  // Listen to change events on elements with the data-binding attribute and proxy
  // them to the PubSub, so that the change is "broadcasted" to all connected objects
  jQuery( document ).on( "change", "[data-" + data_attr + "]", function( evt ) {
    var $input = jQuery( this );

    pubSub.trigger( message, [ $input.data( data_attr ), $input.val() ] );
  });

  // PubSub propagates changes to all bound elements, setting value of
  // input tags or HTML content of other tags
  pubSub.on( message, function( evt, prop_name, new_val ) {
    jQuery( "[data-" + data_attr + "=" + prop_name + "]" ).each( function() {
      var $bound = jQuery( this );

      if ( $bound.is("input, textarea, select") ) {
        $bound.val( new_val );
      } else {
        $bound.html( new_val );
      }
    });
  });

  return pubSub;
}
```

For what concerns the JavaScript object, a minimal implementation of a `User` model for the sake of this experiment could be the following:

```
function User( uid ) {
  var binder = new DataBinder( uid ),

      user = {
        attributes: {},

        // The attribute setter publish changes using the DataBinder PubSub
        set: function( attr_name, val ) {
          this.attributes[ attr_name ] = val;
          binder.trigger( uid + ":change", [ attr_name, val, this ] );
        },

        get: function( attr_name ) {
          return this.attributes[ attr_name ];
        },

        _binder: binder
      };

  // Subscribe to the PubSub
  binder.on( uid + ":change", function( evt, attr_name, new_val, initiator ) {
    if ( initiator !== user ) {
      user.set( attr_name, new_val );
    }
  });

  return user;
}
```

Now, whenever we want to bind a model's property to a piece of UI we just have to set an appropriate `data` attribute on the corresponding HTML element:

```
// javascript
var user = new User( 123 );
user.set( "name", "Wolfgang" );

// html
<input type="number" data-bind-123="name" />
```

The value of the input field will automatically reflect the `name` property of the `user` object, and viceversa. Mission accomplished!

Doing without jQuery