Web Design: Fall 2010

Basic to Advanced Techniques

http://fa10.decal.aw-industries.com

Lecture 8: Lab

Description

You've just learned Ajax, the coolest technology on the web today. Feeling inspired, you decide to put your skills to the test by creating tWDDer so you and your classmates can tWDDt about all the fun you're having in the Web Design Decal.
You will be working with jQuery and Ajax to bring your idea to fulfillment.

Fortunately for you, much of the framework has been completed. I've added a new jQuery plugin (with slight html 5 modifications) from http://timeago.yarp.com/. A jQuery plugin is simply just another JavaScript file that utilizes the jQuery framework to add additional functionality to your website without having to do all the work yourself. They're usually made by jQuery enthusiasts and are often times offered for free (just make sure though, you could get into a lawsuit if it isn't). This plugin will calculate your feed times and continually update as time passes. Check out the site to see what I mean.

To use the jQuery plugin, make sure to add the HTML attribute 'data-time', when receiving a request.

Goals of this lab

- Learn Ajax to both fetch and send data.
- Learn jQuery effects
- Learn HTML forms

Part 1: Setup

1. The files you need are already included. You must link each of the following:
   - JavaScript
     - ✓ jquery.timeago.js
     - ✓ jquery-1.4.3.min.js
     - ✓ myCode.js //This is where your work will be.
   - CSS
     - ✓ do-not-modify.css //You won't have to touch any CSS files today.

   Please refer to previous lectures, labs, and MPs if you don't recall.

Part 2: Ajax submitting

1. Add 2 inputs, text input and textarea to the input form.
   a. The name input should have id 'formName'. The textarea should have id 'formMessage'

2. In p.submit, Add 'submit' and 'cancel' <button> or <a> to give some control over the form effects. (Buttons look nicer by default)
3. In the following, when I mention hide/show, I would like you to use a jQuery effects of your choosing to exhibits these states.
4. Now move onto myCode.js
    a. Start off by initializing some variables and constants
        i. KEY to tWDDer>twitter
        ii. lastRefresh to new Date(2009, 0, 1, 0, 0, 0, 0)
            1. If you look at the specifications of the Date class, you will notice that we are creating a Date in the 2009. This is so that on the first refresh, we make sure to gather all posts ever created. Later on we will update this lastRefresh variable to keep track of the last time we refreshed. This way, when we refresh again we can tell the server that we only need to add new posts created after our last refresh. This saves huge amounts of traffic, since we only get what is needed and nothing more.
5. Fill in the hideForm().
    a. Hide the Form
    b. Show the 'Add a new tWDDt' button. This MUST follow immediately after hiding the form. Not during! (Hint: Think about callbacks).
6. **In $(document).ready**: Please note that you will be coding each of these tasks. Specs such as "dataType: jsonp" do not mean it should appear like that in the code. Refer to the syntax and jQuery API to understand how it should be coded.
    a. By default, the form is hidden.
        i. Hide 'Add a new tWDDt' button.
        ii. Then show form. Again this MUST follow immediately after hiding the button
    b. If you decide to not post any status, you would like to cancel the form.
        i. Hide the form when the cancel button is clicked
    c. Later on, you will add some code to refresh every 30 seconds, but sometimes you want to immediately refresh.
        i. Refresh the feed when the refresh link is clicked.
    d. Now that you have the form done, add some Ajax functionality to it.
        i. Refer to the lecture slide on how to do jQuery Ajax.
        ii. URL: http://twdder.decal.aw-industries.com/messages/create
        iii. Parameters: keys—description of value
            1. name— retrieve the name input value (Hint: .val())
            2. message—retrieve the message input value

3. key—KEY

iv. dataType: jsonp

v. Successs CallBack

1. hide the form
2. refresh the feed

e. You're almost done preparing the document after it loads.

i. When we first land on the page, we want to grab the data, so call refreshFeed

ii. Now we want to have refreshFeed called very 30 seconds. It's easy, so I'll let practice figuring out how to use APIs. http://www.w3schools.com/jsref/met_win_setinterval.asp

Part 3: Ajax retrieving

1. Put the Ajax call under refreshFeed. In the success callback, you must hide the form and refresh the feed. In the callback, you will be using two functions we've written for you:
   - dateToDateTimeString(date) — Takes in a Date and converts it to the proper string format to pass to our servers.
   - dateTimeStringtoDate(string) — Takes in a string and returns a Date string with the time set to the string specified.

   a. URL: http://twdder.decal.aw-industries.com/messages/retrieve

   b. Parameters: keys—description of value

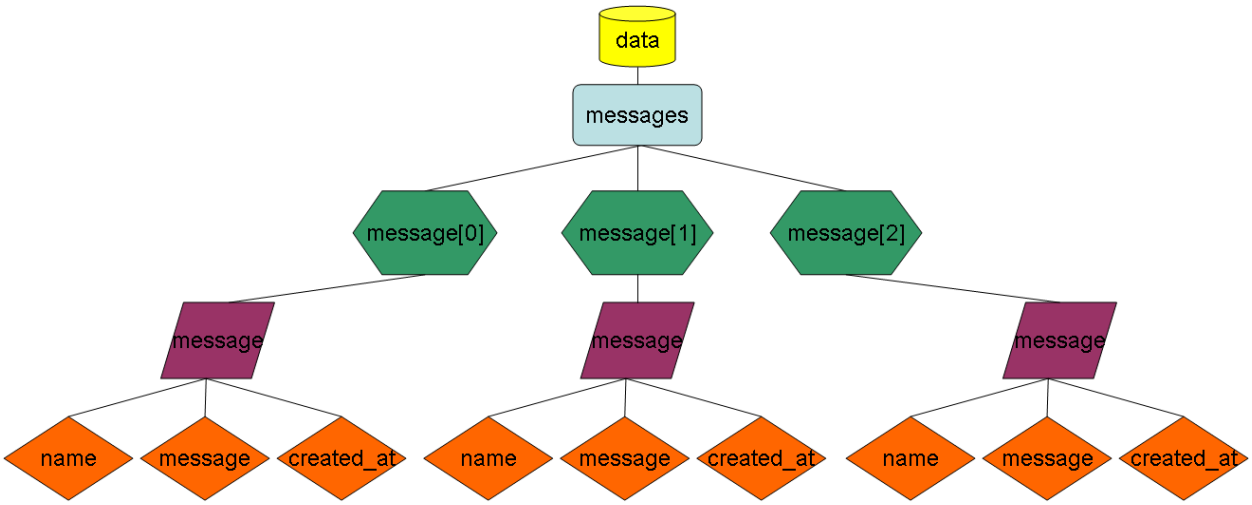
      i. key—KEY

      ii. created_after— the string of value of lastRefresh

   c. dataType: jsonp

   d. Successs CallBack

      i. Your function should have the signature "function(data)", since we only care about the data for this task. The data will be structured like so:



ii. We're going to give you most of the code here, because much of it is actual programming, which can be overwhelming for a lab if you don't have programming experience in a language like Javascript.
iii. Now that you have the data returned from your request to the serer, we need to parse it to get just the stuff we need. As you can see above, each 'data' has 'messages'. We don't know before hand how many messages[i] there will be. So we should loop through it. Here is the for loop syntax:

```
for(var i = 0; i < data.messages.length; i++){
    //code to be executed
}
```

Anything in the braces will be executed at each iteration of the loop. So "var i = 0" specifies the counter variable to begin at 0. Then we loop until "i < data.messages.length" (basically loop through the number of messages). The final specification is "i++" which tells that at the end of each iteration, we increase 'i' by one. All this is to iterate through each message.
iv. In each message, there are 3 items that we care about to add new statuses. They are…
- name, accessed by: data.messages[i].message.name
- message, accessed by: data.messages[i].message.message
- created_at , accessed by: data.messages[i].message.created_at

v. Call addStatus in the for loop and pass those three arguments.
vi. Now select all html abbr (abbreviation) elements that have class "timeago" and call .timeago() on them.
vii. The last step of the call back is to update lastRefresh.
    1. Use an 'if' statement to check if you have greater than 0 messages. Check this out for the syntax: http://www.w3schools.com/JS/js_if_else.asp (Hint: to get the number of messages you'll want to use a snippet from the for loop we gave you above.)
    2. Within the 'if', assign lastRefresh to be the output of dateTimeStringtoDate. dateTimeStringtoDate takes an argument string. You'll want this argument to be the creation time of the last message. (Hint: remember how you access created_at? Now access the last message's created_at item. The last message's number will be: total_messages – 1.)
    3.

2. You're almost done! The next step is to fill in addStatus. This might be tricky if you haven't grasped jQuery's method chaining. Don't forget you're passed in the user, entry, and time. The rest of the work is just constructing the html for the status.
   a. Create a variable "newStatus" to store the html of the status you will add.
   b. Each new status will have html that looks like this

```
<li class="status">
    <span class="pfimage"><img src="images/speechBubble.png" /></span>
    <span class="statusBody">
        <span class="user">Jonathan:</span>
        <span class="statusEntry">Hey this class is sooo awesome!</span>
        <span class="time"><abbr class="timeago" data-time="2008-07-17T09:24:17Z"></abbr></span>
    </span>
</li>
```

   Except that you'll want to dynamically add the user, status entry, and date-time
   c. It may be simpler to first code up the HTML as you see it, then figure out how to break it down to jQuery it.
   d. To concatenate text with variables just add '+'. For example,

```
'<span class="mySpan">' + mySpanText + '</span>'
```

   Then if mySpanText was "hi" you would get this HTML

```
<span class="mySpan">hi</span>
```

   e. After you generate the html, give it some css to hide it, so that when we attach it to the feed later, we can add some show effects.
   f. After you complete the html for the newStatus, add it to the top of the feed. Use one of the jQuery manipulation functions. And in the callback, make sure to add some show effect.
3. Congratulations. That's it. You should have a working tWDDer Feed. If you had only thought about making this 4 years ago, you would have been a millionaire!
4. If you are feeling enthusiastic, think about adding one of those loading icons to show during the ajax calls, so we know our application is refreshing.

Hint: *You'll probably need these methods*

.addClass()
.animate()
.append()
.css()
.before()
.click()
.insertBefore()
. prepend()
.val()