

Spring Data Solr

Christoph Strobl, Oliver Gierke, Mark Pollack, Thomas Risberg

Copyright © 2012 - 2014The original author(s)

Copies of this document may be made for your own use and for distribution to others, provided that you do not charge any fee for such copies and further provided that each copy contains this Copyright Notice, whether distributed in print or electronically.

Table of Contents

Preface	iii
1. Project Metadata	iii
2. Requirements	iii
I. Reference Documentation	1
1. Solr Repositories	2
1.1. Introduction	2
Spring Namespace	2
Annotation based configuration	3
Multicore Support	3
Solr Repositores using CDI	4
Transaction Support	4
1.2. Query methods	5
Query lookup strategies	5
Query creation	5
Using @Query Annotation	7
Using NamedQueries	7
1.3. Document Mapping	7
Mapping Solr Converter	7
2. Miscellaneous Solr Operation Support	10
2.1. Partial Updates	10
2.2. Projection	10
2.3. Faceting	10
2.4. Terms	11
2.5. Filter Query	11
2.6. Time allowed for a search	12
2.7. Boost document Score	12
Index Time Boosts	12
2.8. Select Request Handler	12
2.9. Using Join	13
2.10. Highlighting	13
2.11. Using Functions	13
II. Appendix	15

Preface

The Spring Data Solr project applies core Spring concepts to the development of solutions using the Apache Solr Search Engine. We provide a "template" as a high-level abstraction for storing and querying documents. You will notice similarities to the mongodb support in the Spring Framework.

1 Project Metadata

- Version Control - [git://github.com/spring-projects/spring-data-solr.git](https://github.com/spring-projects/spring-data-solr.git)
- Bugtacker - <https://jira.spring.io/browse/DATASOLR>
- Release repository - <http://repo.spring.io/libs-release>
- Milestone repository - <http://repo.spring.io/libs-milestone>
- Snapshot repository - <http://repo.spring.io/libs-snapshot>

2 Requirements

Requires [Apache Solr](#) 3.6 and above or optional dependency

```
<dependency>
  <groupId>org.apache.solr</groupId>
  <artifactId>solr-core</artifactId>
  <version>${solr.version}</version>
</dependency>
```



Note

If you tend to use the Embedded Version of Solr Server 4.x you will also have to add a version of servlet-api and check your `<lockType>` as well as `<unlockOnStartup>` settings.

Part I. Reference Documentation

1. Solr Repositories

This chapter includes details of the Solr repository implementation.

1.1 Introduction

Spring Namespace

The Spring Data Solr module contains a custom namespace allowing definition of repository beans as well as elements for instantiating a `SolrServer`.

Using the `repositories` element looks up Spring Data repositories as described in ???.

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:solr="http://www.springframework.org/schema/data/solr"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans.xsd
    http://www.springframework.org/schema/data/solr
    http://www.springframework.org/schema/data/solr/spring-solr-1.0.xsd">

  <solr:repositories base-package="com.acme.repositories" />

</beans>
```

Example 1.1 Setting up Solr repositories using Namespace

Using the `solr-server` or `embedded-solr-server` element registers an instance of `SolrServer` in the context.

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:solr="http://www.springframework.org/schema/data/solr"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans.xsd
    http://www.springframework.org/schema/data/solr
    http://www.springframework.org/schema/data/solr/spring-solr-1.0.xsd">

  <solr:solr-server id="solrServer" url="http://localhost:8983/solr" />

</beans>
```

Example 1.2 HttpSolrServer using Namespace

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:solr="http://www.springframework.org/schema/data/solr"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans.xsd
    http://www.springframework.org/schema/data/solr
    http://www.springframework.org/schema/data/solr/spring-solr-1.0.xsd">

  <solr:solr-server id="solrServer" url="http://localhost:8983/solr,http://localhost:8984/
solr" />

</beans>
```

Example 1.3 LBSolrServer using Namespace

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:solr="http://www.springframework.org/schema/data/solr"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans.xsd
    http://www.springframework.org/schema/data/solr
    http://www.springframework.org/schema/data/solr/spring-solr-1.0.xsd">

  <solr:embedded-solr-server id="solrServer" solrHome="classpath:com/acme/solr" />
</beans>
```

Example 1.4 EmbeddedSolrServer using Namespace

Annotation based configuration

The Spring Data Solr repositories support cannot only be activated through an XML namespace but also using an annotation through JavaConfig.

```
@Configuration
@EnableSolrRepositories
class ApplicationConfig {

    @Bean
    public SolrServer solrServer() {
        EmbeddedSolrServerFactory factory = new EmbeddedSolrServerFactory("classpath:com/acme/
solr");
        return factory.getSolrServer();
    }

    @Bean
    public SolrOperations solrTemplate() {
        return new SolrTemplate(solrServer());
    }
}
```

The configuration above sets up an `EmbeddedSolrServer` which is used by the `SolrTemplate`. Spring Data Solr Repositories are activated using the `@EnableSolrRepositories` annotation, which essentially carries the same attributes as the XML namespace does. If no base package is configured, it will use the one the configuration class resides in.

Example 1.5 Spring Data Solr repositories using JavaConfig

Multicore Support

Solr handles different collections within one core. Use `MulticoreSolrServerFactory` to create separate `SolrServer` for each core.

```
@Configuration
@EnableSolrRepositories(multicoreSupport = true)
class ApplicationConfig {

    private static final String PROPERTY_NAME_SOLR_SERVER_URL = "solr.host";

    @Resource
    private Environment environment;

    @Bean
    public SolrServer solrServer() {
        return new
        HttpSolrServer(environment.getRequiredProperty(PROPERTY_NAME_SOLR_SERVER_URL));
    }
}
```

Example 1.6 Multicore Configuration

Solr Repositores using CDI

The Spring Data Solr repositories can also be set up using CDI functionality.

```
class SolrTemplateProducer {

    @Produces
    @ApplicationScoped
    public SolrOperations createSolrTemplate() {
        return new SolrTemplate(new EmbeddedSolrServerFactory("classpath:com/acme/solr"));
    }
}

class ProductService {

    private ProductRepository repository;

    public Page<Product> findAvailableProductsByName(String name, Pageable pageable) {
        return repository.findByAvailableTrueAndNameStartingWith(name, pageable);
    }

    @Inject
    public void setRepository(ProductRepository repository) {
        this.repository = repository;
    }
}
```

Example 1.7 Spring Data Solr repositories using JavaConfig

Transaction Support

Solr supports transactions on server level means create, updaet, delete actions since the last commit/ optimize/rollback are queued on the server and committed/optimized/rolled back at once. Spring Data Solr Repositories will participate in Spring Managed Transactions and commit/rollback changes on complete.

```
@Transactional
public Product save(Product product) {
    Product savedProduct = jpaRepository.save(product);
    solrRepository.save(savedProduct);
    return savedProduct;
}
```

Example 1.8

1.2 Query methods

Query lookup strategies

The Solr module supports defining a query manually as String or have it being derived from the method name.



Note

There is no QueryDSL Support present at this time.

Declared queries

Deriving the query from the method name is not always sufficient and/or may result in unreadable method names. In this case one might make either use of Solr named queries (see the section called “Using NamedQueries”) or use the `@Query` annotation (see the section called “Using @Query Annotation”).

Query creation

Generally the query creation mechanism for Solr works as described in ??? . Here's a short example of what a Solr query method translates into:

```
public interface ProductRepository extends Repository<Product, String> {
    List<Product> findByNameAndPopularity(String name, Integer popularity);
}
```

The method name above will be translated into the following solr query

```
q=name:?0 AND popularity:?1
```

Example 1.9 Query creation from method names

A list of supported keywords for Solr is shown below.

Table 1.1. Supported keywords inside method names

Keyword	Sample	Solr Query String
And	findByNameAndPopularity	q=name:?0 AND popularity:?1
Or	findByNameOrPopularity	q=name:?0 OR popularity:?1
Is	findByName	q=name:?0
Not	findByNameNot	q=-name:?0
IsNull	findByNameIsNull	q=-name:[* TO *]

Keyword	Sample	Solr Query String
IsNotNull	findByNameIsNotNull	q=name:[* TO *]
Between	findByPopularityBetween	q=popularity:[?0 TO ?1]
LessThan	findByPopularityLessThan	q=popularity:[* TO ?0}
LessThanEqual	findByPopularityLessThanEqual	q=popularity:[* TO ?0]
GreaterThan	findByPopularityGreaterThan	q=popularity:{?0 TO *]
GreaterThanEqual	findByPopularityGreaterThanEqual	q=popularity:[?0 TO *]
Before	findByLastModifiedBefore	q=last_modified:[* TO ?0}
After	findByLastModifiedAfter	q=last_modified:{?0 TO *]
Like	findByNameLike	q=name:?0*
NotLike	findByNameNotLike	q=-name:?0*
StartingWith	findByNameStartingWith	q=name:?0*
EndingWith	findByNameEndingWith	q=name:*?0
Containing	findByNameContaining	q=name:*?0*
Matches	findByNameMatches	q=name:?0
In	findByNameIn(Collection<String> names)	q=name:{(?0...)}
NotIn	findByNameNotIn(Collection<String> names)	q=-name:{(?0...)}
Within	findByStoreWithin(GeoLocation Distance)	q={!bbox geofilt pt=?0.latitude,?0.longitude sfield=store d=?1}
Near	findByStoreNear(GeoLocation Distance)	q={!bbox pt=?0.latitude,?0.longitude sfield=store d=?1}
Near	findByStoreNear(BoundingBox)	q=store[?0.start.latitude,?0.start.longitude TO ?0.end.latitude,?0.end.longitude]
True	findByAvailableTrue	q=inStock:true
False	findByAvailableFalse	q=inStock:false
OrderBy	findByAvailableTrueOrderByNameDesc	q=inStock:true&sort=name desc



Note

Collections types can be used along with 'Like', 'NotLike', 'StartingWith', 'EndingWith' and 'Containing'.

```
Page<Product> findByNameLike(Collection<String> name);
```

Using @Query Annotation

Using named queries (the section called “Using NamedQueries”) to declare queries for entities is a valid approach and works fine for a small number of queries. As the queries themselves are tied to the Java method that executes them, you actually can bind them directly using the Spring Data Solr @Query annotation.

```
public interface ProductRepository extends SolrRepository<Product, String> {
    @Query("inStock:?0")
    List<Product> findByAvailable(Boolean available);
}
```

Example 1.10 Declare query at the method using the @Query annotation.

Using NamedQueries

Named queries can be kept in a properties file and wired to the according method. Please mind the naming convention described in ??? or use @Query .

```
Product.findByNameQuery=popularity:?0
Product.findByName=name:?0
```

```
public interface ProductRepository extends SolrCrudRepository<Product, String> {

    List<Product> findByNameQuery(Integer popularity);

    @Query(name = "Product.findByName")
    List<Product> findByNameAnnotatedNamedQuery(String name);

}
```

Example 1.11 Declare named query in properties file

1.3 Document Mapping

Though there is already support for Entity Mapping within SolrJ, Spring Data Solr ships with its own mapping mechanism shown in the following section.



Note

DocumentObjectBinder has superior performance. Therefore usage is recommended if there is not need for custom type mapping. You can switch to DocumentObjectBinder by registering SolrJConverter within SolrTemplate.

Mapping Solr Converter

MappingSolrConverter allows you to register custom converters for your SolrDocument and SolrInputDocument as well as for other types nested within your beans. The Converter is not 100% compatible with DocumentObjectBinder and @Indexed has to be added with readonly=true to ignore fields from being written to solr.

```
public class Product {
    @Field
    private String simpleProperty;

    @Field("somePropertyName")
    private String namedProperty;

    @Field
    private List<String> listOfValues;

    @Indexed(readonly = true)
    @Field("property_*")
    private List<String> ignoredFromWriting;

    @Field("mappedField_*")
    private Map<String, List<String>> mappedFieldValues;

    @Field
    private GeoLocation location;
}
```

Example 1.12 Sample Document Mapping

Taking a look as the above MappingSolrConverter will do as follows:

Table 1.2.

Property	Write Mapping
simpleProperty	<field name="simpleProperty">value</field>
namedProperty	<field name="somePropertyName">value</field>
listOfValues	<field name="listOfValues">value 1</field> <field name="listOfValues">value 2</field> <field name="listOfValues">value 3</field>
ignoredFromWriting	//not written to document
mappedFieldValues	<field name="mapentry[0].key">mapentry[0].value[0]</ field> <field name="mapentry[0].key">mapentry[0].value[2]</field> <field name="mapentry[1].key">mapentry[1].value[0]</ field>
location	<field name="location">48.362893,14.534437</field>

To register a custom converter one must add CustomConversions to SolrTemplate initializing it with own Converter implementation.

```
<bean id="solrConverter"
  class="org.springframework.data.solr.core.convert.MappingSolrConverter">
  <constructor-arg>
    <bean class="org.springframework.data.solr.core.mapping.SimpleSolrMappingContext" />
  </constructor-arg>
  <property name="customConversions" ref="customConversions" />
</bean>

<bean id="customConversions"
  class="org.springframework.data.solr.core.convert.CustomConversions">
  <constructor-arg>
    <list>
      <bean class="com.acme.MyBeanToSolrInputDocumentConverter" />
    </list>
  </constructor-arg>
</bean>

<bean id="solrTemplate" class="org.springframework.data.solr.core.SolrTemplate">
  <constructor-arg ref="solrServer" />
  <property name="solrConverter" ref="solrConverter" />
</bean>
```

Example 1.13

2. Miscellaneous Solr Operation Support

This chapter covers additional support for Solr operations (such as faceting) that cannot be directly accessed via the repository interface. It is recommended to add those operations as custom implementation as described in ??? .

2.1 Partial Updates

PartialUpdates can be done using `PartialUpdate` which implements `Update` .



Note

Partial updates require Solr 4.x. With Solr 4.0.0 it is not possible to update multivalue fields.



Note

With Solr 4.1.0 you have to take care on parameter order when setting null values. Order parameters with nulls last.

```
PartialUpdate update = new PartialUpdate("id", "123");
update.add("name", "updated-name");
solrTemplate.saveBean(update);
```

Example 2.1

2.2 Projection

Projections can be applied via `@Query` using the fields value.

```
@Query(fields = { "name", "id" })
List<ProductBean> findByNameStartingWith(String name);
```

Example 2.2

2.3 Faceting

Faceting cannot be directly applied using the `SolrRepository` but the `SolrTemplate` holds support for this feature.

```
FacetQuery query = new SimpleFacetQuery(new
Criteria(Criteria.WILDCARD).expression(Criteria.WILDCARD))
.setFacetOptions(new FacetOptions().addFacetOnField("name").setFacetLimit(5));
FacetPage<Product> page = solrTemplate.queryForFacetPage(query, Product.class);
```

Example 2.3

Facets on fields and/or queries can also be defined using `@Facet` . Please mind that the result will be a `FacetPage` .



Note

Using `@Facet` allows you to define place holders which will use your input parameter as value.

```
@Query(value = "*:~")
@Facet(fields = { "name" }, limit = 5)
FacetPage<Product> findAllFacetOnName(Pageable page);
```

Example 2.4

```
@Query(value = "popularity:?0")
@Facet(fields = { "name" }, limit = 5, prefix="?1")
FacetPage<Product> findByPopularityFacetOnName(int popularity, String prefix, Pageable
page);
```

Example 2.5

Solr allows definition of facet parameters on a per field basis. In order to add special facet options to defined fields use `FieldWithFacetParameters`.

```
// produces: f.name.facet.prefix=spring
FacetOptions options = new FacetOptions();
options.addFacetOnField(new FieldWithFacetParameters("name").setPrefix("spring"));
```

Example 2.6

2.4 Terms

Terms Vector cannot directly be used within `SolrRepository` but can be applied via `SolrTemplate`. Please mind, that the result will be a `TermsPage`.

```
TermsQuery query = SimpleTermsQuery.queryBuilder().fields("name").build();
TermsPage page = solrTemplate.queryForTermsPage(query);
```

Example 2.7

2.5 Filter Query

Filter Queries improve query speed and do not influence document score. It is recommended to implement geospatial search as filter query.



Note

Please note that in solr, unless otherwise specified, all units of distance are kilometers and points are in degrees of latitude,longitude.

```
Query query = new SimpleQuery(new
Criteria("category").is("supercalifragilisticexpialidocious"));
FilterQuery fq = new SimpleFilterQuery(new Criteria("store")
.near(new GeoLocation(48.305478, 14.286699), new Distance(5)));
query.addFilterQuery(fq);
```

Example 2.8

Simple filter queries can also be defined using `@Query`.



Note

Using `@Query` allows you to define place holders which will use your input parameter as value.

```
@Query(value = "*:~", filters = { "inStock:true", "popularity:[* TO 3]" })
List<Product> findAllFilterAvailableTrueAndPopularityLessThanEqual3();
```

2.6 Time allowed for a search

It is possible to set the time allowed for a search to finish. This value only applies to the search and not to requests in general. Time is in milliseconds. Values less than or equal to zero implies no time restriction. Partial results may be returned, if there are any.

```
Query query = new SimpleQuery(new
SimpleStringCriteria("field_1:value_1"));
// Allowing maximum of 100ms for this search
query.setTimeAllowed(100);
```

Example 2.9

2.7 Boost document Score

Boost document score in case of matching criteria to influence result order. This can be done by either setting boost on Criteria or using @Boost for derived queries.

```
Page<Product> findByNameOrDescription(@Boost(2) String name, String description);
```

Example 2.10

Index Time Boosts

Boosting documents score can be done on index time by using @SolrDocument annotation on classes (for Solr documents) and/or @Indexed on fields (for Solr fields).

```
import org.apache.solr.client.solrj.beans.Field;
import org.springframework.data.solr.repository.Boost;

@SolrDocument(boost = 0.8f)
public class MyEntity {

    @Id
    @Indexed
    private String id;

    @Indexed(boost = 1.0f)
    private String name;

    // setters and getters ...

}
```

Example 2.11

2.8 Select Request Handler

Select the request handler via qt Parameter directly in Query or add @Query to your method signature.

```
@Query(requestHandler = "/instock")
Page<Product> findByNameOrDescription(String name, String description);
```

Example 2.12

2.9 Using Join

Join attributes within one solr core by defining Join attribute of Query.



Note

Join is not available prior to solr 4.x.

```
SimpleQuery query = new SimpleQuery(new SimpleStringCriteria("text:ipod"));
query.setJoin(Join.from("manu_id_s").to("id"));
```

Example 2.13

2.10 Highlighting

To highlight matches in search result add HighlightOptions to the SimpleHighlightQuery. Providing HighlightOptions without any further attributes will highlight apply highlighting on all fields within a SolrDocument.



Note

Field specific highlight parameters can be set by adding FieldWithHighlightParameters to HighlightOptions.

```
SimpleHighlightQuery query = new SimpleHighlightQuery(new
    SimpleStringCriteria("name:with"));
query.setHighlightOptions(new HighlightOptions());
HighlightPage<Product> page = solrTemplate.queryForHighlightPage(query, Product.class);
```

Example 2.14

Not all parameters are available via setters/getters but can be added directly.

```
SimpleHighlightQuery query = new SimpleHighlightQuery(new
    SimpleStringCriteria("name:with"));
query.setHighlightOptions(new
    HighlightOptions().addHighlightParameter("hl.bs.country", "at"));
```

Example 2.15

In order to apply Highlighting to derived queries use @Highlight. If no fields are defined highlighting will be applied on all fields.

```
@Highlight(prefix = "<b>", postfix = "</b>")
HighlightPage<Product> findByName(String name, Pageable page);
```

Example 2.16

2.11 Using Functions

Solr supports several functional expressions within queries. Followig functions are supported out of the box. Custom functions can be added by implementing Function

Table 2.1. Functions

Class	Solr Function
CurrencyFunction	currency(field_name, [CODE])

Class	Solr Function
DefaultValueFunction	def(field function,defaultValue)
DistanceFunction	dist(power, pointA, pointB)
DivideFunction	div(x,y)
ExistsFunction	exists(field function)
GeoDistanceFunction	geodist(sfield, latitude, longitude)
GeoHashFunction	geohash(latitude, longitude)
IfFunction	if(value field function,trueValue,falseValue)
MaxFunction	max(field function,value)
NotFunction	not(field function)
ProductFunction	product(x,y,...)
QueryFunction	query(x)
TermFrequencyFunction	termfreq(field,term)

```
SimpleQuery query = new SimpleQuery(new SimpleStringCriteria("text:ipod"));
query.addFilterQuery(new FilterQuery(Criteria.where(QueryFunction.query("name:sol*"))));
```

Example 2.17

Part II. Appendix