APACHE SOLR COOKBOOK

Hot Recipes for the Apache Solr Platform

VEERAMANI KALYANASUNDARAM

Java Code Geeks
JAVA 2 JAVA DEVELOPERS RESOURCE CENTER

Apache Solr Cookbook

Contents

Preface

Solr (pronounced "solar") is an open source enterprise search platform, written in Java, from the Apache Lucene project. Its major features include full-text search, hit highlighting, faceted search, real-time indexing, dynamic clustering, database integration, NoSQL features and rich document (e.g., Word, PDF) handling. Providing distributed search and index replication, Solr is designed for scalability and fault tolerance. Solr is the second-most popular enterprise search engine after Elasticsearch. (Source: https://en.wikipedia.org/wiki/Apache_Solr)

Solr is highly reliable, scalable and fault tolerant, providing distributed indexing, replication and load-balanced querying, automated failover and recovery, centralized configuration and more. Solr powers the search and navigation features of many of the world's largest internet sites. (Source: https://lucene.apache.org/solr/)

In this ebook, we provide a compilation of Apache Solr tutorials that will help you kick-start your own programming projects. We cover a wide range of topics, from basic usage and installation, to query syntax and synonyms search. With our straightforward tutorials, you will be able to get your own projects up and running in minimum time.

About the Author

Veera is a Software Architect working in telecom domain with rich experience in Java Middleware Technologies. He is a OOAD practitioner and interested in Performance Engineering.

Chapter 1

Apache Solr Tutorial for Beginners

In this example of Apache Solr Tutorial for Beginners, we will discuss about how to install the latest version of Apache Solr and show you how to configure it. Also we will show you how to perform the index using a sample data file. Apache Solr supports indexing from different source formats including various databases, PDF files, XML files, CSV files etc. For this example we will look into how to index data from a CSV file.

Our preferred environment for this example is Windows. Before you begin the Solr installation make sure you have JDK installed and `Java_Home` is set appropriately.

1.1 Why Apache Solr

Apache Solr is a powerful search server, which supports REST like API. Solr is powered by Lucene which enables powerful matching capabilities like phrases, wildcards, joins, grouping and many more across various data types. It is highly optimized for high traffic using Apache Zookeeper. Apache Solr comes with a wide set of features and we have listed a subset of high impact features.

- Advanced Full-Text search capabilities.
- Standards based on Open Interfaces - XML, JSON and Http.
- Highly scalable and fault tolerant.
- Supports both Schema and Schemaless configuration.
- Faceted Search and Filtering.
- Support major languages like English, German, Chinese, Japanese, French and many more
- Rich Document Parsing.

1.2 Installing Apache Solr

To begin with lets download the latest version of Apache Solr from the following location:

`https://lucene.apache.org/solr/downloads.html`

As of this writing, the stable version available is 5.0.0. Apache Solr has gone through various changes from 4.x.x to 5.0.0, so if you have different version of Solr you need to download the 5.x.x. version to follow this example.

Once the Solr zip file is downloaded unzip it into a folder. The extracted folder will look like the below.

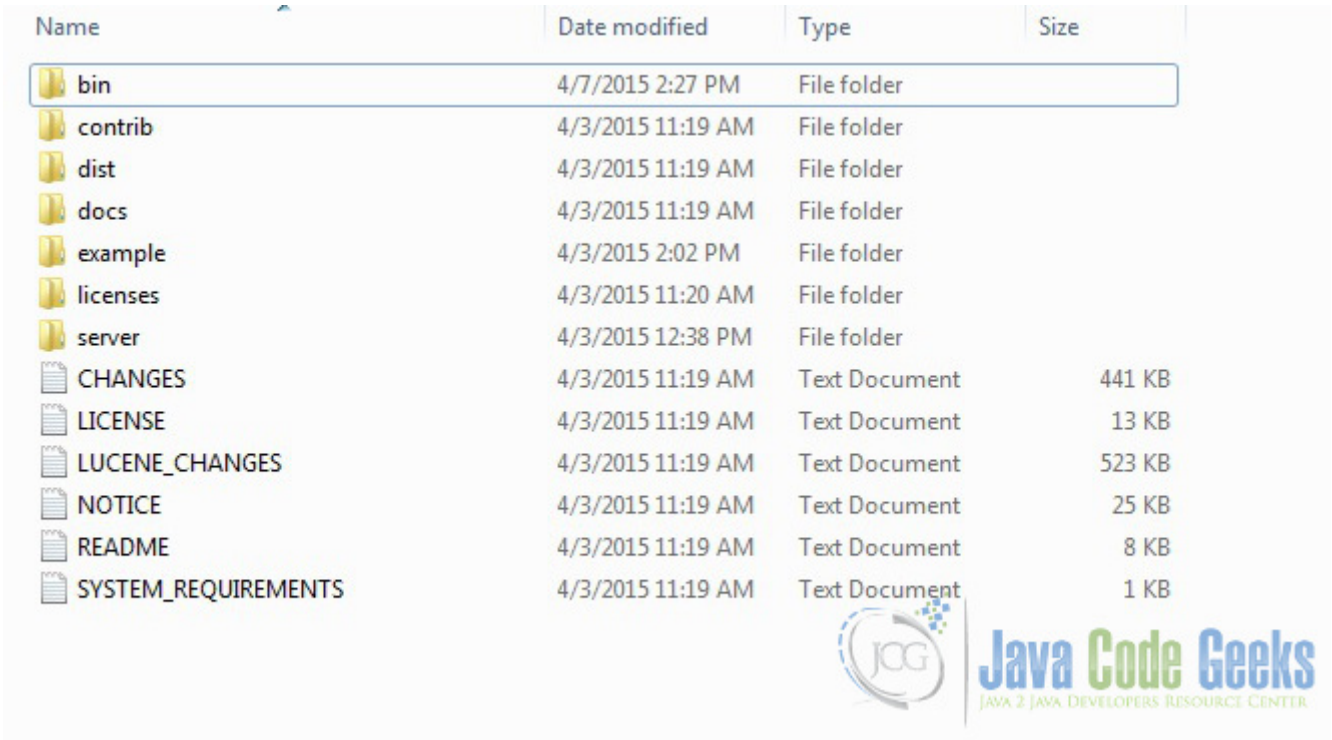

Figure 1.1: Solr folders

The `bin` folder contains the scripts to start and stop the server. The `example` folder contains few example files. We will be using one of them to demonstrate how Solr indexes the data. The `server` folder contains the `logs` folder where all the Solr logs are written. It will be helpful to check the logs for any error during indexing. The `solr` folder under server holds different collection or core. The configuration and data for each of the core/ collection are stored in the respective core/ collection folder.

Apache Solr comes with an inbuilt Jetty server. But before we start the solr instance we must validate the JAVA_HOME is set on the machine.

We can start the server using the command line script. Lets go to the bin directory from the command prompt and issue the following command

```
solr start
```

This will start the Solr server under the default port 8983.

We can now open the following URL in the browser and validate that our Solr instance is running. The specifics of solr admin tool is beyond the scope of the example.

```
https://localhost:8983/solr/
```

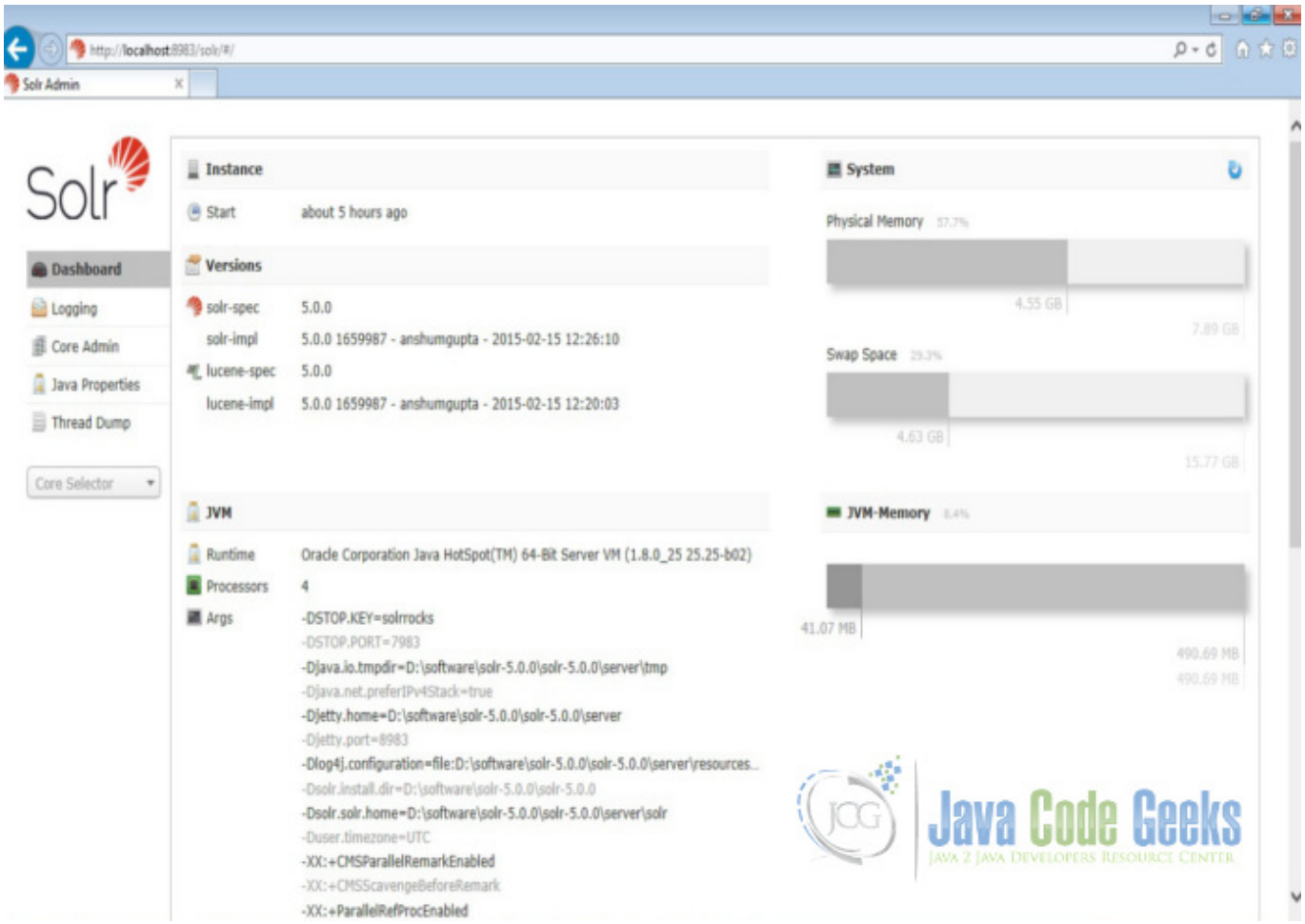

Figure 1.2: Solr admin console

1.3 Configuring Apache Solr

In this section, we will show you how to configure the core/collection for a solr instance and how to define the fields. Apache Solr ships with an option called Schemaless mode. This option allow users to construct effective schema without manually editing the schema file. But for this example we will use the Schema configuration for understanding the internals of the Solr.

1.3.1 Creating a Core

When the Solr server is started in Standalone mode the configuration is called core and when it is started in SolrCloud mode the configuration is called Collection. In this example we will discuss about the standalone server and core. We will park the SolrCloud discussion for later time.

First, we need to create a Core for indexing the data. The Solr create command has the following options:

- -c <name> - Name of the core or collection to create (required).
- -d <confdir> - The configuration directory, useful in the SolrCloud mode.
- -n <configName> - The configuration name. This defaults to the same name as the core or collection.
- -p - Port of a local Solr instance to send the create command to; by default the script tries to detect the port by looking for running Solr instances.

- -s <shards> - Number of shards to split a collection into, default is 1.
- -rf <replicas> - Number of copies of each document in the collection. The default is 1.

In this example we will use the -c parameter for core name and -d parameter for the configuration directory. For all other parameters we make use of default settings.

Now navigate the `solr-5.0.0bin` folder in the command window and issue the following command.

```
solr create -c jcg -d basic_configs
```

We can see the following output in the command window.

```
Creating new core 'jcg' using command:
https://localhost:8983/solr/admin/cores?action=CREATE&name=jcg&instanceDir=jcg

{
 "responseHeader":{
 "status":0,
 "QTime":663},
 "core":"jcg"}
```

Now we navigate to the following URL and we can see jcg core being populated in the core selector. You can also see the statistics of the core.

```
https://localhost:8983/solr
```

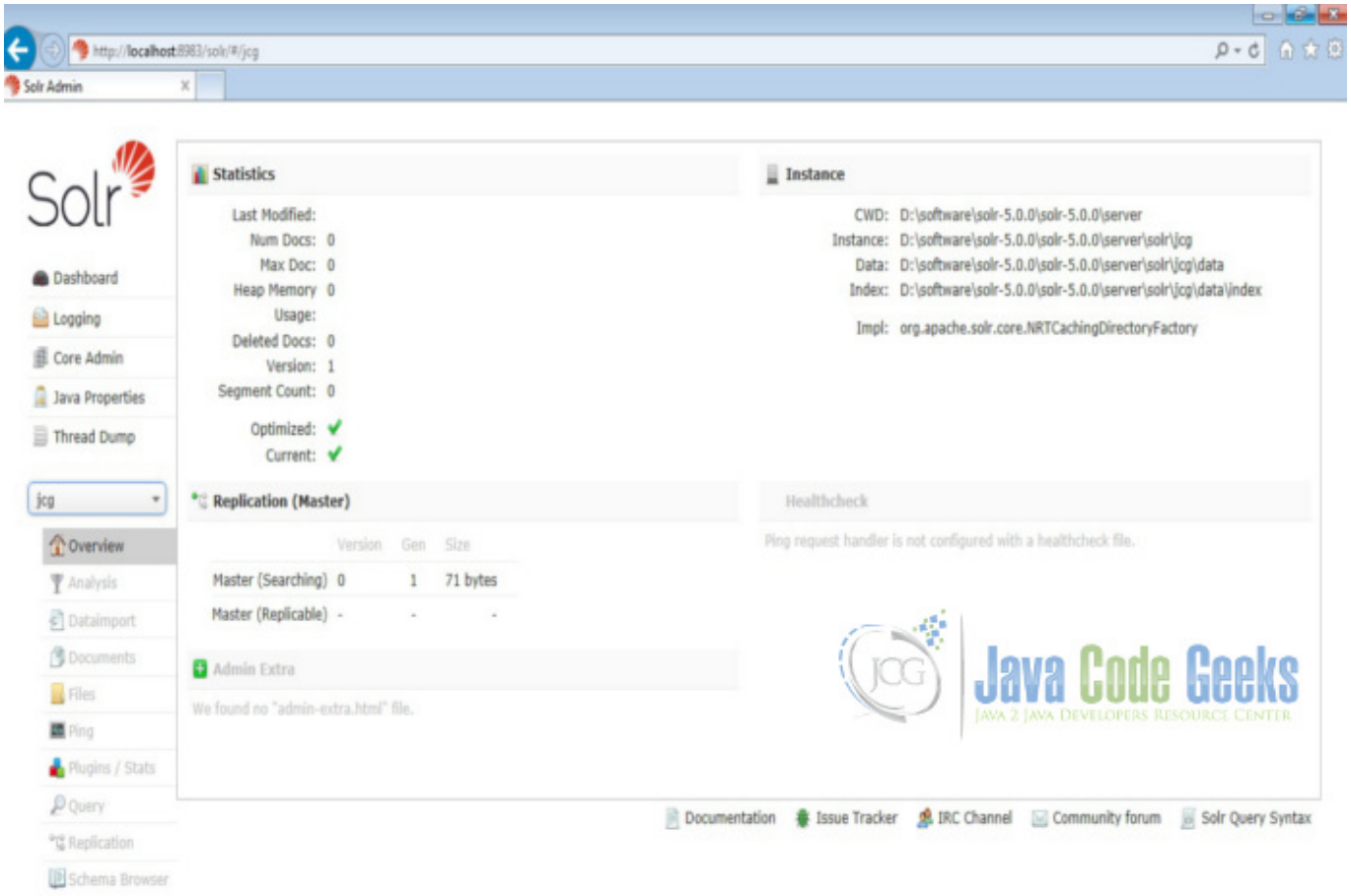

Figure 1.3: Solr JCG core

1.3.2 Modify the schema.xml file

We need to modify the `schema.xml` file under the folder `serversolrjcgconf` to include the fields. We will use one of the example file "books.csv" shipped along with Solr installation for indexing. The file is located under the folder `solr-5.0.0exampleexampledocs`

Now we navigate to the folder `serversolr` directory. You will see a folder called `jcg` created. The sub-folders namely `conf` and `data` have the core's configuration and indexed data respectively.

Now edit the `schema.xml` file in the `serversolrjcgconf` folder and add the following contents after the uniqueKey element.

schema.xml

```
<uniqueKey>id</uniqueKey>
 <!-- Fields added for books.csv load-->
 <field name="cat" type="text_general" indexed="true" stored="true"/>
 <field name="name" type="text_general" indexed="true" stored="true"/>
 <field name="price" type="tdouble" indexed="true" stored="true"/>
 <field name="inStock" type="boolean" indexed="true" stored="true"/>
 <field name="author" type="text_general" indexed="true" stored="true"/>
```

We have set the attribute `indexed` to true. This specifies the field is used for indexing and the record can be retrieved using the index. Setting the value to false will make the field only stored but can't be queried with.

Also note we have another attribute called `stored` and set it to true. This specifies the field is stored and can be returned in the output. Setting this field to false will make the field only indexed and can't be retrieved in output.

We have assigned the type for the fields present in the "books.csv" file here. The first field in the CSV file "id" is automatically taken care by the uniqueKey element of schema.xml file for indexing. If you note, we have missed the fields series_t, sequence_i and genre_s without making any entry. But, when we perform indexing all these fields are also indexed without any issue. If you wonder how that happens take a closer look at the dynamicField section in schema.xml file.

schema.xml

```
<dynamicField name="*_i" type="int" indexed="true" stored="true"/>
 <dynamicField name="*_is" type="ints" indexed="true" stored="true"/>
 <dynamicField name="*_s" type="string" indexed="true" stored="true" />
 <dynamicField name="*_ss" type="strings" indexed="true" stored="true"/>
 <dynamicField name="*_l" type="long" indexed="true" stored="true"/>
 <dynamicField name="*_ls" type="longs" indexed="true" stored="true"/>
 <dynamicField name="*_t" type="text_general" indexed="true" stored="true"/>
 <dynamicField name="*_txt" type="text_general" indexed="true" stored="true"/>
 <dynamicField name="*_b" type="boolean" indexed="true" stored="true"/>
 <dynamicField name="*_bs" type="booleans" indexed="true" stored="true"/>
 <dynamicField name="*_f" type="float" indexed="true" stored="true"/>
 <dynamicField name="*_fs" type="floats" indexed="true" stored="true"/>
 <dynamicField name="*_d" type="double" indexed="true" stored="true"/>
 <dynamicField name="*_ds" type="doubles" indexed="true" stored="true"/>
```

Since we have modified the configuration we have to stop and start the server. To do so, we need to issue the following command from bin directory through command line.

```
solr stop -all
```

The server will be stopped now. Now to start the server issue the following command from bin directory through command line.

```
solr start
```

1.4 Indexing the Data

Apache Solr comes with a Standalone Java program called the SimplePostTool. This program is packaged into JAR and available with the installation under the folder `exampleexampledocs`.

Now we navigate to the `exampleexampledocs` folder in the command prompt and type the following command. You will see a bunch of options to use the tool.

```
java -jar post.jar -h
```

The usage format in general is as follows

```
Usage:java [SystemProperties] -jar post.jar [-h|-] [<file|folder|url|arg> [<file|folde
r|url|arg>...]]
```

As we said earlier, we will index the data present in the "books.csv" file shipped with Solr installation. We will navigate to the `solr-5.0.0exampleexampledocs` in the command prompt and issue the following command.

```
java -Dtype=text/csv -Durl=https://localhost:8983/solr/jcg/update -jar post.jar books.
csv
```

The SystemProperties used here are:

- -Dtype - the type of the data file.
- -Durl - URL for the jcg core.

The file "books.csv" will now be indexed and the command prompt will display the following output.

```
SimplePostTool version 5.0.0
Posting files to [base] url https://localhost:8983/solr/jcg/update using content-
type text/csv...
POSTing file books.csv to [base]
1 files indexed.
COMMITting Solr index changes to https://localhost:8983/solr/jcg/update...
Time spent: 0:00:00.647
```

Now we navigate to the following URL and select the core jcg.

```
https://localhost:8983/solr
```

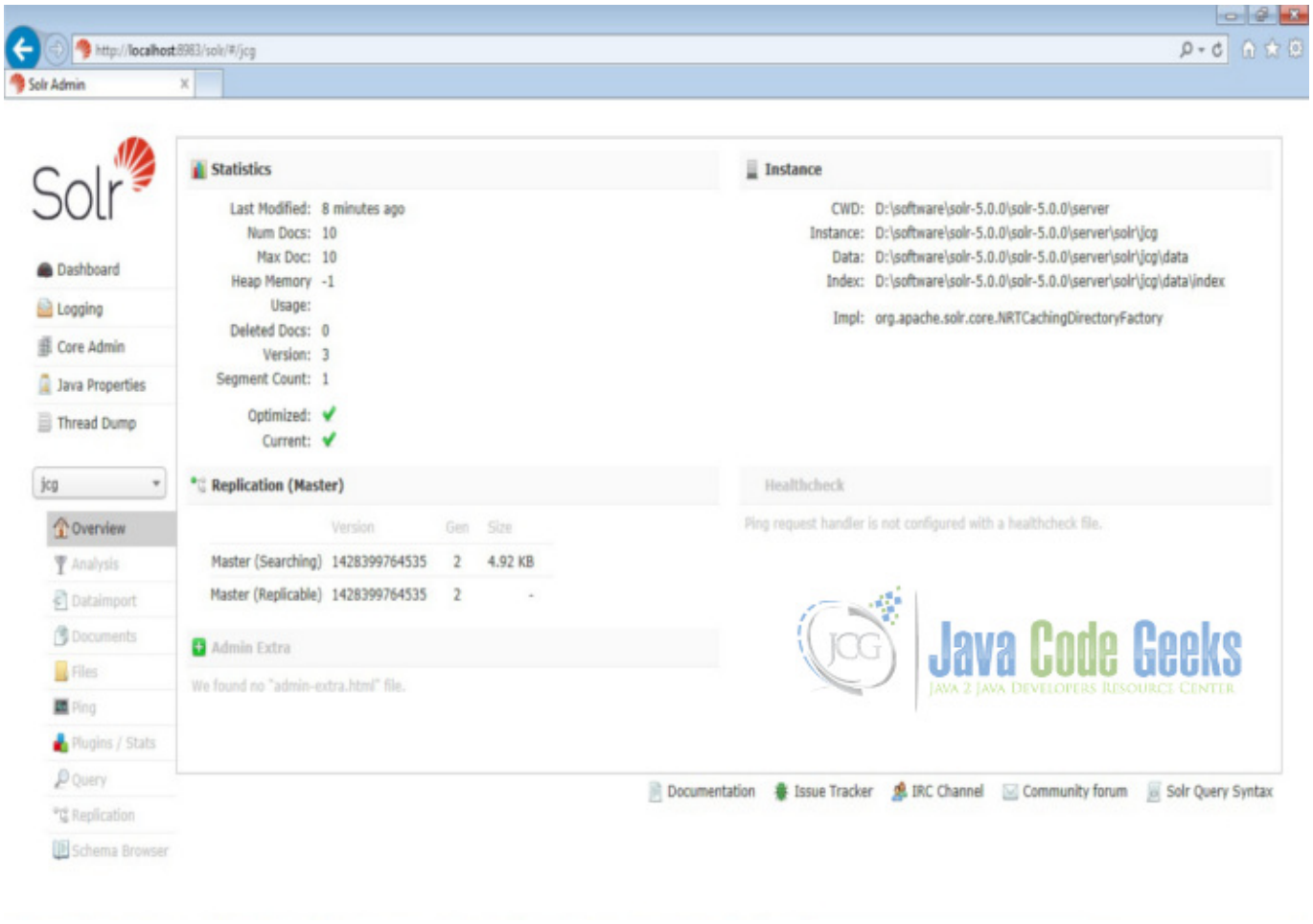

Figure 1.4: Solr JCG core data

Take a closer look at the statistics section, the Num Docs parameter will show the count of the rows indexed.

1.5 Access the Indexed documents

Apache Solr provides a REST based API to access the data and also provides different parameters to retrieve the data. We will show you few scenario based queries.

1.5.1 Search by name

We will retrieve the details of the book by its name. To do so, we will use the following syntax. The parameter "q" in the URL is the query event.

Open the following URL in a browser.

```
https://localhost:8983/solr/jcg/select?q=name:"A Clash of Kings"
```

The output will be as shown below.

Figure 1.5: Solr by name

1.5.2 Search by starting letter

Now we will show you how to search for the record if we know only the starting letter or word and don't remember the full title. We can use the following query to retrieve the result.

```
https://localhost:8983/solr/jcg/select?q=name:"A"
```

The output will list all the books staring with letter A.

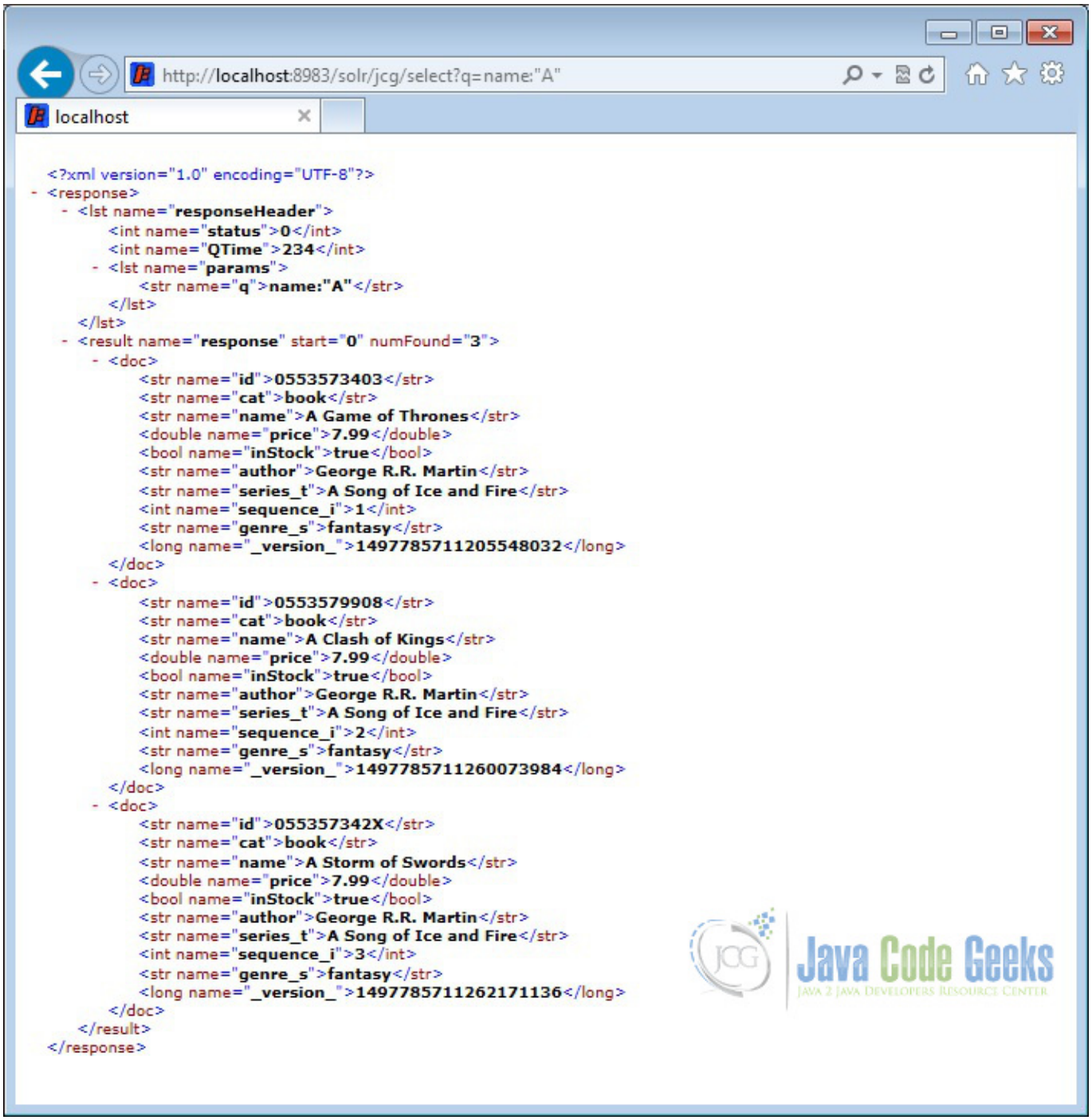

Figure 1.6: Solr starting letter

1.5.3 Search using wildcard

Solr supports wildcard search. We will show in the following query how to retrieve all the books which contains the word "of" in the name.

```
https://localhost:8983/solr/jcg/select?q=name:"*of"
```

The output will list all the books with the word "of" present in it.

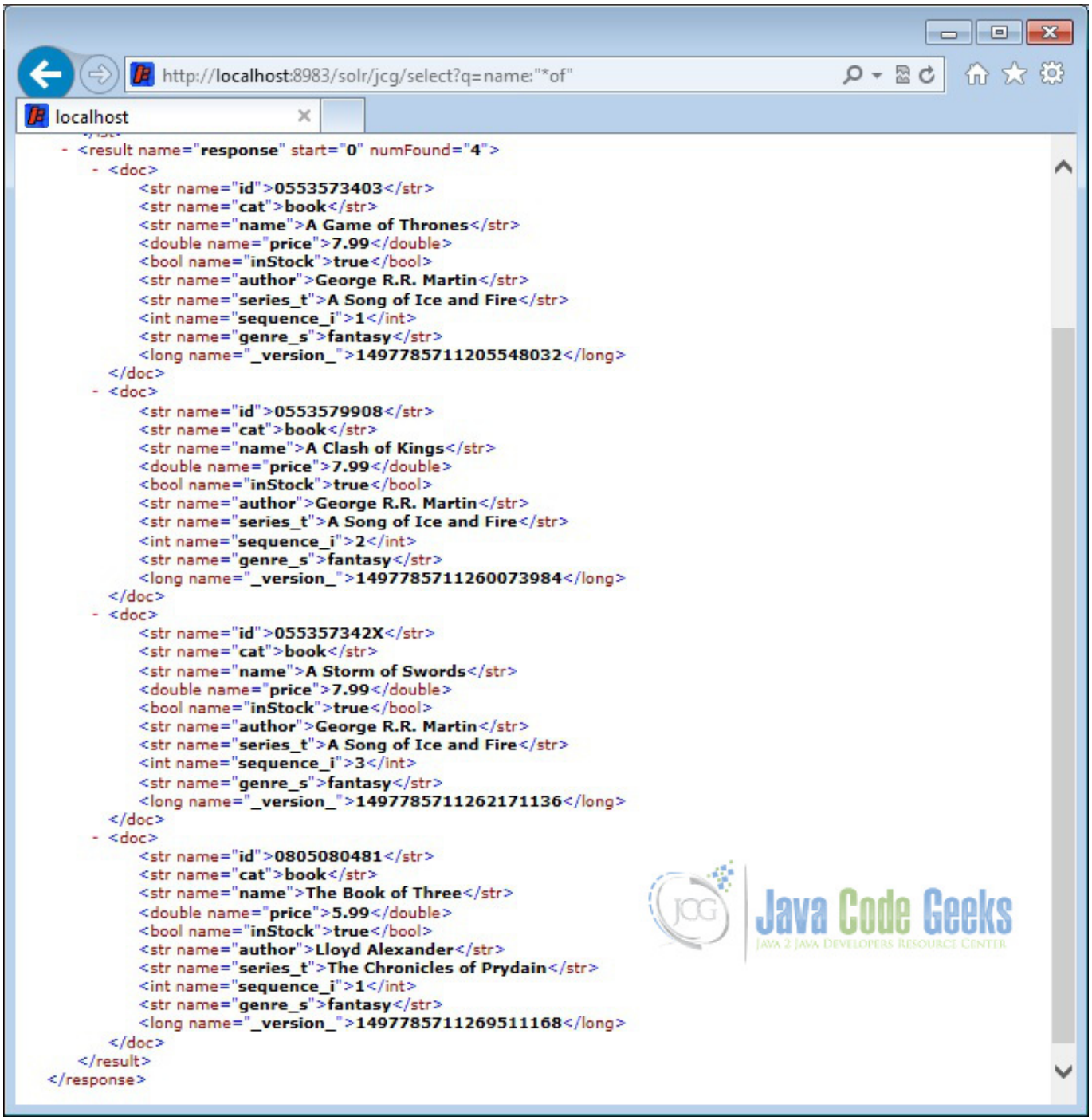

Figure 1.7: Solr wildcard search

1.5.4 Search using a condition

Solr supports conditional search. It provides "fq" parameter using which, we can set condition to our query. We will show you how to find books which are priced less than $6 in the following query.

```
https://localhost:8983/solr/jcg/select?q=*&fq=price:[0 TO 6]
```

The output will list only the books which are less than $6.

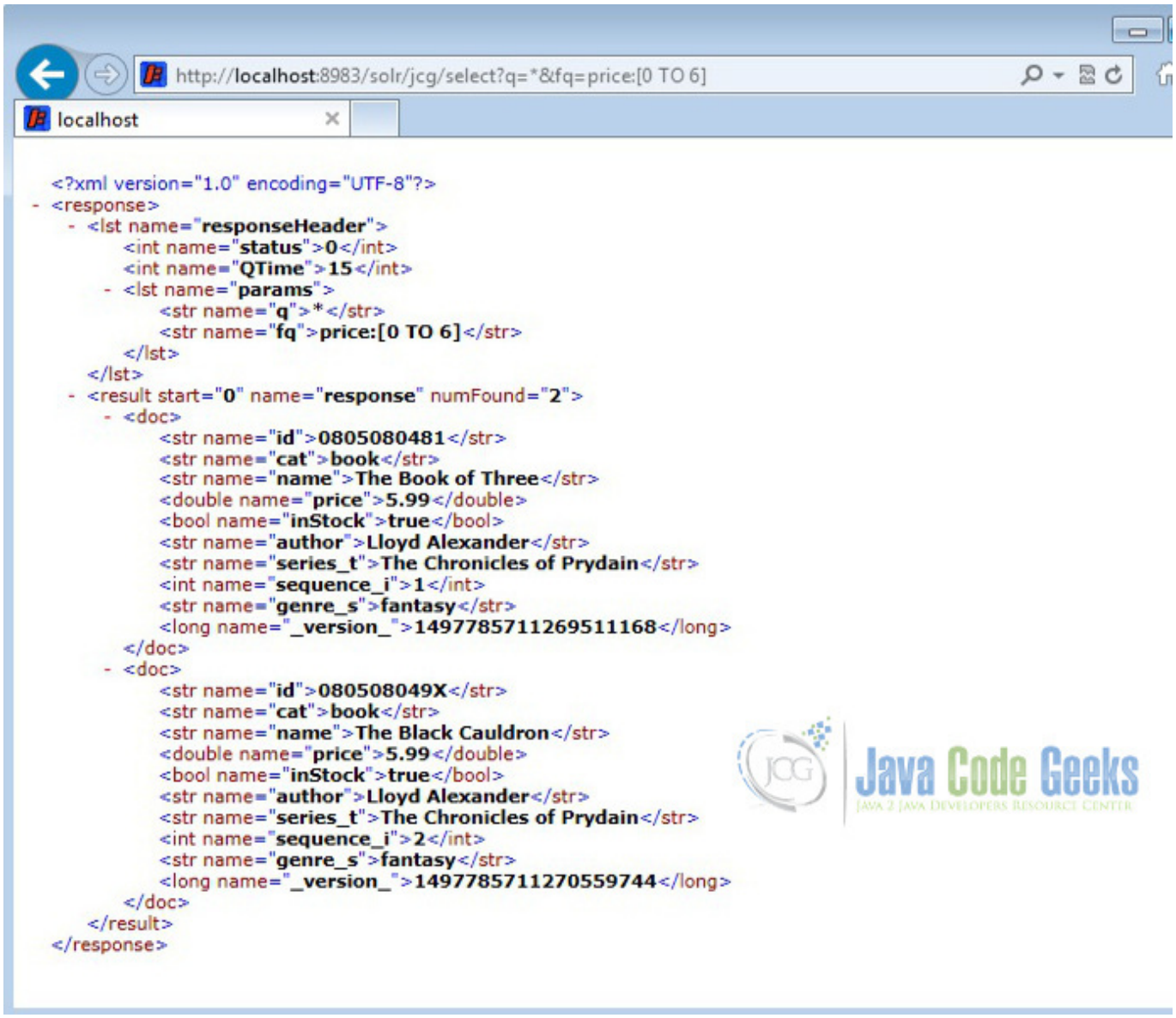

Figure 1.8: Solr search condition

1.6 Solr Client API's

There are different client API's available to connect to the Solr server. We have listed a few widely used Solr client API's.

- SolRuby - To connect from Ruby
- SolPHP - To connect from PHP
- PySolr - To connect from Python
- SolPerl - To connect from Perl
- SolrJ - To connect from Java
- SolrSharp - To connect from C#

Also Solr provides the REST based API which can be directly consumed using the JavaScript.

1.7 Download the Schema file

This was a tutorial on Apache Solr for beginners.

Download

You can download the schema file here : **Solr schema file**

Chapter 2

How to Install Solr on Ubuntu

In this example of "how to install Solr on Ubuntu" we will discuss about how to download and install Solr in Ubuntu operating system. Ubuntu desktop operating system powers millions of PCs and laptops around the world. So this example is dedicated to users who are on Ubuntu and want to install Solr on Ubuntu.

Along with Solr installation, we will also show you how to create a Solr core and index an example file shipped along with Solr. Our preferred environment for this example is Ubuntu 14.x and solr-5.x. Before you begin the Solr installation make sure you have JDK installed and Java_Home is set appropriately.

2.1 Install Apache Solr

To begin with, lets download the latest version of Apache Solr from the following location:

`https://www.eu.apache.org/dist/lucene/solr/5.3.1/`

File: `solr-5.3.1.tgz`

Once the file is downloaded, create a directory called solr under /opt and move the downloaded file. Now navigate to the directory `/opt/solr` and unzip the file using the following command.

```
sudo tar -xvf solr-5.3.1.tgz
```

The Solr commands has to be executed from the bin directory, so navigate to the following path.

`/opt/solr/solr-5.3.1/bin`

The extracted directory will look like the below.

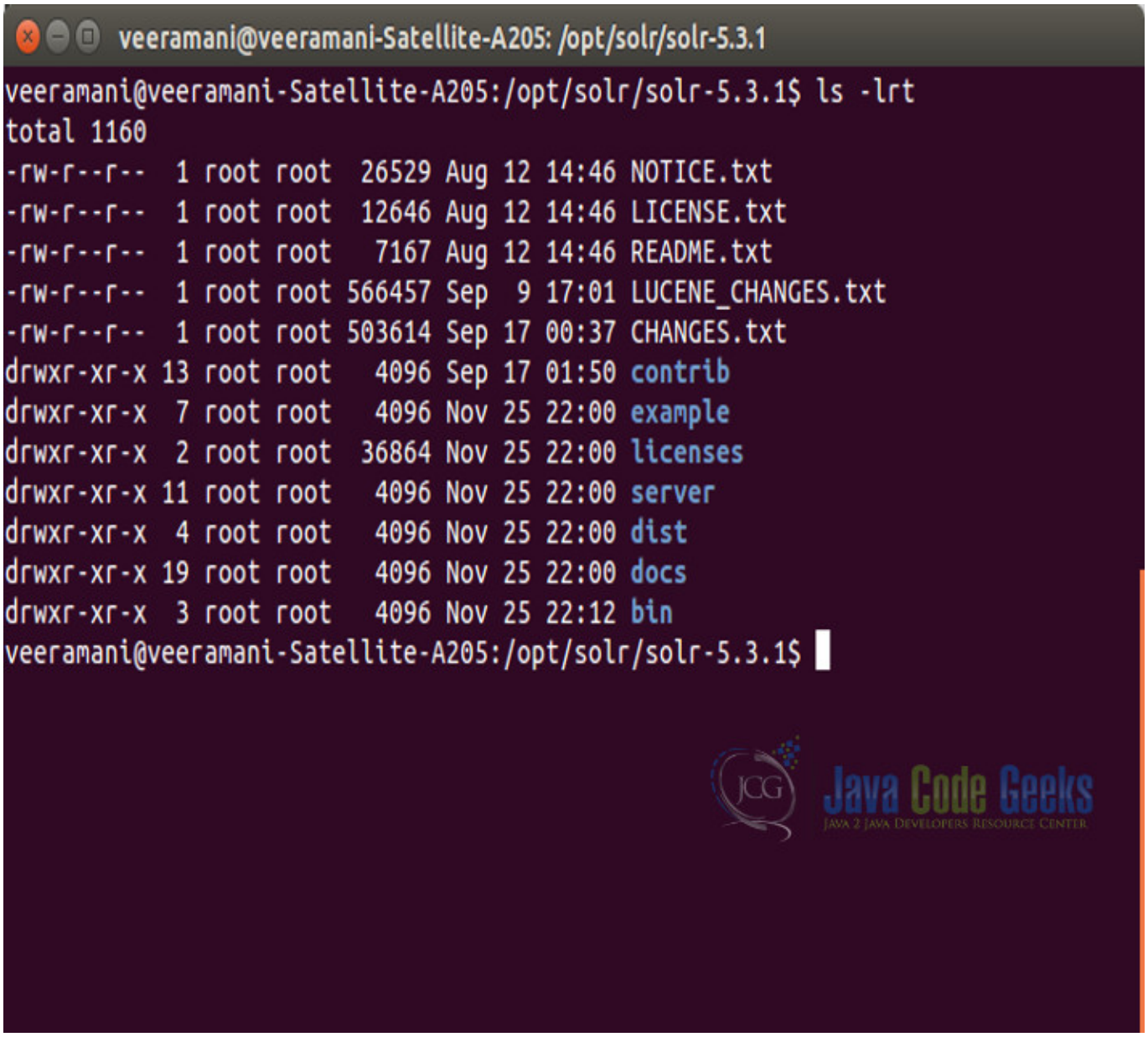

Figure 2.1: Solr Ubuntu folders

The `bin` folder contains the scripts to start and stop the server. The `example` folder contains few example files. We will be using one of them to demonstrate how Solr indexes the data. The `server` folder contains the `logs` folder where all the Solr logs are written. It will be helpful to check the logs for any error during indexing. The `solr` folder under server holds different collection or core. The configuration and data for each of the core/ collection are stored in the respective core/ collection folder.

Apache Solr comes with an inbuilt Jetty server. But before we start the solr instance we must validate the JAVA_HOME is set on the machine.

Now use the following command to start the Solr server.

```
sudo ./solr start
```

This will start the Solr server under the default port 8983. We can now open the following URL in the browser and validate that our Solr instance is running.

```
https://localhost:8983/solr/#/
```

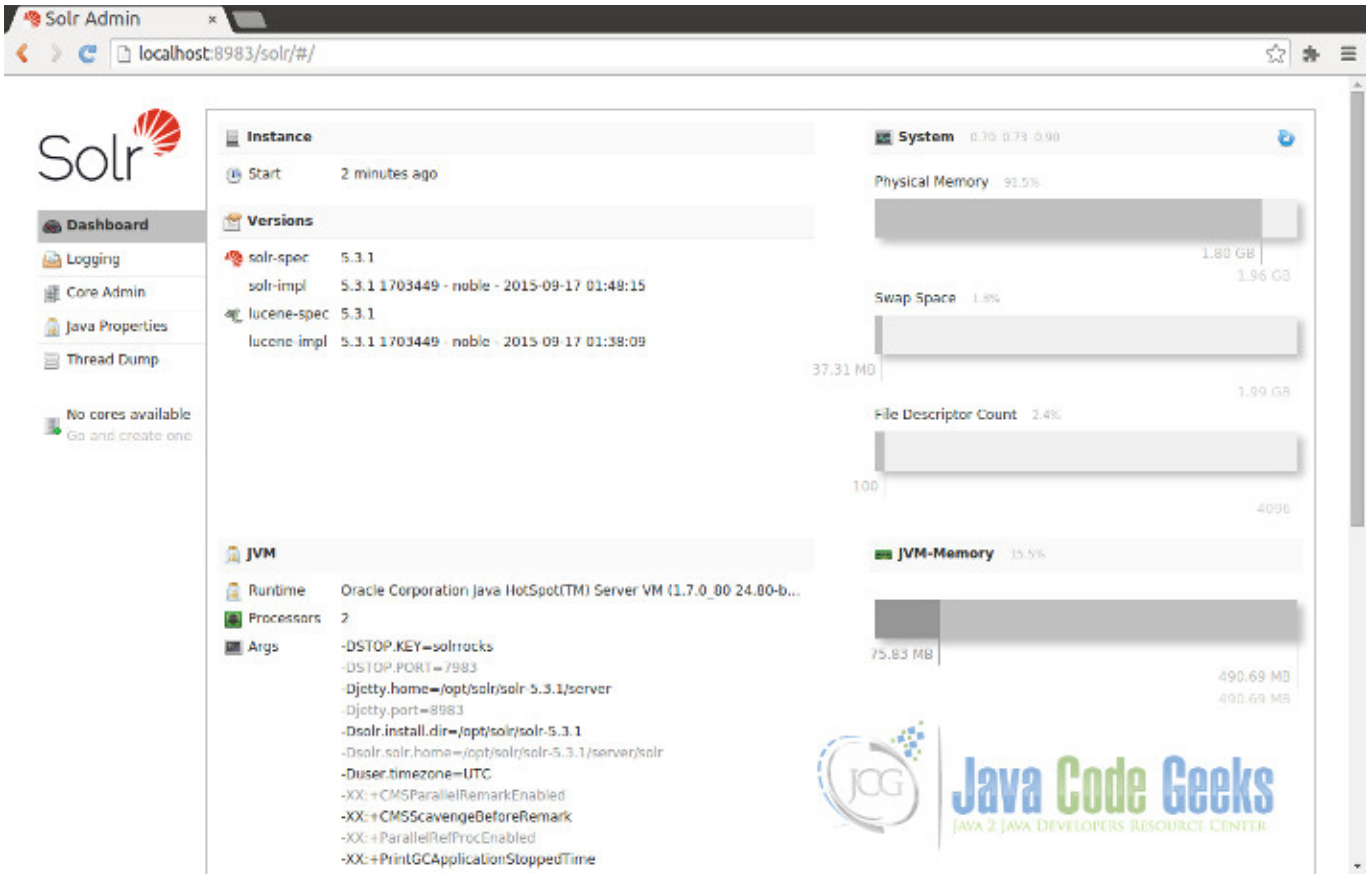

Figure 2.2: Solr Ubuntu Console

2.2 Configure Apache Solr

When the Solr server is started in Standalone mode, the configuration is called core and when it is started in SolrCloud mode, the configuration is called Collection. In this example we will discuss about the standalone server and core. We will park the SolrCloud discussion for later time.

First, we need to create a Core for indexing the data. The Solr create command has the following options:

- **-c <name>** - Name of the core or collection to create (required).
- **-d <confdir>** - The configuration directory, useful in the SolrCloud mode.
- **-n <configName>** - The configuration name. This defaults to the same name as the core or collection.
- **-p <port>** - Port of a local Solr instance to send the create command to; by default the script tries to detect the port by looking for running Solr instances.
- **-s <shards>** - Number of shards to split a collection into, default is 1.
- **-rf <replicas>** - Number of copies of each document in the collection. The default is 1.

In this example we will use the -c parameter for core name and -d parameter for the configuration directory. For all other parameters we make use of default settings.

Now navigate the `solr-5.3.1/bin` directory and issue the following command

```
sudo ./solr create -c jcg -d basic_configs
```

We can see the following output in the command window.

```
Setup new core instance directory:
/opt/solr/solr-5.3.1/server/solr/jcg
Creating new core 'jcg' using command:
https://localhost:8983/solr/admin/cores?action=CREATE&name=jcg&instanceDir=jcg

{
"responseHeader":{
"status":0,
"QTime":5862},
"core":"jcg"}
```

Now edit the schema.xml file in the `/server/solr/jcg/conf` folder and add the following contents after the uniqueKey element.

schema.xml

```
<uniqueKey>id</uniqueKey>
<!-- Fields added for books.csv load-->
<field name="cat" type="text_general" indexed="true" stored="true"/>
<field name="name" type="text_general" indexed="true" stored="true"/>
<field name="price" type="tdouble" indexed="true" stored="true"/>
<field name="inStock" type="boolean" indexed="true" stored="true"/>
<field name="author" type="text_general" indexed="true" stored="true"/>
```

Since we have modified the configuration, we have to stop and start the server. To do so, we need to issue the following command from bin directory through command line:

```
sudo ./solr stop -all
```

The server will be stopped now. Now to start the server issue the following command from bin directory through command line:

```
sudo ./solr start
```

2.3 Indexing the Data

Apache Solr comes with a Standalone Java program called the SimplePostTool. This program is packaged into JAR and available with the installation under the folder example/exampledocs.

Now we navigate to the /example/exampledocs folder in the command prompt and type the following command. You will see a bunch of options to use the tool.

```
java -jar post.jar -h
```

The usage format in general is as follows:

```
Usage:java [SystemProperties] -jar post.jar [-h|-] [<file|folder|url|arg> [<file|folde
r|url|arg>...]]
```

As we said earlier, we will index the data present in the "books.csv" file shipped with Solr installation. We will navigate to the /example/exampledocs in the command prompt and issue the following command.

```
java -Dtype=text/csv -Durl=https://localhost:8983/solr/jcg/update -jar post.jar books.
csv
```

The SystemProperties used here are:

- -Dtype - the type of the data file.
- -Durl - URL for the jcg core.

```
SimplePostTool version 5.0.0
Posting files to [base] url https://localhost:8983/solr/jcg/update using content-type text/ ↩
    csv...
POSTing file books.csv to [base]
1 files indexed.
COMMITting Solr index changes to https://localhost:8983/solr/jcg/update...
Time spent: 0:00:01.149
```

Now, the data from the example file is indexed and stored. Let's open the following URL. We can see the number of documents matching the data count in the example file.

```
https://localhost:8983/solr/#/jcg
```

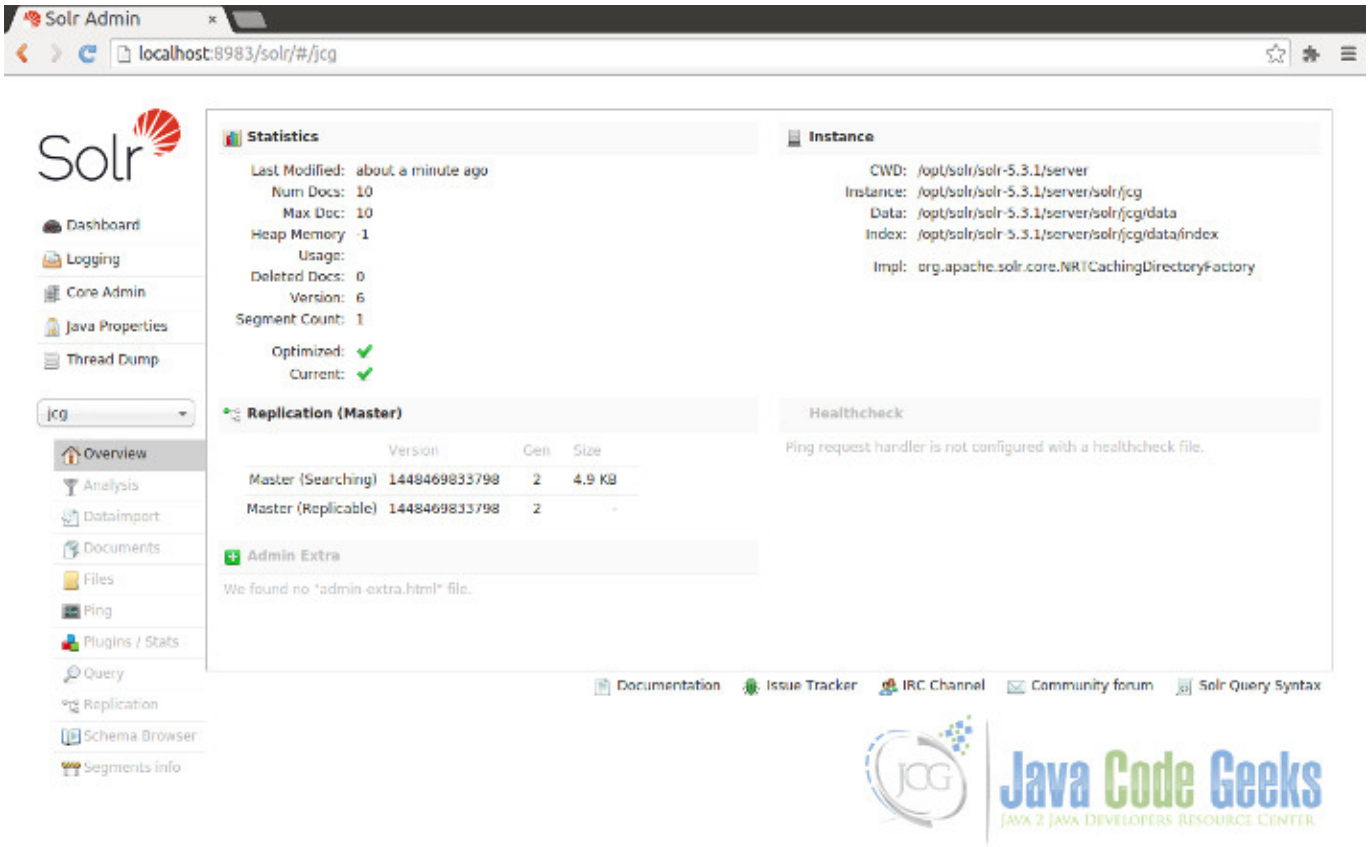

Figure 2.3: Solr Ubuntu Data

2.4 Download the Schema file

Download

You can download the schema file used in this example here: **schema.xml**

Chapter 3

Solr query syntax examples

In this example of Solr query syntax we will discuss about different query formats in Solr. For our discussion, we will be using one of the collection example (techproducts) that comes along with Solr Installation. We will show you, how to use the REST based API's exposed by Solr and show you how to use various querying parameters .

Our preferred environment for this example is Windows. Before you begin the Solr installation make sure you have JDK installed and `Java_Home` is set appropriately.

3.1 Installing Apache Solr

To begin with, lets download the latest version of Apache Solr from the following location:

`https://lucene.apache.org/solr/downloads.html`

As of this writing, the stable version available is 5.0.0. Apache Solr has gone through various changes from 4.x.x to 5.0.0, so if you have different version of Solr you need to download the 5.x.x. version to follow this example.

Once the Solr zip file is downloaded unzip it into a folder. The extracted folder will look like the below.

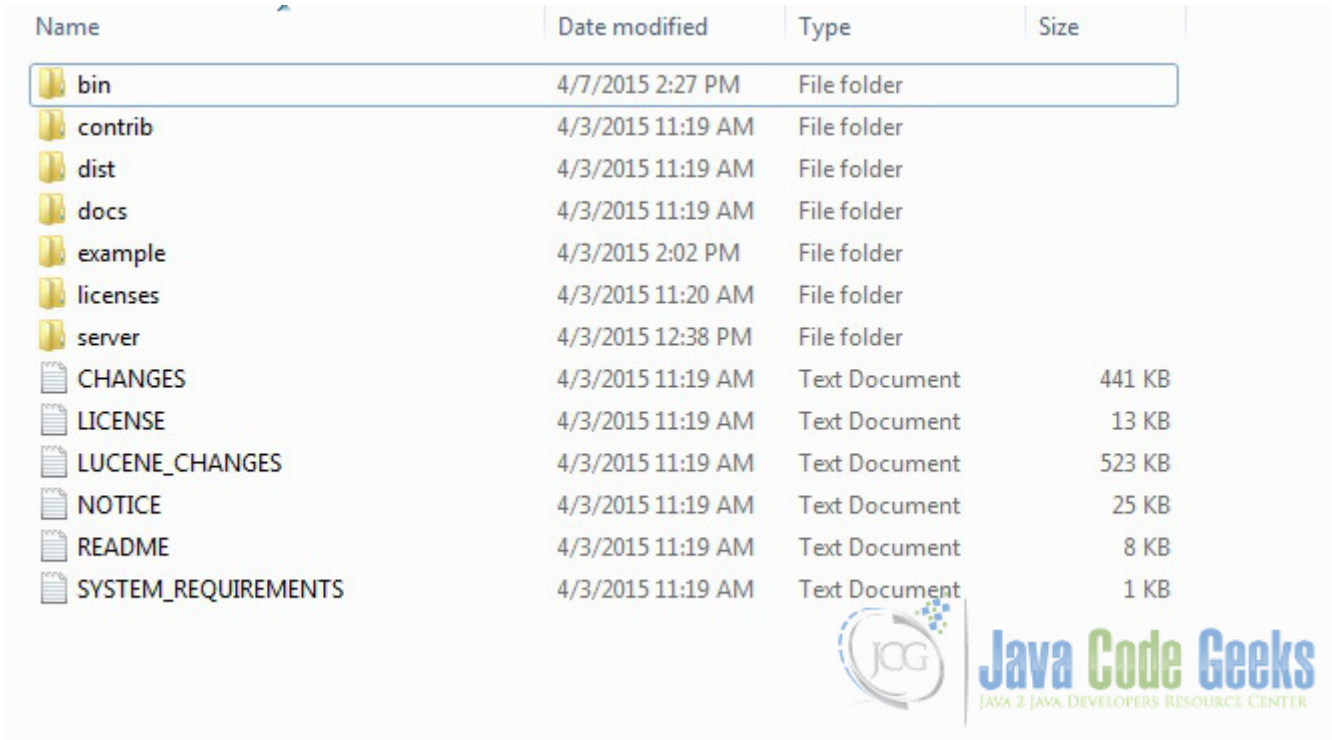

Figure 3.1: Solr folders

The `bin` folder contains the scripts to start and stop the server. The `example` folder contains few example files. We will be using one of them to demonstrate how Solr indexes the data. The `server` folder contains the `logs` folder where all the Solr logs are written. It will be helpful to check the logs for any error during indexing. The `solr` folder under server holds different collection or core. The configuration and data for each of the core/ collection are stored in the respective core/ collection folder.

Apache Solr comes with an inbuilt Jetty server. But before we start the solr instance we must validate the JAVA_HOME is set on the machine.

3.2 Start Solr Server

Solr provides few useful collection example to learn about the key features. We will use the techproducts collection bundled with Solr for our discussion. To start the Solr server with the techproducts collection let's open a command prompt, navigate to bin folder and issue the following syntax.

```
solr -e techproducts
```

This will start the Solr server under the default port 8983.

We can now open the following URL in the browser and validate that our Solr instance is running. You can also notice the collection techproducts being populated.

```
https://localhost:8983/solr/
```

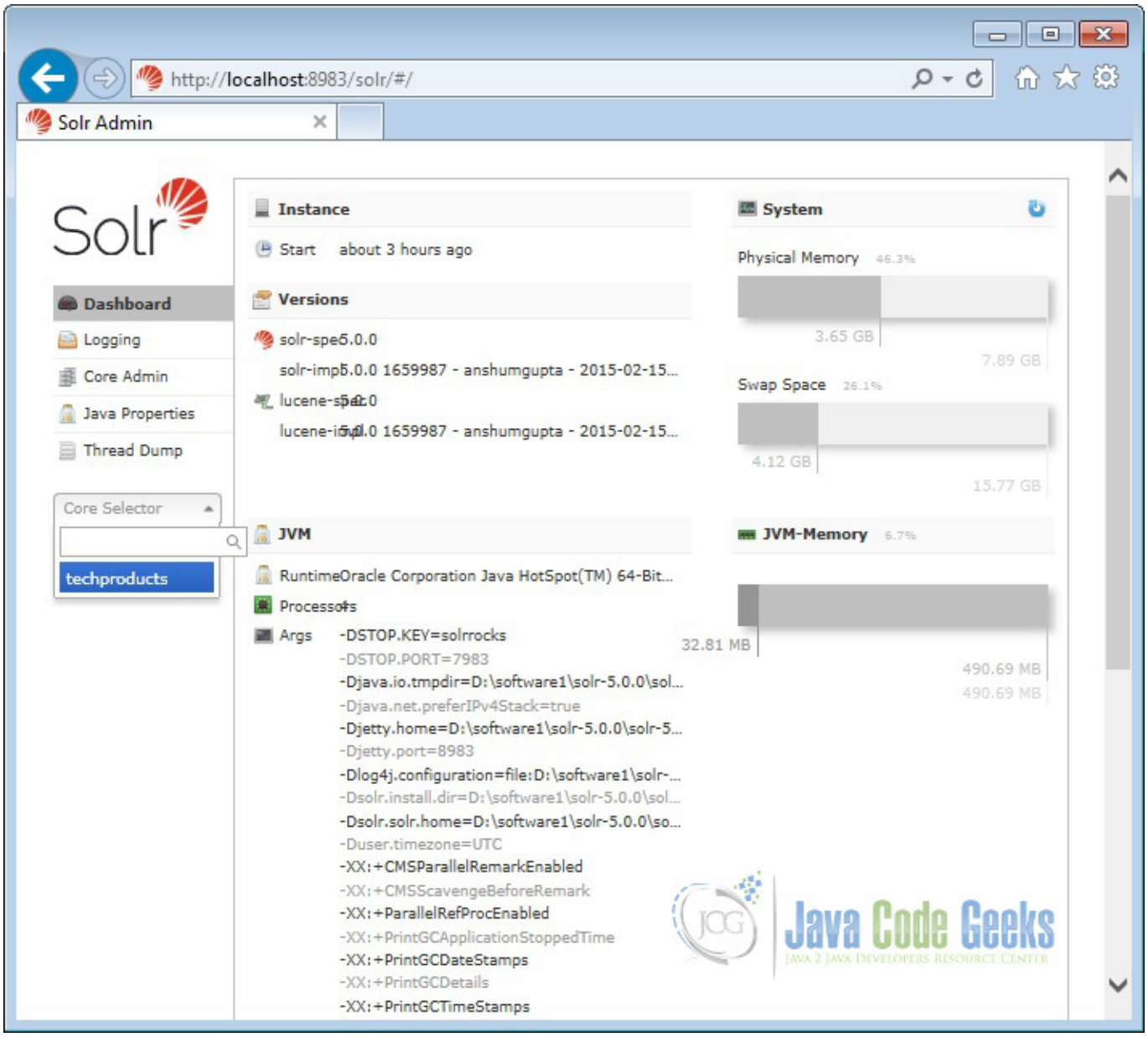

Figure 3.2: Solr admin console

3.3 Solr basic query

Solr provides a simple REST based select query to search on indexed data. We have to provide the context path of the collection (techproducts in our case) and use select in the URL indicating this is a select query. The parameter q is used to specify the search string.

The following query will look for video in all the indexed fields of the techproducts collection. If you notice the video is present in name field of result 1 and present in one of the features for result 2. This type of query can be used for free text searching on documents. Open the following URL in the browser.

```
https://localhost:8983/solr/techproducts/select?q=video
```

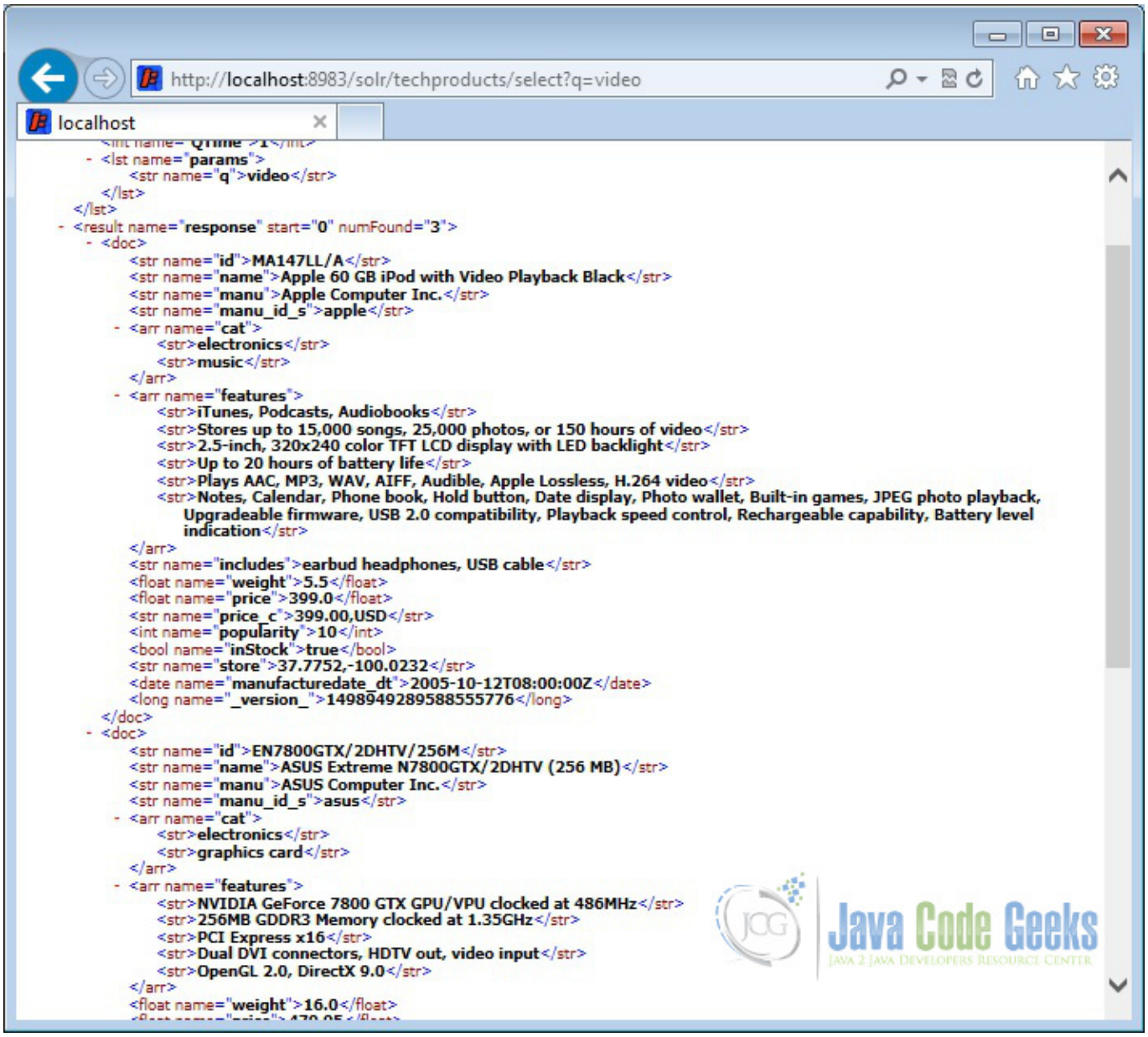

Figure 3.3: Solr query - basic

3.4 Solr query parameters

Solr provides a list of parameters that can be used with queries. The below section explains the available parameters and the purpose.

- **qt** - Query handler for the request. Standard query handler is used if not specified.
- **q** - It is used to specify the query event.
- **fq** - Used to specify filter queries.
- **sort** - Used to sort the results in ascending or descending order.
- **start, rows** - start specifies the staring number of the result set. By default it is zero. rows specify the number of records to return.

- **fl** - Used to return selective fields.
- **wt** - Specifies the response format. Default is XML.
- **indent** - Setting to true makes the response more readable.
- **debugQuery** - Setting the parameter to true gives the debugging information as part of response.
- **dismax** - To specify the dismax parser.
- **edismax** - To specify the edismax parser.
- **facet** - Setting to true enables the faceting.
- **spatial** - Used for geospatial searches.
- **spellcheck** - Setting to true help in searching similar terms.

3.5 Solr advanced queries

We can use one or more parameters provided by Solr to construct the query. In this section we will show you few combinations.

3.5.1 Solr query - selective fields

As we stated earlier, fl parameter can be used to select limited set of fields in the response. This will help to limit the volume of data that pass through system and reduce I/O cost.

We will modify the basic query to return limited set of fields. We have chosen to return id, name and price in the following query.

Open the following URL in the browser. You can notice the result set contains only the selected fields and the size of the response is reduced when measured in bytes.

```
https://localhost:8983/solr/techproducts/select?q=video&fl=id,name,price
```

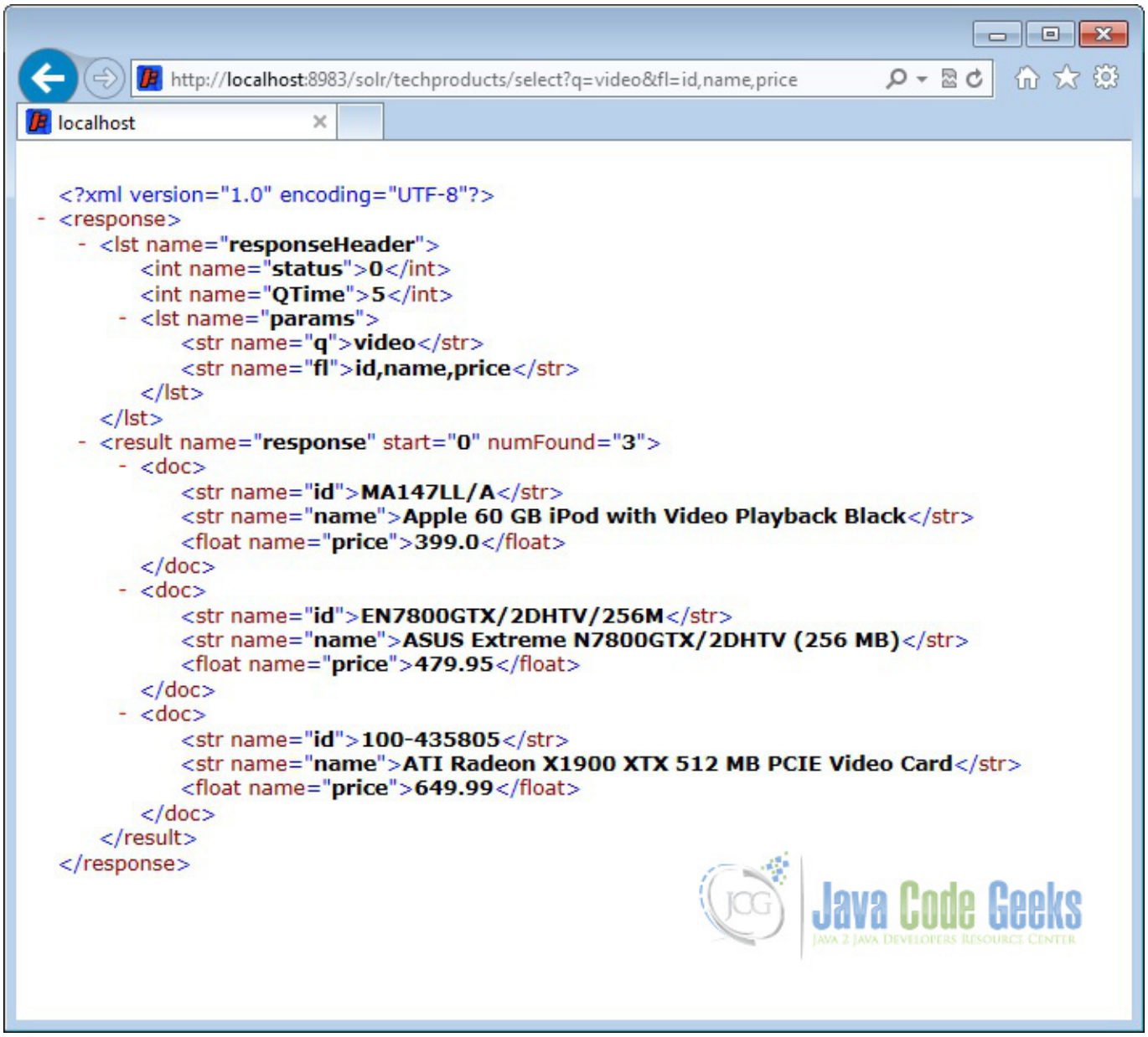

Figure 3.4: Solr query - selected fields

3.5.2 Solr query - filter

We can modify the basic query to add filter. In the basic query, we haven't specified any field to search for the string video and it returned values from name, features etc. But now we will specify where to look for the search string.

Open the following URL in browser. You can notice the result contains only the records that contain video in the name field.

```
https://localhost:8983/solr/techproducts/select?q=name:video
```

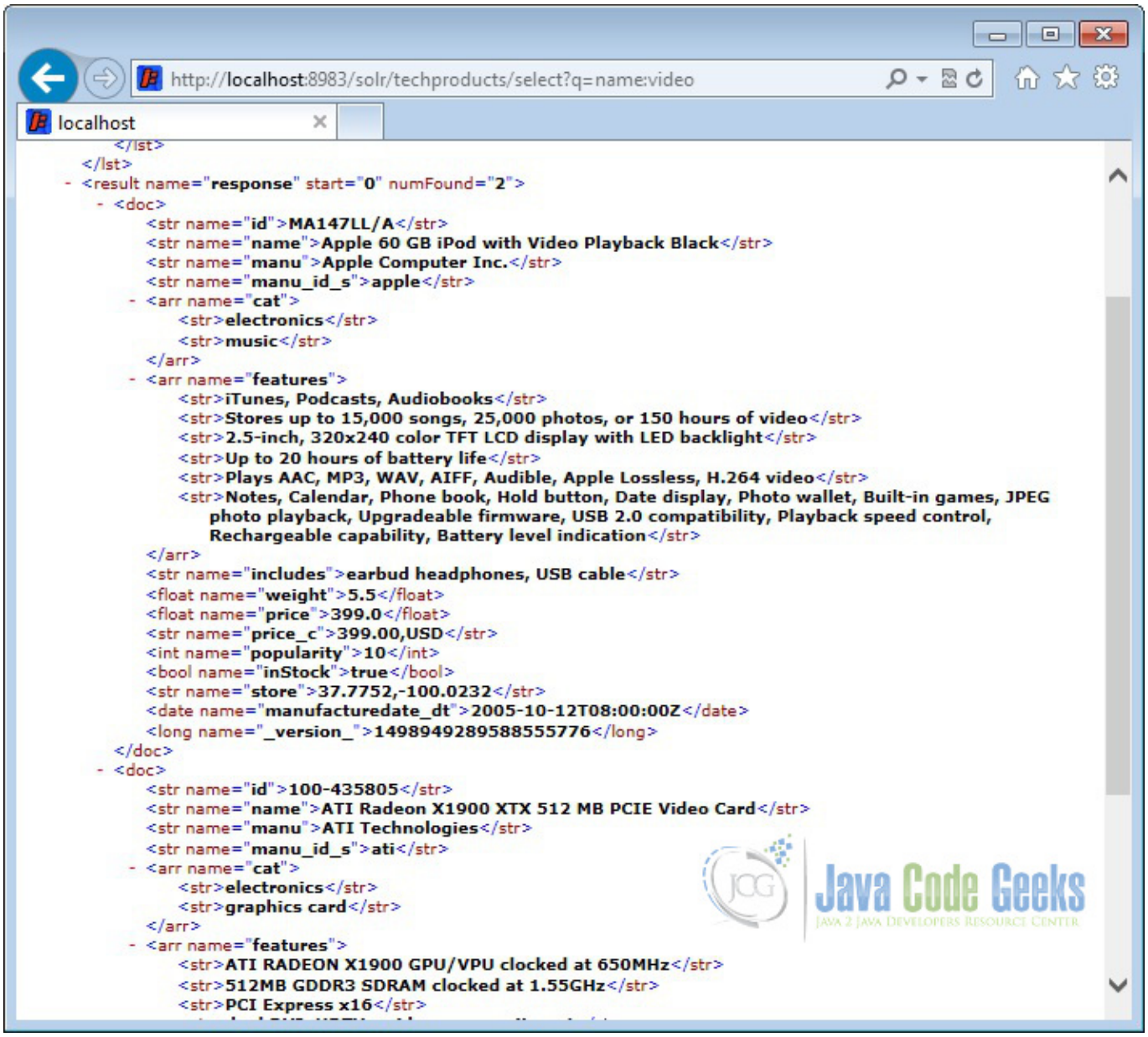

Figure 3.5: Solr query - filter name

Similarly, we can modify the query to select all the products with category as electronics. Open the following URL in browser. You can notice the result set contains only the electronics products. Also, we have combined the fl parameter to select only id, name and price fields.

```
https://localhost:8983/solr/techproducts/select?q=cat:electronics&fl=id,name,price
```

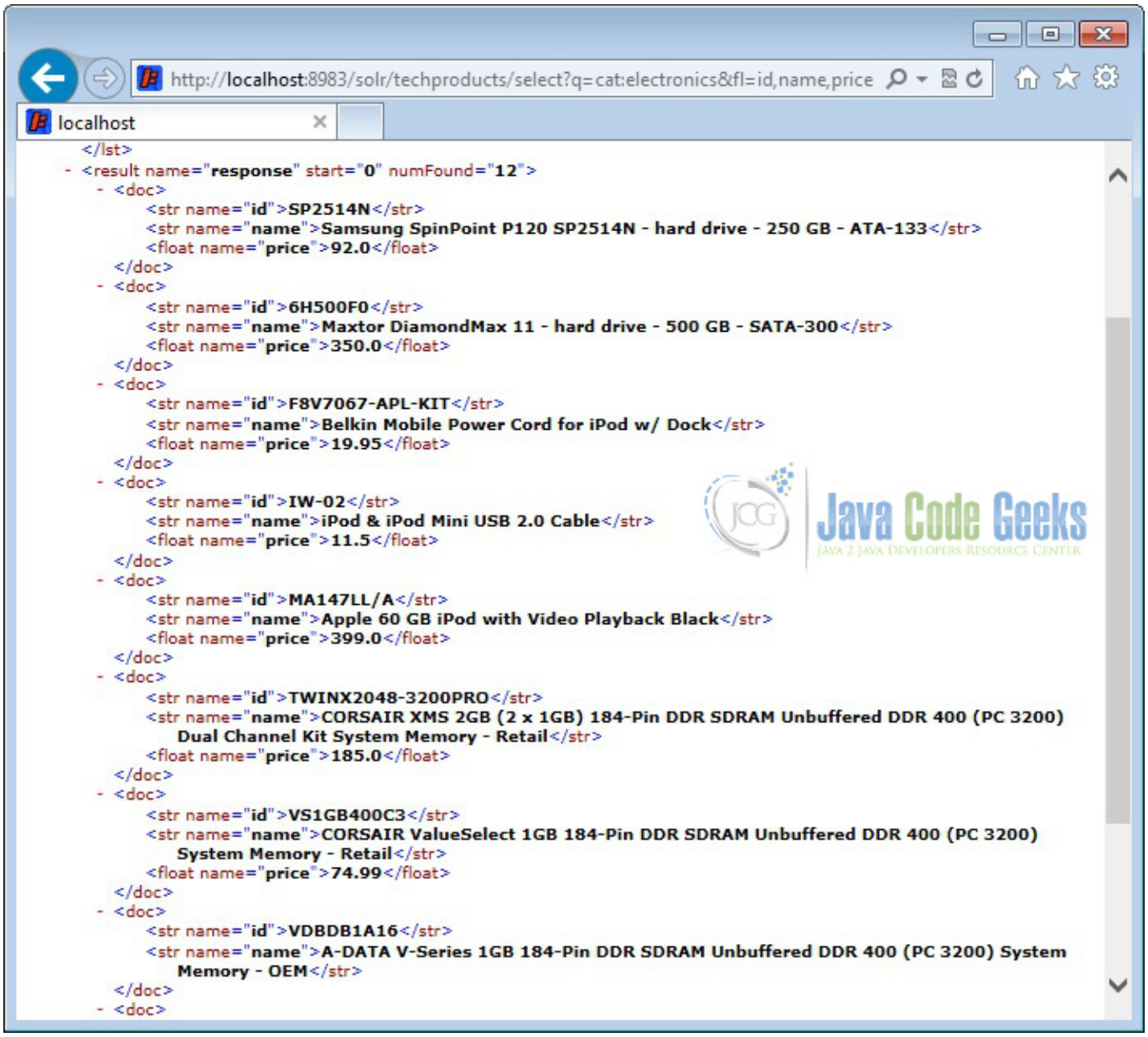

Figure 3.6: Solr query - filter category

3.5.3 Solr query - faceted Search

Faceting is a special type of search used for arranging the search results into categories. Searches are presented with indexed terms along with count of matching documents. Faceting makes it easy for users to explore search results, narrowing in on exactly the results they are looking for.

Open the following query in browser. You will see at the bottom of the response contains facet_counts for each of the category. Also you can notice we have applied filter on price and selected only specified fields.

```
https://localhost:8983/solr/techproducts/select?q=price:[0 TO 400]&fl=id,name,price&facet=true&facet.field=cat
```

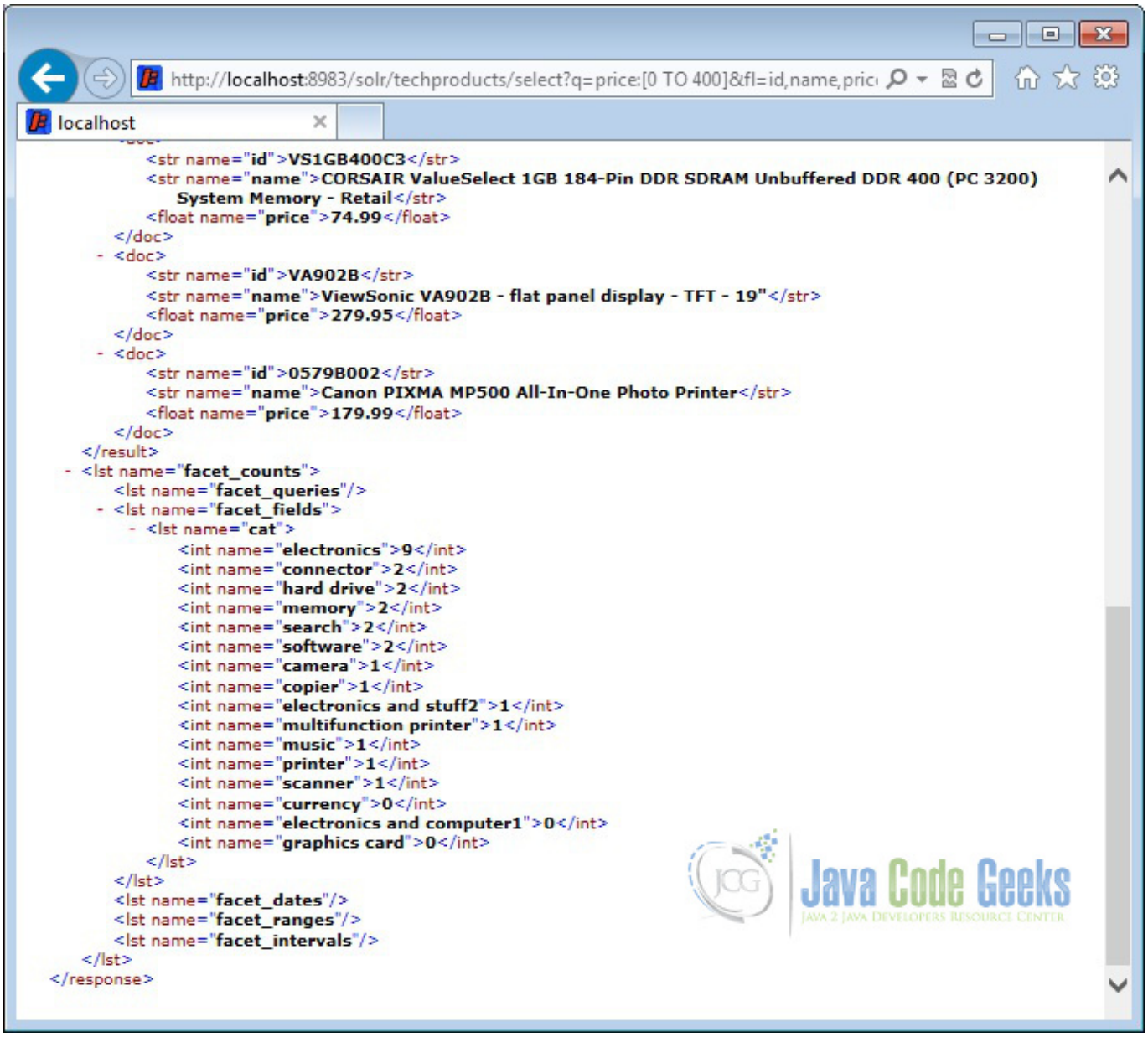

Figure 3.7: Solr query - facet

Chapter 4

Solr autocomplete example

In this example of Solr autocomplete example, we will discuss about how to implement autocomplete functionality for any UI component. We will be using jQuery autocomplete feature along with Solr indexing data to achieve the autocomplete functionality.

Our preferred environment for this example is solr-5.0.0, Eclipse Luna, JDK 8u25, and Tomcat 8 application server. Having said that, we have tested the code against JDK 1.7 and Tomcat 7 as well.

Before you begin the Solr installation make sure you have JDK installed and `Java_Home` is set appropriately.

4.1 Install Apache Solr

To begin with lets download the latest version of Apache Solr from the following location.

`https://lucene.apache.org/solr/downloads.html`

As of this writing, the stable version available is 5.0.0. Apache Solr has gone through various changes from 4.x.x to 5.0.0, so if you have different version of Solr you need to download the 5.x.x. version to follow this example.

Once the Solr zip file is downloaded unzip it into a folder. The extracted folder will look like the below.

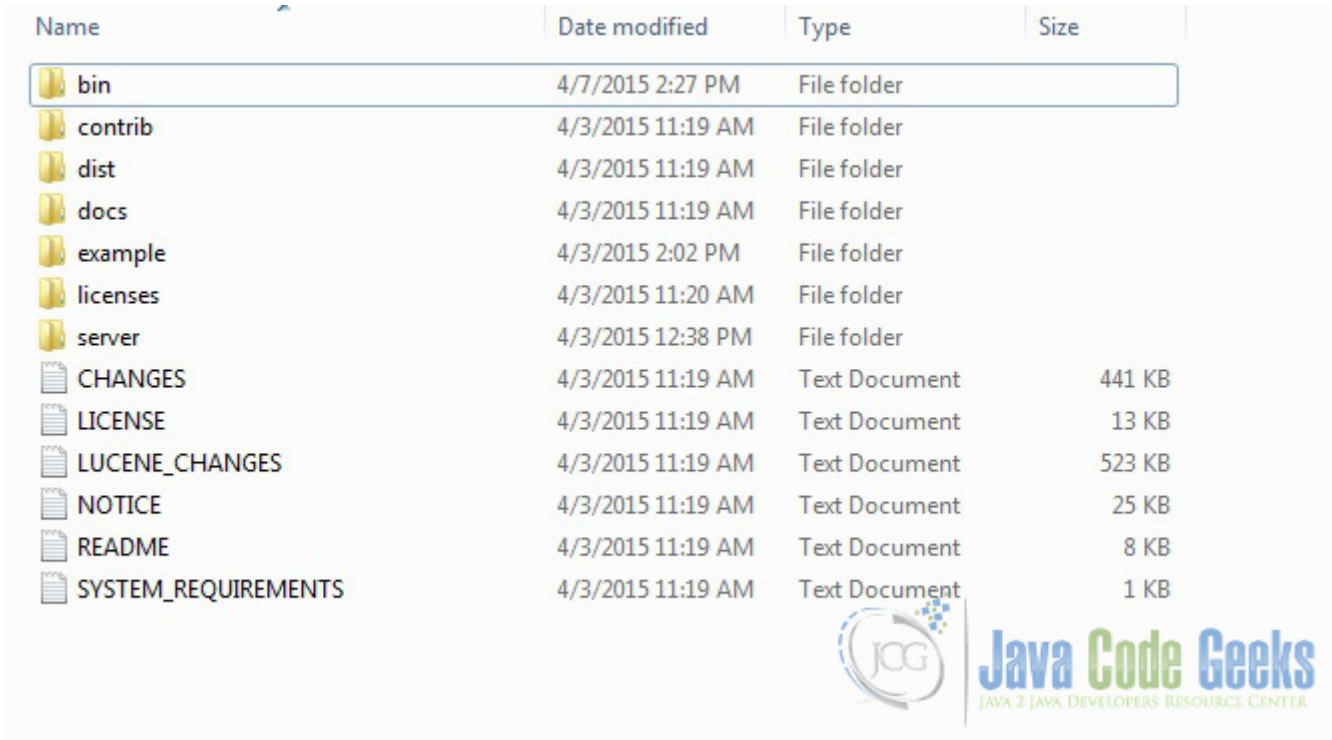

Figure 4.1: Solr folders

The `bin` folder contains the scripts to start and stop the server. The `example` folder contains few example files. We will be using one of them to demonstrate how Solr indexes the data. The `server` folder contains the `logs` folder where all the Solr logs are written. It will be helpful to check the logs for any error during indexing. The `solr` folder under server holds different collection or core. The configuration and data for each of the core/ collection are stored in the respective core/ collection folder.

Apache Solr comes with an inbuilt Jetty server. But before we start the solr instance we must validate the JAVA_HOME is set on the machine.

We can start the server using the command line script. Lets go to the bin directory from the command prompt and issue the following command

```
solr start
```

This will start the Solr server under the default port 8983.

We can now open the following URL in the browser and validate that our Solr instance is running. The specifics of solr admin tool is beyond the scope of the example.

```
https://localhost:8983/solr/
```

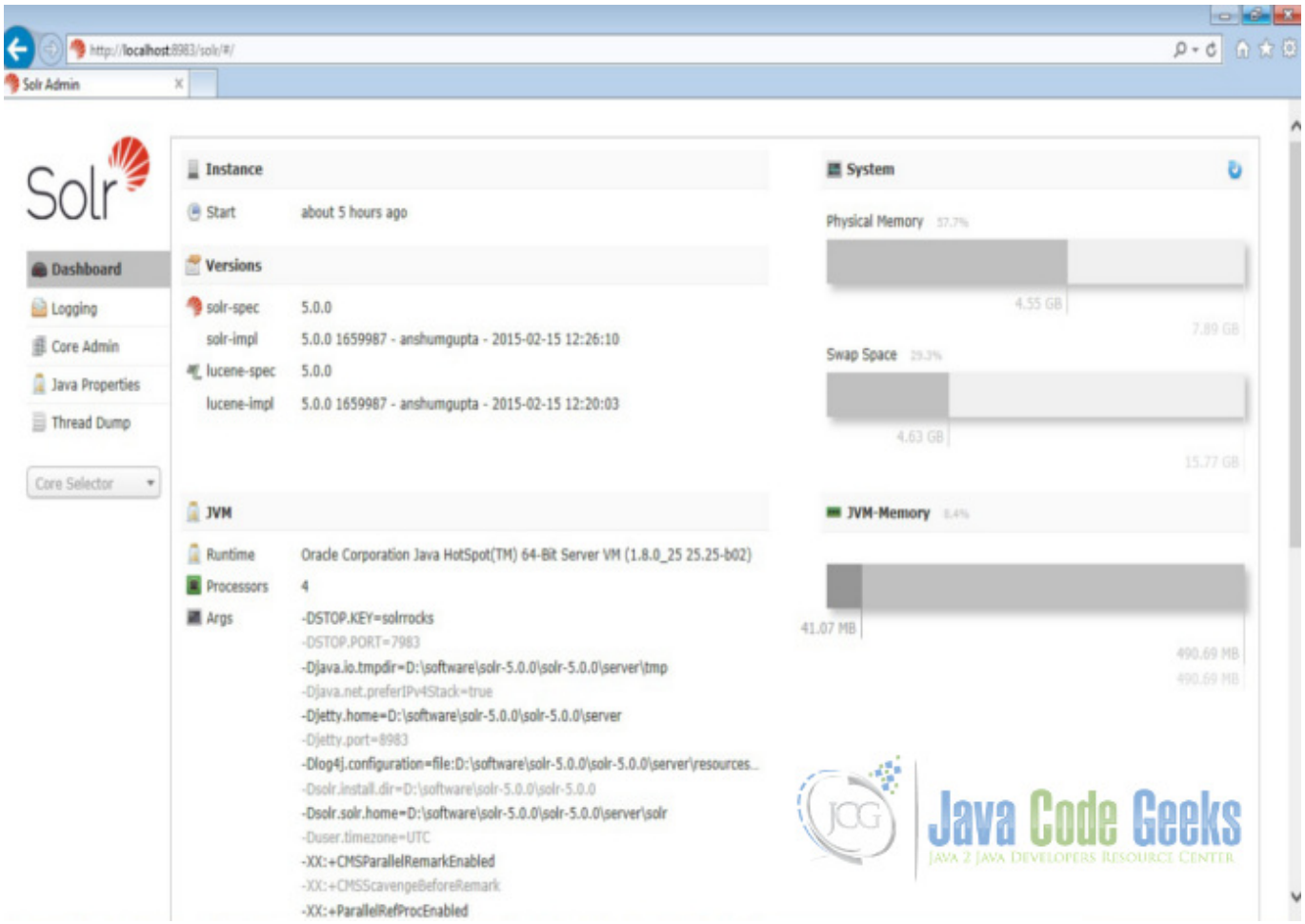

Figure 4.2: Solr admin console

4.2 Configuring Apache Solr

In this section, we will show you how to configure the core/collection for a solr instance and how to define the fields. Apache Solr ships with an option called Schemaless mode. This option allow users to construct effective schema without manually editing the schema file. But for this example we will use the Schema configuration for understanding the internals of the Solr.

4.2.1 Creating a Core

When the Solr server is started in Standalone mode the configuration is called core and when it is started in SolrCloud mode the configuration is called Collection. In this example we will discuss about the standalone server and core. We will park the SolrCloud discussion for later time.

First, we need to create a Core for indexing the data. The Solr create command has the following options:

- **-c <name>** - Name of the core or collection to create (required).
- **-d <confdir>** - The configuration directory, useful in the SolrCloud mode.
- **-n <configName>** - The configuration name. This defaults to the same name as the core or collection.
- **-p <port>** - Port of a local Solr instance to send the create command to; by default the script tries to detect the port by looking for running Solr instances.

- **-s <shards>** - Number of shards to split a collection into, default is 1.
- **-rf <replicas>** - Number of copies of each document in the collection. The default is 1.

In this example we will use the -c parameter for core name and -d parameter for the configuration directory. For all other parameters we make use of default settings.

Now navigate the `solr-5.0.0bin` folder in the command window and issue the following command.

```
solr create -c jcg -d basic_configs
```

We can see the following output in the command window.

```
Creating new core 'jcg' using command:
https://localhost:8983/solr/admin/cores?action=CREATE&name=jcg&instanceDir=jcg

{
 "responseHeader":{
 "status":0,
 "QTime":663},
 "core":"jcg"}
```

Now we navigate to the following URL and we can see jcg core being populated in the core selector. You can also see the statistics of the core.

```
https://localhost:8983/solr
```

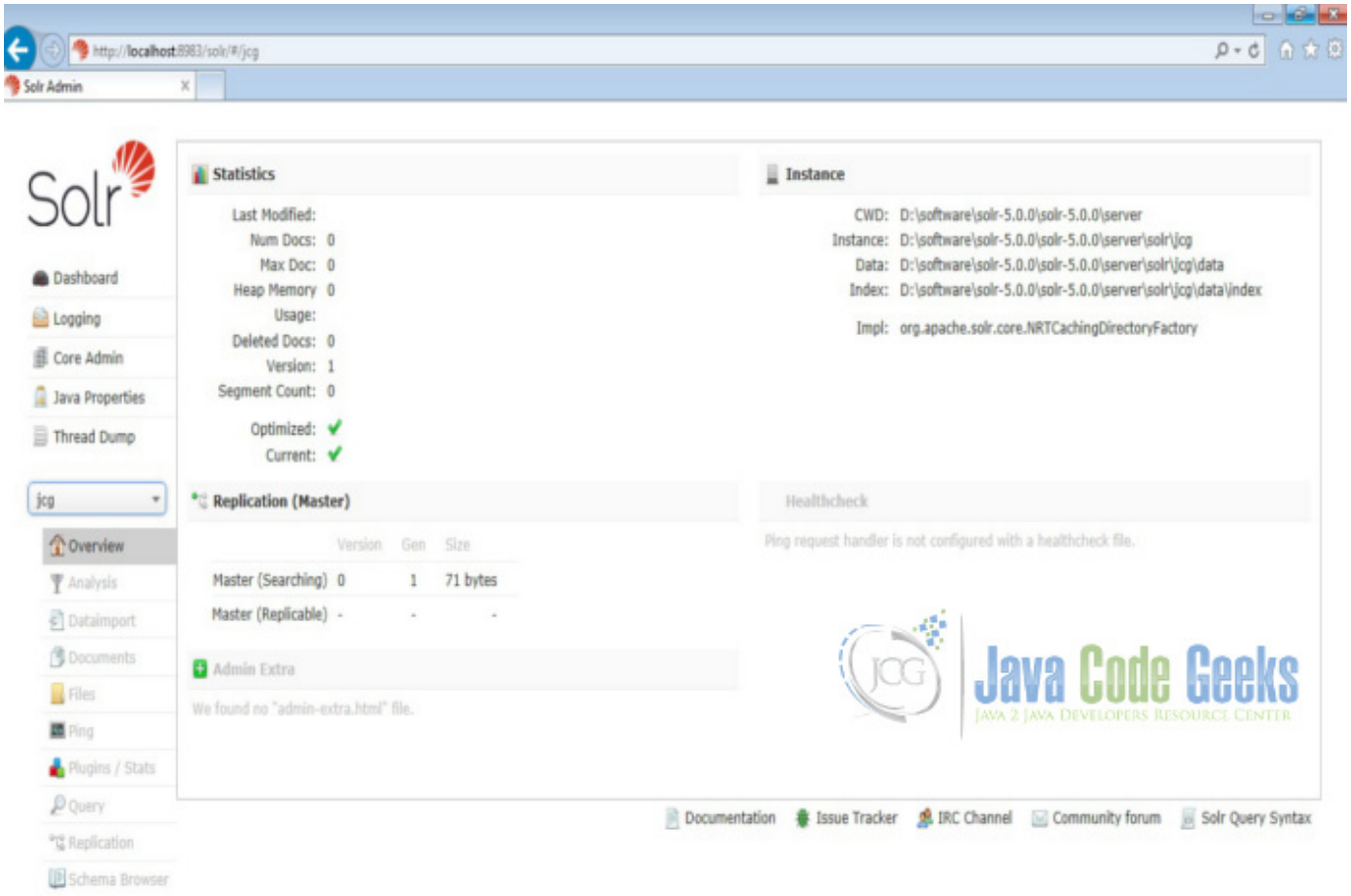

Figure 4.3: Solr jcg core

4.2.2 Modify the schema.xml file

We need to modify the `schema.xml` file under the folder `serversolrjcgconf` to include the fields. We will use one of the example file "books.csv" shipped along with Solr installation for indexing. The file is located under the folder `solr-5.0.0exampleexampledocs`

Now we navigate to the folder `serversolr` directory. You will see a folder called `jcg` created. The sub-folders namely`conf` and `data` have the core's configuration and indexed data respectively.

Now edit the `schema.xml` file in the `serversolrjcgconf` folder and add the following contents after the uniqueKey element.

schema.xml

```
<uniqueKey>id</uniqueKey>
<!-- Fields added for books.csv load-->
<field name="cat" type="text_general" indexed="true" stored="true"/>
<field name="name" type="text_general" indexed="true" stored="true"/>
<field name="price" type="tdouble" indexed="true" stored="true"/>
<field name="inStock" type="boolean" indexed="true" stored="true"/>
<field name="author" type="text_general" indexed="true" stored="true"/>
```

We have set the attribute `indexed` to true. This specifies the field is used for indexing and the record can be retrieved using the index. Setting the value to false will make the field only stored but can't be queried with.

Also note we have another attribute called `stored` and set it to true. This specifies the field is stored and can be returned in the output. Setting this field to false will make the field only indexed and can't be retrieved in output.

We have assigned the type for the fields present in the "books.csv" file here. The first field in the CSV file "id" is automatically taken care by the uniqueKey element of schema.xml file for indexing.

Since we have modified the configuration we have to stop and start the server. To do so, we need to issue the following command from bin directory through command line.

```
solr stop -all
```

The server will be stopped now. Now to start the server issue the following command from bin directory through command line.

```
solr start
```

4.3 Indexing the Data

Apache Solr comes with a Standalone Java program called the SimplePostTool. This program is packaged into JAR and available with the installation under the folder `exampleexampledocs`.

Now we navigate to the `exampleexampledocs` folder in the command prompt and type the following command. You will see a bunch of options to use the tool.

```
java -jar post.jar -h
```

The usage format in general is as follows

```
Usage: java [SystemProperties] -jar post.jar [-h|-] [<file|folder|url|arg> [<file|folder| ↩
    url|arg>...]]
```

As we said earlier, we will index the data present in the "books.csv" file shipped with Solr installation. We will navigate to the `solr-5.0.0exampleexampledocs` in the command prompt and issue the following command.

```
java -Dtype=text/csv -Durl=https://localhost:8983/solr/jcg/update -jar post.jar books.
csv
```

The SystemProperties used here are:

- -Dtype - the type of the data file.

- -Durl - URL for the jcg core.

The file "books.csv" will now be indexed and the command prompt will display the following output.

```
SimplePostTool version 5.0.0
Posting files to [base] url https://localhost:8983/solr/jcg/update using content-
type text/csv...
POSTing file books.csv to [base]
1 files indexed.
COMMITting Solr index changes to https://localhost:8983/solr/jcg/update...
Time spent: 0:00:00.647
```

4.4 Setting up the webproject

We will use the jQuery autocomplete widget to consume the data from Solr. First, we will set up the maven project for a simple web application.

In eclipse go to File → New→Other→ Maven Project.

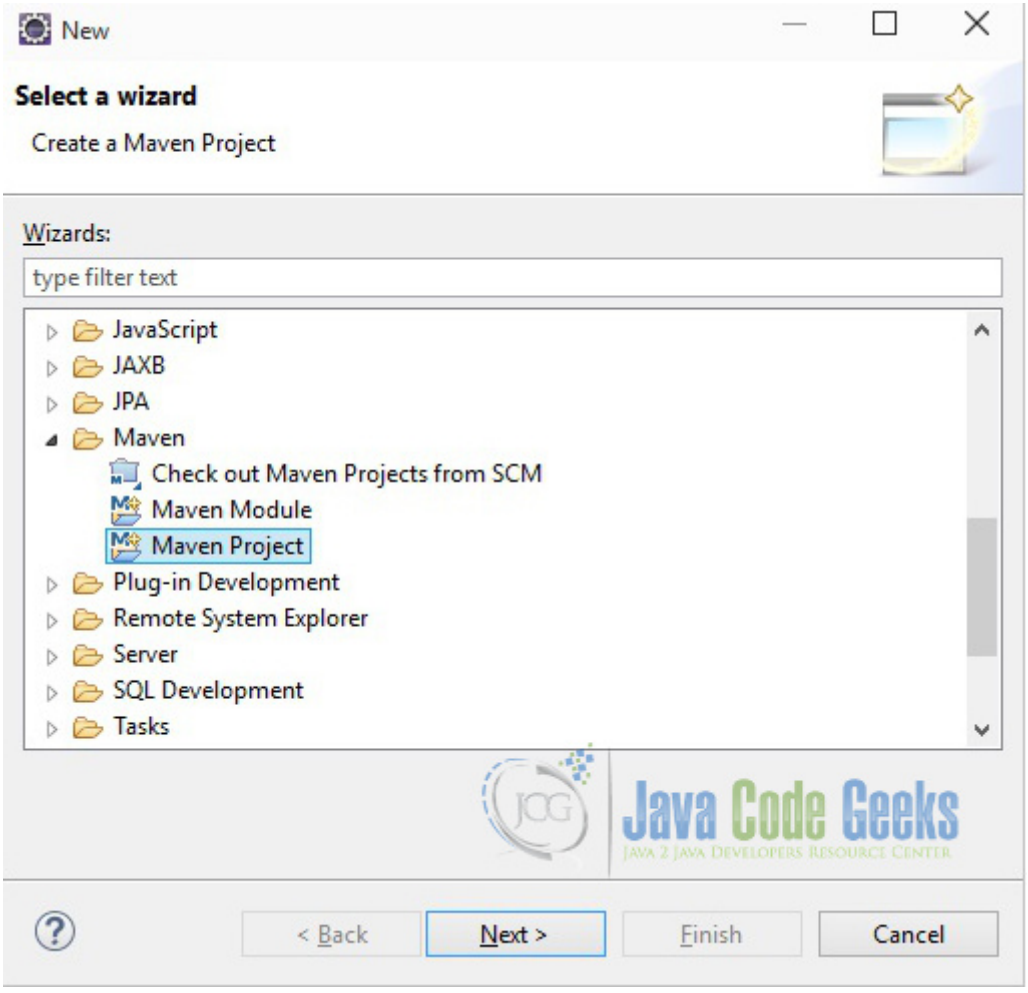

Figure 4.4: Maven - step 1

In the “Select project name and location” page of the wizard, make sure that “Create a simple project (skip archetype selection)” option is **unchecked**, hit “Next” to continue with default values.

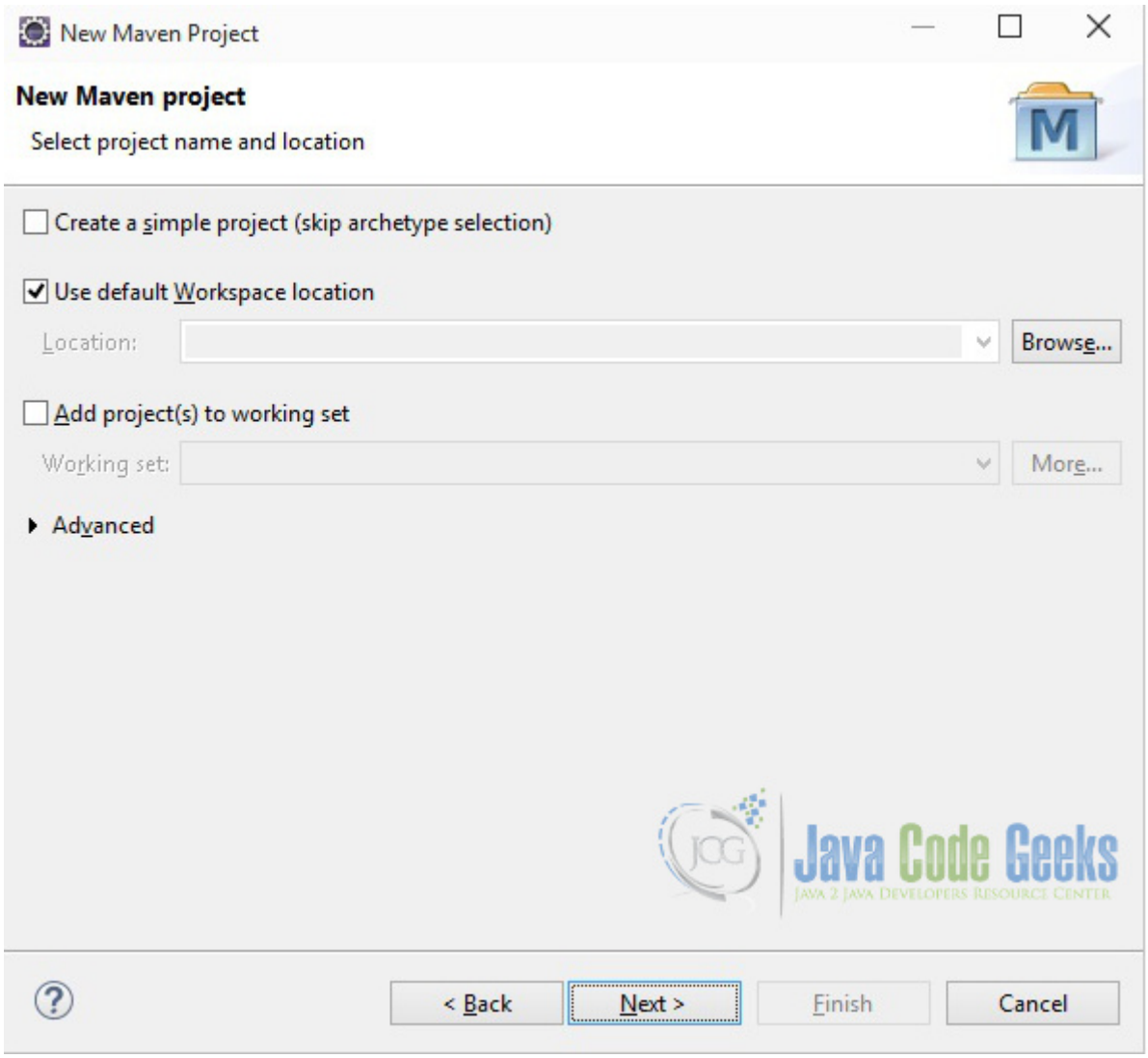

Figure 4.5: Maven - step 2

Here choose “maven-archetype-webapp” and click on Next.

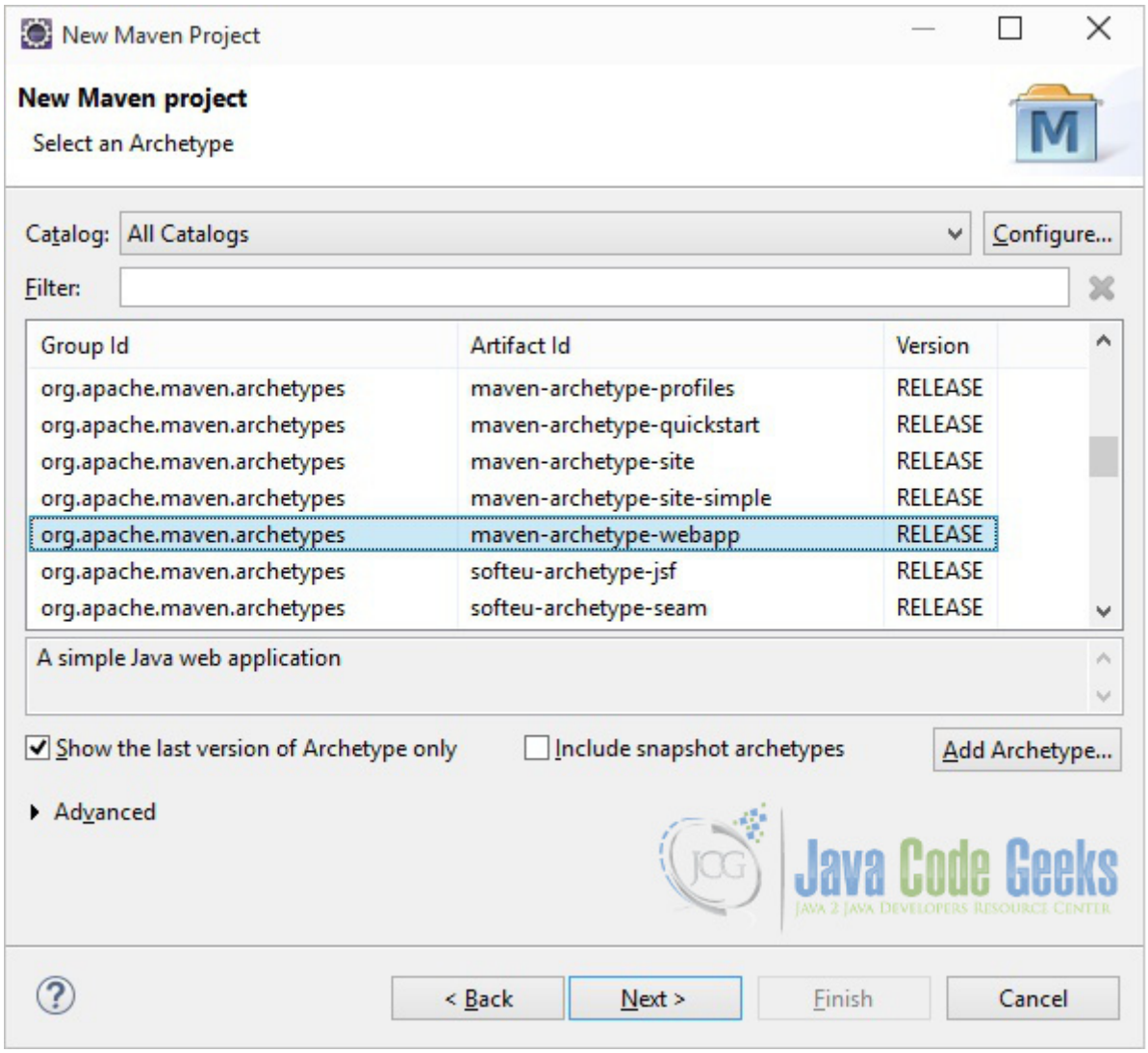

Figure 4.6: Maven - step 3

In the “Enter an artifact id” page of the wizard, you can define the name and main package of your project. Set the “Group Id” variable to `"com.javacodegeeks.snippets.enterprise"` and the “Artifact Id” variable to `"solrautocomplete"`. For package enter `"com.javacodegreeks.solrautocomplete"` and hit “Finish” to exit the wizard and to create your project.

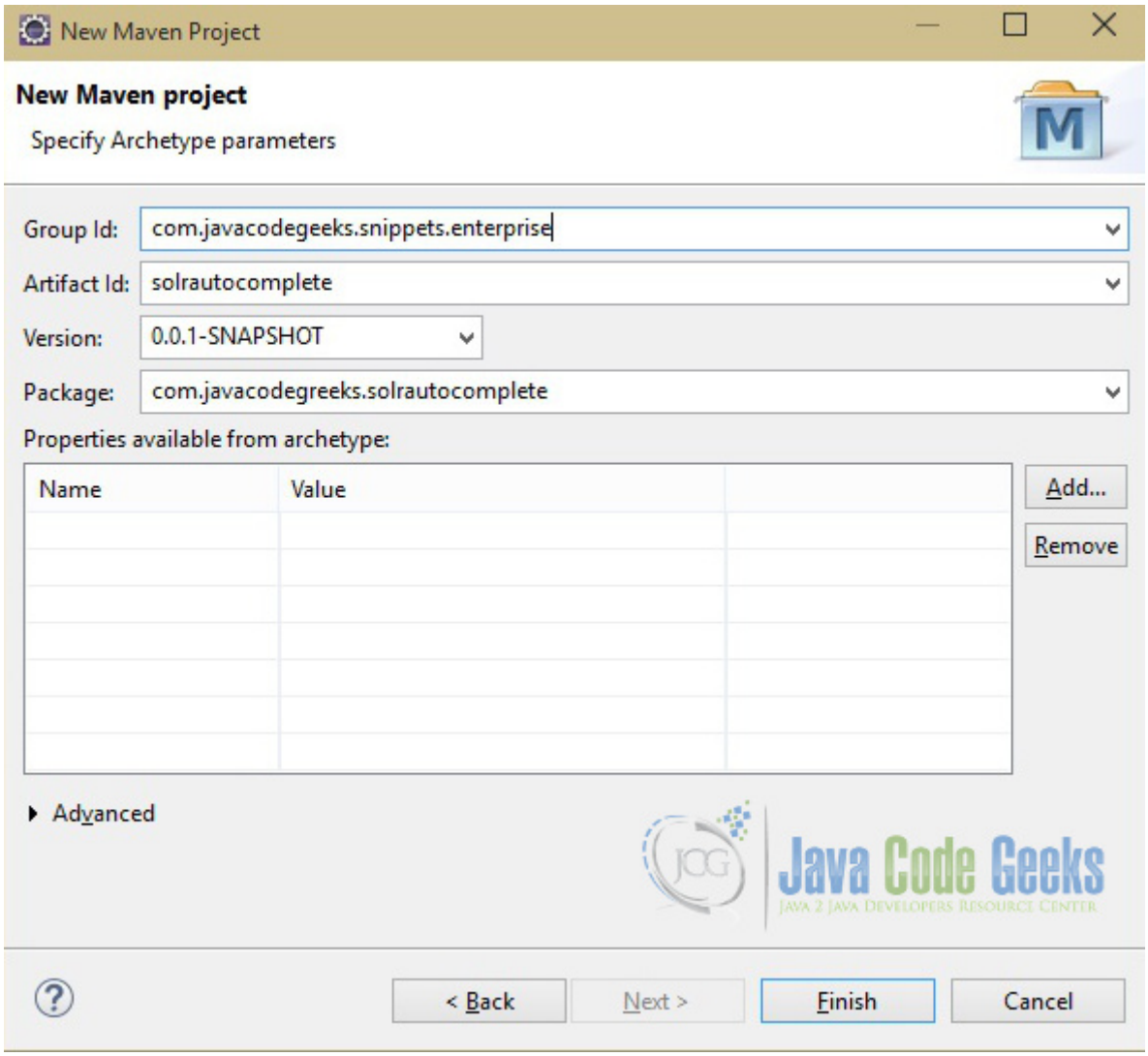

Figure 4.7: Maven - step 4

If you see any errors in the index.jsp , set target runtime for the project.

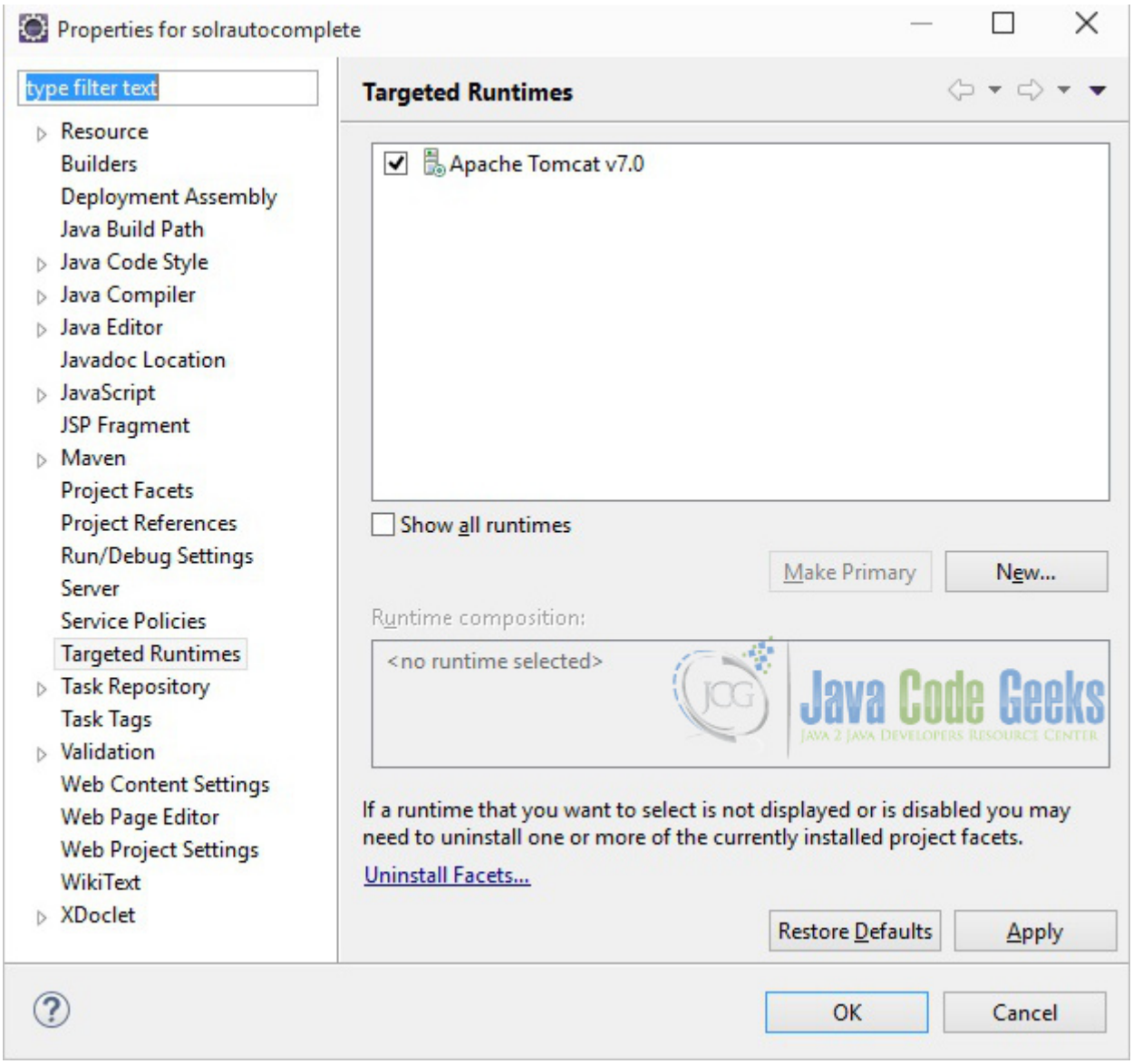

Figure 4.8: Maven - step 5

Now create a file called `search.html` in webapp folder. We are using the jQuery hosted on the cloud. We will use the jQuery AJAX to fetch the data from Solr and bind to the source of the autocomplete function.

search.html

```
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Solr auto complete</title>
<link
 href="https://code.jquery.com/ui/1.10.4/themes/ui-lightness/jquery-ui.css"
 rel="stylesheet"></link>
<script src="https://code.jquery.com/jquery-1.10.2.js"></script>
<script src="https://code.jquery.com/ui/1.10.4/jquery-ui.js"></script>
<script>
 $(function() {
 var URL_PREFIX = "https://localhost:8983/solr/jcg/select?q=name:";
```

```
 var URL_SUFFIX = "&wt=json";
 $("#searchBox").autocomplete({
 source : function(request, response) {
 var URL = URL_PREFIX + $("#searchBox").val() + URL_SUFFIX;
 $.ajax({
 url : URL,
 success : function(data) {
 var docs = JSON.stringify(data.response.docs);
 var jsonData = JSON.parse(docs);
 response($.map(jsonData, function(value, key) {
 return {
 label : value.name
 }
 }));
 },
 dataType : 'jsonp',
 jsonp : 'json.wrf'
 });
 },
 minLength : 1
 })
 });
</script>
</head>
<body>

 <p>Type The or A
 <label for="searchBox">Tags: </label> <input id="searchBox"></input>

</body>
</html>
```

Since Solr runs on a different port and the request (webpage) is initiated from another port, we might end up with cross domain issue. To overcome this we have to use `jsonp`. The minLength parameter specify after how many characters typed the search has to begin. Over here we have specified the value as 1 which means when a single character is typed the results are bound.

Now we can create the deployment package using Run as → Maven clean and then Run as → Maven install. This will create a war file in the target folder. The `war` file produced must be placed in `webapps` folder of tomcat. Now we can start the server.

Open the following URL and type *A* . This will bring results with books having title A ..

`https://localhost:8080/solrautocomplete/search.html`

Figure 4.9: Output 1

Now type *The* in the search box. This will return the books having word The.

Figure 4.10: Output 2

The problem with the above indexing technique is we could not able to get results based on phrases. Say if we type *The black* it doesn't fetch any result. Also when we type *bla* no results are bound. To overcome this issue we will use NGramFilterFactory and re-index the data.

4.5 Indexing using NGramFilterFactory

We will copy the field `name` to a new field called `name_ngram`. The copyField command copy one field to another at the time a document is added to the index. It's used either to index the same field differently, or to add multiple fields to the same field for easier/faster searching.

Now modify the `schema.xml` file in the `serversolrjcgconf` folder and add the following highlighted content.

schema.xml

```
<!--
<copyField source="title" dest="text"/>
<copyField source="body" dest="text"/>
-->
<copyField source="name" dest="name_ngram"/>
```

In the same file, we need to add a field called `name_ngram` and mark it for indexing. For it, we need to add the highlighted line.

schema.xml

```
<uniqueKey>id</uniqueKey>
<!-- Fields added for books.csv load-->
<field name="cat" type="text_general" indexed="true" stored="true"/>
<field name="name" type="text_general" indexed="true" stored="true"/>
```

```
<field name="price" type="tdouble" indexed="true" stored="true"/>
<field name="inStock" type="boolean" indexed="true" stored="true"/>
<field name="author" type="text_general" indexed="true" stored="true"/>
<field name="name_ngram" type="text_ngram" indexed="true" stored="true"/>
```

Take a note we have changed the type of the new field as `text_ngram`. We will define the type `text_ngram` subsequently.

Now we add the definition for the field `text_ngram` in the `schema.xml` file. We have set the minimum ngram size as 2 and maximum ngram size as 10.

schema.xml

```
<!-- Added for NGram field-->
<fieldType name="text_ngram" class="solr.TextField" positionIncrementGap="100">
<analyzer type="index">
<tokenizer class="solr.NGramTokenizerFactory" minGramSize="2" maxGramSize="10"/>
<filter class="solr.LowerCaseFilterFactory"/>
</analyzer>
<analyzer type="query">
<tokenizer class="solr.EdgeNGramTokenizerFactory" minGramSize="2" maxGramSize="10"/>
<filter class="solr.LowerCaseFilterFactory"/>
</analyzer>
</fieldType>
```

We have combined the features of `NGramTokenizerFactory` and `EdgeNGramTokenizerFactory` to achieve the best of indexing. Since we have modified the configuration we have to stop and start the server. To do so, we need to issue the following command from `bin` directory through command line.

```
solr stop -all
```

The server will be stopped now. Now to start the server issue the following command from bin directory through command line.

```
solr start
```

We will re-index the data present in the `books.csv` file. We will navigate to the `solr-5.0.0exampleexampledocs` in the command prompt and issue the following command.

```
java -Dtype=text/csv -Durl=https://localhost:8983/solr/jcg/update -jar post.jar books.
csv
```

The file `books.csv` will now be re-indexed and the command prompt will display the following output.

```
SimplePostTool version 5.0.0
Posting files to [base] url https://localhost:8983/solr/jcg/update using content-type text/ ↩
    csv...
POSTing file books.csv to [base]
1 files indexed.
COMMITting Solr index changes to https://localhost:8983/solr/jcg/update...
Time spent: 0:00:02.325
```

4.6 Modify search.html

Now we will modify the `search.html` to include another search box to test the NGram indexing. We will create search box with id `ngrambox` and write another javascript function for the new search box.

search.html

```
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Solr auto complete</title>
```

```
<link
 href="https://code.jquery.com/ui/1.10.4/themes/ui-lightness/jquery-ui.css"
 rel="stylesheet"></link>
<script src="https://code.jquery.com/jquery-1.10.2.js"></script>
<script src="https://code.jquery.com/ui/1.10.4/jquery-ui.js"></script>
<script>
 $(function() {
 var URL_PREFIX = "https://localhost:8983/solr/jcg/select?q=name:";
 var URL_SUFFIX = "&wt=json";
 $("#searchBox").autocomplete({
 source : function(request, response) {
 var URL = URL_PREFIX + $("#searchBox").val() + URL_SUFFIX;
 $.ajax({
 url : URL,
 success : function(data) {
 var docs = JSON.stringify(data.response.docs);
 var jsonData = JSON.parse(docs);
 response($.map(jsonData, function(value, key) {
 return {
 label : value.name
 }
 }));
 },
 dataType : 'jsonp',
 jsonp : 'json.wrf'
 });
 },
 minLength : 1
 })
 });
 $(function() {
 var URL_PREFIX = "https://localhost:8983/solr/jcg/select?q=name:";
 var URL_MIDDLE = "OR name_ngram:";
 var URL_SUFFIX = "&wt=json";
 $("#ngramBox").autocomplete(
 {
 source : function(request, response) {
 var searchString = "\"" + $("#ngramBox").val() + "\"";
 var URL = URL_PREFIX + searchString + URL_MIDDLE
 + searchString + URL_SUFFIX;
 $.ajax({
 url : URL,
 success : function(data) {
 var docs = JSON.stringify(data.response.docs);
 var jsonData = JSON.parse(docs);
 response($.map(jsonData, function(value, key) {
 return {
 label : value.name
 }
 }));
 },
 dataType : 'jsonp',
 jsonp : 'json.wrf'
 });
 },
 minLength : 1
 })
 });
</script>
</head>
<body>
```

```
 Type 'A' or 'The'
 <label for="searchBox">Tags: </label> <input id="searchBox"></input>


 Type 'Th' or 'Bla' or 'The Black'
 <label for="ngramBox">Tags: </label> <input id="ngramBox"></input>

</body>
</html>
```

Now again package using maven and copy the `war` to the apache tomcat `webapps` folder. Open the following URL in the browser and type *Bla*.

`https://localhost:8080/solrautocomplete/search.html`

Figure 4.11: Output 3

4.7 Download the Eclipse Project

This was an example of Solr autocomplete.

Download

You can download the full source code of this example here: **solr autocomplete** and download the schema file here: **schema file**.

Chapter 5

Solr replication example

In this example of Solr replication example, we will show you how to set up replication in Apache Solr and demonstrate how a new record gets replicated from master to slave cores. For this example we will consider one master and two slave servers. In production environment we will use different machines for hosting the master and slave servers. Over here we will run both master and slave Solr servers on the same machine by using different ports.

Our preferred environment for this example is Windows. Before you begin the Solr installation make sure you have JDK installed and Java_Home is set appropriately.

5.1 Install Apache Solr

To begin with lets download the latest version of Apache Solr from the following location.

`https://lucene.apache.org/solr/downloads.html`

Apache Solr has gone through various changes from 4.x.x to 5.0.0, so if you have different version of Solr you need to download the 5.x.x. version to follow this example. Once the Solr zip file is downloaded unzip it into a folder. The extracted folder will look like the below.

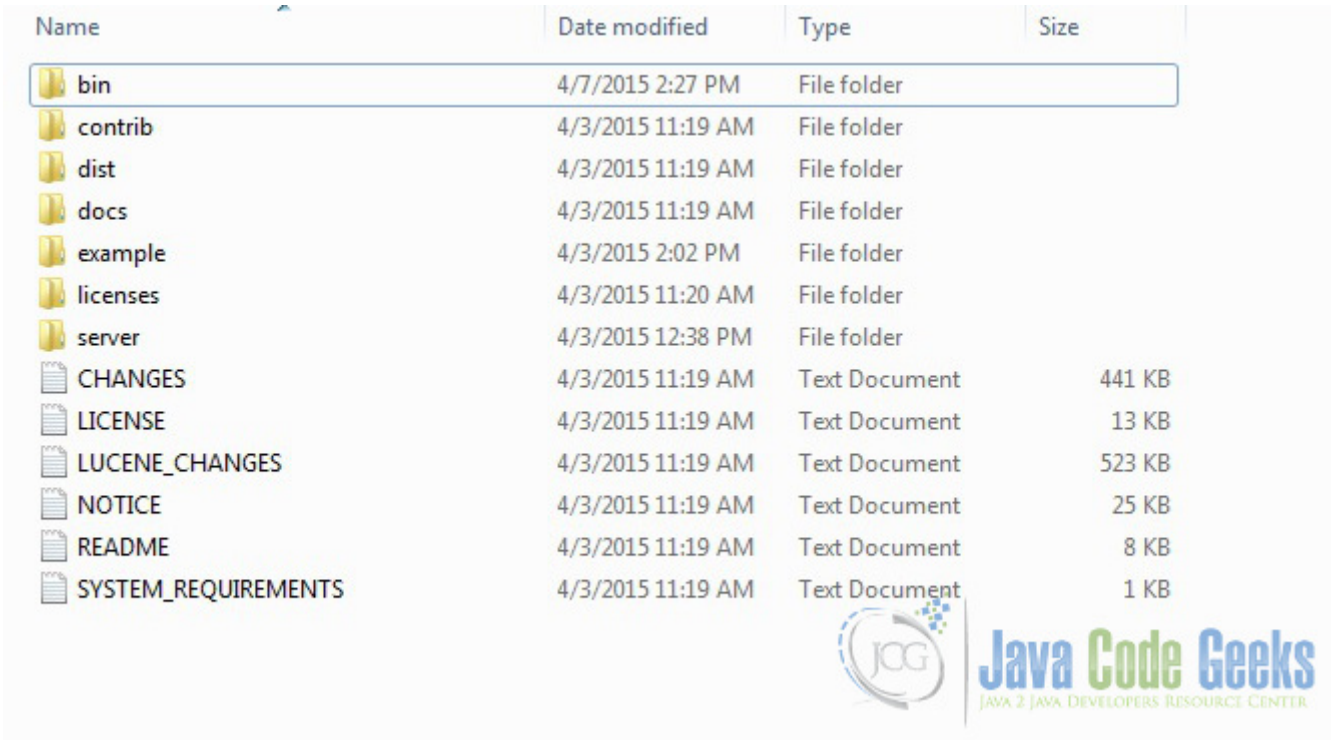

Figure 5.1: Solr folders

The `bin` folder contains the scripts to start and stop the server. The `example` folder contains few example files. We will be using one of them to demonstrate how replication works. The `server` folder contains the `logs` folder where all the Solr logs are written. It will be helpful to check the logs for any error during indexing. The `solr` folder under server holds different collection or core. The configuration and data for each of the core/ collection are stored in the respective core/ collection folder.

Apache Solr comes with an inbuilt Jetty server. But before we start the solr instance we must validate the JAVA_HOME is set on the machine.

We can start the server using the command line script. Lets go to the bin directory from the command prompt and issue the following command

```
solr start
```

This will start the Solr server under the default port 8983.

We can now open the following URL in the browser and validate that our Solr instance is running. The specifics of solr admin tool is beyond the scope of the example.

```
https://localhost:8983/solr/
```

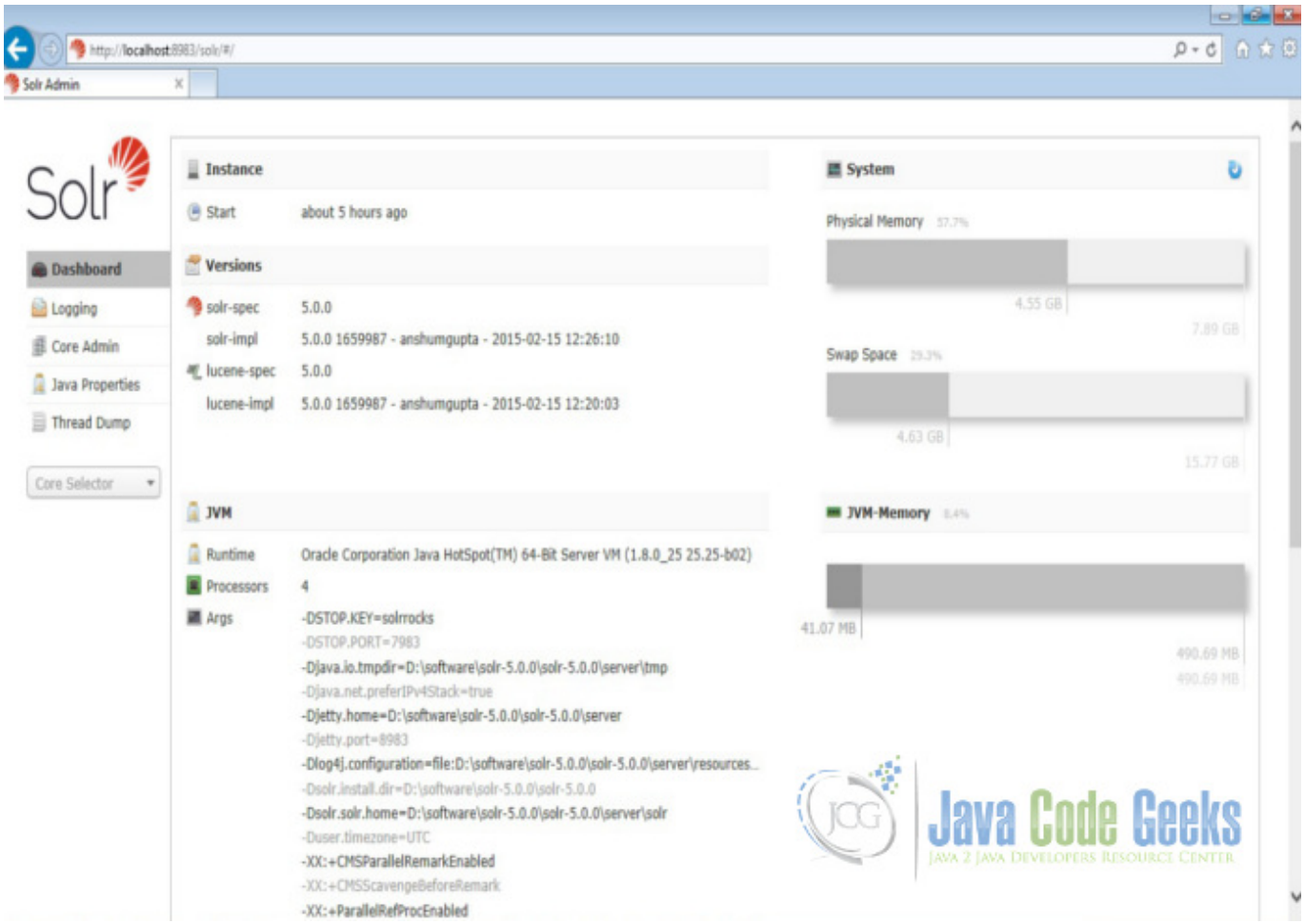

Figure 5.2: Solr admin console

5.2 Configuring Solr - master

In this section, we will show you how to configure the master core for a Solr instance. Apache Solr ships with an option called Schemaless mode. This option allow users to construct effective schema without manually editing the schema file. For this example we will use the reference configset `sample_techproducts_configs`.

5.2.1 Creating master Core

First, we need to create a core for indexing the data. The Solr create command has the following options:

- **-c <name>** - Name of the core or collection to create (required).
- **-d <confdir>** - The configuration directory, useful in the SolrCloud mode.
- **-n <configName>** - The configuration name. This defaults to the same name as the core or collection.
- **-p <port>** - Port of a local Solr instance to send the create command to; by default the script tries to detect the port by looking for running Solr instances.
- **-s <shards>** - Number of shards to split a collection into, default is 1.
- **-rf <replicas>** - Number of copies of each document in the collection. The default is 1.

In this example we will use the -c parameter for core name, -rf parameter for replciation and -d parameter for the configuration directory.

Now navigate the `solr-5.0.0bin` folder in the command window and issue the following command.

```
solr create -c master -d sample_techproducts_configs -p 8983 -rf 3
```

We can see the following output in the command window.

```
Creating new core 'master' using command:
https://localhost:8983/solr/admin/cores?action=CREATE&name=master&instanceDi
r=master

{
 "responseHeader":{
 "status":0,
 "QTime":1563},
 "core":"master"}
```

Now we can navigate to the following URL and see master core being populated in the core selector. You can also see the statistics of the core.

```
https://localhost:8983/solr/#/master
```

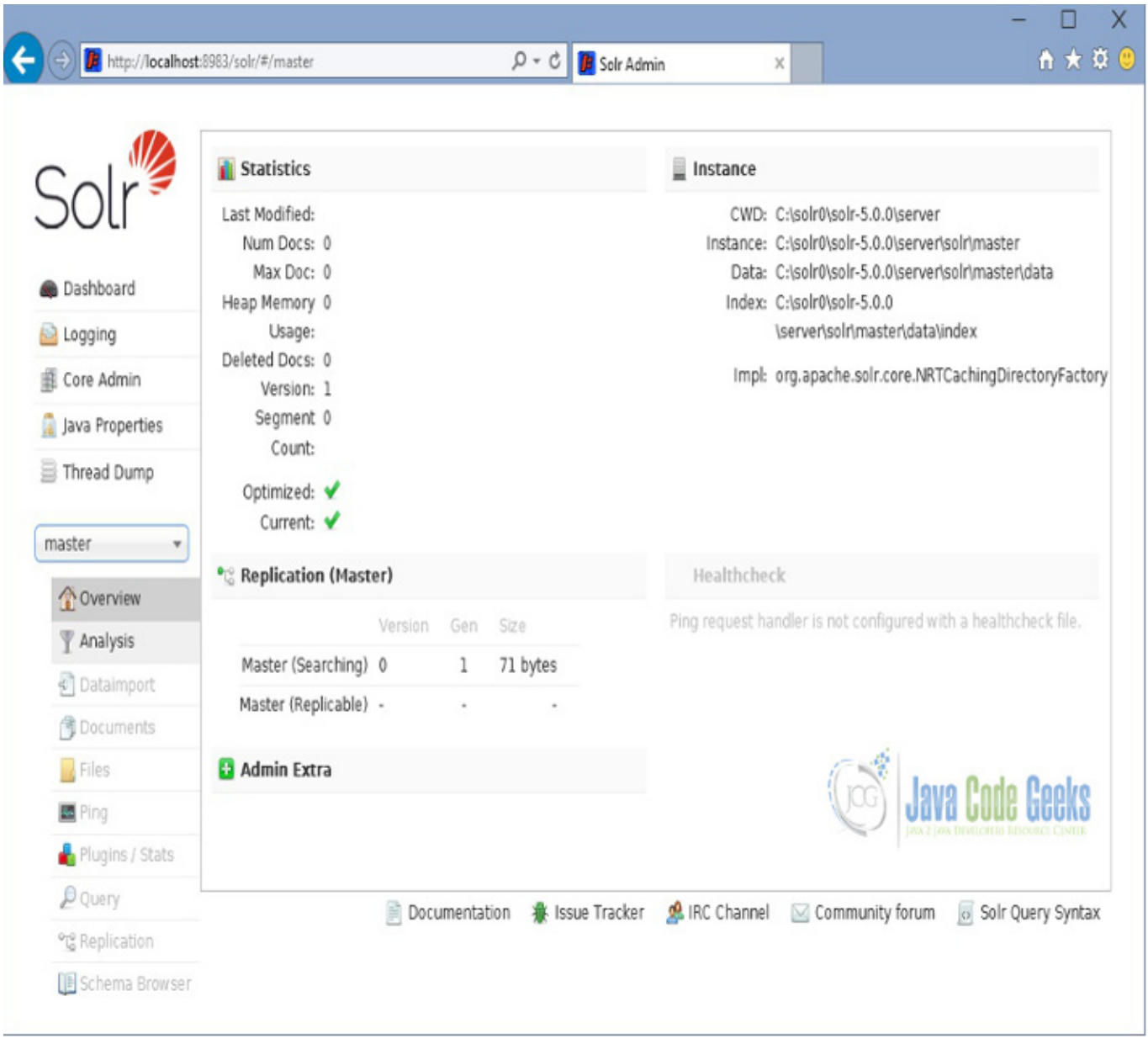

Figure 5.3: master console

5.2.2 Modify solrconfig

Open the file `solrconfig.xml` under the folder `serversolrmasterconf` and add the configuration for the master under the requestHandler tag. We will set the values for `replicateAfter` and `backAfter` to `optimize`. The `confFiles` parameter value is set according to the slave collection name we are going to create.

solrconfig.xml

```
<!-- Replication Handler -->
     <requestHandler name="/replication" class="solr.ReplicationHandler" >
          <lst name="master">
               <str name="replicateAfter">optimize</str>
               <str name="backupAfter">optimize</str>
               <str name="confFiles">solrconfig_slave.xml:solrconfig.xml,x.xml,y.xml</str>
               <str name="commitReserveDuration">00:00:10</str>
```

```
          </lst>
          <int name="maxNumberOfBackups">2</int>
          <lst name="invariants">
               <str name="maxWriteMBPerSec">16</str>
          </lst>
     </requestHandler>
```

Since we have modified the `solrconfig` we have to restart the solr server. Issue the following commands in the command window navigating to `solr-5.0.0bin`.

```
solr stop -all
```

```
solr start
```

5.3 Configuring Solr - slave

For this example, we will create two slave cores. The data from the master core will get replicated into both slaves. We will run the two slaves on the same machine with different ports along with the master core. To do so, extract another copy of solr server to a folder called solr1. Navigate to the `solr-5.0.0bin` folder of solr1 in the command window and issue the following command.

```
solr start -p 9000
```

The -p option will start the solr server in a different port. For the first slave we will use port 9000. Now navigate to the `solr-5.0.0bin` folder of the slave in the command window and issue the following command.

```
solr create -c slave -d sample_techproducts_configs -p 9000
```

We can see the following output in the command window.

```
Creating new core 'slave' using command:
https://localhost:9000/solr/admin/cores?action=CREATE&name=slave&instanceDir=slave

{
 "responseHeader":{
 "status":0,
 "QTime":1778},
 "core":"slave"}
```

Now open the file `solrconfig.xml` under the folder `serversolrslaveconf` and add the configuration for the slave under the requestHandler tag. In the configuration we will point the slave to the `masterUrl` for replication. The `pollInterval` is set to 20 seconds. It is the time difference between two poll requests made by the slave.

solrconfig.xml

```
<!-- Replication Handler -->
     <requestHandler name="/replication" class="solr.ReplicationHandler" >
          <lst name="slave">
               <!--fully qualified url for the replication handler of master. It is ↩
                   possible
               to pass on this as
               a request param for the fetchindex command-->
               <str name="masterUrl">https://localhost:8983/solr/master/replication</str>
               <!--Interval in which the slave should poll master .Format is HH:mm:ss . If
               this is absent slave does not
               poll automatically.
               But a fetchindex can be triggered from the admin or the http API -->
               <str name="pollInterval">00:00:20</str>
          </lst>
     </requestHandler>
```

Since we have modified the solrconfig we have to restart the solr server. Issue the following commands in the command window navigating to `solr-5.0.0bin`.

```
solr stop -all
```

```
solr start -p 9000
```

Now open the slave console using the following URL. The replication section will show the configuration reflecting the configuration we made in the solrconfig.

```
https://localhost:9000/solr/#/slave/replication
```

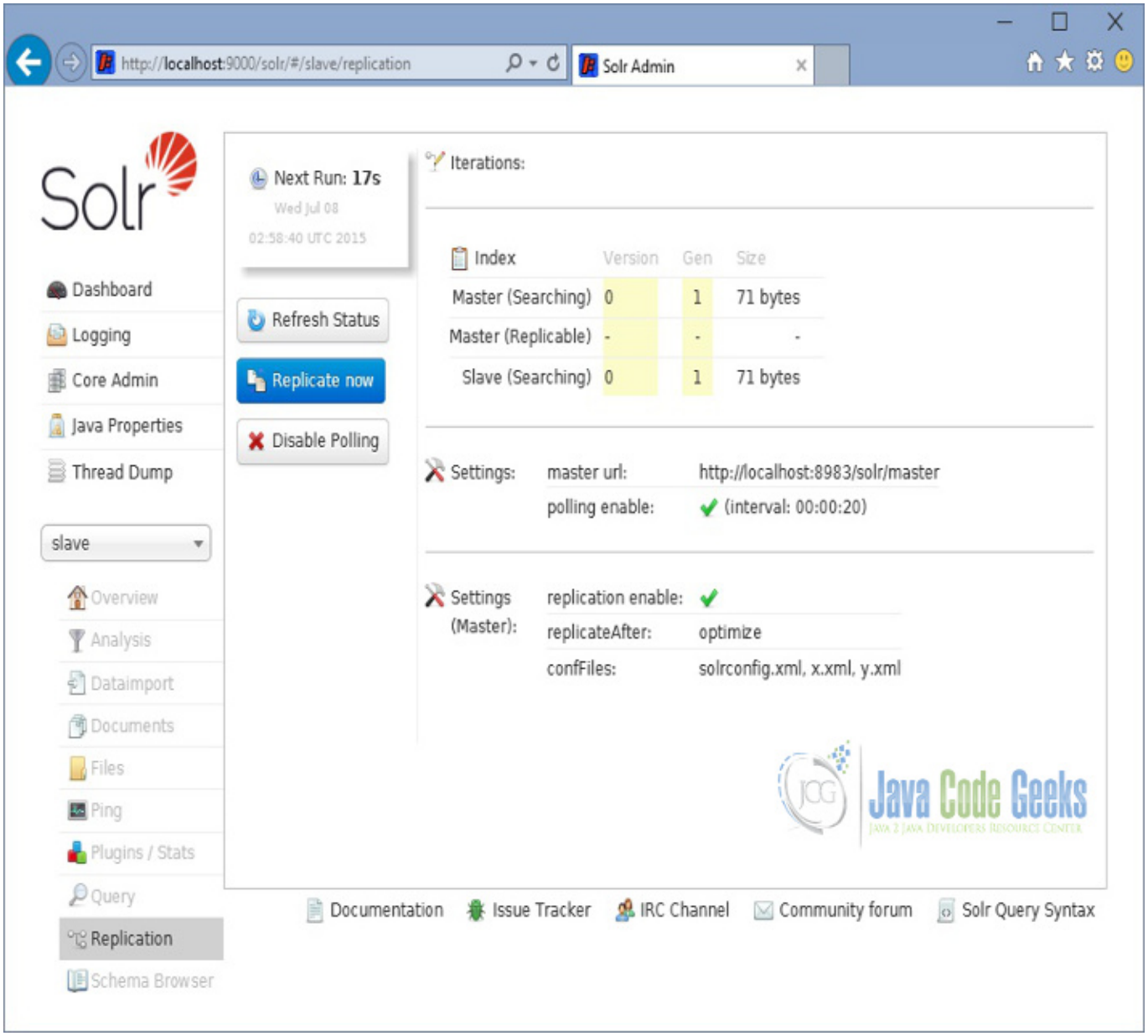

Figure 5.4: slave-1 replication console

To create another slave server, follow the same steps and configure the server in port 9001. We can now open the console using the following URL and validate the configuration in the replication section.

```
https://localhost:9001/solr/#/slave/replication
```

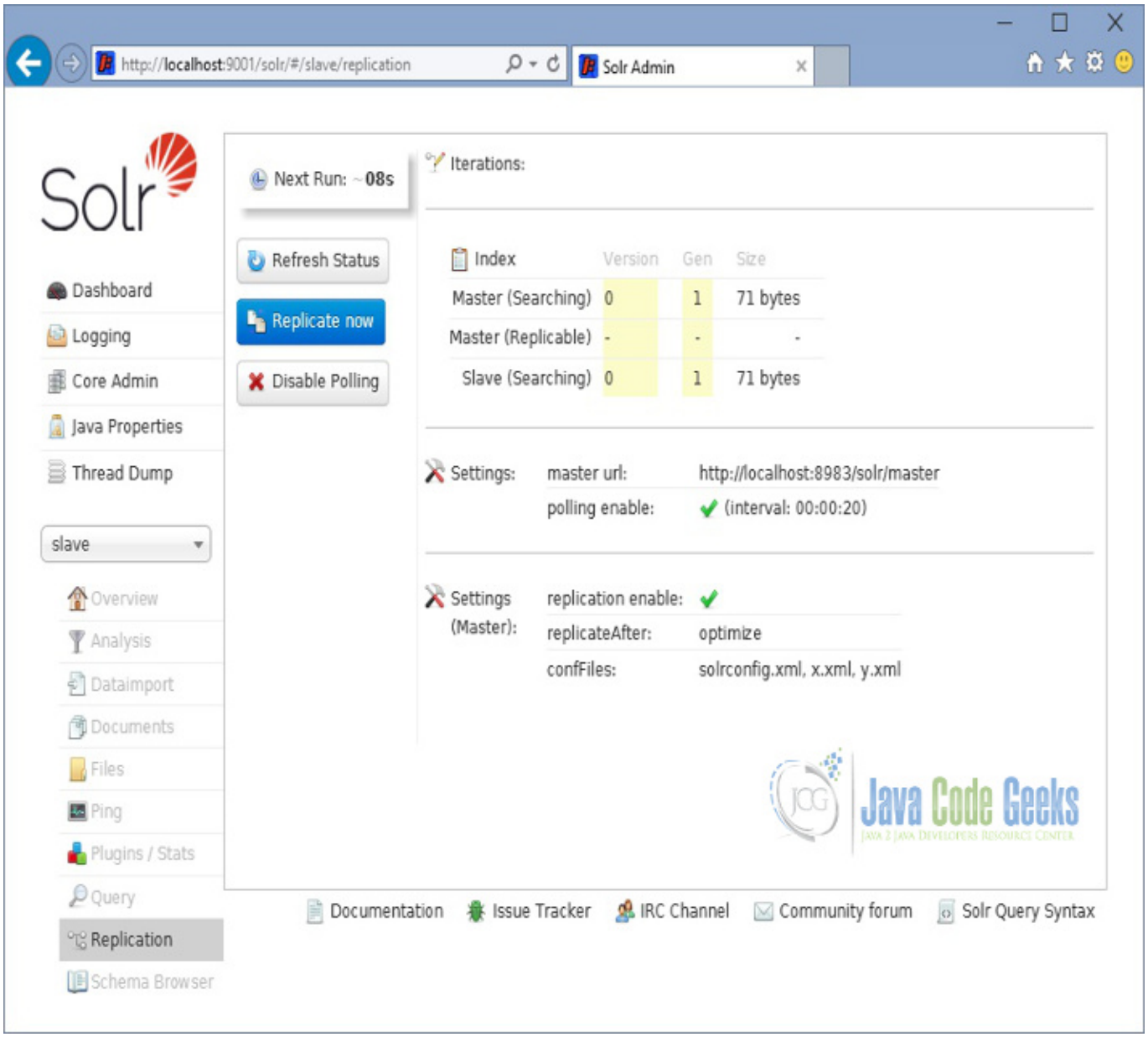

Figure 5.5: slave-2 replication console

5.4 Indexing and Replication

Now we will index the example data pointing to the master core. Apache Solr comes with a Standalone Java program called the SimplePostTool. This program is packaged into JAR and available with the installation under the folder `exampleexampledocs`.

Now we navigate to the `exampleexampledocs` folder in the command prompt and type the following command. You will see a bunch of options to use the tool.

```
java -jar post.jar -h
```

The usage format in general is as follows

```
Usage: java [SystemProperties] -jar post.jar [-h|-] [<file|folder|url|arg> [<file|folder|url|arg>...]]
```

As we said earlier, we will index the data present in the “books.csv” file shipped with Solr installation. We will navigate to the `solr-5.0.0exampleexampledocs` in the command prompt and issue the following command.

```
java -Dtype=text/csv -Durl=https://localhost:8983/solr/master/update -jar post.jar  books. ↩
    csv
```

The SystemProperties used here are:

- -Dtype - the type of the data file.
- -Durl - URL for the jcg core.

The file “books.csv” will now be indexed and the command prompt will display the following output.

```
SimplePostTool version 5.0.0
 Posting files to [base] url https://localhost:8983/solr/master/update using content-type  ↩
     text/csv...
 POSTing file books.csv to [base]
 1 files indexed.
 COMMITting Solr index changes to https://localhost:8983/solr/master/update...
 Time spent: 0:00:00.604
```

Now open the console of the slave cores and we can see the data replicated automatically.

`https://localhost:9000/solr/#/slave`

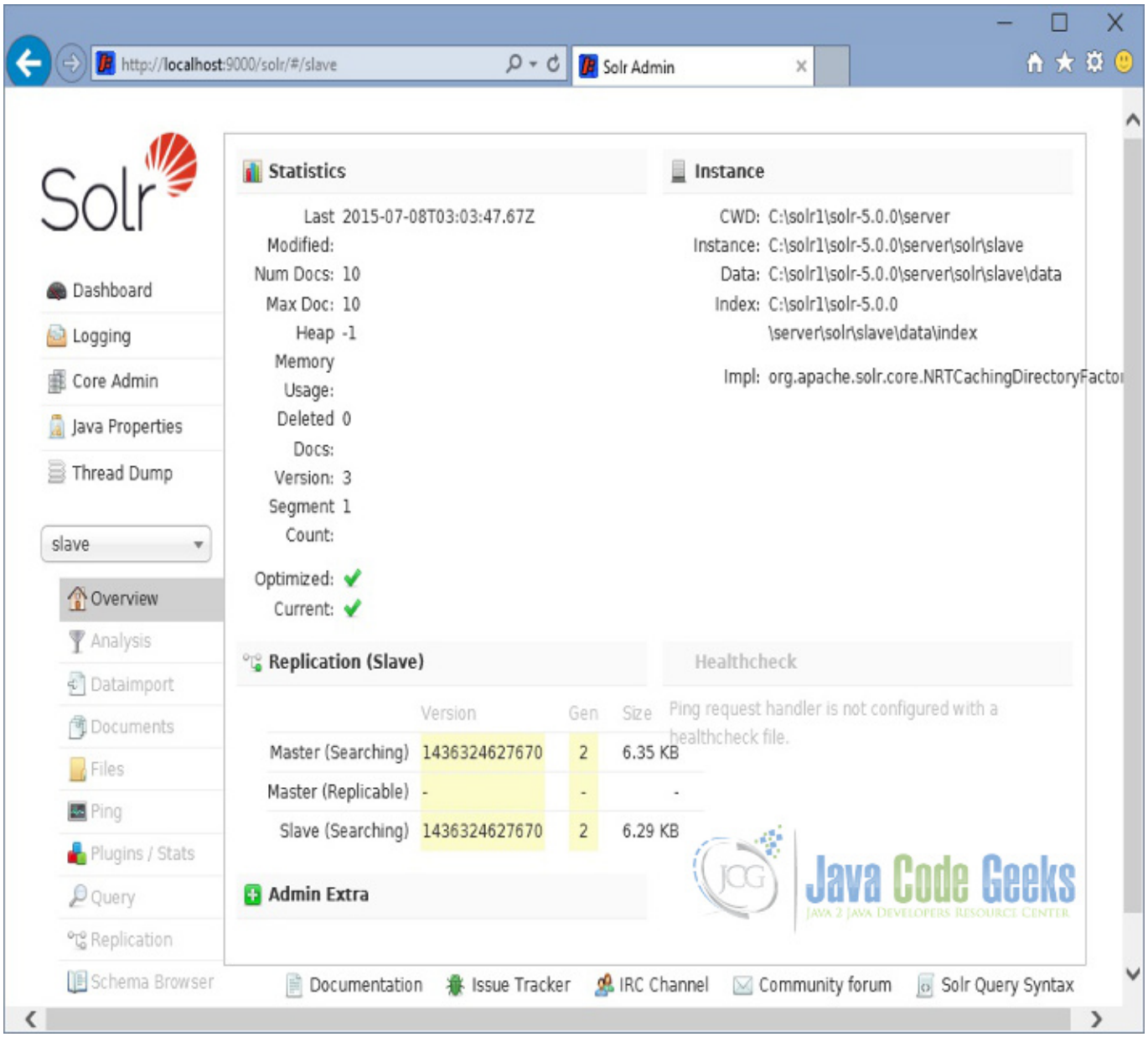

Figure 5.6: slave console - data replicated

5.5 Add new record

Now we validate the replication further by adding a record to the master core. To do it, lets open the master console URL.

`https://localhost:8983/solr/#/master/documents`

Navigate to the documents section and choose the document type as CSV and input the following content into the document text area and click on Submit.

```
id,cat,name,price,inStock,author,series_t,sequence_i,genre_s
123,book,Apache Solr,6.99,TRUE,Veera,JCG,1,Technical
```

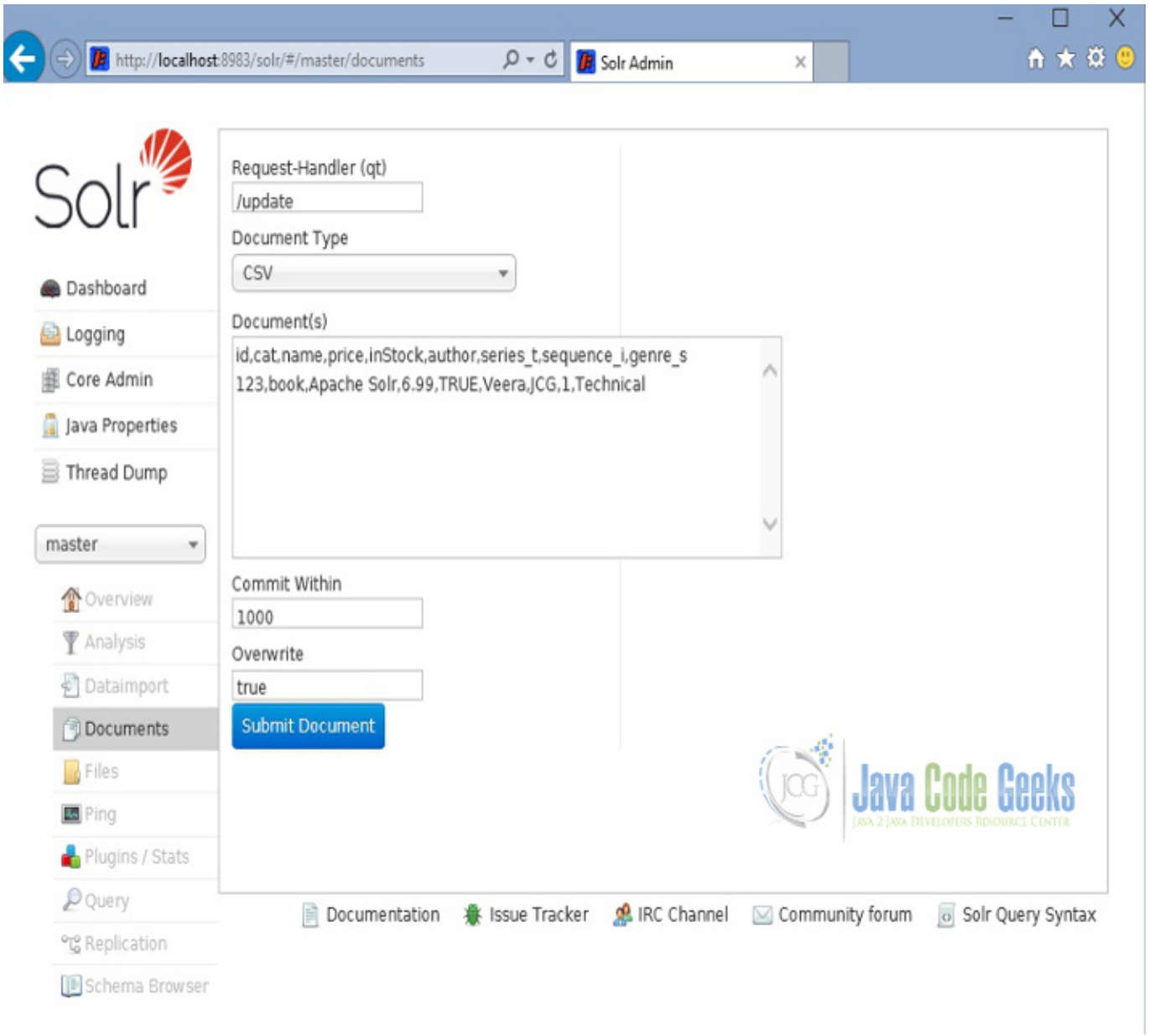

Figure 5.7: master console - add new record

The data will be added to master core and get replicated to the slave servers. To validate it lets navigate to the slave core. We can find the count of documents getting increased to 11. We can also use the query section in the slave admin console to validate it. Open the following URL.

`https://localhost:9000/solr/#/slave/query`

Input the values `name:apache` in the `q` text area and click on Execute Query. The new record we inserted on the master core will get reflected in the slave core.

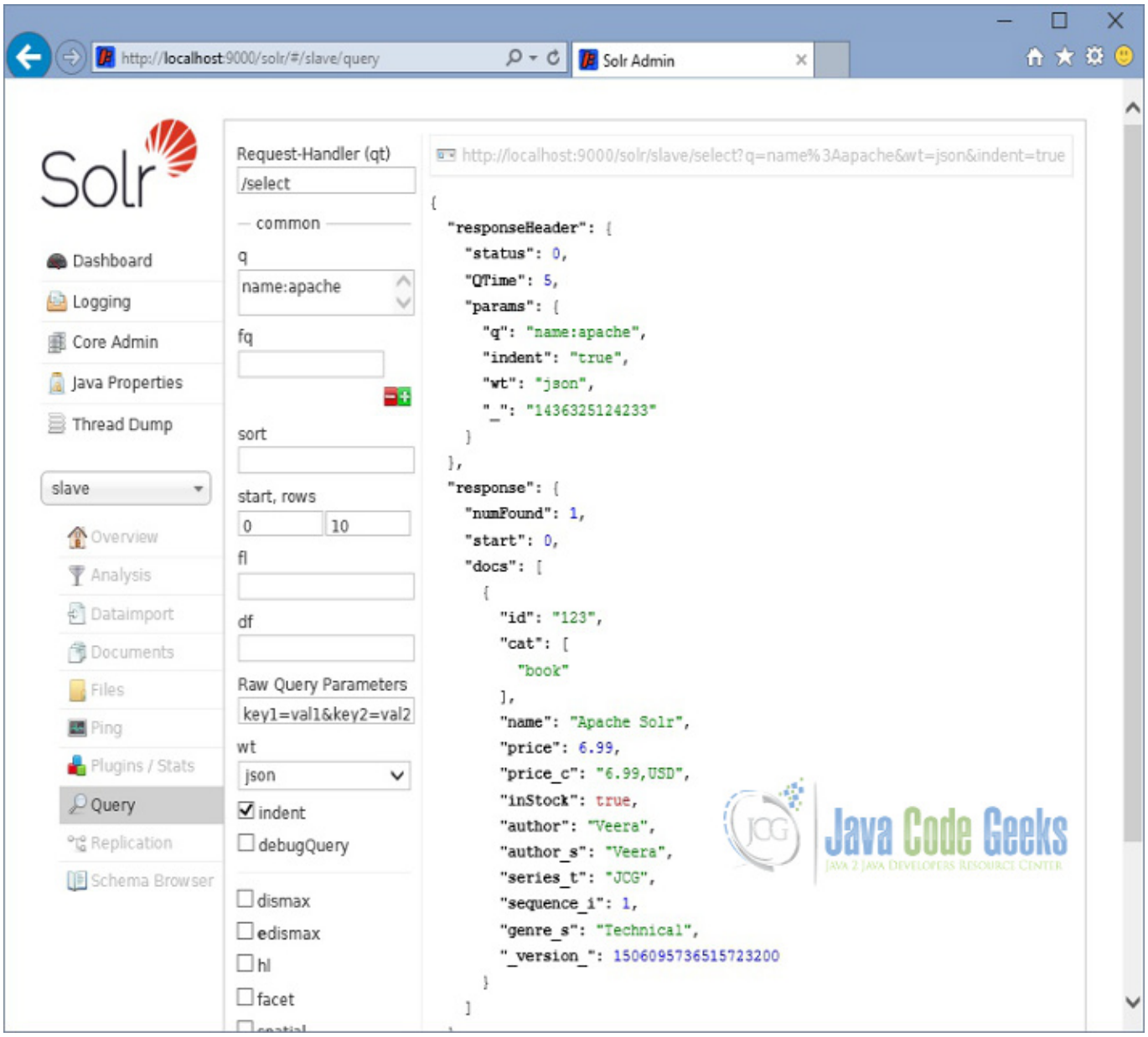

Figure 5.8: slave console - query

5.6 Download the Configuration

This was an example of Apache Solr replication.

Download

You can download the master configuration here: **solrconfig master** and slave configuration here: **solrconfig slave**

Chapter 6

Solr Synonyms Example

In this example of Solr Synonyms we will show you how to use the Solr synonym feature to substitute words with the relevant words of the data we index. This feature helps in providing better user experience by identifying different usage for a word in the given data context.

Solr ships with a filter factory called `SynonymFilterFactory` to achieve this functionality. Also, it provides a configuration file called `synonyms.txt` to add our synonyms. In this example, we will discuss how to configure the synonyms for our books data.

Our preferred environment for this example is solr-5.0.0. Before you begin the Solr installation make sure you have JDK installed and`Java_Home` is set appropriately.

6.1 Install Apache Solr

To begin with lets download the latest version of Apache Solr from the following location.

`https://lucene.apache.org/solr/downloads.html`

Apache Solr has gone through various changes from 4.x.x to 5.0.0, so if you have different version of Solr you need to download the 5.x.x. version to follow this example.

Once the Solr zip file is downloaded unzip it into a folder. The extracted folder will look like the below.

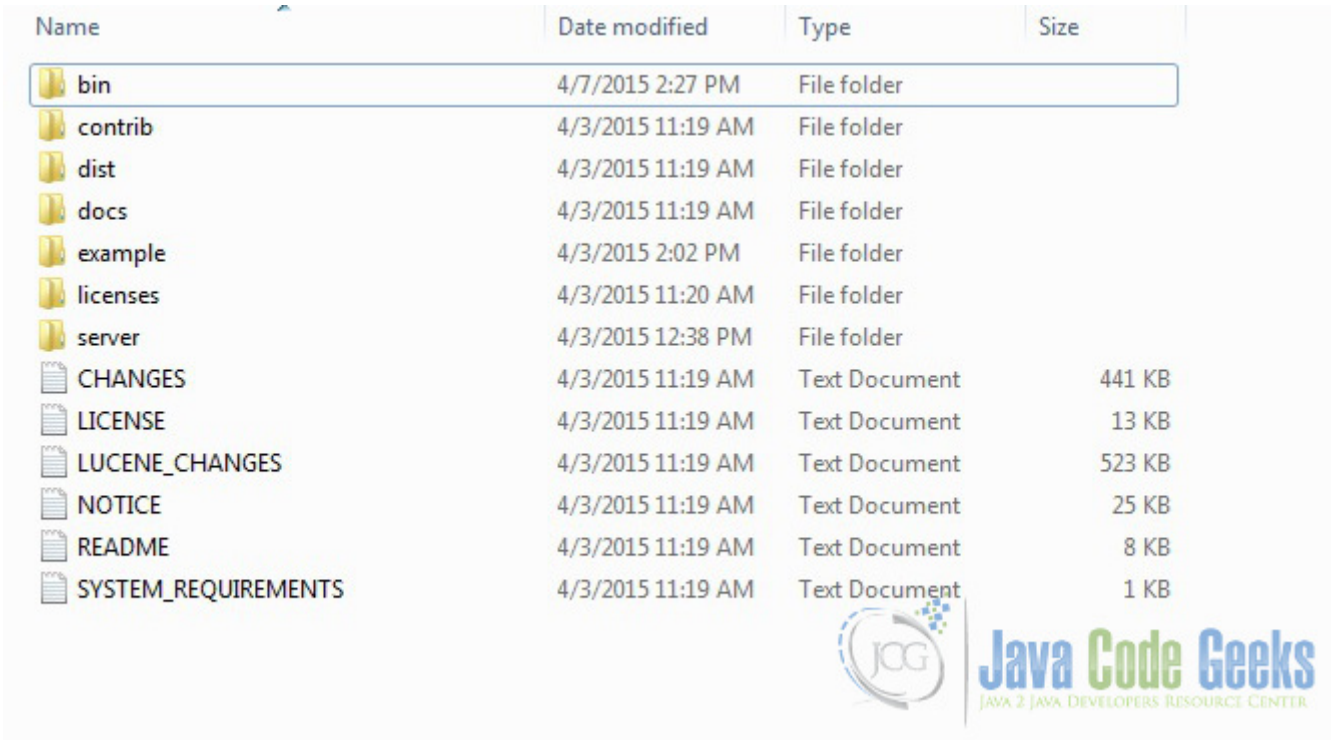

Figure 6.1: Solr folders

The `bin` folder contains the scripts to start and stop the server. The `example` folder contains few example files. We will be using one of them to demonstrate how Solr indexes the data. The `server` folder contains the `logs` folder where all the Solr logs are written. It will be helpful to check the logs for any error during indexing. The `solr` folder under server holds different collection or core. The configuration and data for each of the core/ collection are stored in the respective core/ collection folder.

Apache Solr comes with an inbuilt Jetty server. But before we start the solr instance we must validate the JAVA_HOME is set on the machine.

We can start the server using the command line script. Lets go to the bin directory from the command prompt and issue the following command:

```
solr start
```

This will start the Solr server under the default port 8983.

We can now open the following URL in the browser and validate that our Solr instance is running. The specifics of solr admin tool is beyond the scope of the example.

```
https://localhost:8983/solr/
```

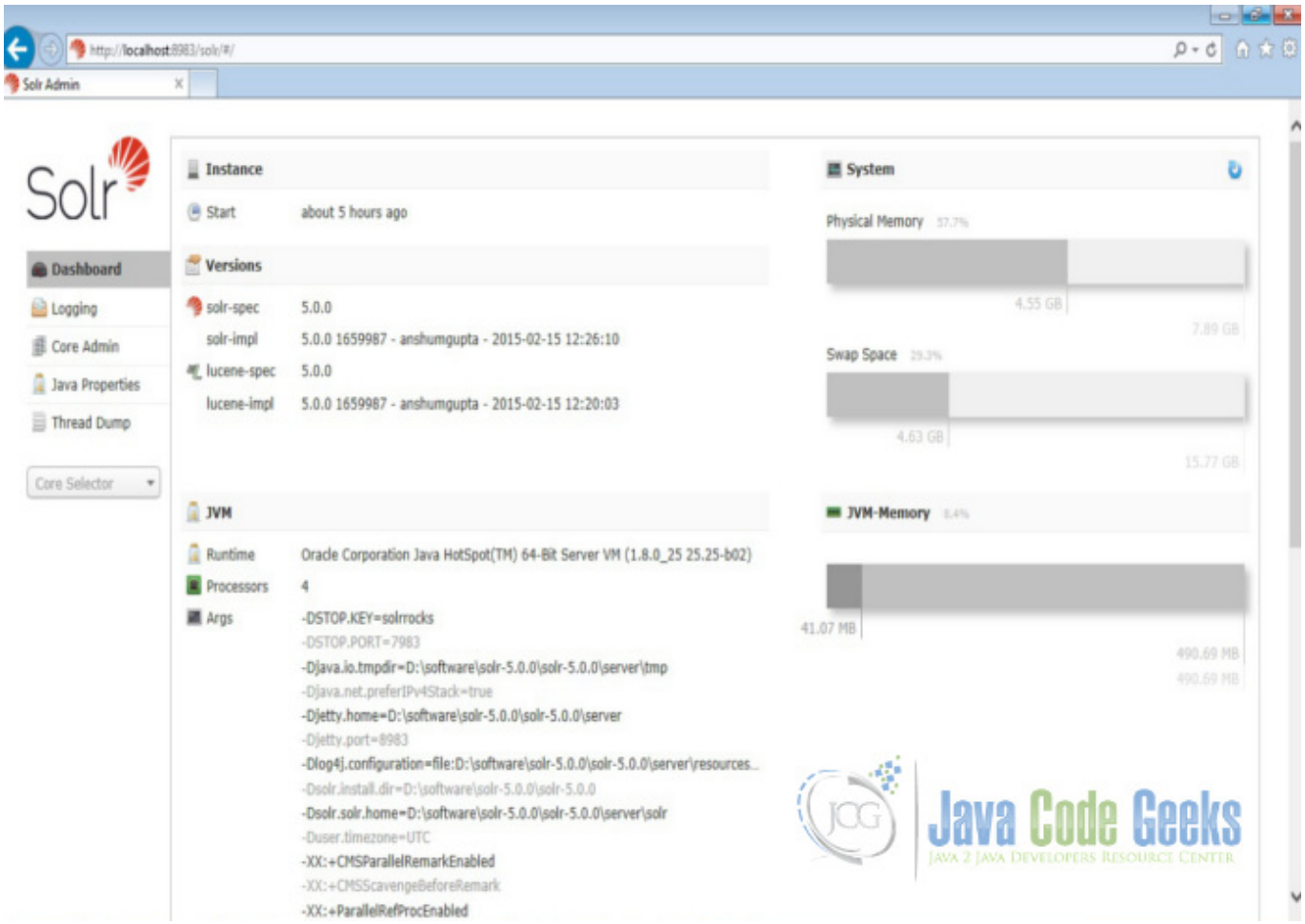

Figure 6.2: Solr admin console

6.2 Configuring Apache Solr

In this section, we will show you how to configure the core/collection for a solr instance and how to define the fields. Apache Solr ships with an option called Schemaless mode. This option allow users to construct effective schema without manually editing the schema file. For this example we will use the reference configset `sample_techproducts_configs`.

First, we need to create a Core for indexing the data. The Solr create command has the following options:

- **-c <name>** - Name of the core or collection to create (required).
- **-d <confdir>** - The configuration directory, useful in the SolrCloud mode.
- **-n <configName>** - The configuration name. This defaults to the same name as the core or collection.
- **-p <port>** - Port of a local Solr instance to send the create command to; by default the script tries to detect the port by looking for running Solr instances.
- **-s <shards>** - Number of shards to split a collection into, default is 1.
- **-rf <replicas>** - Number of copies of each document in the collection. The default is 1.

In this example we will use the -c parameter for core name and -d parameter for the configuration directory. For all other parameters we make use of default settings.

Now navigate the `solr-5.0.0bin` folder in the command window and issue the following command.

```
solr create -c jcg -d sample_techproducts_configs
```

We can see the following output in the command window.

```
Creating new core 'jcg' using command:
 https://localhost:8983/solr/admin/cores?action=CREATE&name=jcg&instanceDir=jcg{
 "responseHeader":{
 "status":0,
 "QTime":1377},
 "core":"jcg"}
```

Now we navigate to the following URL and we can see jcg core being populated in the core selector. You can also see the statistics of the core.

```
https://localhost:8983/solr
```

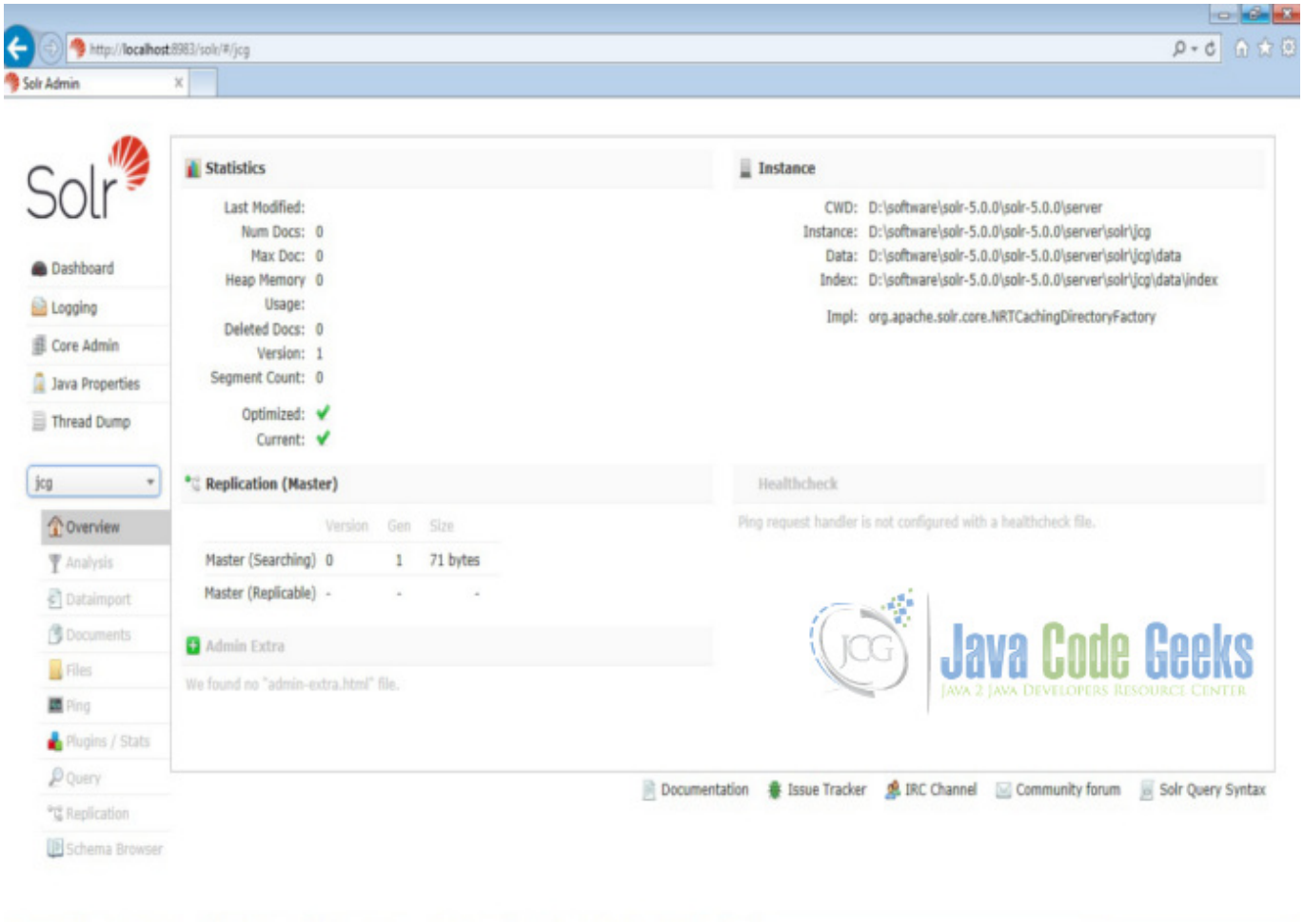

Figure 6.3: Solr jcg core

6.3 Indexing the Data

Apache Solr comes with a Standalone Java program called the SimplePostTool. This program is packaged into JAR and available with the installation under the folder `exampleexampledocs`.

Now we navigate to the `exampleexampledocs` folder in the command prompt and type the following command. You will see a bunch of options to use the tool.

```
java -jar post.jar -h
```

The usage format in general is as follows:

```
Usage:java [SystemProperties] -jar post.jar [-h|-] [<file|folder|url|arg> [<file|folde
r|url|arg>...]]
```

As we said earlier, we will index the data present in the "books.csv" file shipped with Solr installation. We will navigate to the `solr-5.0.0exampleexampledocs` in the command prompt and issue the following command.

```
java -Dtype=text/csv -Durl=https://localhost:8983/solr/jcg/update -jar post.jar  books.csv
```

The SystemProperties used here are:

- -Dtype - the type of the data file.
- -Durl - URL for the jcg core.

The file "books.csv" will now be indexed and the command prompt will display the following output.

```
SimplePostTool version 5.0.0
 Posting files to [base] url https://localhost:8983/solr/jcg/update using content-type text ↩
    /csv...
 POSTing file books.csv to [base]
 1 files indexed.
 COMMITting Solr index changes to https://localhost:8983/solr/jcg/update...
 Time spent: 0:00:00.604
```

6.4 Configure synonym

Now we modify the `synonyms.txt` file located under the folder `serversolrjcgconf` to add the synonym for our data. There are two ways to specify synonym mappings as listed below. We will discuss both the options with example.

- Two comma-separated lists of words with the symbol "⇒" between them. If the token matches any word on the left, then the list on the right is substituted. The original token will not be included unless it is also in the list on the right.
- A comma-separated list of words. If the token matches any of the words, then all the words in the list are substituted, which will include the original token.

6.4.1 With symbol "⇒"

We will first set up the synonym for correcting the spelling. Open the synonyms.txt file and add common spelling mistakes happens to the context of the data. In this example we will take the word *the*.

synonyms.txt

```
# Synonym mappings can be used for spelling correction too
pixima => pixma
teh => the
```

Since we have modified the configuration we have to restart the Solr server. To do so, issue the following commands:

```
solr stop -all
```

```
solr start
```

Now we query the books with wrong spelling as *teh*. Open the following URL:

```
https://localhost:8983/solr/jcg/select?q=name:"teh"
```

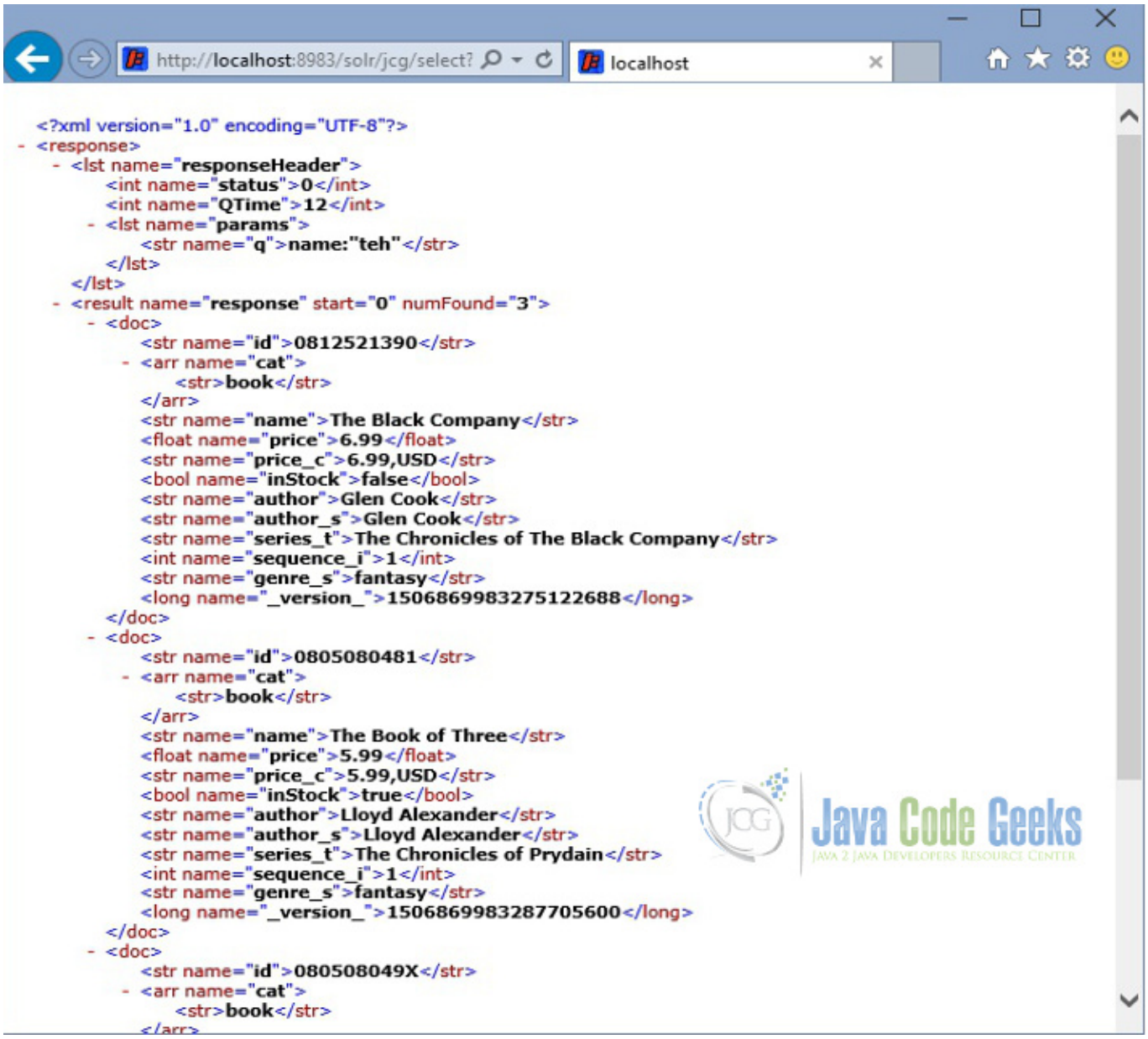

Figure 6.4: Solr Synonym - Output 1

6.4.2 Comma-separated list

Now let's implement another feature of Solr synonym. We will provide list of synonym for a word (clash in our case). When the user types any of the relevant word, the book with title `clash` will be returned. Similarly, we can add MB for MegaByte, GB for GigaByte etc depending on the context of the data we need to index.

When we perform the Solr query, each token is looked up in the list of synonyms and if a match is found then the synonym is emitted in place of the token. The position value of the new tokens are set such that they all occur at the same position as the original token.

synonyms.txt

```
# Some synonym groups specific to this example
```

```
GB,gib,gigabyte,gigabytes
MB,mib,megabyte,megabytes
Television, Televisions, TV, TVs

clash, battle, fight
```

Since we have modified the configuration we have to do a restart. Issue the following commands:

`solr stop -all`

`solr start`

Now query the books for title `fight` or `battle`, it would fetch the book "`A Clash of Kings`".

`https://localhost:8983/solr/jcg/select?q=name:"A fight"`

Figure 6.5: Solr Synonym - Output 2

6.5 Schema configuration

The configuration to use synonym is located in the file called `schema.xml` in the Solr server. To view the configuration let's open the file from the location `serversolrjcgconf` and take a look at the following section. You can notice we have used SynonymFilterFactory filter for the fieldType text_general. Also we can notice it is only used during the query time.

schema.xml

```
<!-- A general text field that has reasonable, generic
 cross-language defaults: it tokenizes with StandardTokenizer,
 removes stop words from case-insensitive "stopwords.txt"
 (empty by default), and down cases. At query time only, it
 also applies synonyms. -->
 <fieldType name="text_general" class="solr.TextField" positionIncrementGap="100">
 <analyzer type="index">
 <tokenizer class="solr.StandardTokenizerFactory"/>
 <filter class="solr.StopFilterFactory" ignoreCase="true" words="stopwords.txt" />
 <!-- in this example, we will only use synonyms at query time
 <filter class="solr.SynonymFilterFactory" synonyms="index_synonyms.txt" ignoreCase="true"  ↩
    expand="false"/>
 -->
 <filter class="solr.LowerCaseFilterFactory"/>
 </analyzer>
 <analyzer type="query">
 <tokenizer class="solr.StandardTokenizerFactory"/>
 <filter class="solr.StopFilterFactory" ignoreCase="true" words="stopwords.txt" />
 <filter class="solr.SynonymFilterFactory" synonyms="synonyms.txt" ignoreCase="true" expand ↩
    ="true"/>
 <filter class="solr.LowerCaseFilterFactory"/>
 </analyzer>
 </fieldType>
```

6.6 Download the Configuration

This was an example of Apache Solr Synonym.

Download

You can download the synonym configuration here: **synonyms.txt**

Chapter 7

Solr Faceted Search Example

In this example of Solr faceted search, we will discuss about the use of faceting the data and also discuss different facet options available in Solr. For our discussion, we will be using one of the collection example (techproducts) that comes with the Solr Installation for easy set up. We will show you how to make use of the Solr facet parameters to achieve the desired search results.

Our preferred environment for this example is Windows. Before you begin the Solr installation make sure you have JDK installed and Java_Home is set appropriately.

7.1 Installing Apache Solr

To begin with, lets download the latest version of Apache Solr from the following location:

`https://lucene.apache.org/solr/downloads.html`

As of this writing, the stable version available is 5.0.0. Apache Solr has gone through various changes from 4.x.x to 5.0.0, so if you have different version of Solr you need to download the 5.x.x. version to follow this example.

Once the Solr zip file is downloaded unzip it into a folder. The extracted folder will look like the below.

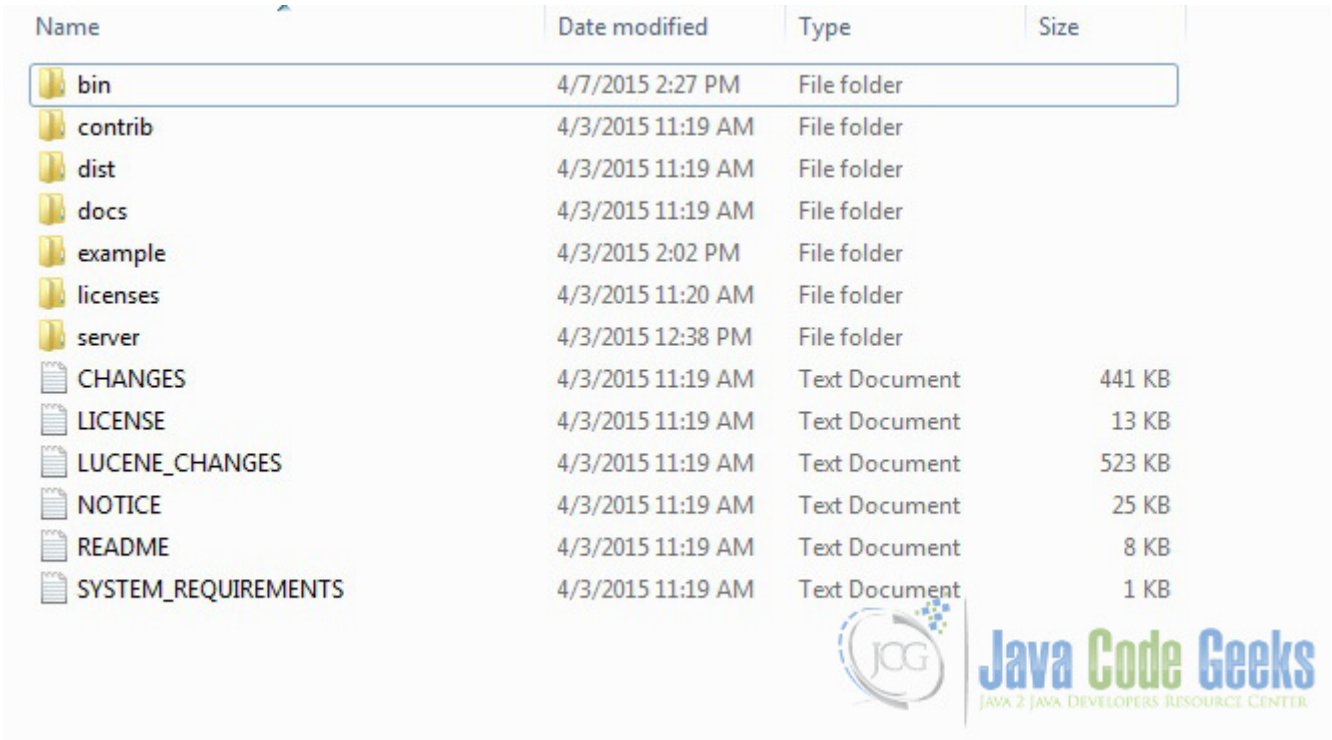

Figure 7.1: Solr folders

The `bin` folder contains the scripts to start and stop the server. The `example` folder contains few example files. We will be using one of them to demonstrate how Solr indexes the data. The `server` folder contains the `logs` folder where all the Solr logs are written. It will be helpful to check the logs for any error during indexing. The `solr` folder under server holds different collection or core. The configuration and data for each of the core/ collection are stored in the respective core/ collection folder.

Apache Solr comes with an inbuilt Jetty server. But before we start the solr instance we must validate the JAVA_HOME is set on the machine.

7.2 Start Solr Server

Solr provides few useful collection example to learn about the key features. We will use the techproducts collection bundled with Solr for our discussion. To start the Solr server with the techproducts collection let's open a command prompt, navigate to bin folder and issue the following syntax.

```
solr -e techproducts
```

This will start the Solr server under the default port 8983.

We can now open the following URL in the browser and validate that our Solr instance is running. You can also notice the collection techproducts being populated.

```
https://localhost:8983/solr/
```

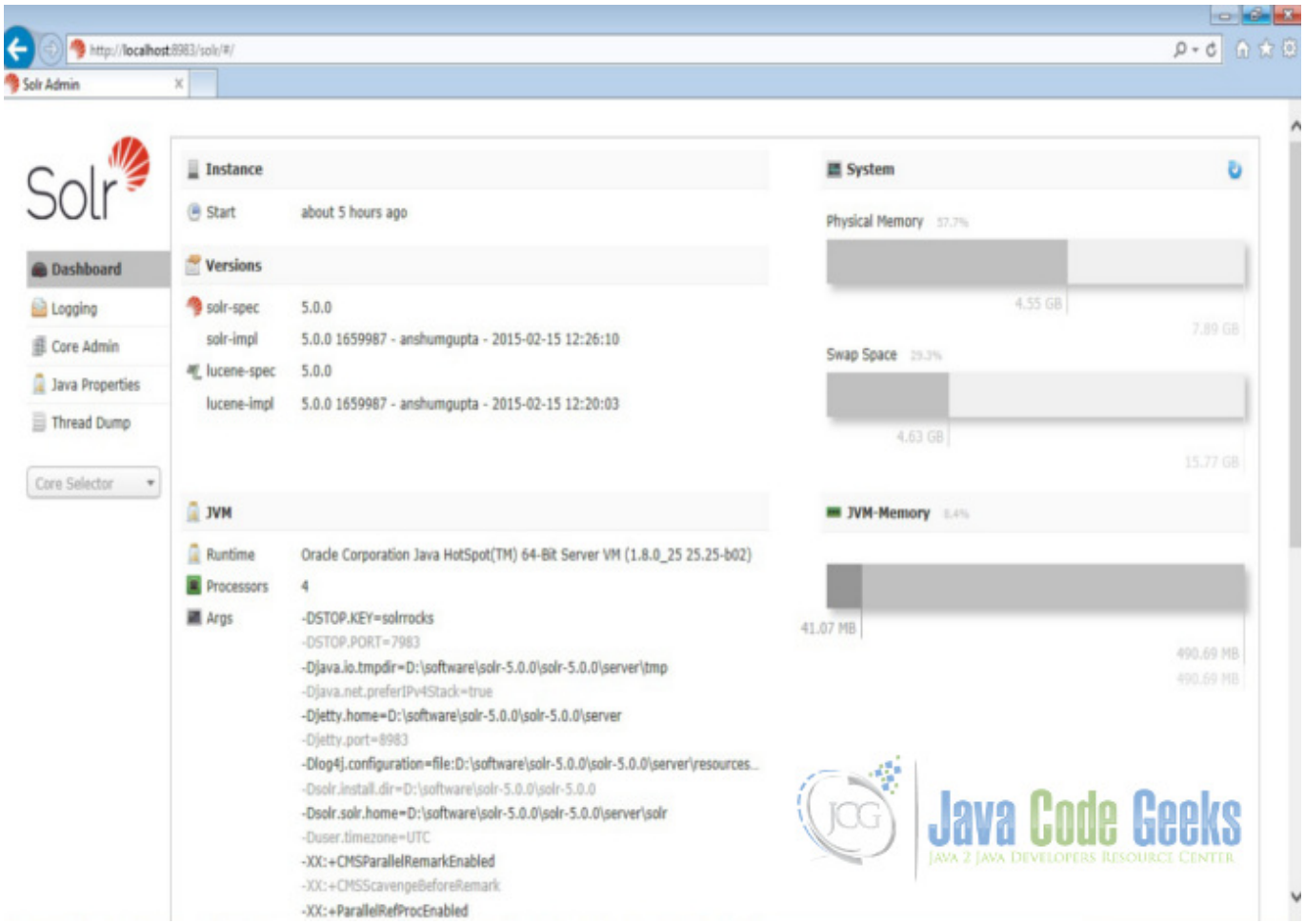

Figure 7.2: Solr admin console

7.3 Facet Search

Faceting is the process of arranging the search results into categories based on indexed terms. The output of the facet search is the numerical count found for each search term. This feature is very useful in providing better user experience during search by narrowing in on the results.

The following are the general parameters for facet.

- **facet** - If set to true, enables faceting.
- **facet.query** - Specifies a Lucene query to generate a facet count.

7.3.1 Field-Value Faceting

In this example we will set the facet value to true and set the facet.field parameter. The facet.field parameter identifies a field to be treated as a facet. The other parameters used in the query are Solr general query parameters and not related to faceting. To get more information on those parameters please look into our previous examples.

Now navigate to the following URL. This will bring the products with price range 0 to 400 and group the results by category.

```
https://localhost:8983/solr/techproducts/select?q=price:[0 TO 400]&fl=id,name,price&facet=true&facet.field=cat
```

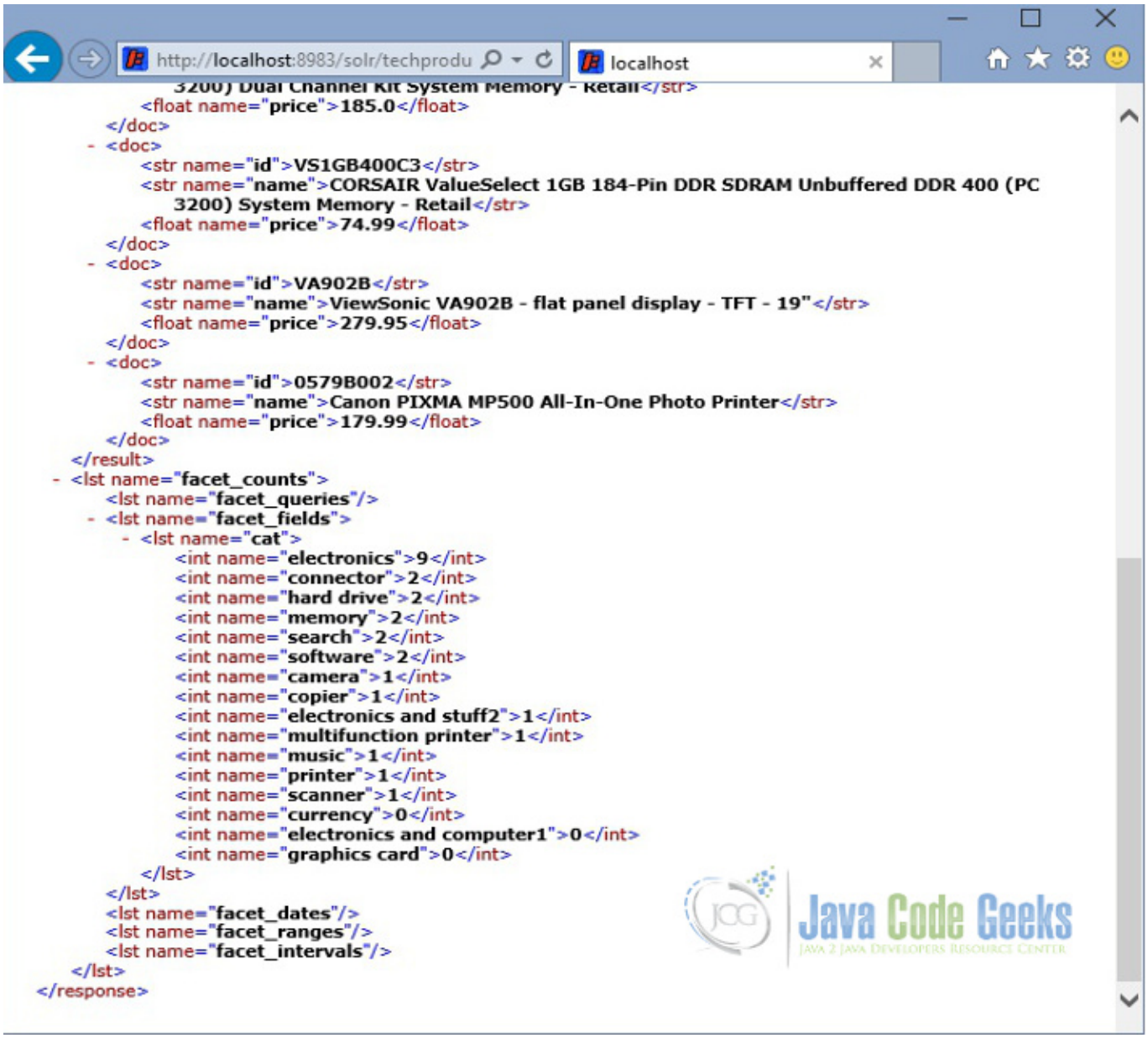

Figure 7.3: Solr Facet Field Value

There are other handful of facet parameters available to tune the search results when using Field-Value faceting.

7.3.2 Range Faceting

We can use range faceting on date or numeric fields that support range queries. This feature is very helpful in providing better user experience by bucketing the reference field in ranges. In this example we will use the price field to do the range faceting. The following parameters are used in the query.

- **facet.range** - Specifies the field to facet by range.
- **facet.range.start** - Specifies the start of the facet range.
- **facet.range.end** - Specifies the start of the facet range.

- **facet.range.gap** - Specifies the span of the range as a value to be added to the lower bound.

Now navigate to the following URL. This will bring the numerical products with price range bucketed into ranges of 100 for the results.

```
https://localhost:8983/solr/techproducts/select?q=price:[0 TO 4000]&fl=id,name,price&facet= ↩
    true&facet.field=cat&facet.range=price&f.price.facet.range.start=0.0&f.price.facet.range ↩
    .end=1000.0&f.price.facet.range.gap=100
```

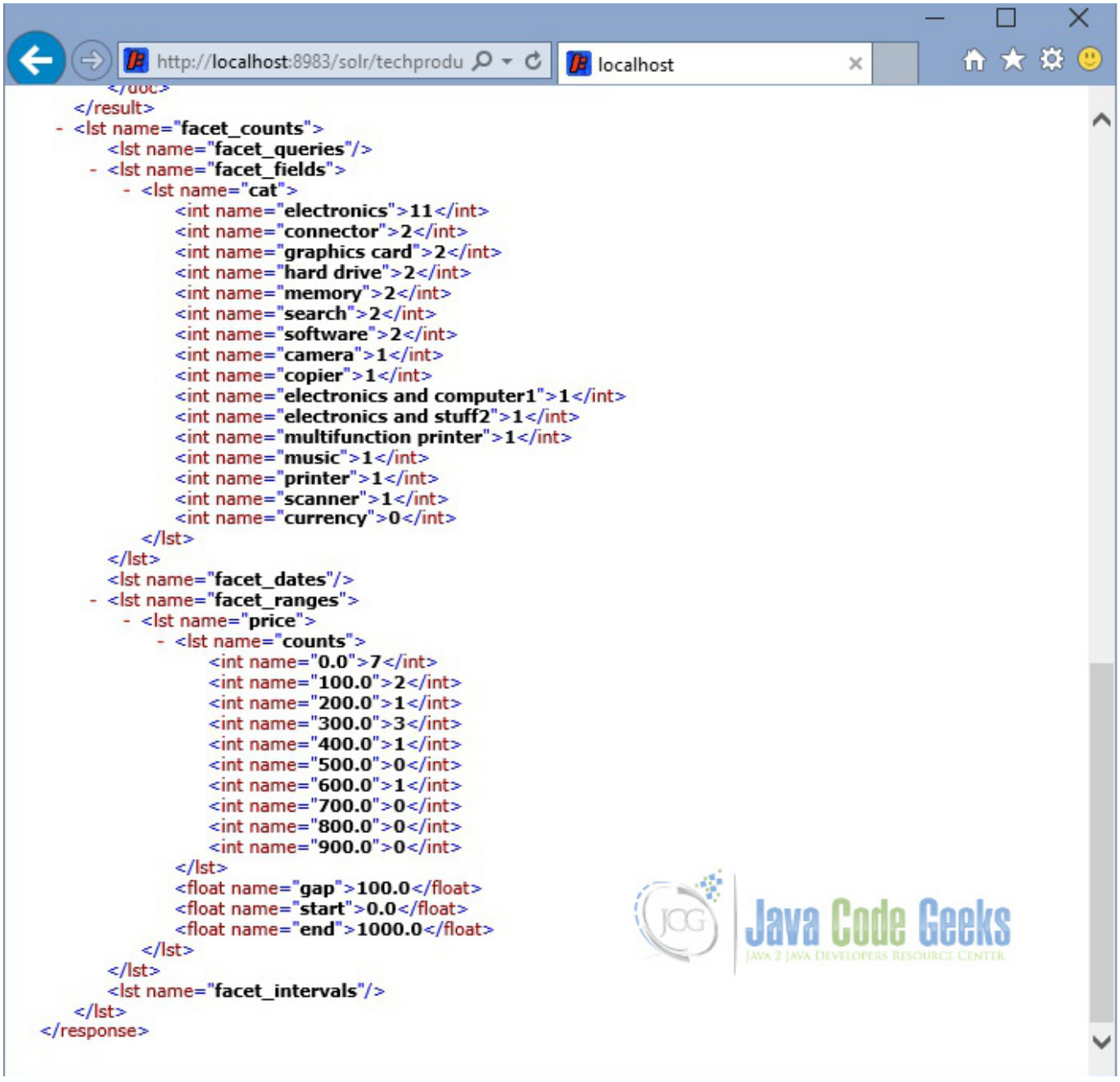

Figure 7.4: Solr Facet Range

7.3.3 Interval Faceting

Another feature in Solr is Interval faceting. This looks similar to Range faceting but Interval faceting gives options to set variable range as against the former which can set only a fixed gap. In order to use Interval Faceting on a field, it is required that the field has “docValues” enabled.

To modify the field lets navigate to `exampletechproductssolrtechproductsconf` and set the "docValues" attribute to true in the schema.xml file as shown below.

schema.xml

```
<field name="weight" type="float" indexed="true" stored="true"/>
 <field name="price" type="float" indexed="true" stored="true" docValues="true"/>
 <field name="popularity" type="int" indexed="true" stored="true" />
 <field name="inStock" type="boolean" indexed="true" stored="true" />
```

Since we have modified the configuration we have to restart the Solr instance. Open a command prompt, navigate to bin folder and issue the following commands.

```
solr stop -all
```

```
solr -e techproducts
```

In this example we will use the following faceting parameters.

- **facet.interval** - Specifies the field to facet by interval.
- **facet.interval.set** - Sets the intervals for the field.

We can use the following syntax to include or exclude the values provided in the set interval.

(1,10) → will include values greater than 1 and lower than 10.

[1,10) → will include values greater or equal to 1 and lower than 10.

[1,10] → will include values greater or equal to 1 and lower or equal to 10.

Now navigate to the following URL. This will bring the numerical count of the products for the intervals provided in the query.

```
https://localhost:8983/solr/techproducts/select?q=*:*&fl=id,name,price&facet=true&facet. ↩
    field=cat&facet.interval=price&f.price.facet.interval.set=[0,10]&f.price.facet.interval. ↩
    set=(10,100]
```

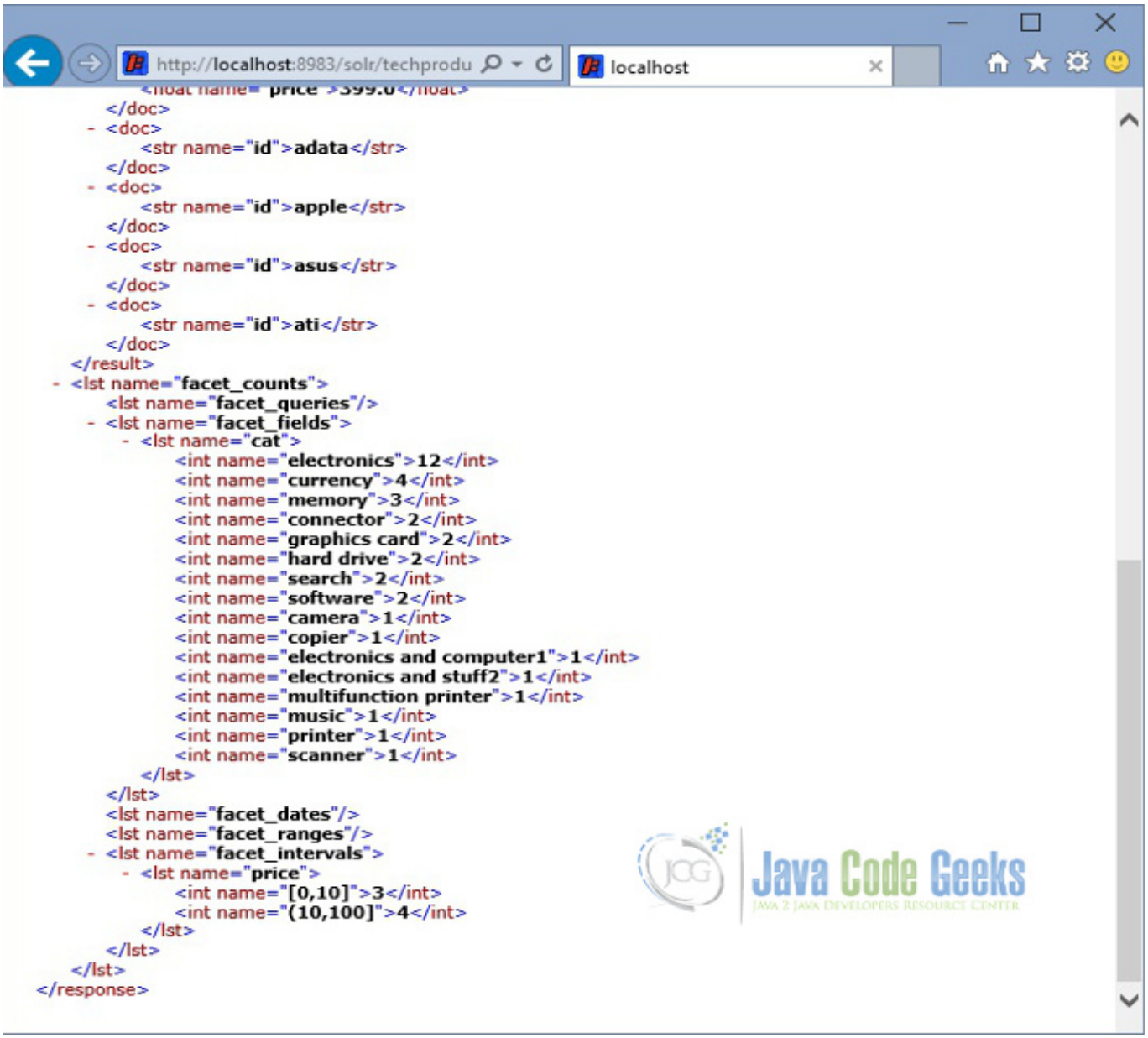

Figure 7.5: Solr Facet Interval

7.4 Download the Configuration

This was an example of solr faceted search.

Download

You can download the schema for the example here: **Schema.xml**

Chapter 8

Solr Filter Query Example

In this example of Solr filter query, we will discuss about how to implement filter queries functionality provided by Apache Solr. We will discuss how to use single and multiple filter queries to achieve the desired results. Also we will show the various filter query syntax offered by Solr and discuss the advantages of using one over other.

To demonstrate the filter query usage, we will create a core in Solr using basic configuration and index a sample file shipped along with Solr installation.

Our preferred environment for this example is solr-5.0.0. Before you begin the Solr installation make sure you have JDK installed and `Java_Home` is set appropriately.

8.1 Install Apache Solr

To begin with, lets download the latest version of Apache Solr from the following location:

`https://lucene.apache.org/solr/downloads.html`

Apache Solr has gone through various changes from 4.x.x to 5.0.0, so if you have a different version of Solr you need to download the 5.x.x. version to follow this example.

Once the Solr zip file is downloaded, unzip it into a folder. The extracted folder will look like the below:

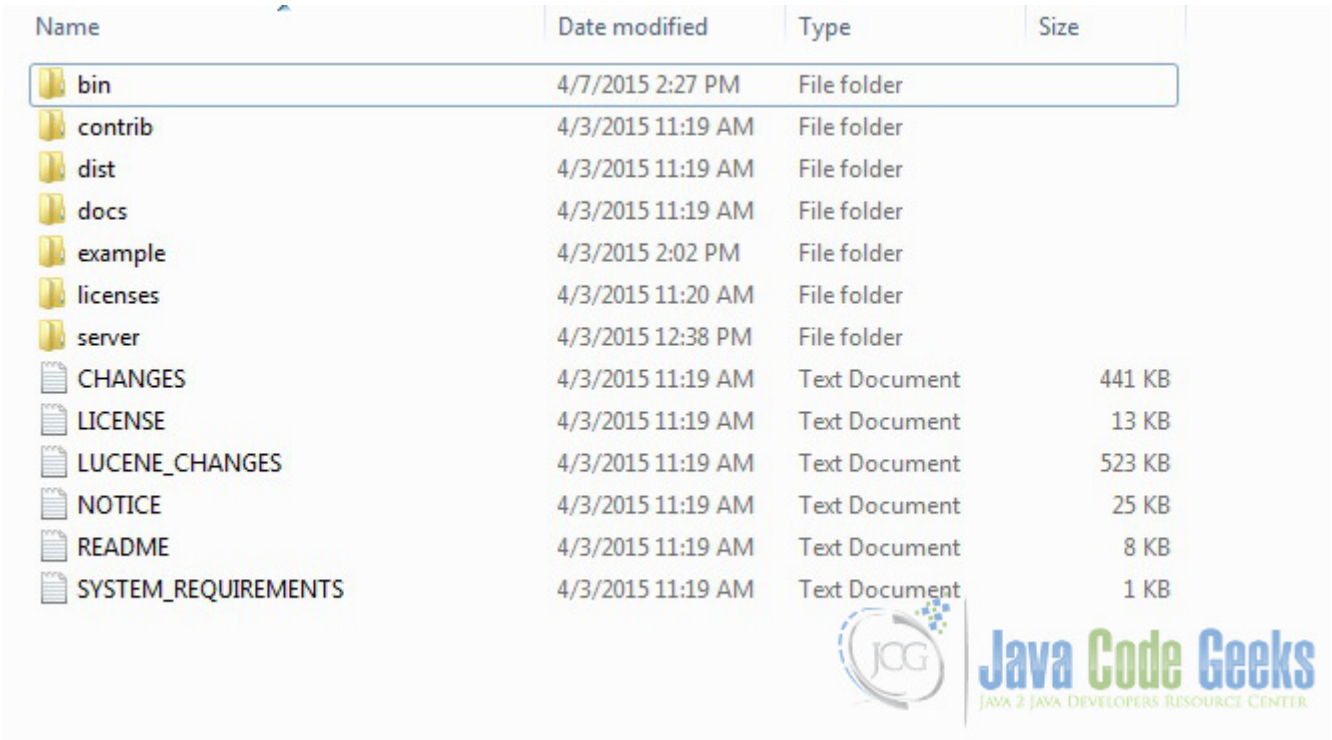

Figure 8.1: Solr folders

The `bin` folder contains the scripts to start and stop the server. The `example` folder contains few example files. We will be using one of them to demonstrate how Solr indexes the data. The `server` folder contains the `logs` folder where all the Solr logs are written. It will be helpful to check the logs for any error during indexing. The `solr` folder under server holds different collection or core. The configuration and data for each of the core/ collection are stored in the respective core/ collection folder.

Apache Solr comes with an inbuilt Jetty server. But before we start the solr instance we must validate the JAVA_HOME is set on the machine.

We can start the server using the command line script. Lets go to the bin directory from the command prompt and issue the following command:

```
solr start
```

This will start the Solr server under the default port 8983.

We can now open the following URL in the browser and validate that our Solr instance is running. The specifics of solr admin tool is beyond the scope of the example.

```
https://localhost:8983/solr/
```

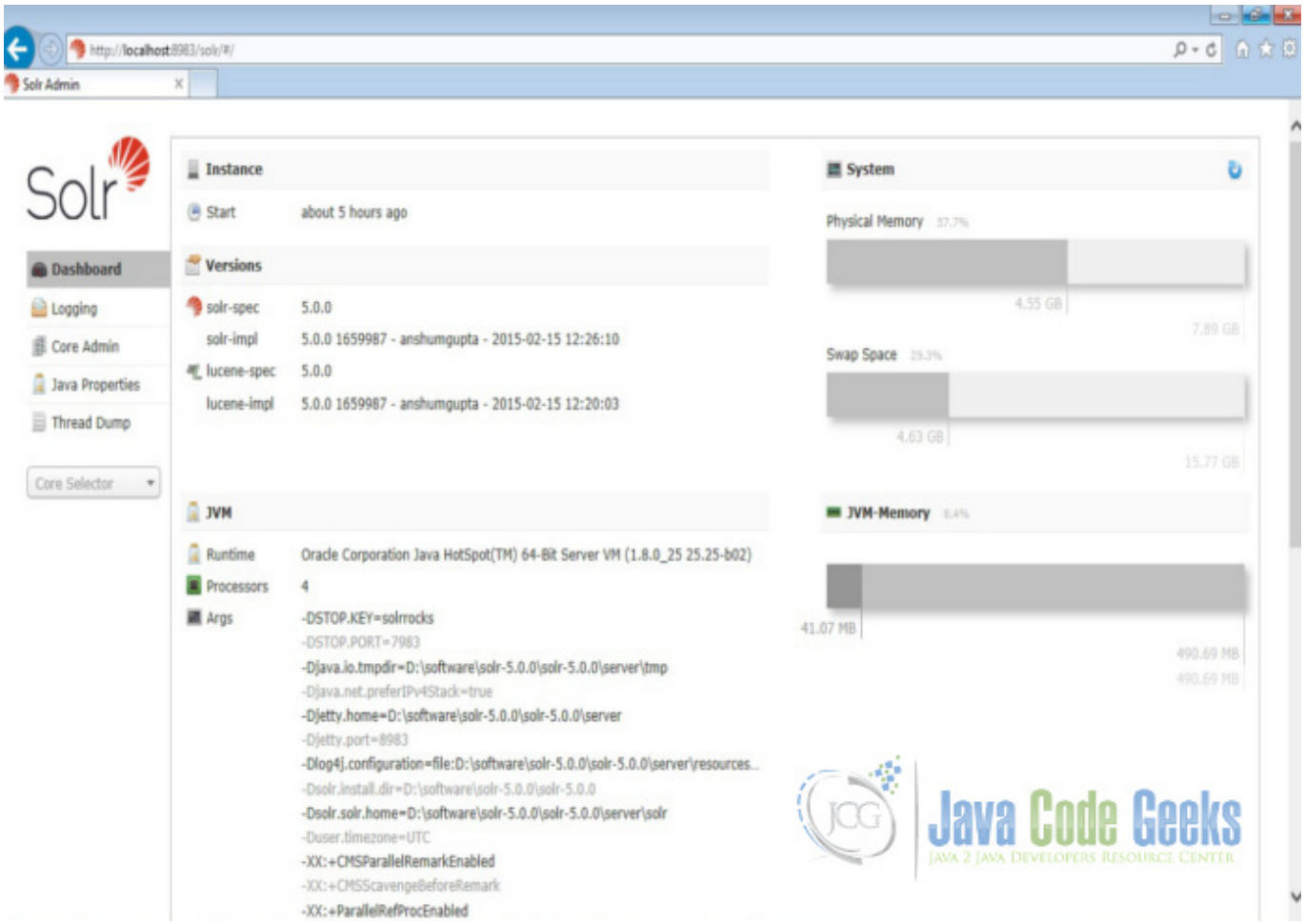

Figure 8.2: Solr admin console

8.2 Configuring Apache Solr

In this section, we will show you how to configure the core/collection for a solr instance and how to define the fields. Apache Solr ships with an option called Schemaless mode. This option allow users to construct effective schema without manually editing the schema file. But for this example we will use the Schema configuration for understanding the internals of the Solr.

8.2.1 Creating a Core

When the Solr server is started in Standalone mode, the configuration is called core and when it is started in SolrCloud mode, the configuration is called Collection. In this example we will discuss about the standalone server and core. We will park the SolrCloud discussion for later time.

First, we need to create a Core for indexing the data. The Solr create command has the following options:

- **-c <name>** - Name of the core or collection to create (required).
- **-d <confdir>** - The configuration directory, useful in the SolrCloud mode.
- **-n <configName>** - The configuration name. This defaults to the same name as the core or collection.
- **-p <port>** - Port of a local Solr instance to send the create command to; by default the script tries to detect the port by looking for running Solr instances.

- **-s <shards>** - Number of shards to split a collection into, default is 1.
- **-rf <replicas>** - Number of copies of each document in the collection. The default is 1.

In this example we will use the -c parameter for core name and -d parameter for the configuration directory. For all other parameters we make use of default settings.

Now navigate the `solr-5.0.0bin` folder in the command window and issue the following command:

```
solr create -c jcg -d basic_configs
```

We can see the following output in the command window.

```
Creating new core 'jcg' using command:
https://localhost:8983/solr/admin/cores?action=CREATE&name=jcg&instanceDir=jcg

{
 "responseHeader":{
 "status":0,
 "QTime":663},
 "core":"jcg"}
```

Now we navigate to the following URL and we can see jcg core being populated in the core selector. You can also see the statistics of the core.

```
https://localhost:8983/solr
```

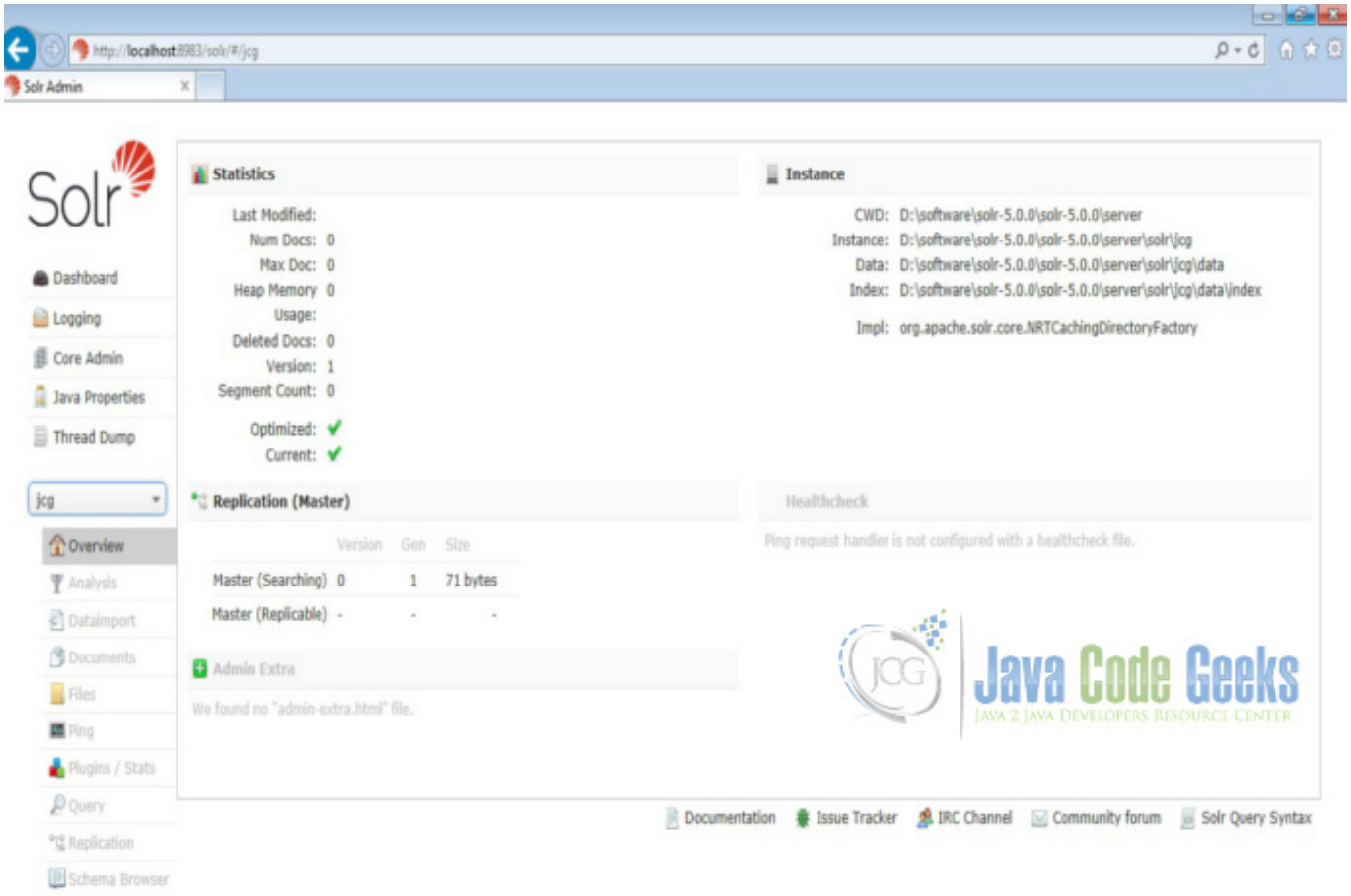

Figure 8.3: Solr jcg core

8.2.2 Modify the schema.xml file

We need to modify the `schema.xml` file under the folder `serversolrjcgconf` to include the fields. We will use one of the example file "books.csv" shipped along with Solr installation for indexing. The file is located under the folder `solr-5.0.0exampleexampledocs`

Now we navigate to the folder `serversolr` directory. You will see a folder called `jcg` created. The sub-folders namely`conf` and `data` have the core's configuration and indexed data respectively.

Now edit the `schema.xml` file in the `serversolrjcgconf` folder and add the following contents after the uniqueKey element.

schema.xml

```
<uniqueKey>id</uniqueKey>
<!-- Fields added for books.csv load-->
<field name="cat" type="text_general" indexed="true" stored="true"/>
<field name="name" type="text_general" indexed="true" stored="true"/>
<field name="price" type="tdouble" indexed="true" stored="true"/>
<field name="inStock" type="boolean" indexed="true" stored="true"/>
<field name="author" type="text_general" indexed="true" stored="true"/>
```

We have set the attribute `indexed` to true. This specifies the field is used for indexing and the record can be retrieved using the index. Setting the value to false will make the field only stored but can't be queried with.

Also note we have another attribute called `stored` and set it to true. This specifies the field is stored and can be returned in the output. Setting this field to false will make the field only indexed and can't be retrieved in output.

We have assigned the type for the fields present in the "books.csv" file here. The first field in the CSV file "id" is automatically taken care by the uniqueKey element of schema.xml file for indexing.

Since we have modified the configuration we have to stop and start the server. To do so, we need to issue the following command from bin directory through command line:

```
solr stop -all
```

The server will be stopped now. Now to start the server issue the following command from bin directory through command line:

```
solr start
```

8.3 Indexing the Data

Apache Solr comes with a Standalone Java program called the SimplePostTool. This program is packaged into JAR and available with the installation under the folder `exampleexampledocs`.

Now we navigate to the `exampleexampledocs` folder in the command prompt and type the following command. You will see a bunch of options to use the tool.

```
java -jar post.jar -h
```

The usage format in general is as follows:

<code>Usage: java [SystemProperties] -jar post.jar [-h|-] [<file|folder|url|arg> [<file|folder|url|arg>...]]</code>

As we said earlier, we will index the data present in the "books.csv" file shipped with Solr installation. We will navigate to the `solr-5.0.0exampleexampledocs` in the command prompt and issue the following command.

```
java -Dtype=text/csv -Durl=https://localhost:8983/solr/jcg/update -jar post.jar books.csv
```

The SystemProperties used here are:

- -Dtype - the type of the data file.
- -Durl - URL for the jcg core.

The file “books.csv” will now be indexed and the command prompt will display the following output.

```
SimplePostTool version 5.0.0
Posting files to [base] url https://localhost:8983/solr/jcg/update using content-
type text/csv...
POSTing file books.csv to [base]
1 files indexed.
COMMITting Solr index changes to https://localhost:8983/solr/jcg/update...
Time spent: 0:00:00.647
```

8.4 Filter queries

Solr provides the following parameter to filter the queries. This parameter can be used with other common query parameters to achieve the desired output.

- **fq** - Applies a filter query to the search results.

The fq parameter defines a query that can be used to restrict the superset of documents that can be returned, without influencing score. The fq parameterized queries are cached independent of the main query. When the same filter is used in the subsequent queries its a cache hit and the data is returned quickly from the cache.

8.4.1 Single filter query

Let’s form a query to search for the keyword Game in the name field. Open the following URL in the browser. This query will fetch two records as shown in the screenshot.

```
https://localhost:8983/solr/jcg/select?q=name
```

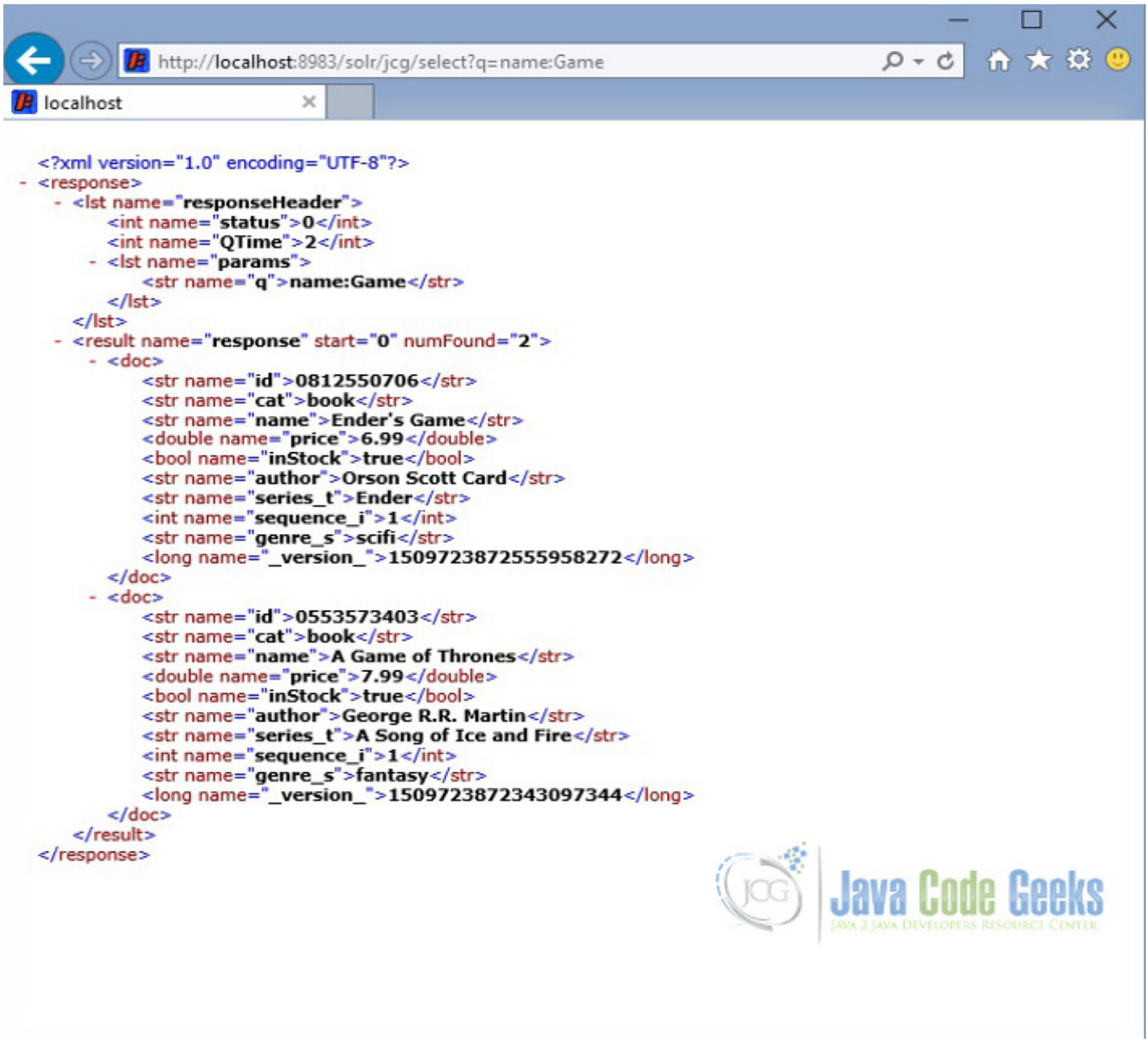

Figure 8.4: Without filter

We will modify the query to filter the result for the book's price between 1.00 to 7.00. With the filter parameter we will get only a single record.

Open the following URL in the browser:

```
https://localhost:8983/solr/jcg/select?q=name:Game&fq=price:[1.00 TO 7.00]
```

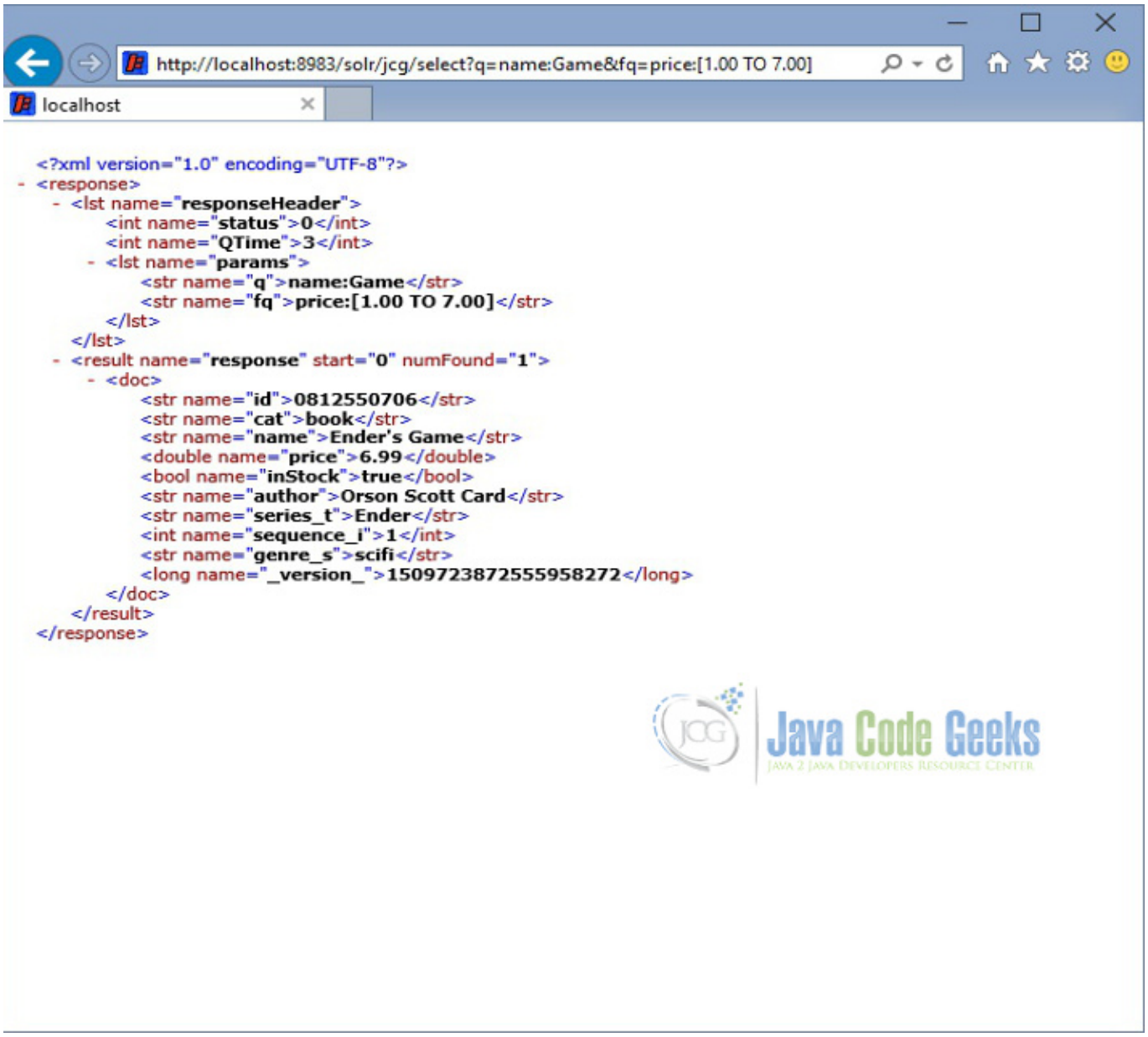

Figure 8.5: Single filter

8.4.2 Multiple filters

The fq parameter can be specified multiple times in a query. The documents will only be included in the result if they are in the intersection of the document sets resulting from each instance of the parameter. In the example below, only documents which are in price between 1.00 and 7.00 and also in stock will be returned.

Open the following URL in the browser.

```
https://localhost:8983/solr/jcg/select?q=name:The&fq=price:[1.00 TO 7.00]&fq=inStock:true
```

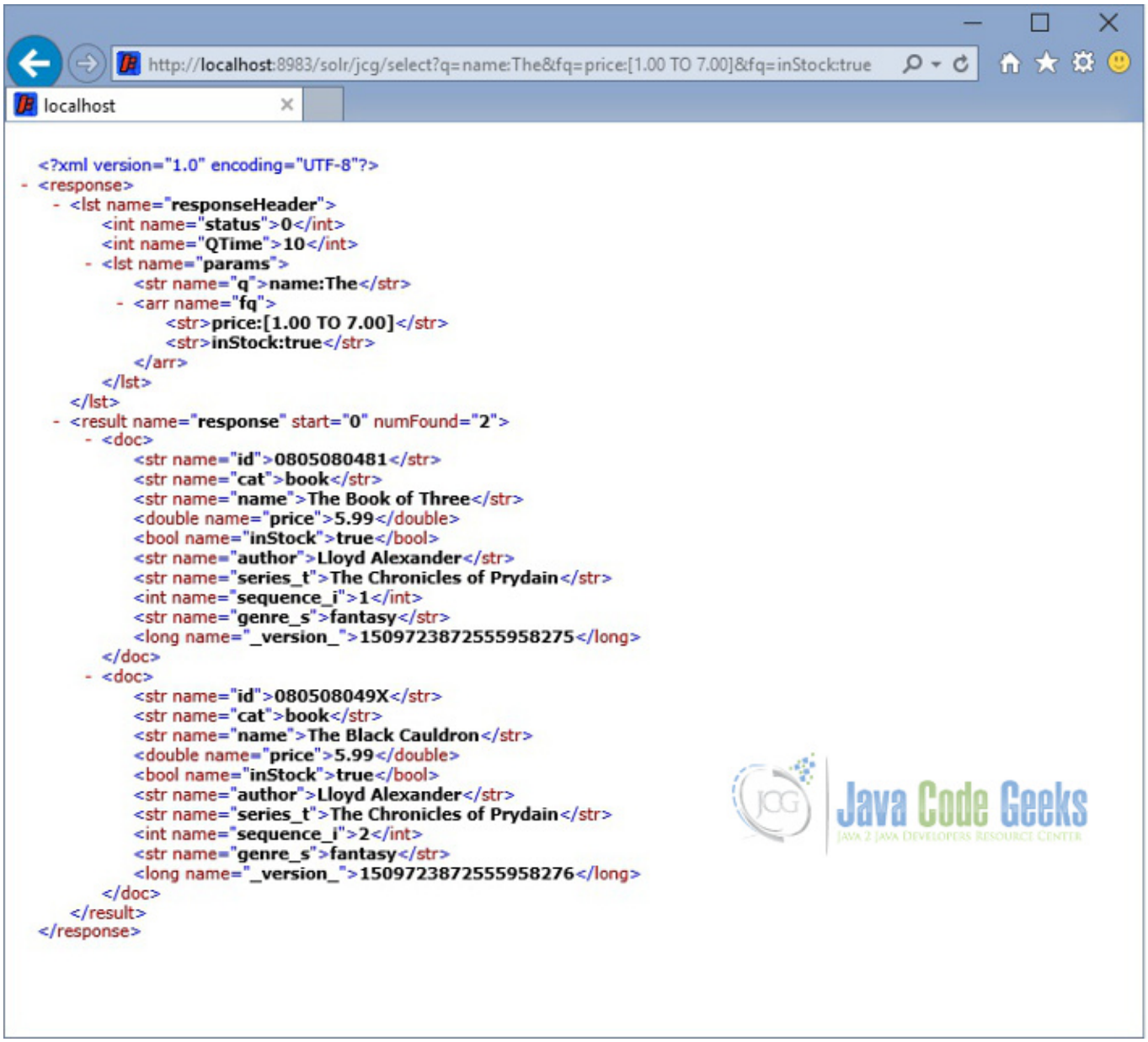

Figure 8.6: Multiple filter

Since the cache works on individual parameters it is suggested to use multiple fq parameters for better caching.

8.5 Download the source code

This was an example of solr filter queries.

Download the Configuration

You can download the schema file of this example here: **filter_schema**