SolrConfigXml

solrconfig.xml

solrconfig.xml is the file that contains most of the parameters for configuring Solr itself.

⚠ Solr4.7 Solrconfig.xml, schema.xml and all the files that are in the Solr conf directory are editable through the admin UI screen. See Editing configuration files in the admin UI

TODO: Document /get - see [RealTimeGet]

A sample solrconfig.xml with comments can be found in the Source Repository.

lib

<lib> directives can be specified in your solrconfig.xml to specify where Solr should look to load SolrPlugins. The example solrconfig.xml contains main examples of the syntax.

dataDir parameter

Used to specify an alternate directory to hold all index data other than the default ./data under the Solr home. If replication is in use, this should match the replication configuration. If this directory is not absolute, then it is relative to the directory you're in when you start SOLR.

```
<dataDir>/var/data/solr</dataDir>
```

⚠ NOTE: begining with Live Reload] attempting to change the <dataDir> and then RELOAD the core will not work because it is a [CoreAdmin#LiveReload.

codecFactory

⚠ Solr4.0 Codec factories allow your schema to use custom codecs. For example, if you want to define per-field postings and/or DocValues formats, you can use the built-in SchemaCodecFactory:

```
<codecFactory name="CodecFactory" class="solr.SchemaCodecFactory" />
```

Please note that there are some limitations and core reloading might not work as expected when custom codec factories are defined (SOLR-3610). For detailed instructions on how to configure SimpleTextCodec, see: SimpleTextCodec Example

indexConfig Section

⚠ Solr3.6 These settings control low-level behavior of indexing. Please see comments in the example solrconfig.xml for details.

⚠️ NOTE: begining with Live Reload] attempting to change Index Writer related settings in `<indexConfig>` and then RELOAD the core will not work because it is a [CoreAdmin#LiveReload.

Note: In versions prior to 3.6, there were two different sections named <indexDefaults> and <mainIndex>, which are now replaced with this one <indexConfig> section. You can still use the old config as is in 3.6 and 4.x, but need to migrate when bumping luceneMatchVersion. Easiest way to migrate is to start with the example <indexConfig> section which has everything commented out, using the defaults, and then customize if necessary.

Update Handler Section

The Update Handler section mostly relates to low level information about how updates are handled internally (do not confuse with higher level configuration of "Request Handlers" for dealing with updates sent by clients)

```
<updateHandler class="solr.DirectUpdateHandler2">

    <!-- Limit the number of deletions Solr will buffer during doc updating.

        Setting this lower can help bound memory use during indexing.
    -->
    <maxPendingDeletes>100000</maxPendingDeletes>

    <!-- autocommit pending docs if certain criteria are met.  Future versions may expand the available
     criteria -->
    <autoCommit>
      <maxDocs>10000</maxDocs> <!-- maximum uncommited docs before autocommit triggered -->
      <maxTime>15000</maxTime> <!-- maximum time (in MS) after adding a doc before an autocommit is triggered --
>
      <openSearcher>false</openSearcher> <!-- SOLR 4.0.  Optionally don't open a searcher on hard commit.  This
is useful to minimize the size of transaction logs that keep track of uncommitted updates. -->
    </autoCommit>

    ...
```

⚠️ Solr4.0

```
   ...
     <autoCommit>
       <maxTime>1000</maxTime>
     </autoCommit>

    <!-- SoftAutoCommit

         Perform a 'soft' commit automatically under certain conditions.
         This commit avoids ensuring that data is synched to disk.

         maxDocs - Maximum number of documents to add since the last
                   soft commit before automaticly triggering a new soft commit.

         maxTime - Maximum amount of time in ms that is allowed to pass
                   since a document was added before automaticly
                   triggering a new soft commit.
      -->

     <autoSoftCommit>
       <maxTime>1000</maxTime>
     </autoSoftCommit>

    ...
```

"Update" Related Event Listeners

Within the Update Handler Section, you can define listeners for particular "update" related events: "postCommit" and "postOptimize". Listeners can be used to fire-off any special code; they are typically used to exec snapshooter.

```
  ...

  <!-- The RunExecutableListener executes an external command.
       exe  - the name of the executable to run
       dir  -  dir to use as the current working directory. default="."
       wait - the calling thread waits until the executable returns.
              default="true"
       args - the arguments to pass to the program.  default=nothing
       env  - environment variables to set.  default=nothing
    -->
  <!-- A postCommit event is fired after every commit
    -->
  <listener event="postCommit" class="solr.RunExecutableListener">
    <str name="exe">snapshooter</str>
    <str name="dir">solr/bin</str>
    <bool name="wait">true</bool>
    <!--
    <arr name="args"> <str>arg1</str> <str>arg2</str> </arr>
    <arr name="env"> <str>MYVAR=val1</str> </arr>
      -->
  </listener>
</updateHandler>
```

The Query Section

Controls everything query-related.

```
<query>
  <!-- Maximum number of clauses in a boolean query... can affect range
       or wildcard queries that expand to big boolean queries.
       An exception is thrown if exceeded.
  -->
  <maxBooleanClauses>1024</maxBooleanClauses>
```

Caching Section

You can change these caching parameters as your index grows and changes. See the SolrCaching page for details on configuring the caches.

```
  <!-- Cache used by SolrIndexSearcher for filters (DocSets),
       unordered sets of *all* documents that match a query.
       When a new searcher is opened, its caches may be prepopulated
       or "autowarmed" using data from caches in the old searcher.
       autowarmCount is the number of items to prepopulate.  For LRUCache,
       the autowarmed items will be the most recently accessed items.
     Parameters:
       class - the SolrCache implementation (currently only LRUCache)
       size - the maximum number of entries in the cache
       initialSize - the initial capacity (number of entries) of
         the cache.  (seel java.util.HashMap)
       autowarmCount - the number of entries to prepopulate from
         and old cache.
       -->
  <filterCache
    class="solr.LRUCache"
    size="512"
    initialSize="512"
    autowarmCount="256"/>

 <!-- queryResultCache caches results of searches - ordered lists of
       document ids (DocList) based on a query, a sort, and the range
       of documents requested.  -->
  <queryResultCache
    class="solr.LRUCache"
    size="512"
    initialSize="512"
```

```
    autowarmCount="256"/>

<!-- documentCache caches Lucene Document objects (the stored fields for each document).
     Since Lucene internal document ids are transient, this cache will not be autowarmed.  -->
  <documentCache
    class="solr.LRUCache"
    size="512"
    initialSize="512"
    autowarmCount="0"/>

  <!-- Example of a generic cache.  These caches may be accessed by name
       through SolrIndexSearcher.getCache().cacheLookup(), and cacheInsert().
       The purpose is to enable easy caching of user/application level data.
       The regenerator argument should be specified as an implementation
       of solr.search.CacheRegenerator if autowarming is desired.  -->
  <!--
  <cache name="myUserCache"
    class="solr.LRUCache"
    size="4096"
    initialSize="1024"
    autowarmCount="1024"
    regenerator="org.mycompany.mypackage.MyRegenerator"
    />
  -->

  <!-- An optimization that attempts to use a filter to satisfy a search.
       If the requested sort does not include a score, then the filterCache
       will be checked for a filter matching the query.  If found, the filter
       will be used as the source of document ids, and then the sort will be
       applied to that.
    -->
  <useFilterForSortedQuery>true</useFilterForSortedQuery>

  <!-- An optimization for use with the queryResultCache.  When a search
       is requested, a superset of the requested number of document ids
       are collected.  For example, of a search for a particular query
       requests matching documents 10 through 19, and queryWindowSize is 50,
       then documents 0 through 50 will be collected and cached. Any further
       requests in that range can be satisfied via the cache.
  -->
  <queryResultWindowSize>50</queryResultWindowSize>

  <!-- This entry enables an int hash representation for filters (DocSets)
       when the number of items in the set is less than maxSize. For smaller
       sets, this representation is more memory efficient, more efficient to
       iterate over, and faster to take intersections.
   -->
  <HashDocSet maxSize="3000" loadFactor="0.75"/>


  <!-- boolToFilterOptimizer converts boolean clauses with zero boost
       cached filters if the number of docs selected by the clause exceeds the
       threshold (represented as a fraction of the total index)
  -->
  <boolTofilterOptimizer enabled="true" cacheSize="32" threshold=".05"/>

  <!-- Lazy field loading will attempt to read only parts of documents on disk that are
       requested.  Enabling should be faster if you aren't retrieving all stored fields.
  -->
  <enableLazyFieldLoading>false</enableLazyFieldLoading>
```

"Query" Related Event Listeners

Withing the Query section, you can define listeners for particular "query" related events — listeners can be used to fire-off special code — such as invoking some common queries to warm-up caches.

newSearcher

A New Searcher is opened when a (current) Searcher already exists. In the example below, the listener is of the class, QuerySenderListener, which takes lists of queries and sends them to the new searcher being opened, thereby warming it.

```
    <!-- a newSearcher event is fired whenever a new searcher is being
         prepared and there is a current searcher handling requests
         (aka registered).
     -->
    <!-- QuerySenderListener takes an array of NamedList and
         executes a local query request for each NamedList in sequence.
     -->
    <!--
    <listener event="newSearcher" class="solr.QuerySenderListener">
      <arr name="queries">
        <lst> <str name="q">solr</str>
              <str name="start">0</str>
              <str name="rows">10</str>
        </lst>
        <lst> <str name="q">rocks</str>
              <str name="start">0</str>
              <str name="rows">10</str>
        </lst>
      </arr>
    -->
```

firstSearcher

A First Searcher is opened when there is *no* existing (current) Searcher. In the example below, the listener is of the class, QuerySenderListener, which takes lists of queries and sends them to the new searcher being opened, thereby warming it. (If there is no Searcher, you cannot use auto-warming because auto-warming requires an existing Searcher.)

```
    <!-- a firstSearcher event is fired whenever a new searcher is being
         prepared but there is no current registered searcher to handle
         requests or to gain prewarming data from.
     -->
    <!--
    <listener event="firstSearcher" class="solr.QuerySenderListener">
      <arr name="queries">
        <lst> <str name="q">fast_warm</str>
              <str name="start">0</str>
              <str name="rows">10</str>
        </lst>
      </arr>
    </listener>
    -->
  </query>
```

Request Dispatcher

The Request Dispatcher section is the only section of the solrconfig.xml to be used directly by Solr's HTTP RequestDispatcher to configure how it should deal with various aspects of HTTP requests, including wether it should handle "/select" urls (for Solr 1.1 compatibility); HTTP Request Parsing; remote streaming support; max multipart file upload size, etc...

```
  <!--
    Let the dispatch filter handler /select?qt=XXX
    handleSelect=true will use consistent error handling for /select and /update
    handleSelect=false will use solr1.1 style error formatting
    -->
  <requestDispatcher handleSelect="true" >
    <!--Make sure your system has some authentication before enabling remote streaming!  -->
    <requestParsers enableRemoteStreaming="false" multipartUploadLimitInKB="2048" formdataUploadLimitInKB="
2048" />

  ...
```

HTTP Caching

⚠️ Solr1.3

Within the main Request Dipatcher section, the HTTP Caching subsection contains configuration options for controlling how the Solr Request Dispatcher responds to HTTP Requests that include cache validation headers, and what kinds of responses Solr will generate. More information can be found in SolrAndHTTPCaches

```
  ...
   <!-- Set HTTP caching related parameters (for proxy caches and clients).

        To get the behaviour of Solr 1.2 (ie: no caching related headers)
        use the never304="true" option and do not specify a value for
        <cacheControl>
   -->
   <!-- <httpCaching never304="true"> -->
   <httpCaching lastModFrom="openTime"
                etagSeed="Solr">
      <!-- lastModFrom="openTime" is the default, the Last-Modified value
           (and validation against If-Modified-Since requests) will all be
           relative to when the current Searcher was opened.
           You can change it to lastModFrom="dirLastMod" if you want the
           value to exactly corrispond to when the physical index was last
           modified.

           etagSeed="..." is an option you can change to force the ETag
           header (and validation against If-None-Match requests) to be
           differnet even if the index has not changed (ie: when making
           significant changes to your config file)

           lastModifiedFrom and etagSeed are both ignored if you use the
           never304="true" option.
      -->
      <!-- If you include a <cacheControl> directive, it will be used to
           generate a Cache-Control header, as well as an Expires header
           if the value contains "max-age="

           By default, no Cache-Control header is generated.

           You can use the <cacheControl> option even if you have set
           never304="true"
      -->
      <!-- <cacheControl>max-age=30, public</cacheControl> -->
    </httpCaching>
  </requestDispatcher>
```

The value for *max-age* should be set based on how often your index changes and how long your application can live with an outdated cached response. To force a shared (or browser) cache to recheck that the cached response is still valid you can add the parameter *must-revalidate* to the Cache-Control header. According to the W3C specification *max-age* should not be higher than 31536000 (1 year).

Request Handler Plug-in Section

⚠️ :TODO: ⚠️ Add details on supplying init parameters.

This is where multiple request handlers can be registered.

```
<!-- requestHandler plugins... incoming queries will be dispatched to
     the correct handler based on the qt (query type) param matching the
     name of registered handlers. The "standard" request handler is the
     default and will be used if qt is not specified in the request.
  -->
  <requestHandler name="standard" class="solr.StandardRequestHandler" />
  <requestHandler name="custom" class="your.package.CustomRequestHandler" />
```

Configuration of Shard Handlers for Distributed searches

Inside requestHandlers it is possible to configure and specify the shard handler used for distributed search, it is possible to plug in custom shard handlers as well as configure the provided solr version.

To configure the standard handler one needs to provide configuration like so

```
<requestHandler name="standard" class="solr.SearchHandler" default="true">
    <!-- other params go here -->
     <shardHandlerFactory class="HttpShardHandlerFactory">
        <int name="socketTimeout">1000</int>
        <int name="connTimeout">5000</int>
      </shardHandlerFactory>
  </requestHandler>
```

The parameters that can be specified are as follows:

- socketTimeout. default: 0 (use OS default) - The amount of time in ms that a socket is allowed to wait for
- connTimeout. default: 0 (use OS default) - The amount of time in ms that is accepted for binding / connection a socket
- maxConnectionsPerHost. default: 20 - The maximum number of connections that is made to each individual shard in a distributed search
- corePoolSize. default: 0 - The retained lowest limit on the number of threads used in coordinating distributed search
- maximumPoolSize. default: Integer.MAX_VALUE - The maximum number of threads used for coordinating distributed search
- maxThreadIdleTime. default: 5 seconds - The amount of time to wait for before threads are scaled back in response to a reduction in load
- sizeOfQueue. default: -1 - If specified the thread pool will use a backing queue instead of a direct handoff buffer. This may seem difficult to grasp, essentially high throughput systems will want to configure this to be a direct hand off (with -1). Systems that desire better latency will want to configure a reasonable size of queue to handle variations in requests.
- fairnessPolicy. default: false - Chooses in the JVM specifics dealing with fair policy queuing, if enabled distributed searches will be handled in a First in First out fashion at a cost to throughput. If disabled throughput will be favoured over latency.

UpdateRequestProcessorChain section

This allows you to configure a processing chain that processes a document before indexing. The chain consists of UpdateRequestProcessors, each doing some processing on the document. You can define multiple chains to use in different update request handlers.

Example chain, commented out in solrconfig.xml:

```
  <updateRequestProcessorChain name="dedupe">
    <processor class="org.apache.solr.update.processor.SignatureUpdateProcessorFactory">
      <bool name="enabled">true</bool>
      <str name="signatureField">id</str>
      <bool name="overwriteDupes">false</bool>
      <str name="fields">name,features,cat</str>
      <str name="signatureClass">org.apache.solr.update.processor.Lookup3Signature</str>
    </processor>
    <processor class="solr.LogUpdateProcessorFactory" />
    <processor class="solr.RunUpdateProcessorFactory" />
  </updateRequestProcessorChain
```

The chain is then referenced from your UpdateRequestHandler:

```
  <requestHandler name="/update "class="solr.XmlUpdateRequestHandler">
    <lst name="defaults">
      <str name="update.chain">dedupe</str>
    </lst>
  </requestHandler>
```

⚠️ Solr3.1 NOTE: Prior to Solr3.1 you need to use update.processor instead of update.chain

The Highlighter plugin configuration section

You can configure custom fragmenters and formatters for use in highlighting. Each may be configured with its own set of highlighting parameter defaults. See also HighlightingParameters.

```
 <highlighting>
  <!-- Configure the standard fragmenter -->
  <!-- This could most likely be commented out in the "default" case -->
  <fragmenter name="gap" class="org.apache.solr.highlight.GapFragmenter" default="true">
   <lst name="defaults">
    <int name="hl.fragsize">100</int>
   </lst>
  </fragmenter>

  <!-- A regular-expression-based fragmenter (f.i., for sentence extraction) -->
  <fragmenter name="regex" class="org.apache.solr.highlight.RegexFragmenter">
   <lst name="defaults">
     <!-- slightly smaller fragsizes work better because of slop -->
     <int name="hl.fragsize">70</int>
     <!-- allow 50% slop on fragment sizes -->
     <float name="hl.regex.slop">0.5</float>
     <!-- a basic sentence pattern -->
     <str name="hl.regex.pattern">[-\w ,/\n\"']{20,200}</str>
   </lst>
  </fragmenter>

  <!-- Configure the standard formatter -->
  <formatter name="html" class="org.apache.solr.highlight.HtmlFormatter" default="true">
   <lst name="defaults">
    <str name="hl.simple.pre"><![CDATA[<em>]]></str>
    <str name="hl.simple.post"><![CDATA[</em>]]></str>
   </lst>
  </formatter>
 </highlighting>
```

Providing a Custom Highlighter

⚠️ Solr1.3

Since Solr 1.3, applications can provide their own highlighting code. To take advantage of this, configure the HighlightComponent as above, but, in the `<highlighting` declaration, pass in fully qualified Java class name, as in class="com.me.MyHighlighter". The implementing class must extend the SolrHighlighter class.

The Admin/GUI Section

This section handles the administration web page.

Defines “Gettable” files — allows the defined files to be accessed through the web interface. Also specifies the default search to be filed in on the admin form.

The `<pingQuery>` defines what the "ping" query should be for monitoring the health of the Solr server. The URL of the "ping" query is **/admin/ping**. It can be used by a load balancer in front of a set of Solr servers to check response time of all the Solr servers in order to do response time based load balancing.

Including the optional `<healthcheck type="file">` will add a **ENABLE**/**DISABLE** link on the administration web page which can be used to create /remove a file at the path defined by the value of `<healthcheck>`. When the file is removed then the "ping" query will automatically fail. A load balancer in front of a set of Solr servers can then determine when it should keep/add/remove a server from rotation by pinging the server.

```
<admin>
    <defaultQuery>solr</defaultQuery>
    <gettableFiles>
         solrconfig.xml
         schema.xml
    </gettableFiles>
    <pingQuery>q=solr&amp;version=2.0&amp;start=0&amp;rows=0</pingQuery>

    <!-- configure a healthcheck file for servers behind a loadbalancer
      -->
    <healthcheck type="file">server-enabled</healthcheck>
  </admin>
```

System property substitution

Solr supports system property substitution, allowing the launching JVM to specify string substitutions within either of Solr's configuration files. The syntax ${property[:default value]}. Substitutions are valid in any element or attribute text. Here's an example of allowing the runtime to dictate the data directory:

```
   <dataDir>${solr.data.dir:./solr/data}</dataDir>
```

And using the example application, Solr could be launched in this manner:

```
java -Dsolr.data.dir=/data/dir -jar start.jar
```

If no default value is provided, the system property MUST be specified otherwise a Solr startup failure occurs indicating what property has no value.

⚠️ Solr1.4 All the properties which need to be substituted can be put into a properties file and can be put into the <solr.home>/conf/solrcore.properties. example :

```
#solrcore.properties
data.dir=/data/solrindex
```

and in the solrconfig.xml it can be used as follows

```
   <dataDir>${data.dir}</dataDir>
```

Enable/disable components

⚠️ Solr1.4

Every component can have an extra attribute enable which can be set as true/false.

example:

```
<requestHandler name="/replication" class="solr.ReplicationHandler" enable="{enable.replication:true}">

</requestHandler>
```

here the value of 'enable.replication' can be provided from outside and it can be enabled/disabled at runtime

XInclude

⚠️ Solr1.4

Portions of the solrconfig.xml file can be externalized and included using a standard xml feature called XInclude. The easy to read specification for XInclude can be found at http://www.w3.org/TR/xinclude/

This might be useful in cases where there are a number of settings that only differ if the server is a master or a slave. These settings could be factored out into two separate files and then included as needed to avoid duplication of common portions of solrconfig.xml

The following sample attempts to include solrconfig_master.xml. If it doesn't exist or can't be loaded, then solrconfig_slave.xml will be used instead. If solrconfig_slave.xml can't be loaded an XML parsing exception will result.

```
    <xi:include href="solrconfig_master.xml" xmlns:xi="http://www.w3.org/2001/XInclude">
      <xi:fallback>
        <xi:include href="solrconfig_slave.xml"/>
      </xi:fallback>
    </xi:include>
```

solrconfig_master.xml might contain:

```
<requestHandler name="/replication" class="solr.ReplicationHandler" >
    <lst name="master">
      <str name="replicateAfter">commit</str>
      <str name="replicateAfter">startup</str>
      <str name="confFiles">schema.xml,stopwords.txt</str>
    </lst>
</requestHandler>
```

Notes:

- The included XML file must be valid self-contained XML with a single root node.
- Since the xinclude elements are handled by the XML parser and not by solr, properties expansion is not available.
- File paths in href attributes can be absolute or relative to the current file's directory (this changed since Solr 3.1: before, the XIncludes were not resolved using the base directory of the current XML file, so the CWD was used - if you upgrade to 3.1, you may need to change you XInclude hrefs). If you want to ensure absolute links, use file:-URIs (e.g., "file:/etc/solr/solrconfig.xml") to refer to those files. By default Solr 3.1 will use the SolrResourceLoader to search for the referenced files, so it behaves like loading stopword files in the schema. XIncludes can also be HTTP URLs to fetch resources from remote servers if desired.
- The Xerces parser, used by default in Solr, doesn't support the xpointer="xpointer()" scheme. http://xerces.apache.org/xerces2-j/faq-xinclude.html

Includes via Document Entities

An alternate way of doing includes is to use Document Entities.

At the top of solrconfig.xml:

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE config [
        <!ENTITY exampleinclude SYSTEM "example.xml">
]>
```

Where example.xml contains the XML segment you want to import. To use it, just use &entityname; (eg. &exampleinclude😉 wherever you like.

See: http://lucene.472066.n3.nabble.com/XInclude-Multiple-Elements-td3167658.html