

SolrSherlock Getting Started

Latest edit: 20130619

Background

SolrSherlock is a *society of agents* framework which serves two primary use cases:

1. A Merge Agent for a Topic Map
2. A DeepQA research platform

Both of those use cases share agent components; merge and harvesting (*machine reading*) require many of the same agent functions. Those functions will be described later; for now, our goal is to sketch the high-level picture of what is inside SolrSherlock. As the following figures will illustrate, the core platform is based on an installation of Apache Solr, with certain additions which we describe here.

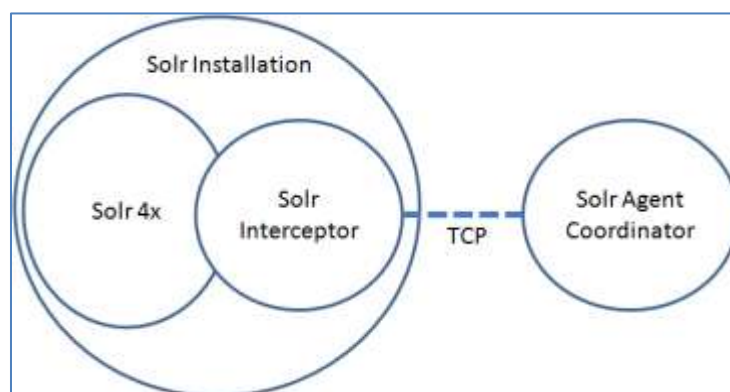


Figure 1. Core Solr Installation Framework

As Figure 1 shows, Solr itself is installed in a particular directory, which we describe below, including certain additional features and changes to the Solr's initial distribution configurations. Outside of that installation is a Solr Agent Coordinator, a kind of *blackboard* which listens to the Solr Interceptor through a TCP socket. Since the agent coordinator listens over a TCP socket, it need not be installed on the same computer as Solr. Many simple installations can have both on the same computer.

Solr, itself, is useful for indexing things, but we have made additions to it such that Solr is now indexing particular kinds of documents, each of which is a *topic* in a *topic map*. We will explain the details of this terminology elsewhere, but for now, the next figure illustrates the first agents installed in SolrSherlock, the topic map agents.

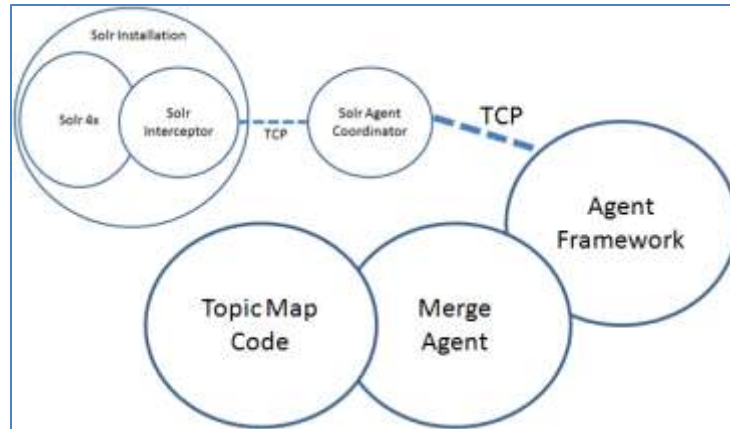


Figure 2. Solr as a Topic Map

Figure 2 shows that we install agents which serve the purpose of maintaining the topic map. An instance of Agent Framework uses a TCP socket to watch the Solr Agent Coordinator for new topic documents. Each new document must be scanned, as if asking this question: *have I seen this before?*

That question is answered by making comparisons between the new document and documents already indexed in Solr. When the document is found to be *about* an already-known topic, then a *merge process* (to be described elsewhere) is started.

That completes the first use case for SolrSherlock. In performing that use case, the merge agent will make use of other agents throughout the harvesting and question-answering features of SolrSherlock. In the next figure, we sketch how the DeepQA use case is satisfied with more components.

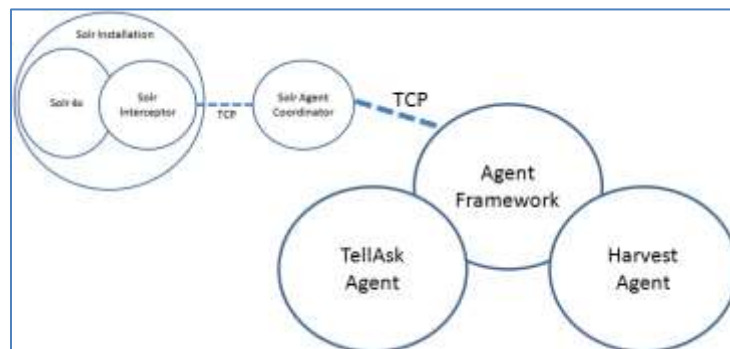


Figure 3. SolrSherlock DeepQA Agents

Figure 3 shows that we use a different instance of the Agent Framework to anchor several (just two are shown) agents. Each different agent performs a different task. For instance, the TellAsk agent is the conversational interface through which users *tell* information to SolrSherlock, e.g. answer questions, or *ask* questions to SolrSherlock. A Harvest agent, as illustrated, is one of many different kinds of agents performing tasks such as reading documents and converting what is harvested into entries in the topic map.

Solr Installation (Single server)

Key point to keep in mind with what follows:

This section does not describe installation of a SolrCloud; it is for installation of a single instance of Solr, as one would perform during development and testing.

General Installation and Adaptation to SolrSherlock

In general, Solr is installed precisely as described in the Solr Tutorial¹. In that regard, there are choices:

- Simply use the Solr distribution's `/example` directory, which includes a working copy of the Jetty servlet container, ready to run (after modifications we outline here)
- Create a Solr installation by opening the distribution's `war` file and distributing the files into a Jetty container.

In any case, the ultimate layout is a *Solr Installation Directory* as illustrated in the next figure.

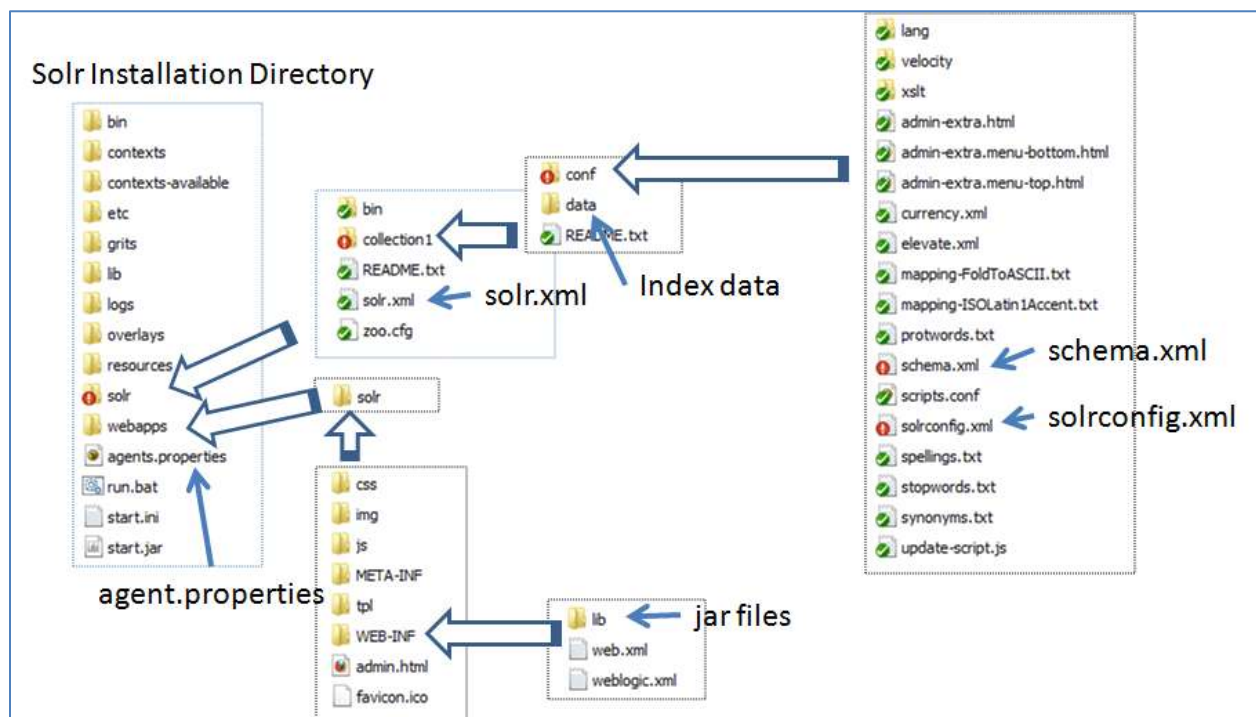


Figure 4. Solr Installation Directory (annotated)

Figure 4, on the left hand side, is a Jetty container with various directories added to it to create a Solr installation. There are several key points about this illustration:

- the file `agent.properties` from SolrSherlock's Solr Interceptor distribution is added in the root directory
- the directory `/solr` is added in which the `/collection1` directory is important

¹ Solr Tutorial: http://lucene.apache.org/solr/4_3_0/tutorial.html

- Two files must be replaced from the SolrSherlock distribution
 - `schema.xml`
 - `solrconfig.xml`
- the directory `/webapps/solr/WEB-INF/lib` is important since it contains jar files used by Solr and by SolrSherlock
 - the jar file `SolrUpdateResponseHandler.jar` (Solr Interceptor) is added to this directory
 - the jar file `json_simple-1.2.jar`
 - Note: upgrading to Solr 4.3+ and we need to add more jars to this directory
 - `slf4j-api-1.6.4.jar`
 - `slf4j-jdk14-1.6.4.jar`
 - `commons-logging-1.1.jar`

Runtime Considerations

The `run.bat` file looks like this:

```
java -Xms256M -Xmx1024m -Dsystem.properties=conf\solrcore.properties -jar start.jar
```

where two JVM directives have been highlighted. They determine the amount of memory this instance of Solr will have available to it. In the Solr literature, there is a large volume of conversation about setting those values; the simplest distillation of that is this: the more memory you can give Solr, the better.

Solr's Index

All of the Solr index data is in the directory `/collection1/data/index`. When first installed, that directory will be empty (Figure 5); the moment Solr is booted, there will be a small set of index files created. As we shall see later, SolrSherlock, when it first boots, will add to that index.

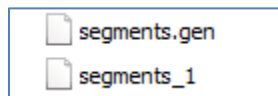


Figure 5. Empty Solr Index

Solr as a Topic Map

As we shall see, Solr begins life as a simple indexing system. To turn Solr into a topic map, we perform these tasks at installation:

- Install Solr
- Modify Solr
 - Swap in from the SolrSherlock distribution (replace these files)
 - `schema.xml`
 - `solrconfig.xml`
 - Add these files from the SolrSherlock distribution, as illustrated:
 - `agent.properties`

- `SolrUpdateResponseHandler.jar`
- `json_simple-1.2.jar`

With those modifications, Solr is now prepared to serve as a topic map. What remains is to install two agents:

- Solr Agent Coordinator
- Merge Agent

Those installations are the topic of the next section.

Agent Installation

The context of this installation is purely that of booting the topic map. In a more expansive document, we will cover agents associated with SolrSherlock as a DeepQA system.

Solr Agent Coordinator

There are two considerations when installing the agent coordinator:

- Where it is to be installed (that is, on which computer)
- Configuration settings which relate to where it is installed

Where it is installed

In the simplest installation, the agent coordinator can be installed on the same computer on which Solr is installed. Each uses a different JVM; issues which relate to this are these:

- Available memory
- Performance (two systems running and busy at the same time)

Regardless of where the agent is installed, its installation directory is illustrated next.

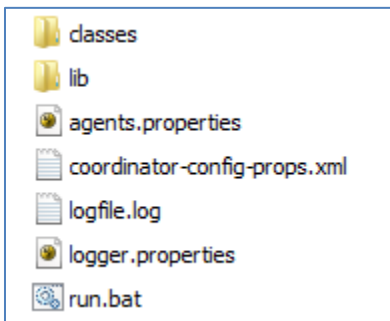


Figure 6. Solr Agent Coordinator Installation Directory

Figure 6 shows the agent coordinator's current directory layout. In a future update, the /classes directory will be removed in favor of a jar file for the agent itself. When the system is booted, it will create and maintain a log file.

Configuration settings

There are many considerations related to the configuration values associated with the agent coordinator. They are these:

- A unique name given to the coordinator, which is defined now for later logging purposes
- The port to which other agents communicate with this coordinator
- Which interceptors it listens to, their specific address; this allows for future expansion in the case where one agent coordinator is listening to more than one Solr Interceptor

The configuration file defines these properties:

```
<properties>
```

<pre> <!-- The name given to this agent, for use in creating Trace files. The name should be unique for all agents running on the same computer --> <parameter name="SemiSpaceName" value="MySpace94026" /> <!-- The port on which we listen for requests from SolrAgentFrameworks and other agents who want to talk to us --> <parameter name="TupleListenPort" value="2930" /> <!-- provide a list of the server IPs on which SolrInjectors to which we are supposed to listen; value is the port number for that server. We require possibly many since we might be listening to many Solr cores in SolrCloud --> <list name="SolrInjectors"> <parameter name="127.0.0.1" value="2925" /> </list> </properties> </pre>

Table 1. Agent Coordinator Properties

In this case, there is just one Solr Interceptor (labeled `SolrInjectors` in the name field), and it is defined in this instance as *localhost* talking on the port 2925, as defined by the Solr Interceptor to which it is listening. This is the case where the agent coordinator is installed on the same computer as is Solr.

Merge Agent

Configuration settings for the merge agent, like the agent coordinator, are concerned, in the same fashion, with locations of the merge agent and the agent coordinator. If they are on the same computer, then *localhost* is appropriate. Otherwise, the IP address must be different.

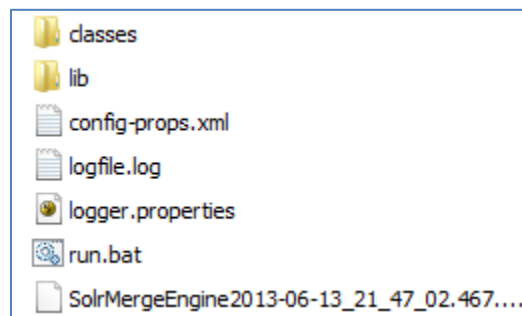


Figure 7. Merge Agent Directory

Figure 7 shows the current configuration of the merge agent, using the `/classes` directory; eventually, the jar file for this agent will reside in `/lib`. Notice that the merge agent produces two files during runtime:

- `logfile.log`
- `SolrMergeEngine<date>`

`SolrMergeEngine<date>` is a gzipped file; it is a trace file which is different from a debug and error log file; it records a trace of specific merge operations performed for later study. The file is closed and a new one started after a certain number of trace lines. Configuration properties are important to this agent. There are many agent properties here, some specific to the merge agent and some general to the Agent Framework which supports the Merge Agent. We list them all here.

```
<properties>

  <!--
    ShouldBootstrap:
      Yes = test for bootstrap the topic map
      No = do not test for bootstrap
    Only one config-props.xml file should say "Yes" in
      an installation; typically on the SolrAgentFramework that includes
      the MergeAgent. Other installations of SolrAgentFramework
      in the area or network should say "No".
      It is unlikely that there will be problems, the probability
      that two agents, while booting, will decide to begin the
      bootstrap process at the same time is not nil.
  -->
  <parameter name="ShouldBootstrap" value="Yes" /> <!-- should be Yes, can be No -->

  <!--
    The core TopicQuests topic map platform has an internal merge function which
    is plug-in swappable. "MergeImplementation" is the key to get the
    classpath (value) for that object
  -->
  <parameter name="MergeImplementation" value="org.topicquests.solr.merge.MergeBean" />

  <!--
    AgentBehavior value is a string which determines certain behaviors.
    There are presently two types of behaviours:
      "harvest" where the system stays tightly coupled with Solr update events
      "model" where the system watches other SolrAgentEnvironment instances
        for requests for models or results
    Behaviors are defined in IAgentFrameworkBehavior
      though it might be possible to make the behavior extensible such
      that new behaviors can be "plugged in" by way of the agents
      themselves.
  -->
```



```

<parameter name="AgentBehavior" value="harvest" />

<!--
    AgentName gives this agent a unique identity when talking to
    SolrAgentCoordinator(s). The string can be anything, just so long as it is
    unique in a society of agents all polling the same blackboard.
-->
<parameter name="AgentName" value="94026" />

<!--
    This platform includes an instance of SemiSpace blackboard (tuplespace).
    It needs a unique name for its Tracer log
-->
<parameter name="SemiSpaceName" value="SemiSpace94026" />

<!-- if it's localhost, this should work: http://localhost:8983/solr/ "http://10.1.10.80:8993/solr/" -
->
<parameter name="SolrURL" value="http://localhost:8983/solr/" /> <!-- CHANGE ME -->

<!-- number of nodes cached internally -->
<parameter name="MapCacheSize" value="1024" />

<!--
    provide a list of agents, which must be in the classpath, for booting.
    Each agent implements the org.topicquests.solr.api.IPluggableAgent interface.
    MockAgent is for testing.
    used in SolrEnvironment; they are the last to boot since they
    go right to work
    Note that the "name" field in each parameter is the name given to this
    agent for its log tracer; each name must be unique in the group,
    and unique in terms of SemiSpaceName and AgentName above
-->
<list name="Agents">
    <parameter name="Merger"
value="org.topicquests.solr.agents.merge.SolrMergeAgent" />
</list>

<!--
    These are individual IPortfolioAgents which are used by the
    "Merger" merge agent. If a merge agent is not installed in the "Agents" list
    then these will not be called
-->
<list name="PortfolioAgents">org.topicquests.solr.agents.merge.agents
    <parameter name="LabelAgent"
value="org.topicquests.solr.agents.merge.agents.LabelMergeAgent" />
    <parameter name="DetailsAgent"
value="org.topicquests.solr.agents.merge.agents.DetailsMergeAgent" />
<!--    <parameter name="TuplesAgent"

```

```

value="org.topicquests.solr.agents.merge.agents.TupleMergeAgent" /> -->
    <parameter name="WebResourceAgent"
value="org.topicquests.solr.agents.merge.agents.WebResourceMergeAgent" />
</list>

<!--
    port for the TupleSpaceConnector
    used in AgentEnvironment
-->
<parameter name="TuplespacePort" value="2930" /> <!-- CHANGE ME -->
<!--
    server for the TupleSpaceConnector
    NOTE: if this system must listen to multiple instances of
    SolrAgentCoordinator, then there may be different instances
    of this parameter with slightly different name and value
    used in AgentEnvironment
-->
<parameter name="TuplespaceServer" value="localhost" /> <!-- CHANGE ME -->

</properties>

```

ShouldBootstrap

This property is set to `Yes` if this agent is responsible for running the *bootstrapping process* to initialize the topic map. Other agents will not say `Yes`; they will say `No`; we leave bootstrapping to the merge agent since maintaining the topic map is its job. Bootstrapping is the process of adding pre-defined topics to the topic map to establish its *baseline typology*.

MergeImplementation

The merge engine is a *plug-in implementation* which means we can swap in different implementations for testing and development. The value of this property is precisely the class path and class name for the chosen merge implementation. It is important to distinguish between the *merge implementation* and the *merge agent*.

- The *merge implementation* performs the surgical tasks on the topic map which constitute a merge between topics which are deemed to require merging.
- The *merge agent* itself, is a society of agents, each of which is responsible for examining new topics, answering the question *have I seen this before* and giving reasons why a merge should be performed. These agents only make decisions; they do not perform the merge action.

AgentBehavior

Agent behaviors are defined in a Java interface class. There are two options at this time:

- `model`
- `harvest`

In fact, `harvest` fires up an instance of `HarvestBehavior` which is named by the property `AgentName`, described next.

AgentName

The name given to an agent which is booted as an `AgentBehavior`, described above.

SemiSpaceName

A `SemiSpace` is a *blackboard* agent coordinator, the same as is used in the Solr Agent Coordinator. The Agent Framework provides a local blackboard for its agents to use. This property gives that space a name

SolrURL

Agents operating in the Agent space of SolrSherlock, from time to time, need to query the topic map. This property provides the URL for forming such queries.

MapCacheSize

Since the merge agent is responsible for maintaining the topic map, this property declares an internal cache size for holding topics returned from the topic map following a query. Solr does its own caching as well. This cache is cleared of any topic which gets updated. Other agents will ignore this property.

Agents

This is a *list property* in which *agents* are declared by their class path and name. The only agent defined here is the *merge agent*.

PortfolioAgents

This property will generally be ignored by most agents. In this case, the merge agent is in play, and it requires a list of what are called *portfolio agents*. That is, the merge agent has a *portfolio* of agents, each of which performs a specific merge test. The list given here is obviously that of a very impoverished merge system; there will be many more portfolio agents declared as SolrSherlock matures

TuplespacePort

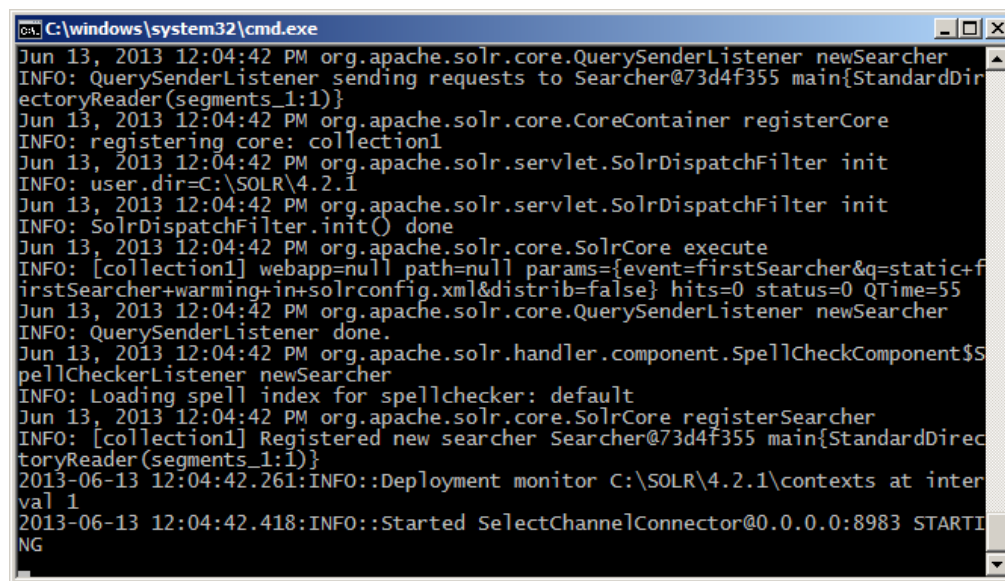
<ToDo>

TuplespaceServer

<ToDo>

Booting Solr

Booting Solr is simple: just fire up the batch file or shell script. The console (Windows version) looks like the following figure after booting.



```
C:\windows\system32\cmd.exe
Jun 13, 2013 12:04:42 PM org.apache.solr.core.QuerySenderListener newSearcher
INFO: QuerySenderListener sending requests to Searcher@73d4f355 main{StandardDir
ectoryReader(segments_1:1)}
Jun 13, 2013 12:04:42 PM org.apache.solr.core.CoreContainer registerCore
INFO: registering core: collection1
Jun 13, 2013 12:04:42 PM org.apache.solr.servlet.SolrDispatchFilter init
INFO: user.dir=C:\SOLR\4.2.1
Jun 13, 2013 12:04:42 PM org.apache.solr.servlet.SolrDispatchFilter init
INFO: SolrDispatchFilter.init() done
Jun 13, 2013 12:04:42 PM org.apache.solr.core.SolrCore execute
INFO: [collection1] webapp=null path=null params={event=firstSearcher&q=static+f
irstSearcher+warming+in+solrconfig.xml&distrib=false} hits=0 status=0 QTime=55
Jun 13, 2013 12:04:42 PM org.apache.solr.core.QuerySenderListener newSearcher
INFO: QuerySenderListener done.
Jun 13, 2013 12:04:42 PM org.apache.solr.handler.component.SpellCheckComponent$S
pellCheckerListener newSearcher
INFO: Loading spell index for spellchecker: default
Jun 13, 2013 12:04:42 PM org.apache.solr.core.SolrCore registerSearcher
INFO: [collection1] Registered new searcher Searcher@73d4f355 main{StandardDir
ectoryReader(segments_1:1)}
2013-06-13 12:04:42.261:INFO::Deployment monitor C:\SOLR\4.2.1\contexts at inter
val 1
2013-06-13 12:04:42.418:INFO::Started SelectChannelConnector@0.0.0.0:8983 STARTI
NG
```

Figure 8. Solr Server Console

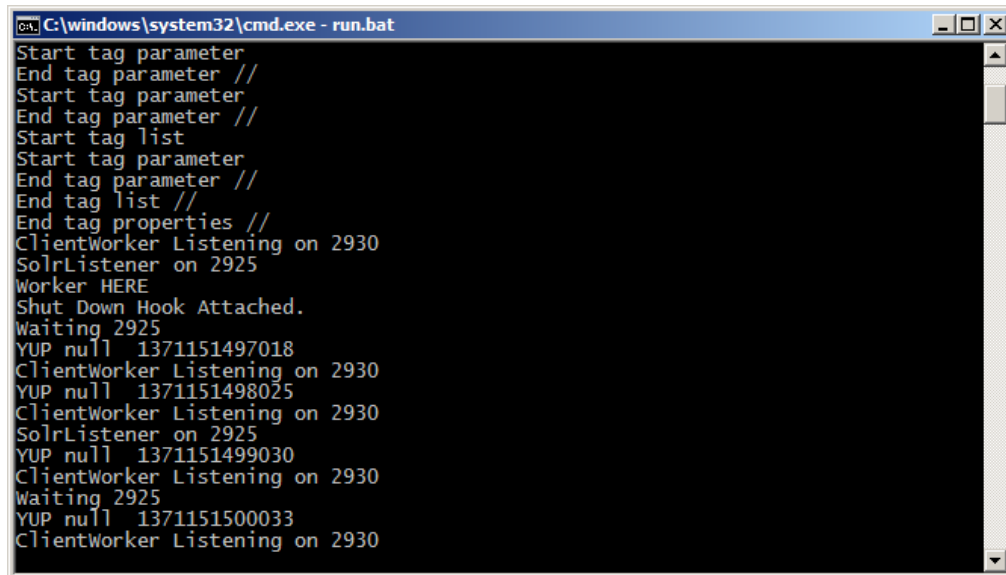
Booting Agents

In all cases, there is a sequence of booting:

1. Boot Solr First
2. Boot the Solr Agent Coordinator
3. Boot the Merge Agent
4. Boot other agents -- they may have sequences of their own, but must follow the above

Solr Agent Coordinator

Booting the agent coordinator is as simple as booting the batch file or shell script. When it starts, it immediately begins to monitor the Solr Interceptor.



```
C:\windows\system32\cmd.exe - run.bat
Start tag parameter
End tag parameter //
Start tag parameter
End tag parameter //
Start tag list
Start tag parameter
End tag parameter //
End tag list //
End tag properties //
ClientWorker Listening on 2930
SolrListener on 2925
Worker HERE
Shut Down Hook Attached.
Waiting 2925
YUP null 1371151497018
ClientWorker Listening on 2930
YUP null 1371151498025
ClientWorker Listening on 2930
SolrListener on 2925
YUP null 1371151499030
ClientWorker Listening on 2930
Waiting 2925
YUP null 1371151500033
ClientWorker Listening on 2930
```

Figure 9. Solr Agent Coordinator Console

Merge Agent

Querying the Topic Map from a Browser

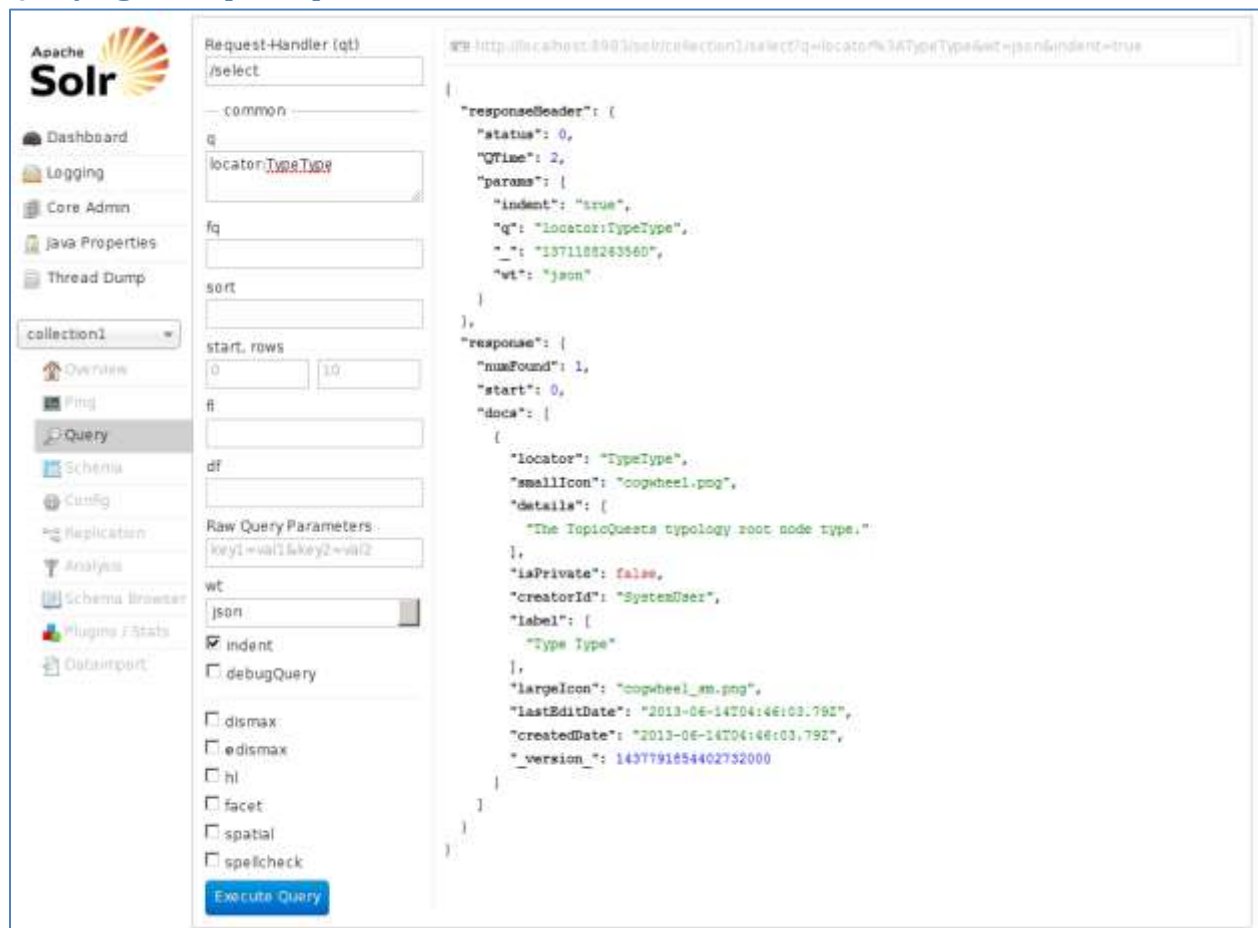


Figure 12. A Browser Query

We are able to view the topic map from a browser with the URL

```
http://localhost:8983/solr/admin.html#/collection1/query
```

as described in the Solr Tutorial. Here is a brief sketch of that process.

We are asking the topic map to show us its *root topic* which has the *unique identifier*, called a *locator*, of *TypeType*. So, we posed this query:

```
locator:TypeType
```

and selected JSON as the output. The result is this:

```
{
  "locator": "TypeType",
  "smallIcon": "cogwheel.png",
  "details": [
    "The TopicQuests typology root node type."
  ],
}
```

```
    "isPrivate": false,  
    "creatorId": "SystemUser",  
    "label": [  
        "Type Type"  
    ],  
    "largeIcon": "cogwheel_sm.png",  
    "lastEditDate": "2013-06-14T04:46:03.79Z",  
    "createdDate": "2013-06-14T04:46:03.79Z",  
    "_version_": 1437791854402732000  
}
```

From that JSON string, one can examine what a topic *document* looks like in Solr. In fact, the document is very much a classic *frame structure*, an identified object with a collection of *key/value pairs*. In this case, the *keys* are specific *fields* declared in our topic map's `schema.xml` file. There are two fields required by Solr; the rest are optional, but without them, we would not have much of a topic map. The two fields are:

- **locator** required by the system and declared by the topic map code
- **_version_** required by the system and maintained solely by Solr