Dynamically adding markers to Google maps

Posted At : 15 January 2011 19:24 | Posted By : Shaun McCran
Related Categories: JQuery, Google, Development, Coldfusion, AJAX

Following on from a previous article I wrote about (Google maps panning the next step in my Google mapping project is to be able to add markers to a Google map dynamically.

This article deals with how to translate a location into a latitude and longitude using Google, and how to send and add markers from a database into a Google maps via a remote service, using AJAX and JSON.

There is a full example of the finished application here: Demo of dynamically adding markers to Google maps

Much like the previous article about Google map panning we have to load the JQuery and the Google maps API, using an API key related to our domain.

```
<s/cript type="text/javascript"
src="http://www.google.com/jsapi?key= your API key here">
</script>

<s/cript type="text/javascript">
google.load("jquery", '1.3');
google.load("maps", "2.x");
</script>
```

Next we will pick a point on the map, and set it as the centre, with a predefined zoom level. Below that we are creating references to the Google API 'bounds' object, and the 'Geocoder' object. I am also creating an Array of response codes for remote service responses.

Next the getJSON() JQuery method fires on page load. This communicates with our remote service, in this case a CFC (ColdFusion) and looks for a function called 'listpoints'. I'll drop the code for that in later, but it returns a JSON object of data that contains the name of a location and its lat-long values.

```
<s/cript type="text/javascript">
$(document).ready(function(){

// set the element 'map' as container
var map = new GMap2($("#map").get(0));

// set the lat long for the center point
var cheltenham = new GLatLng(51.89487062921521,-2.084484100341797);
// value 1 is the center, value 2 is the zoom level
map.setCenter(cheltenham, 9);

var bounds = new GLatLngBounds();
var geo = new GClientGeocoder();

// status codes
var reasons=[];
reasons[G_GEO_SUCCESS]            = "Success";
reasons[G_GEO_MISSING_ADDRESS]    = "Missing Address";
reasons[G_GEO_UNKNOWN_ADDRESS]    = "Unknown Address.";
reasons[G_GEO_UNAVAILABLE_ADDRESS]= "Unavailable Address";
reasons[G_GEO_BAD_KEY]            = "Bad API Key";
reasons[G_GEO_TOO_MANY_QUERIES]   = "Too Many Queries";
reasons[G_GEO_SERVER_ERROR]       = "Server error";

// initial load points
$.getJSON("map-service.cfc?method=listpoints", function(json) {
if (json.Locations.length > 0) {
for (i=0; i<json.Locations.length; i++) {
var location = json.Locations[i];
addLocation(location);
}
zoomToBounds();
}
});
```

The page display itself is very small, there are only three elements, the 'map' div, where our map function will be inserted, a 'list', which will populated with a list of our locations and a 'form' to allow a user to submit new locations.

```
<div id="wrapper">

<div id="map"></div>
<ul id="list"></ul>
<div id="message" style="display:none;"></div>

<form id="add-point" action="map-service.cfc?method=accept" method="post">
<input type="hidden" name="action" value="savepoint" id="action">

<fieldset>
<legend>Add a Point to the Map</legend>
<div class="error" style="display:none;"></div>
<div class="input">
<label for="name">Location Name</label>
<input type="text" name="name" id="name" value="">
</div>

<div class="input">
<label for="address">Address</label>
<input type="text" name="address" id="address" value="">
</div>

<button type="submit">Add Point</button>
</fieldset>
</form>

</div>
```

All the data in my example is stored in a mySQL database. The script to create it is below. There are only a few fields, and they are reasonably easy to recognise.

```
CREATE TABLE `locations` (
`intId` int(11) NOT NULL AUTO_INCREMENT,
`varName` varchar(100) DEFAULT NULL,
`varLat` varchar(25) DEFAULT NULL,
`varLong` varchar(25) DEFAULT NULL,
`varAddress` varchar(55) DEFAULT NULL,
PRIMARY KEY (`intId`)
) ENGINE=InnoDB AUTO_INCREMENT=12 DEFAULT CHARSET=latin1;
```

At this point if you were build this app you would have a page displaying a map and a series of map points from a database (if you had some in the table).

The next few code snippets are the parts that accept the form values, and translate them into useful map data to store then display them.

A user will enter the place label and location, this is then geo coded and sent to the remote service to be saved in our database. Here we can perform any validation and actually store the data. The same service returns the newly saved data to our map and we dynamically add the new marker to the map.

The first new function is a JQuery selector triggered when a user submits the form. It simply fires the geoEncode() function below it. The geoCode function tries to lookup the location the user entered from the form. If there is an error the corresponding message is displayed, otherwise the savePoint() function fires, using the location values.

```
$("#add-point").submit(function(){
geoEncode();
return false;
});

function geoEncode() {
var address = $("#add-point input[name=address]").val();
geo.getLocations(address, function (result){
if (result.Status.code == G_GEO_SUCCESS) {
geocode = result.Placemark[0].Point.coordinates;
savePoint(geocode);
} else {
var reason="Code "+result.Status.code;
if (reasons[result.Status.code]) {
reason = reasons[result.Status.code]
}
$("#add-point .error").html(reason).fadeIn();
geocode = false;
}
});
}
```

The savePoint() function serialises the inputs from the form and performs the Geo encoding. Now we try and post the form. If it fails we show a message, otherwise we run the addLocation() and zoomToBounds() functions. (almost there, stick with it!).

```
function savePoint(geocode) {
var data = $("#add-point :input").serializeArray();
data[data.length] = { name: "lng", value: geocode[0] };
data[data.length] = { name: "lat", value: geocode[1] };

$.post($("#add-point").attr('action'), data, function(json){
$("#add-point .error").fadeOut();

if (json.status == "fail") {
$("#add-point .error").html(json.message).fadeIn();
}
if (json.status == "success") {
$("#add-point :input[name!=action]").val("");
var location = json.data;
addLocation(location);
zoomToBounds();
}
}, "json");
}
```

The addLocation() function below creates a map marker object and adds it to the map. It also adds the new location to the 'list' we created in the display layer above.

The zoomToBounds() function zooms to the new marker, and sets the center of the map on it.

The last function is a click event that pans the map to the marker and displays the marker label when a user clicks on one of the list item locations, or the marker itself.

```
function addLocation(location) {
var point = new GLatLng(location.lat, location.lng);
var marker = new GMarker(point);
map.addOverlay(marker);
```

```
                                bounds.extend(marker.getPoint());

                                        $("<li />")
                                    .html(location.name)
                                     .click(function(){
                                showMessage(marker, location.name);
                                                        })

                                     .appendTo("#list");

                            GEvent.addListener(marker, "click", function(){
                                   showMessage(this, location.name);
                                                   });
                                               }

                           function zoomToBounds() {
                          map.setCenter(bounds.getCenter());
                  map.setZoom(7); // map.getBoundsZoomLevel(bounds)-1
                                                       }

       $("#message").appendTo( map.getPane(G_MAP_FLOAT_SHADOW_PANE) );


                     function showMessage(marker, text){
           var markerOffset = map.fromLatLngToDivPixel(marker.getPoint());

                               map.panTo(marker.getLatLng());

                            $("#message").hide().fadeIn()
                      .css({ top:markerOffset.y, left:markerOffset.x })
                                     .html(text);
                                               }
                                        });
                                   </script>
```

The server side code used in this example is below. I have used ColdFusion to query a database and return JSON objects to the map, but you can use any server side language.

```
ame="accept" access="remote" output="true" returntype="void" hint="handler for goo
                           <cfquery datasource="database-name">
                                       INSERT INTO locations
                              (varName, varlat, varlong, varAddress)
                                                VALUES(
      <cfqueryparam cfsqltype="cf_sql_varchar" value="#arguments.name#">,
       <cfqueryparam cfsqltype="cf_sql_varchar" value="#arguments.lat#">,
       <cfqueryparam cfsqltype="cf_sql_varchar" value="#arguments.lng#">,
    <cfqueryparam cfsqltype="cf_sql_varchar" value="#arguments.address#">)
                                          </cfquery>

 '{"status":"success","data":{"lat":"#arguments.lat#","lng":"#arguments.lng#","nam
```

```
                                    <cfoutput>#response#</cfoutput>
                                         </cffunction>

stpoints" access="remote" output="true" returntype="void" hint="returns data to a

                <cfquery datasource="database-name" name="qGetMarkers">
                       SELECT intId, varName, varlat, varlong, varAddress
                                           FROM locations
                                           ORDER BY intId
                                         </cfquery>

ers.varName#","lat":"#qGetMarkers.varlat#","lng":"#qGetMarkers.varlong#"}<cfif qG
                                         </cffunction>
```

ColdFusion purists will note that I'm not returning the data in the most efficient JSON way, but I'm running this code on ColdFusion server 7, so I can't specify a returnformat of JSON.

The next (and last) article in this series will bring AJAX polling, map panning and dynamically adding markers together to create the finished dynamic Google map application.

There is a full example of the finished application here: [Demo of dynamically adding markers to Google maps]