Observer Pattern in web applications (JavaScript): A walkthrough with an AJAX example

By **Albin Abel**, 2 Dec 2010

Download source code - 2.88 KB

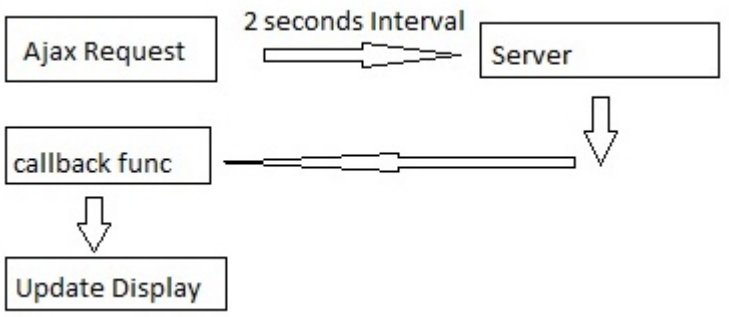
Introduction

Design Patterns are helpful to create scalable and flexible applications. They help decouple objects and ensure we follow OOP design principles like an object should be closed for modification and open for extension.

The Observer Pattern defines a one to many dependency between objects so that when an object changes state, all of its dependents are notified and updated automatically. The dependency is established at runtime; at design time, the objects don't know each other and are decoupled. We call the source object 'observable' and the dependants 'observers'.

The Pattern Explained

Let us walkthrough the Observer pattern with a simple AJAX application. This demo application is a simple PHP code, but the design concept is not restricted to a language. The scope of this tiny application is updating the temperature display in a webpage whenever a temperature value change occurs on the server side. The work flow is simple; the webpage makes an AJAX request at two second intervals and gets the temperature value updated by a callback function.



The server responds with a temperature value in degree Celsius and the webpage displays it. Let us say that the application works fine and the client is happy. The next day, the client wants to add one more option for customers to display the temperature in Fahrenheit as well. This is optional, and the customer might decide to turn it on or off. What do we do?. Should we modify the callback function, or write a separate function to convert Celsius to Fahrenheit and for the display logic and call these functions from the callback function? Either way, the parent function needs to know which functions are to be called at design time.

```
//pseudocode

function ajaxcallback(temperature){
      myCelsiusDisplay(temperature);
      myFarentheitDisplay(temperature);
}

function myCelsiusDisplay(temperature){
     displayLogic
}

function myFarenheitDisplay(temperature){
    call conversion function to convert celsius to farenheit
    display logic
}
```

In the above code, the AJAX callback is highly coupled with the other display functions. So every time you want to add a display logic, you need to edit the AJAX call back function, which will not be reusable as it needs to know about other functions at design time. It is subject to change whenever a new function depends on it. It can't be moved to an another page where the display logic might be different. Also, now the callback may need to know about a hidden field, say if a check box is checked or not. This situation could get more complicated in bigger applications.

Our clients often change their minds. You may wonder when the next day he asks to display the temperature in Kelvin. We need to find a better solution. How about the AJAX callback only knows the function to be called at runtime, and not at design time. We can then ask the AJAX callback to call any function at runtime. This way, it is decoupled and reusable in any page. That sounds like a good idea. We call the AJAX callback as 'Observable', and the functions to be called are 'Observers'. Observerble will notify the Observers when some change has occurred in the Observable.

Using the Code

The Display

When we click the Celsius check box, it displays the temperature in Celsius, and when we click the Fahrenheit checkbox, it displays the temperature in Fahrenheit. You can make the display more fancy, like a thermometer, and display the number in the form of a graphic, say mercury level.

The Script

```
//THIS PART DOESN'T WHAT FUNCTIONS TO BE CALLED, WHAT KIND OF DATA IT IS HANDLING
//IT OBSERVERS WILL DO THAT JOB. IT ONLY ACT AS A MODEL-----------------------------

//Observable object which is observe by observers

observable=new Object();

//room to keep the observers
observable.observers=new Array();

//communicate with observers
observable.notify=function(message){

     for(i=0;i<this.observers.length;i++){

          //all observers must have the doWork function
          this.observers[i].doWork(message);
     }
}

//register an observer
observable.addObservers=function(observer){
      this.observers[this.observers.length]=observer;
}

//remove an observer
observable.removeObservers=function(observerName){

      for(i=0;i<=this.observers.length;i++){

         if(this.observers[i].Name=observerName){
              this.observers.splice(i,1);
         }
     }
}

//-------------------------------------------------------------
```

Above is the Observable JavaScript object which has a 'notify' function. The AJAX request callback passes the temperature value to all of its observers. It has the 'observers' array which is where the observers will be registered or removed at run time.

```
//An Observer object.THE OBSERVABLE DOESN'T
  KNOW ABOUT IT UNTILL IT REGISTERS ITSELF --------------------

function addCelsiusDisplay(checkbox){

     if(checkbox.checked==true){
            observer1=new Object();

             //an unique identity for the observer
            observer1.Name="celsiusdisplay"

           observer1.doWork= function(information){
              document.getElementById("celsius_display").value=information;
          }
          //register the observer

           observable.addObservers(observer1);
```

```
     }else{

          //remove the observer
         observable.removeObservers("celsiusdisplay");

     }
}

//-------------------------------------------------------------
```

This function is called on the checkbox click event at runtime. If the checkbox is checked, then it adds an observer to its observable. If unchecked, it removes the observer from the observable's observer list. All these coupling happens only at runtime; the objects don't know about each other at design time.

In the same way, we can add any number of observers at runtime and display the temperature value in different ways. We will not have to change the observable logic any more. Any change in the observable is critical because this relationship follows one to many; changing the observable may lead to changes in all observers. We are safe now; we are not going to change the observable. The below code is going to update a Fahrenheit display without changing anything at the observable.

```
//Another Observer Object--------------------------------

function addFarenheitDisplay(checkbox){

     if(checkbox.checked==true){
           observer1=new Object();

           observer1.Name="farenheitdisplay"

          observer1.doWork= function(information){

                var farenheit=information*(9/5)+32;
               document.getElementById("farenheit_display").value=farenheit;
          }

          observable.addObservers(observer1);

    }else{
          observable.removeObservers("farenheitdisplay");
    }

}

//-------------------------------------------------------
```

Here the observable doesn't know what kind of data it handles. It just passes over the server response. So this model is reusable with any web application. This is good news.

The below code is the AJAX request which requests the server side asynchronously; the page will not refresh, and our observable object preserves. It calls our Observable's function when a response arrives.

```
function getData(){

    //creates XML Http object
   var ajaxRequest=createXMLHttp();

    //make open request with unique url to avoid cache related issues
    ajaxRequest.open("GET",
```

```
      "temperature.php?unique_request="+Math.random(), true);

    ajaxRequest.onreadystatechange=function()
    {
        if(ajaxRequest.readyState==4 &&
        ajaxRequest.status==200){
           //call the observable function which inturn notify its observers
           observable.notify(ajaxRequest.responseText);
        }
    }
    ajaxRequest.send(null);
}

//The below code executes the ajax request
//every two seconds in javascript's multithreading mode.
//communicate server in every 2 secs
t=setInterval(getData,2000);
```

The Server Side

```
<?php

if (file_exists('data.xml')) {
      $xml = simplexml_load_file('data.xml');
} else {
      exit('Failed to open data.');
}

$data=$xml->temperature;
$xml=null;
echo $data;

?>
```

The server side does simple work. It reads an XML file and echoes the temperature.

Another server side code is provided to update this temperature in XML:

```
if(isset($_REQUEST["action_id"]) && $_REQUEST["action_id"]!=""){
    $data=$_REQUEST["temperature"];
    $sxe = new SimpleXMLElement('data.xml', NULL, TRUE);
    $sxe->temperature=$data;
    file_put_contents('data.xml', $sxe->asXML());
    echo "Temperature updated sucessfully";
}

if (file_exists('data.xml')) {
    $xml = simplexml_load_file('data.xml');
} else {
    xit('Failed to open data.');
}

$data=$xml->temperature;

?>
```

Testing

Now it is time for testing what we have done. Unzip the attached source code. Copy it to your web server's root

folder. Run *observer.php* in the browser using the correct URL from your server. Check those checkboxes; the temperature value will be displayed in the text boxes. Now run *dataeditor.php* in another browser tab. Update the temperature value. Look at the display page (*observer.php*), it now gets updated.

That's it. Now you may explore the code and use this model in your applications with your creativity. Thanks for reading this article.

Points of Interest

There are hundreds of books and thousands of articles on Design Patterns. A few focus on web development. Though many resources are available, Design Patterns are learned only in theory level in many software development projects. I thought CodeProject is the right place to write a series of articles on Design Patterns to reach programmers. Thanks again for reading this.

License

About the Author

Albin Abel
Software Developer

India
Member

I am developer in .Net and GIS. albin_gis@yahoo.com

Comments and Discussions

3 messages have been posted for this article Visit **http://www.codeproject.com/Articles/132942/Observer-Pattern-in-web-applications-JavaScript-A** to post and view comments on this article, or click **here** to get a print view with messages.