SolrJS

SolrJS is deprecated. See AJAX Solr for a Solr JavaScript client.

SolrJS is a JavaScript client library that was initially developed as a 2008 Google Summer of Code project. It is currently work in progress, this page is intended to show the project's status, some technical documentation and a collection of thoughts about future features.

⚠ Solr1.4

Online example and docs

Refer to http://solrjs.solrstuff.org/ to view an online example and online jsdocs. Note that both example and docs can easily be created locally using ant.

Development

Source code

solrjs can be found in Solr releases under client/javascript. The latest development code can be located at http://svn.apache.org/repos/asf/lucene/dev/trunk/solr/client/javascript/

Creating the documentation

We use jsdoc (http://jsdoc.sourceforge.net/) to create JavaDoc like documentation. Just go to client/javascript, execute "ant docs" and point to "client/javascript/dist/doc/index.html".

Creating the distribution

After execution of "ant dist", a "dist" directory will be created containing 2 files:

- solrjs-1.4-dev.js -> an aggregated file composed of all *.js files in the src tree in correct order.
- solrjs-1.4-dev-templates.jar -> solrjs-1.4-dev-templates.jar velocity files used by server side widgets. This jar can be put into solr/lib when needed.

Creating the reuters example

To explore the code, it's best to investigate and debug the reuters single page example locally. To achieve this, the following steps are needed:

- ant reuters-start -> starts a testsolr instance under port 8983 including the schema for reuters business articles.
- go to "client/javascript/example/reuters/testdata" and execute "./download-dataset.sh". This script will download the dataset from http://kdd.ics.uci.edu/databases/reuters21578/reuters21578.html and extracts the *.sgm files.
- ant reuters-import -> imports the sgm files into the running testsolr
- point the browser to "client/javascript/example/reuters/index.html" and enjoy.

Architectural Overview

The library is written using the javascript toolkit jQuery http://jquery.com/ (minmal version 1.2.5). After distribution, SolrJS is included in one additional javascript file "solrjs.js". The idea is to create several (reusable and extensible) "widgets" that represent solr queries. A widget is a javascript object that is responsible for creating the according solr query as well as render the result from the server to html. One manager object acts as a container that holds these widgets, performs the actual query using jQueries getJSON http://docs.jquery.com/Ajax/jQuery.getJSON#urldatacallback method. This method creates a dynamic script tag, making cross-domain solr requests possible.

There are two base types of widgets:

- client side widgets: They get a Javascript JSON response from the server and render the needed html for the widget using jQuery javascript code.
- server side widgets: They move the rendering logic to the server, using the VelocityResponseWriter (see https://issues.apache.org/jira/browse/SOLR-620. The widget then only "copies" the html reponse from the server into the target div.

To make the framework more clearly arranged, the every js "class" lies inside a single JS file, they are organized in "packages" like in a Java Application:

- core: The manager and other base classes.
- server: server side widgets
- client: client side widgets

For distribution, there is a small ant build script provided taht creates one single js file out of these snippets. Just run "ant dist". In the future, there might be some js compression steps included.

Examples and source code

The best way to get an idea how a widget may look like and how it can be integrated into html is to explore the documented source code and the test*.html example pages at http://solrstuff.org/svn/solrjs/trunk. Use "ant testsolr-import" to get some testdata and "ant testsolr start" to start the included testserver.

Despite this, I'll try to describe some important implementation further on this page.

⚠️ :TODO: ⚠️ ... is this section still relevant? solrstuff.org and explanation of building the reuters demo are already mentioned in above sections.

? ? ?

Implementation

Javascript matters

All SolrJS objects are created inside the jQuery.solrjs namspace. Given JQueries support for overriding the $() function, it is easily possible to use SolrJS alongside other javascript toolkits like prototype or custom javascript code.

Just include the following includes in the header section of your html:

```
<script src="jquery-1.2.5.js"/>
<script src="solrjs.js"/>
<script> var $sj = jQuery.noConflict();        </script>
```

After that, all SolrJS objects are accessible using $sj.solrjs.*

The manager object

The manager object acts as "Controller" for the user input. It holds a state of the current solr query. The current query is represented by QueryItems objetcs. A query item simply is a field:value pair. A user may select or deselect multiple items. In the current work in progress, all query itemsa are concatinated using "AND" operators. More complex query generation may be implemented in the future. The manager currently provides the following methods:

- addWidget: add a new widget
- addQueryItems: add a list of field:value pairs that restrict the current query
- removeQueryItems: remove the given items from the current query
- doRequest(start): performs the actual solr requests
- doRequestAll(): clears queryItems and requests all documents

How a query request works

Every widget implements a "getSolrUrl" method and performs its own http request. It also has to provide a "handleResult" method that gets the json data response to render. So if a user changes the selection:

- manager.doRequest(startindex) is called
- a q=... parameter is created by the manager according to the current selection
- the call is delegated to all the widgets. They may extend the query url in the getSolrUrl method (eg. adding facet=true..) and then make the http call to solr.

- json or html data is returned to the widget and widget.handleResponse(data) is called.

Widget Inheritance

A main goal of the implementation was to create clean easy to understand code that can be easily extended by other developers. So I introduced a small OO inheritance pattern inspired by this thread http://groups.google.com/group/jquery-dev/msg/12d01b62c2f30671 and created the factory method:

jQuery.solrjs.createClass = function(subClassName, baseClassName, subClass)

An "AbstractWidget" class is provided that acts as base class for all further widget implementations. In the source code, all methods that should be implemented by the child class are documented and marked as "abstract methods". An example syntax for a simple widget may be:

```
/*
 * Simple faceted view for one field.
 *
 * @param "id": the widgetId
 * @param "target": the target html element in jquery notation
 * @param "fieldName": The facet field name
 */
jQuery.solrjs.createClass ("SimpleFacetWidget", "AbstractWidget", {

  getSolrUrl : function(start) {
    return "&facet=true&facet.field=" + this.fieldName;
  },

  handleResult : function(data) {
    var values = data.facet_counts.facet_fields[this.fieldName];
    jQuery(this.target).html("");

    for (var i = 0; i < values.length; i = i + 2) {
      var items =  "[new jQuery.solrjs.QueryItem({field:'" + this.fieldName + "',value:'" +  values[i] + "'})]";
      var label = values[i] + "(" + values[i+1] + ")";
      jQuery('<a/>').html(label).attr("href","javascript:solrjsManager.selectItems('" + this.id + "'," + items
+ ")").appendTo(this.target);
        jQuery('<br/>').appendTo(this.target);
      }
  },

  handleSelect : function(data) {
    jQuery(this.target).html(this.selectedItems[0].value);
    jQuery('<a/>').html("(x)").attr("href","javascript:solrjsManager.deselectItems('" + this.id + "')").appendTo
(this.target);
  },

  handleDeselect : function(data) {
    // do nothing
  }
});
```

To add this widget to your html page, just create this script tag:

```
<script>
  var solrjsManager;
  $sj(function(){
    solrjsManager = new $sj.solrjs.Manager({solrUrl:"http://localhost:8983/solr/select"});
    solrjsManager.addWidget(new $sj.solrjs.SimpleResultWidget({id:"result", target:"#result", rows:
20}));
    solrjsManager.addWidget(new $sj.solrjs.SimpleFacetWidget({id:"cat", target:"#cat", fieldName:"cat"}));
    solrjsManager.doRequestAll();
  });
</script>
```

Example for an easily customized widget: ExtensibleResultWidget

The ExtensibleResultWidget is a showcase of a flexible widget. It provides 2 abstract methods:

- renderDataItem
- renderPageStatus

After the document result for the current page is returned, these methods are called and the user is able to use custom javascript code to render the result properly.

```
javascript:

var resultWidget = new $sj.solrjs.ExtensibleResultWidget({
  id:"result",
  target:"#result",
  rows:20,
  renderDataItem : function(item) {
    jQuery('<div/>').html(item.id).appendTo(this.target);
    // more custom code for rendering one single result item
  },
  renderPageStatus : function(first, last, total) {
    // custom code for rendering the pagination status
    jQuery('#pageStatus_start').html(first + "");
    jQuery('#pageStatus_last').html(last + "");
    jQuery('#pageStatus_total').html(total + "");
    var navigation = $sj('#navigation').html("");
    var pageSize = last - first + 1;
    var lastpageFirst = Math.ceil(total / pageSize) * pageSize - pageSize;
    // create navigation buttons for paging
  }
});
solrjsManager.addWidget(resultWidget);


html:

<body>
  <div id="pageStatus">
          Showing <span id="pageStatus_start"></span> to <span id="pageStatus_last"></span> of <span id="
pageStatus_total"></span>
  </div>
  <div id="navigation"></div>
  <div id="result"></div>
</body>
```

Ideas for widgets

Some of this widgets will be implemented during this years Summer of Code and weill be provided alongode the framework.

- Server and client side result views.
- Facet widget for datefields using a timeline chart
- Facet widget for datefields using a javascript calendar
- Facet widget for hierarchical fields represented as a tree (category_0, category_1,..)
- fulltext search
- autocomplete box for facet values
- Geographical selection using google maps or the like.
- Facet widgets that depend on each other
- A single document view that shows similar documents
- .....

Feel free to extend this list.

Security concerns

As we have to access the solr web app directly to get the json data, a discussion about security was raised on the user list. See Ryan's posting about a single, restricted request handler that only allows selects (NO updates and deletes) http://www.nabble.com/Announcement-of-Solr-Javascript-Client-to17462581.html#a17462581. This "ProxyServlet" even could check and modify some parameters to avoid dos attacks or similar (eg. restricting &rows= to a reasonable maximum). Using the jQuery-getJSON method makes it possible to create cross site json requests, so we even can attach this restricted read-only url path to the solr webapp. Every user may then point the javascript client to eg.http://solrserver/readonly/