

# JRedisearch Workshop

By David Parry



Principal Software Engineer at Mutualink Inc.

Worked for a range of companies from startups to big four consulting firms.

Contributes to JRedisearch and Jedis

<http://www.davidparry.com> <- more here



# Redis Overview

- Data Structures Server.
- Commands to access mutable data structures.
- Prefers to store all data in memory faster.
- Data structures stress on memory efficiency, smaller footprint.
- Features typical database; replication, tunable durability, cluster, high availability.
- CRDB – Conflict Free Replicated Database

# Redisearch Module Overview

- Developed and maintained by Redis Labs
- Redis Powered Search Engine
  - Open-Source Full-Text and Secondary Index engine
- Does not use internal data structures like sorted sets
  - Exact phrase matching and numeric filtering for text queries, that are not possible or efficient with traditional Redis search approaches example for a key `'(hello world)|(hola mundo)'`
- Reference Material: <https://oss.redislabs.com/redisearch>

# Main Features

Full-Text indexing of multiple fields in documents

Incremental indexing without performance loss

Document ranking (manually at index time).

Optional query clauses

Complex boolean queries with AND, OR, NOT operators between sub-queries

Auto-complete suggestions (with fuzzy prefix suggestions)

Prefix based searches

Field weights

Exact Phrase Search, Slop based search.

Numeric filters and ranges

Support for custom functions for query expansion and scoring (see Extensions)

Stemming based query expansion in many languages (using Snowball)

Geo filtering using Redis' own Geo-commands

Unicode support (UTF-8 input required)

Retrieve full document content or just ids

Partial and conditional document updates

Document deletion and updating with index garbage collection

Optional query clauses

# JRedisearch Overview

➤ Java library abstracting the API of the RediSearch Redis module

➤ <https://github.com/RedisLabs/JRediSearch>

## ➤ **Topics to Cover**

- Getting Started
- Client Usage
- Schema
- Index Usage
- Documents
- Query Usage
- Suggestion Usage

```

1 import io.redisearch.SearchResult;
2 import io.redisearch.Suggestion;
3 import io.redisearch.client.AddOptions;
4 import io.redisearch.client.SuggestionOptions;
5 import org.apache.commons.lang3.StringUtils;
6 import redis.clients.jedis.exceptions.JedisConnectionException;
7 import redis.clients.jedis.exceptions.JedisDataException;
8
9 import java.util.HashMap;
10 import java.util.List;
11 import java.util.Map;
12
13 public class RedisDemo {
14
15     public Client getClient() { return new io.redisearch.client.Client( "localhost", port: 6379); }
16
17     public boolean checkConnection() {
18         boolean flag = true;
19         try {
20             getClient().getInfo();
21         } catch (JedisConnectionException je) {
22             flag = false;
23         } catch (JedisDataException jex) {
24             // index not present or some other data exception :-|
25         }
26         return flag;
27     }
28
29     public boolean createSchema() {
30         Schema schema = new Schema().addTextField( name: "id", weight: 0.05)
31             .addTextField( name: "text", weight: 0.5)
32             .addNumericField( "chapter")
33             .addNumericField( "line")
34             .addTextField( name: "title", weight: 1.0);
35         return getClient().createIndex(schema, io.redisearch.client.Client.IndexOptions.Default());
36     }
37
38     public boolean validateIndexSchema() {
39         List b = (List) getClient().getInfo().get("fields");
40         assert StringUtils.equals(new String[(byte[])] ((List) b.get(0)).get(0), "id");
41         return StringUtils.equals((CharSequence) getClient().getInfo().get("index_name"), "book");
42     }
43
44     public int addSampleBookDocument() {
45         createSchema();
46         Map<String, Object> fields = new HashMap<>();
47         fields.put("text", "I am an example sentence.");
48         fields.put("chapter", 1);
49         fields.put("line", 1);
50         fields.put("title", "Title of the book");
51         getClient().addDocument(new Document( "id:123-abc", fields), new AddOptions());
52         Map<String, Object> info = getClient().getInfo();
53         return Integer.parseInt((String) info.get("num_docs"));
54     }
55
56     public int addDocumentsWithPayload() {
57         createSchema();
58         Map<String, Object> fields = new HashMap<>();
59         fields.put("text", "A sentence in the book.");
60         fields.put("chapter", 3);
61         fields.put("line", 56);
62         fields.put("title", "Title of the book is");
63         fields.put("large", "I am more data that really do not want to be indexed but stored in redis as a payload");
64         Gson gson = new Gson();
65         getClient().addDocument(new Document( "id:123-abc", fields, score: 1.0, gson.toJson(fields).getBytes()), new AddOptions());
66         Map<String, Object> info = getClient().getInfo();
67         return Integer.parseInt((String) info.get("num_docs"));
68     }
69
70     public long countNumberOfDocumentsHaveBookTerm() {
71         createSchema();
72         Map<String, Object> fields = new HashMap<>();
73         fields.put("text", "I am some text that is on one line of the book.");
74         fields.put("chapter", 23);
75         fields.put("line", 380);
76         fields.put("title", "I am a title of the book");
77         getClient().addDocument(new Document( "id:123", fields), new AddOptions());
78         fields.put("line", 381);
79         fields.put("text", "I am a title but something missing");
80         getClient().addDocument(new Document( "id:124", fields), new AddOptions());
81         fields.put("text", "Another line of text that will not have the same terms we will be searching on.");
82         getClient().addDocument(new Document( "id:124", fields), new AddOptions());
83
84         SearchResult searchResult = getClient().search(new Query( queryString: "book"));
85         Map<String, Object> info = getClient().getInfo();
86         assert 2 == Integer.parseInt((String) info.get("num_docs"));
87
88         return searchResult.totalResults();
89     }
90
91     public List<Suggestion> suggestionLoadAndRetrieve() {
92         createSchema();
93         getClient().addSuggestion(Suggestion.builder().score(0.5).str("hello").build(), increment: false);
94         getClient().addSuggestion(Suggestion.builder().score(0.5).str("happy").build(), increment: false);
95         getClient().addSuggestion(Suggestion.builder().score(0.5).str("henry").build(), increment: false);
96         getClient().addSuggestion(Suggestion.builder().score(0.5).str("hair").build(), increment: false);
97         getClient().addSuggestion(Suggestion.builder().score(0.5).str("howdy").build(), increment: false);
98         List<Suggestion> list = getClient().getSuggestion( prefix: "he", SuggestionOptions.builder().fuzzy()
99             .with(SuggestionOptions.With.SCORES).build());
100         assert list.size() == 5;
101         getClient().addSuggestion(Suggestion.builder().score(0.5).str("hello").build(), increment: true);
102         list = getClient().getSuggestion( prefix: "he", SuggestionOptions.builder()
103             .fuzzy().with(SuggestionOptions.With.SCORES).build());
104
105         return list;
106     }
107
108 }

```

```

1 import java.util.List;
2 import java.util.Map;
3 import java.util.Set;
4 import java.util.stream.Collectors;
5
6 public class BookDemo {
7     private Client client = new io.redisearch.client.Client( "localhost", host: "localhost", port: 6379);
8     private Schema schema = new Schema().addTextField( name: "id", weight: 0.05)
9         .addTextField( name: "text", weight: 0.5)
10         .addNumericField( "chapter")
11         .addNumericField( "line")
12         .addTextField( name: "title", weight: 1.0);
13
14     public void createSearchableIndexBook() {
15         // clean up from other examples but leave it for the rest of the example
16         client.dropIndex( missingOk: true);
17         client.createIndex(schema, io.redisearch.client.Client.IndexOptions.Default());
18         createDocuments().forEach(doc -> {
19             client.addDocument(doc, new AddOptions());
20         });
21     }
22
23     public SearchResult search(String term) {
24         Query query = new Query(term).setWithScores();
25         return client.search(query);
26     }
27
28     public SearchResult searchFirstLines(String term, int lines) {
29         Query query = new Query(term).setWithScores().limit( offset: 0, limit: 100)
30             .addFilter(new Query.NumericFilter( property: "line", min: 0, lines));
31         return client.search(query);
32     }
33
34     public void primeSuggestions() {
35         createSuggestionSet().forEach(suggestion -> {
36             client.addSuggestion(suggestion, increment: false);
37         });
38     }
39
40     public List<String> getSuggestions(String partial) {
41         List<Suggestion> suggestions = client.getSuggestion(partial, SuggestionOptions.builder().fuzzy().build());
42         return suggestions.stream().map(suggestion -> suggestion.getString())
43             .collect(Collectors.toList());
44     }
45
46     private List<Document> createDocuments() {
47         final List<Document> documents = new ArrayList<>();
48         Book.INSTANCE.getLines().forEach(line -> {
49             Map<String, Object> fields = new HashMap<>();
50             fields.put("text", line.getText());
51             fields.put("chapter", line.getChapter());
52             fields.put("line", line.getLine());
53             fields.put("title", "title of the book");
54             documents.add(new Document(line.getId(), fields));
55         });
56         return documents;
57     }
58
59     private Set<Suggestion> createSuggestionSet() {
60         final Set<Suggestion> terms = new HashSet<>();
61         Book.INSTANCE.getLines().forEach(line -> {
62             String[] values = StringUtils.split(line.getText());
63             for (int v = 0; v < values.length; v++) {
64                 // cleanse from things like colons
65                 if (StringUtils.isAlpha(values[v])) {
66                     terms.add(Suggestion.builder().str(values[v]).score(0.5).build());
67                 }
68             }
69         });
70         return terms;
71     }
72
73 }

```

# Getting Started

➤ Steps to readme:

➤ Git install - <https://git-scm.com/downloads>

➤ Verify – `$ git clone git@github.com:davidparry/JRedisearchWorkshop.git`

➤ Follow the README.md for further instructions

# Client Usage



## Tag v1.0

**Goal:** Connect to Redisearch and verify connectivity

*Reference Material:*

[https://oss.redislabs.com/redisearch/java\\_client/](https://oss.redislabs.com/redisearch/java_client/)

<http://davidparry.com/storage/jredisearch-javadoc-v0-19-0/docs/io/redisearch/Client.html>



# Index Usage



## Tag v1.1

**Goal:** Demonstrate a better approach to check connectivity

*See post @* <http://davidparry.com/blog/2018/12/2/testing-conductivity-in-jredissearch-to-redis-without-a-prev.html>

# Schema

## Tag v2.0

**Goal:** Define and create a Schema in Redisearch

*Reference Material:* <https://oss.redislabs.com/redisearch/Commands/#ftcreate>

<https://oss.redislabs.com/redisearch/Commands/#ftinfo>

<http://davidparry.com/storage/jredisearch-javadoc-v0-19-0/docs/io/redisearch/client/Client.html#createIndex-io.redisearch.Schema-io.redisearch.client.Client.IndexOptions->

# Documents

## Tag v3.0

**Goal:** To add a simple Document to the Index and then a more complex with a Payload

*Reference Material:*

<https://oss.redislabs.com/redisearch/Commands/#ftadd>

<https://oss.redislabs.com/redisearch/payloads/>

<http://davidparry.com/storage/jrediseach-javadoc-v0-19-0/docs/io/redisearch/>

[Client.html#addDocument-io.redisearch.Document-io.redisearch.client.AddOptions-](#)

<http://davidparry.com/storage/jrediseach-javadoc-v0-19-0/docs/io/redisearch/>

[Document.html#Document-java.lang.String-java.util.Map-double-byte:A-](#)

# Query Usage



## Tag 4.0

**Goal:** retrieve a document using a simple term using a query object

*Reference Material:*

[https://oss.redislabs.com/redisearch/Query\\_Syntax/](https://oss.redislabs.com/redisearch/Query_Syntax/)

<http://davidparry.com/storage/jrediseach-javadoc-v0-19-0/docs/io/redisearch/Client.html#search-io.redisearch.Query->

# Suggestion Usage



## Tag 5.0

**Goal:** add suggestions to the index and retrieve suggested words that are fuzzy

**Note:** *Adds a suggestion string to an auto-complete suggestion dictionary. This is disconnected from the index definitions, and leaves creating and updating suggestions dictionaries to the user*

*Reference Material:*

<https://oss.redislabs.com/redisearch/Commands/#ftsugadd>

<http://davidparry.com/storage/jredisearch-javadoc-v0-19-0/docs/io/redisearch/Client.html#addSuggestion-io.redisearch.Suggestion-boolean->

<http://davidparry.com/storage/jredisearch-javadoc-v0-19-0/docs/io/redisearch/Client.html#getSuggestion-java.lang.String-io.redisearch.client.SuggestionOptions->

# Example on Art Book

## Tag 6.0

**Goal:** Use our new knowledge and go deeper with indexing a book and using the api in more depth.

# Recap

- Getting Started
- Client Usage
- Schema
- Index Usage
- Documents
- Query Usage
- Suggestion Usage

# Q&A

