Google Highlighting

This article was published by the Applied Thinking Group

You can read it online, here:
http://www.appliedthinking.org/articles/google_highlighting/

Abstract:

When you view a cached google page, google highlights the terms you searched for. I find this really useful, but it only happens when people look at the cached version, so how can we enable search term highlighting all the time? Easy - we use Javascript and the DOM.

The first thing we need to determine is whether the page referer was google. If it is, then we'll need to find out what the search was for. To do this we look at the *document.referrer* property, using a simple regular expression. If it matches, we the use *String.split()* to break apart the url and find the parameter we want (called 'q' in google's case). If we find a value for q, then we can enable keyword highlighting for the page by calling out highlighter function.

```
function init_google(){
 var pattern = /google\./i;
 if (pattern.exec(document.referrer) != null){
  var url_parts = document.referrer.split('?');
  if (url_parts[1]){
   var url_args = url_parts[1].split('&');
   for(var i=0; i<url_args.length; i++){
    var keyval = url_args[i].split('=');
    if (keyval[0] == 'q'){
     go_google(decode_url(keyval[1]));
     return;
    }
   }
  }
 }
}

function decode_url(url){
 return unescape(url.replace(/\+/g,' '));
}
```

We call this function in the <body> tag's *onload* event. If the page is referered from a google search, the highlighter function, *go_google()*, gets called with the search term(s).

The *decode_url()* function simply helps us to convert the url into human readable format, since search terms often contain spaces and other characters than are encoded when used in urls.

The next step involves splitting out the individual terms and doing the actual highlighting. In true google style, we can use a different color for each term, using a simple array.

```
var google_link_colors = new
Array('#ffff66','#a0ffff','#99ff99','#ff9999','#ff66ff');

function go_google(terms){
 terms = terms.replace(/\"/g,"");
 var terms_split = terms.split(' ');
 var c = 0;
 for(var i=0; i<terms_split.length; i++){
  highlight_goolge(terms_split[i], document.body,google_link_colors[c]);
  c = (c == google_link_colors.length-1)?0:c+1;
 }
}
```

Note than before splitting the terms up we remove any quotes. We could be more intelligent here and treat phrases as phrases and not several distinct words, but i'll leave that as an exercise for the reader.

A local variable, *c*, is used to move between colors. If there are more terms that we have colors defined, the color choice wraps back round to the beginnning.

To do the actual highlighting, we'll use some DOM trickery. By traversing the document tree, we can search all the pieces of text on the page. We do this using a recursive function since the document is arranged in a "tree" which can be many levels deep. By testing the *nodeType* attribute of a node we can see if it contains text (3) or if it contains other nodes (1).

```
function highlight_goolge(term, container, color){

 for(var i=0; i<container.childNodes.length; i++){
  var node = container.childNodes[i];
```

```
  if (node.nodeType == 3){
   var data = node.data;
   // do something here
  }else{
   //recurse
   highlight_goolge(term, node, color);
  }
 }
}
```

To do the highlighting, we will require further DOM techniques. After finding a term with a piece of data, we create some new nodes. The portion of the data before the match is added to a new text element (A), as is the text following the match (C). The matching text is then created as a text node (D), inside a SPAN node (B) with it's colors set. The three nodes A, B and C are then put into one container node (E). This container node is swapped with the current text node we're searching and the term thus becomes highlighted. (Maybe a diagram might be useful)

Now, at this point we could use a regular expression again to match the terms. Unfortunately regular expressions for replacing text have a habit of crashing IE on a mac, so we have to be a little less elegant. By converting both the data we are searching and the search term, we can find the position of the term within the (normal cased) data. With this position infom we can use the *String.substr()* function to chop up the data.

```
var term_low = term.toLowerCase();
var data_low = data.toLowerCase();

if (data_low.indexOf(term_low) != -1){
 //term found!
 var new_node = document.createElement('SPAN');
 node.parentNode.replaceChild(new_node,node);

 var result;
 while((result = data_low.indexOf(term_low)) != -1){
  new_node.appendChild(document.createTextNode(data.substr(0,result)));
  new_node.appendChild(create_node_google(document.createTextNode(
   data.substr(result,term.length)),color));
  data = data.substr(result + term.length);
  data_low = data_low.substr(result + term.length);
 }
 new_node.appendChild(document.createTextNode(data));
}
```

A simple while loop manages multiple matches within a single node, not adding the final trailing text node until the scan has been completed.

The function as a whole now looks like this:

```
function highlight_goolge(term, container, color){
 var term_low = term.toLowerCase();

 for(var i=0; i<container.childNodes.length; i++){
  var node = container.childNodes[i];

  if (node.nodeType == 3){
   var data = node.data;
   var data_low = data.toLowerCase();
   if (data_low.indexOf(term_low) != -1){
    //term found!
    var new_node = document.createElement('SPAN');

    node.parentNode.replaceChild(new_node,node);

    var result;
    while((result = data_low.indexOf(term_low)) != -1){
     new_node.appendChild(document.createTextNode(
      data.substr(0,result)));
     new_node.appendChild(create_node_google(
      document.createTextNode(data.substr(
      result,term.length)),color));
```

```
     data = data.substr(result + term.length);
     data_low = data_low.substr(result + term.length);
    }
    new_node.appendChild(document.createTextNode(data));
   }
  }else{
   //recurse
   highlight_goolge(term, node, color);
  }
 }
}
```

The final piece of the puzzle is then the function that creates the actual highlighted text node. As mentioned previsously, we create a SPAN element then place a text node inside it, creating highlighted text. Since we're setting the background color, it's also prudent to set the foreground color - if someone's text is yellow (maybe they are mad?) then highlighting it with a yellow background is not going to help. As with the background colors, we use a global variable for the text color. That was, changing settings is very easy.

```
var google_text_color = '#000000';

function create_node_google(child, color){
 var node = document.createElement('SPAN');
 node.style.backgroundColor = color;
 node.style.color = google_text_color;
 node.appendChild(child);
 return node;
}
```

Ok, we're done. The whole of the code looks like this:

```
var google_text_color = '#000000';
var google_link_colors = new
Array('#ffff66','#a0ffff','#99ff99','#ff9999','#ff66ff');

function init_google(){
 var pattern = /google\./i;
 if (pattern.exec(document.referrer) != null){
  var url_parts = document.referrer.split('?');
  if (url_parts[1]){
   var url_args = url_parts[1].split('&');
   for(var i=0; i<url_args.length; i++){
    var keyval = url_args[i].split('=');
    if (keyval[0] == 'q'){
     go_google(decode_url(keyval[1]));
     return;
    }
   }
  }
 }
}

function decode_url(url){
 return unescape(url.replace(/\+/g,' '));
}

function go_google(terms){
 terms = terms.replace(/\"/g,"");
 var terms_split = terms.split(' ');
 var c = 0;
 for(var i=0; i<terms_split.length; i++){
  highlight_goolge(terms_split[i], document.body,google_link_colors[c]);
  c = (c == google_link_colors.length-1)?0:c+1;
 }
}

function highlight_goolge(term, container, color){
 var term_low = term.toLowerCase();
```

```
 for(var i=0; i<container.childNodes.length; i++){
  var node = container.childNodes[i];

  if (node.nodeType == 3){
   var data = node.data;
   var data_low = data.toLowerCase();
   if (data_low.indexOf(term_low) != -1){
    //term found!
    var new_node = document.createElement('SPAN');

    node.parentNode.replaceChild(new_node,node);

    var result;
    while((result = data_low.indexOf(term_low)) != -1){
     new_node.appendChild(document.createTextNode(
      data.substr(0,result)));
     new_node.appendChild(create_node_google(
      document.createTextNode(data.substr(
      result,term.length)),color));
     data = data.substr(result + term.length);
     data_low = data_low.substr(result + term.length);
    }
    new_node.appendChild(document.createTextNode(data));
   }
  }else{
   //recurse
   highlight_goolge(term, node, color);
  }
 }
}

function create_node_google(child, color){
 var node = document.createElement('SPAN');
 node.style.backgroundColor = color;
 node.style.color = google_text_color;
 node.appendChild(child);
 return node;
}
```

An extension of this script, which would be quite simple, would be to enable the higlighter for other search engines. This would involve searching for additonal domains in the referer, and chopping out a different query parameter.

The full code is downloadable for you to use here. Code is free for non commercial or educational use only. For any other purposes, please contact the author.

Thanks go to Ian beveridge of yeahbutisitart.com who suggested a number of bugfixes to my code.