Spring Data Solr

Christoph Strobl, Oliver Gierke, Mark Pollack, Thomas Risberg

Version 1.5.0.M1
2015-06-02

Table of Contents

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.

Chapter 1. Project Metadata

- Version Control - https://github.com/spring-projects/spring-data-solr
- Bugtacker - https://jira.spring.io/browse/DATASOLR
- Release repository - https://repo.springsource.org/libs-release
- Milestone repository - https://repo.springsource.org/libs-milestone
- Snapshot repository - https://repo.springsource.org/libs-snapshot

Chapter 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.

Chapter 3. New & Noteworthy

3.1. What's new in Spring Data Solr 1.4

- Upgraded to recent Solr 4.10.x distribution (requires Java 7).
- Add support for Realtime Get.
- Get Field Stats (max, min, sum, count, mean, missing, stddev and distinct calculations).
- Use `@Score` to automatically add projection on document score (See Special Fields).

Unresolved directive in index.adoc - include::../../../../spring-data-commons/src/main/asciidoc/repositories.adoc[] :leveloffset: -1

Chapter 4. Reference Documentation

4.1. Solr Repositories

This chapter includes details of the Solr repository implementation.

4.1.1. 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 [repositories.create-instances] .

Example 1. Setting up Solr repositories 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:repositories base-package="com.acme.repositories" />
</beans>
```

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

Example 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://locahost:8983/solr" />
</beans>
```

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

Example 4. EmbeddedSolrServer 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>
```

Annotation based configuration

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

Example 5. Spring Data Solr repositories using 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.

Multicore Support

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

Example 6. Multicore Configuration

```
@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));
  }

}
```

Solr Repositores using CDI

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

Example 7. Spring Data Solr repositories using JavaConfig

```
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;
  }
}
```

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;
}
```

4.1.2. 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 Using NamedQueries) or use the `@Query` annotation (see Using @Query Annotation).

Query creation

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

Example 8. Query creation from method names

```
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
```

A list of supported keywords for Solr is shown below.

Table 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 *]`
`IsNotNull`	`findByNameIsNotNull`	`q=name:[* TO *]`
`Between`	`findByPopularityBetween`	`q=popularity:[?0 TO ?1]`
`LessThan`	`findByPopularityLessThan`	`q=popularity:[* TO ?0}`

Keyword	Sample	Solr Query String
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(Point, Distance)	q={!geofilt pt=?0.latitude,?0.longitude sfield=store d=?1}
Near	findByStoreNear(Point, Distance)	q={!bbox pt=?0.latitude,?0.longitude sfield=store d=?1}
Near	findByStoreNear(Box)	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 (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.

Example 9. Declare query at the method using the `@Query` annotation.

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

Using NamedQueries

Named queries can be kept in a properties file and wired to the accroding method. Please mind the naming convention described in [repositories.query-methods.query-lookup-strategies] or use `@Query` .

Example 10. Declare named query in properties file

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

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

  List<Product> findByNamedQuery(Integer popularity);

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

}
```

4.1.3. 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% compartible with `DocumentObjectBinder` and `@Indexed` has to be added with `readonly=true` to ignore fields from beeing written to solr.

Example 11. Sample Document Mapping

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

  @Field("somePropertyName")
  private String namedPropery;

  @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;

}
```

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

Property	**Write Mapping**
simpleProperty	`<field name="simpleProperty">value</field>`
namedPropery	`<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>
```

4.2. 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 [repositories.custom-implementations] .

4.2.1. 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 mulitvalue 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);
```

4.2.2. Projection

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

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

4.2.3. 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);
```

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);
```

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

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"));
```

Pivot Faceting

Pivot faceting (Decision Tree) are also supported, and can be queried using `@Facet` annotation as follows:

```
public interface {

@Facet(pivots = @Pivot({ "category", "dimension" }, pivotMinCount = 0))
FacetPage<Product> findByTitle(String title, Pageable page);

@Facet(pivots = @Pivot({ "category", "dimension" }))
FacetPage<Product> findByDescription(String description, Pageable page);

}
```

Alternatively it can be queried using `SolrTemplate` as follows:

```
FacetQuery facetQuery = new SimpleFacetQuery(new SimpleStringCriteria("title:foo"));
FacetOptions facetOptions = new FacetOptions();
facetOptions.setFacetMinCount(0);
facetOptions.addFacetOnPivot("category","dimension");
facetQuery.setFacetOptions(facetOptions);
FacetPage<Product> facetResult = solrTemplate.queryForFacetPage(facetQuery, Product
.class);
```

In order to retrieve the pivot results the method `getPivot` can be used as follows:

```
List<FacetPivotFieldEntry> pivot = facetResult.getPivot(new SimplePivotField(
"categories","available"));
```

4.2.4. 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);
```

4.2.5. Result Grouping / Field Collapsing

Result grouping cannot directly be used within `SolrRepository` but can be applied via `SolrTemplate`. Please mind, that the result will be a `GroupPage`.

```
Field field = new SimpleField("popularity");
Function func = ExistsFunction.exists("description");
Query query = new SimpleQuery("inStock:true");

SimpleQuery groupQuery = new SimpleQuery(new SimpleStringCriteria("*:*"));
GroupOptions groupOptions = new GroupOptions()
.addGroupByField(field)
.addGroupByFunction(func)
.addGroupByQuery(query);
groupQuery.setGroupOptions(groupOptions);

GroupPage<Product> page = solrTemplate.queryForGroupPage(query, Product.class);

GroupResult<Product> fieldGroup = page.getGroupResult(field);
GroupResult<Product> funcGroup = page.getGroupResult(func);
GroupResult<Product> queryGroup = page.getGroupResult(query);
```

4.2.6. Field Stats

Field stats are used to retrieve statistics (max, min, sum, count, mean, missing, stddev and distinct calculations) of given fields from Solr. It is possible by providing `StatsOptions` to your query and reading the `FieldStatsResult` from the returned `StatsPage`. This could be achieved for instance, using `SolrTemplate` as follows:

```
// simple field stats
StatsOptions statsOptions = new StatsOptions().addField("price");

// query
SimpleQuery statsQuery = new SimpleQuery("*:*");
statsQuery.setStatsOptions(statsOptions);
StatsPage<Product> statsPage = solrTemplate.queryForStatsPage(statsQuery, Product
.class);

// retrieving stats info
FieldStatsResult priceStatResult = statResultPage.getFieldStatsResult("price");
Object max = priceStatResult.getMax();
Long missing = priceStatResult.getMissing();
```

The same result could be achieved annotating the repository method with `@Stats` as follows:

```
@Query("name:?0")
@Stats(value = { "price" })
Stats<Product> findByName(String name, Pageable page);
```

Distinct calculation and faceting are also supported:

```
// for distinct calculation
StatsOptions statsOptions = new StatsOptions()
    .addField("category")
    // for distinct calculation
    .setCalcDistinct(true)
    // for faceting
    .addFacet("availability");

// query
SimpleQuery statsQuery = new SimpleQuery("*:*");
statsQuery.setStatsOptions(statsOptions);
StatsPage<Product> statsPage = solrTemplate.queryForStatsPage(statsQuery, Product
.class);

// field stats
FieldStatsResult categoryStatResult = statResultPage.getFieldStatsResult("category");

// retrieving distinct
List<Object> categoryValues = priceStatResult.getDistinctValues();
Long distinctCount = categoryStatResult.getDistinctCount();

// retrieving faceting
Map<String, StatsResult> availabilityFacetResult = categoryStatResult
.getFacetStatsResult("availability");
Long availableCount = availabilityFacetResult.get("true").getCount();
```

The annotated version of the sample above would be:

```
@Query("name:?0")
@Stats(value = "category", facets = { "availability" }, calcDistinct = true)
StatsPage<Product> findByName(String name);
```

In order to perform a selective faceting or selective distinct calculation, `@SelectiveStats` may be used as follows:

```
// selective distinct faceting
...
Field facetField = getFacetField();
StatsOptions statsOptions = new StatsOptions()
    .addField("price")
    .addField("category").addSelectiveFacet("name").addSelectiveFacet(facetField);
...
// or annotating repository method as follows
...
@Stats(value = "price", selective = @SelectiveStats(field = "category", facets = {
"name", "available" }))
...

// selective distinct calculation
...
StatsOptions statsOptions = new StatsOptions()
    .addField("price")
    .addField("category").setSelectiveCalcDistinct(true);
...
// or annotating repository method as follows
...
@Stats(value = "price", selective = @SelectiveStats(field = "category", calcDistinct
= true))
...
```

4.2.7. 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 Point(48.305478, 14.286699), new Distance(5)));
query.addFilterQuery(fq);
```

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();
```

4.2.8. Time allowed for a search

It it 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);
```

4.2.9. 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);
```

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 ...

}
```

4.2.10. 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);
```

4.2.11. 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"));
```

4.2.12. 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);
```

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"));
```

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

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

4.2.13. 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. Functions

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

Class	Solr Function
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*")
)));
```

4.2.14. Realtime Get

The realtime get allows retrieval of the latest version of any document using the unique-key, without the need to reopen searchers.

NOTE realtime get relies on the update log feature.

Example 12. Realtime get

```
Product product = solrTemplate.getById("123", Product.class);
```

Multiple documents can be retrieved by providing a collection of ids as follows:

Example 13. Realtime multi-get

```
Collection<String> ids = Arrays.asList("123", "134");
Collection<Product> products = solrTemplate.getById(ids, Product.class);
```

4.2.15. Special Fields

@Score

In order to load score information of a query result, a field annotated with `@Score` annotation could be added, indicating the property holding the documents score.

NOTE The score property needs to be numerical and can only appear once per document.

```java
public class MyEntity {

    @Id
    private String id;

    @Score
    private Float score;

    // setters and getters ...

}
```

Chapter 5. Appendix

Unresolved directive in index.adoc - include::../../../../spring-data-commons/src/main/asciidoc/repository-namespace-reference.adoc[] Unresolved directive in index.adoc - include::../../../../spring-data-commons/src/main/asciidoc/repository-populator-namespace-reference.adoc[] Unresolved directive in index.adoc - include::../../../../spring-data-commons/src/main/asciidoc/repository-query-keywords-reference.adoc[] Unresolved directive in index.adoc - include::../../../../spring-data-commons/src/main/asciidoc/repository-query-return-types-reference.adoc[] :leveloffset: -1