ScriptUpdateProcessor

ScriptUpdateProcessor

⚠️ Solr4.0

Introduction

The ScriptUpdateProcessor enables update processing code to be written in a scripting language. The script can be written in any scripting language supported by your JVM (such as JavaScript), and executed dynamically so no pre-compilation is necessary.

Configuration

The UpdateRequestProcessor is configured in solrconfig.xml. All parameters listed may also be overridden on the update request itself. A minimal configuration specifies the script file to use:

```
<updateRequestProcessorChain name="script">
  <processor class="solr.StatelessScriptUpdateProcessorFactory">
    <str name="script">update-script.js</str>
  </processor>
  <!--  optional parameters passed to script
    <lst name="params">
      <str name="config_param">example config parameter</str>
    </lst>
  -->
  <processor class="solr.LogUpdateProcessorFactory" />
  <processor class="solr.RunUpdateProcessorFactory" />
</updateRequestProcessorChain>
```

NOTE: The processor supports the `defaults/appends/invariants` concept for its config. However, it is also possible to skip this level and configure the parameters directly underneath the `<processor>` tag.

Below follows a list of each configuration parameters and their meaning:

script

The script file name. The script file must be placed in the conf/ directory. There can be one or more "script" parameters specified; multiple scripts are executed in the order specified.

engine

Optionally specifies the scripting engine to use. This is only needed if the extension of the script file is not a standard mapping to the scripting engine. For example, if your script file was coded in JavaScript but the file name was called update-script.foo, use "javascript" as the engine name.

params

Optional parameters that are passed into the script execution context. This is specified as a named list (<lst>) structure with nested typed parameters. If specified, the script context will get a "params" object, otherwise there will be no "params" object available.

Script execution context

Provided variables

logger

Logger (org.slf4j.Logger) instance. This is useful for logging information from the script.

req

SolrQueryRequest (org.apache.solr.request.SolrQueryRequest) instance.

rsp

SolrQueryResponse (org.apache.solr.response.SolrQueryResponse) instance.

params

The "params" object, if any specified, from the configuration.

Examples

The processAdd() and the other script methods can return false to skip further processing of the document. All methods must be defined, though generally the processAdd() method is where the action is.

Here's a URL that works with the collection1 example demonstrating specifying the "script" update chain: http://localhost:8983/solr/collection1/update?commit=true&stream.contentType=text/csv&fieldnames=id,description&stream.body=1,foo&update.chain=script which logs the following:

```
INFO: update-script#processAdd: id=1
```

JavaScript

Note: There is a JavaScript example (update-script.js) built into the Solr 4 and up example collection1 configuration. Check solrconfig.xml and uncomment the update request processor definition to enable this feature.

```
function processAdd(cmd) {

  doc = cmd.solrDoc;  // org.apache.solr.common.SolrInputDocument
  id = doc.getFieldValue("id");
  logger.info("update-script#processAdd: id=" + id);

// Set a field value:
//  doc.setField("foo_s", "whatever");

// Get a configuration parameter:
//  config_param = params.get('config_param');  // "params" only exists if processor configured with <lst name="
params">

// Get a request parameter:
// some_param = req.getParams().get("some_param")

// Add a field of field names that match a pattern:
//   - Potentially useful to determine the fields/attributes represented in a result set, via faceting on
field_name_ss
//  field_names = doc.getFieldNames().toArray();
//  for(i=0; i < field_names.length; i++) {
//    field_name = field_names[i];
//    if (/attr_.*/.test(field_name)) { doc.addField("attribute_ss", field_names[i]); }
//  }

}

function processDelete(cmd) {
  // no-op
}

function processMergeIndexes(cmd) {
  // no-op
}

function processCommit(cmd) {
  // no-op
}

function processRollback(cmd) {
  // no-op
}

function finish() {
  // no-op
}
```

JRuby

To use JRuby as the scripting engine, add jruby.jar to your system (in a lib/ directory under a collection, or via <lib> directives in solrconfig.xml).

Here's an example JRuby update processing script (note that all variables passed require prefixing with $, such as $logger):

```
def processAdd(cmd)
  doc = cmd.solrDoc  # org.apache.solr.common.SolrInputDocument
  id = doc.getFieldValue('id')

  $logger.info "update-script#processAdd: id=#{id}"

  doc.setField('source_s', 'ruby')

  $logger.info "update-script#processAdd: config_param=#{$params.get('config_param')}"
end

def processDelete(cmd)
  # no-op
end

def processMergeIndexes(cmd)
  # no-op
end

def processCommit(cmd)
  # no-op
end

def processRollback(cmd)
  # no-op
end

def finish()
  # no-op
end
```

Known issues

The following in JRuby do not work as expected for some reason, though it does work properly in JavaScript:

```
#  $logger.info "update-script#processAdd: request_param=#{$req.params.get('request_param')}"
#  $rsp.add('script_processed',id)
```

Groovy

Put all JARs from a Groovy distro's lib/ directory into Solr's resource loader (core lib/ directory, <lib> directive in solrconfig, etc). All JARs from Groovy's distro probably aren't required, but more than just the main Groovy .jar file is needed (at least when this was tested using Groovy 2.0.6)

example update-script.groovy:

```
def processAdd(cmd) {
  doc = cmd.solrDoc  // org.apache.solr.common.SolrInputDocument
  id = doc.getFieldValue('id')

  logger.info "update-script#processAdd: id=" + id

  doc.setField('source_s', 'groovy')

  logger.info "update-script#processAdd: config_param=" + params.get('config_param')

  logger.info "update-script#processAdd: request_param=" + req.params.get('request_param')
  rsp.add('script_processed',id)
}

def processDelete(cmd) {
 //  no-op
}

def processMergeIndexes(cmd) {
 // no-op
}

def processCommit(cmd) {
 //  no-op
}

def processRollback(cmd) {
 // no-op
}

def finish() {
 // no-op
}
```

Jython

Put the *standalone* jython JAR (the JAR that contains all the dependencies) into Solr's resource loader (core lib/ directory, <lib> directive in solrconfig, etc).

example update-script.py:

```
def processAdd(cmd):
  doc = cmd.solrDoc
  id = doc.getFieldValue("id")
  logger.info("update-script#processAdd: id=" + id)

def processDelete(cmd):
    logger.info("update-script#processDelete")

def processMergeIndexes(cmd):
    logger.info("update-script#processMergeIndexes")

def processCommit(cmd):
    logger.info("update-script#processCommit")

def processRollback(cmd):
    logger.info("update-script#processRollback")

def finish():
    logger.info("update-script#finish")
```

Caveats

TBD: Add more on the "stateless" nature, running multiple scripts that can share state, and more about other scripting languages available.

Resources

- SOLR-1725: original JIRA issue tracking the initial development of this feature.