

# spring-data-examples (0.0.1)

Maksim Kostromin

Version 0.0.1, 2019-06-09 14:45:28 UTC

# Table of Contents

1. spring-boot-webflux + cassandra .....	2
1.1. test .....	2
1.2. build and run .....	2
2. fix of spring-data-jpa issue when using <code>#{#entityName}</code> in <code>countQuery</code> .....	3
3. spring expression language (issue) .....	4
3.1. how to reproduce issue .....	4
3.2. run app .....	4
3.3. links .....	4
4. spring expression language (fixes) .....	5
5. redis .....	6
5.1. stack: .....	6
6. QueryDSL .....	7
6.1. functional REST API testing using SoapUI and Gradle plugin .....	7
6.2. functional SOAP API testing using SoapUI and Gradle plugin .....	7
6.3. unit/integration testing using spring-boot-test and Docker .....	7
6.4. JPA: persisting Collections of Enum .....	7
6.5. spring HATEOAS resources assembler page metadata .....	8
6.6. Event Sourcing using spring application events .....	8
6.7. Event Sourcing (history) using spring data-rest .....	8
6.8. Embedded primitive <code>@OneToMany</code> and <code>@ManyToMany</code> relationships .....	9
6.9. Optimization: 3NF .....	9
6.10. examine REST API using HTTPie: .....	9
6.11. generate Q-classes from JPA: .....	10
6.12. quick startup .....	10
6.13. integration tests .....	10
6.14. spring data jpa auditing .....	10
6.15. stack: .....	10
7. Derby create-drop for development) .....	12
8. Reactive Redis .....	13
9. Boot your data - RDBMS (derby, h2, hsql, mysql, postgres) .....	14
10. Listening spring-data events .....	15
11. Elasticsearch .....	16
12. using elastic .....	17
13. problem solving .....	18
14. resources .....	20
15. Spring Data Key-Value (webflux / kotlin) .....	21
16. Spring Data Hazelcast .....	22
17. Spring data reactive (mongo, solr, elastic) .....	24

18. Spring Data (spring-data-rest) advanced audit.....	25
19. MapDB   Spring Webflux.....	26
20. links.....	27

# Introduction

This documentation contains some help to [examples from spring-data-examples repository](#) is contains some node.js playground projects

# Chapter 1. spring-boot-webflux + cassandra

## 1.1. test

TODO...

## 1.2. build and run

*docker (gradle / maven)*

```
./gradlew build composeUp
./gradlew composeDown

./mvnw; ./mvnw com.dkanejs.maven.plugins:docker-compose-maven-plugin:1.0.1:up -P docker
./mvnw com.dkanejs.maven.plugins:docker-compose-maven-plugin:1.0.1:down -P docker
```

*gradle*

```
./gradlew
java -jar build/libs/*.jar
bash build/libs/*.jar
```

*maven*

```
./mvnw
java -jar target/*.jar
bash target/*.jar
```

Initially generated by using [generator-jvm](#) yeoman generator (java-spring-boot)

# Chapter 2. fix of spring-data-jpa issue when using `#{#entityName}` in `countQuery`

*testing using httpie*

```
http :8080/api/
```

*run using gradle*

```
./gradlew  
java -jar build/libs/*.jar  
bash build/libs/*.jar
```

*run using maven*

```
./mvnw  
java -jar target/*.jar  
bash target/*.jar
```

*run using docker*

```
# gradle  
./gradlew build composeUp  
./gradlew composeDown  
  
# maven  
./mvnw; ./mvnw com.dkanejs.maven.plugins:docker-compose-maven-plugin:1.0.1:up -P docker  
./mvnw com.dkanejs.maven.plugins:docker-compose-maven-plugin:1.0.1:down -P docker
```

Initially generated by using [generator-jvm](#) yeoman generator (java-spring-boot)

# Chapter 3. spring expression language (issue)

## 3.1. how to reproduce issue

### 3.1.1. update custom countQuery

in file app/src/main/java/daggerok/domain/MyEntityRepository.java:

```
// ...
@Query(
    value = " select me.name from #{#entityName} me ",
    countQuery = " select count(me.id) from #{#entityName} me "
)
Page<String> findAllNames(final Pageable pageable);
// ...
```

### 3.1.2. verify bootstrapping fail with exception

```
...
Caused by: java.lang.IllegalArgumentException: org.hibernate.QueryException:
unexpected char: '#' [ select count(me.id) from #{#entityName} me ]
...
Caused by: org.hibernate.QueryException: unexpected char: '#' [ select count(me.id)
from #{#entityName} me ]
...
```

## 3.2. run app

```
bash gradlew bootRun # installed docker and compose are required
curl -sS localhost:8080 | jq
curl -sS localhost:8080/names | jq
bash gradlew stop
bash gradlew --stop
```

## 3.3. links

1. [stackoverflow question](#)
2. [DATAJPA-1163: spring data jpa JIRA bug](#)

# Chapter 4. spring expression language (fixes)

Unresolved directive in index.adoc - include::.../spel-fixed/README.adoc[tags=content]



# Chapter 5. redis

bootstrapping docker before bootRun

*build, run, test*

```
gradle redisUp
gradle redis:bootRun
http :8080/redisObjs data=test
http :8080/redisObjs
gradle redisDown

gradle embedded-redis:bootRun
http :8082/embeddedRedisObjs data=embedded-test
http :8082/embeddedRedisObjs

gradle --stop
```

## 5.1. stack:

1. spring-boot
2. spring-data-rest
3. [spring HATEOAS](#)
4. [spring-data-keyvalue](#)
5. spring-data-redis
6. embedded redis server
7. gradle
8. Docker
9. Redis
10. Redis web UI
11. install spring app as linux service

links:

1. [Reactive Java Redis Client](#)
2. [Embedded Redis](#)
3. [Rector Reference](#)

# Chapter 6. QueryDSL

## 6.1. functional REST API testing using SoapUI and Gradle plugin

see `soaptest` subproject

```
gradle clean assemble soaptestRest soaptestWs
```

## 6.2. functional SOAP API testing using SoapUI and Gradle plugin

```
gradle wsServiceRun
curl --header "content-type: text/xml" -d @services/ws-
service/src/test/resources/request.xml http://localhost:8080/ws | xmllint --format -
# ctrl+c
gradle dockerDown
gradle --stop

gradle clean assemble soaptestWs
```

## 6.3. unit/integration testing using spring-boot-test and Docker

```
gradle clean build
```

## 6.4. JPA: persisting Collections of Enum

```
gradle restServiceRun

http :8080/api/v6
http :8080/api/v6/catalog
http :8080/api/v6/catalog\?size=1

http :8080/api/v6/enum-collection/TEST_ENTITY_1
http :8080/api/v6/enum-collection/TEST_ENTITY_2
http :8080/api/v6/enum-collection/not_found

http :8080/api/v6/map-catalog/type/not-found
http :8080/api/v6/map-catalog/type/TEST_ENTITY_2
http :8080/api/v6/map-catalog/status/NOK
http :8080/api/v6/map-catalog/status/OK
http :8080/api/v6/map-catalog/status/OK\?size=1

http :8080/api/v6/jpa-enum
http :8080/api/v6/jpa-enum\?size=1

gradle --stop
```

## 6.5. spring HATEOAS resources assembler page metadata

```
gradle restServiceRun
http :8080/api/v5/engineers/page-metadata
```

## 6.6. Event Sourcing using spring application events

see:

1. `service/**/src/main/java/daggerok/history/applicationevent`
2. `service/**/src/main/java/daggerok/history/service`

```
http :8080/rest/engineers username=tttest | jq '._links.self'
http :8080/rest/histories | jq '._embedded.histories'
```

## 6.7. Event Sourcing (history) using spring data-rest

see:

1. `service/**/src/main/java/daggerok/history/springdatarest`
2. `service/**/src/main/java/daggerok/history/service`

```
http :8080/rest/domains firstName=1 lastName=1 username=1 | jq '._links.self'
http :8080/rest/domains firstName=2 lastName=2 username=2 | jq '._links.self'
http :8080/rest/otherDomains test=1 | jq '._links.self'
http :8080/rest/histories | jq '._embedded.histories'
```

## 6.8. Embedded primitive @OneToMany and @ManyToMany relationships

see: [service/\\*\\*/src/main/java/daggerok/relationships](#)

@OneToMany → @Embeddable Set emails (also could be a list)

@ManyToMany → @Embeddable Map tags (same for labels)

```
gradle compileQuerydsl xjc # gradle assemble
gradle bootRun

http ":8080/api/v4/engineers?size=2&page=0&sort=username,desc"

gradle --stop
gradle composeDown
```

## 6.9. Optimization: 3NF

see: [service/\\*\\*/src/main/java/daggerok/embedded](#)

## 6.10. examine REST API using HTTPie:

```
http :8080/api/v3/predicate
http ":8080/api/v3/predicate?second.secondField1=1"

# bash
http ":8080/api/v3/predicate?createdDate=$(date +%Y-%m-%d)"
# fish
http ":8080/api/v3/predicate?createdDate="(date +%Y-%m-%d)

http ":8080/api/v2/pagination?page=0&size=1&sort=first.firstField1,desc"
http ":8080/api/v2/sorted?sort=id,desc"

http :8080/api/v2/flatten
http :8080/api/v2/flatten/2
```

note: see [.travis.yml](#) for cURL examples

## 6.11. generate Q-classes from JPA:

```
gradle compileQuerydsl xjc
```

## 6.12. quick startup

bootstrapping docker before bootRun

```
gradle bootRun
open http://localhost:8080 # press enter
...
gradle composeDown
gradle --stop
```

## 6.13. integration tests

see `docker` subproject

```
gradle clean assemble test
gradle --stop
```

## 6.14. spring data jpa auditing

see `service/**/src/main/java/daggerok/audit` package

id	created_date	modified_at	de_normalized_field	first_field_1	first_field_2	second_field1	second_field2
1	2017-06-10	2017-06-10 22:18:35.516000	1	1	1	1	1
2	2017-06-10	2017-06-10 22:18:35.545000	2	2	2	2	2

## 6.15. stack:

- spring-boot, spring-data, spring-web, fallback 404 handler
- JPA auditing
- Performance optimization: de-normalize JPA NF4 → NF3, @Embedded, @Embeddable

- QueryDSL (spring-data integration)
- Event sourcing using spring data-rest and spring application events
- gradle, SoapUI
- Postgres, Docker
- QueryDSL [reference documentation](#) and [example](#)

# Chapter 7. Derby create-drop for development)

This repo is contains simple example of usage spring-boot devtools reload/restart with derby

```
gradle bootRun
http :8080
http post :8080 id=user2 name=user2

# 1. update some code (remove mail2 from User.class and from schema.sql)
# 2. rebuild project inside IDEA oe STS to handle devtools
# 3. check logs....

http :8080 # 2 items again
gradle --stop
```

# Chapter 8. Reactive Redis

this repository is containng modern spring 5 web application which is using reactive spring webflux and spring data redis

```
gradle composeUp bootRun
```

```
http :8080/tasks
```

```
http :8080/activities
```

```
http delete :8080
```

```
http :8080/tasks
```

```
http :8080/activities
```

```
gradle composeDown
```

```
gradle --stop
```



## Chapter 9. Boot your data - RDBMS (derby, h2, hsql, mysql, postgres)

This repository contains examples of usage relation databases with spring-data-rest

in progress...

# Chapter 10. Listening spring-data events

this repository is containng modern spring 5 web application which is listening spring-data events

```
bash gradew clean build
```

# Chapter 11. Elasticsearch

This repository contains spring-data elastic examples

in fucking progress...

*run it all using docker-compose*

```
bash gradlew assemble composeUp -Ddocker=compose-all  
  
open http://localhost/  
  
bash gradlew composeDown -Ddocker=compose-all
```

*run app in idea*

```
cd data/  
bash gradlew bootRun  
  
http -a elastic:changeme :9200  
  
http post :8080 name=max  
http :8080  
http :8080/users
```

*explore with kibana*

```
gradle clean bootRun  
http -a elastic:changeme :9200
```

*testing elasticsearch db with curl / httpie*

```
gradle clean bootRun  
http -a elastic:changeme :9200  
curl -u elastic:changeme localhost:9200 | jq
```

*manual run*

```
gradle clean build
```

# Chapter 12. using elastic

*create few users*

```
export auth=" -a elastic:changeme "  
echo '{"username":"daggerok","name":"Maksim Kostromin"}' | http $auth  
:9200/user/customer
```

*helper fish fuctions*

```
function pes  
    echo $argv[2] | http -a elastic:changeme post :9200/$argv[1]  
end  
  
function ges  
    http -a elastic:changeme get :9200/$argv[1]  
end  
  
pes user/customer '{  
  "username": "ololo",  
  "name": "Trololo"  
}'  
  
http -a elastic:changeme :9200/user/_search?q=ololo
```

*more elastic*

```
ges _cluster/health?pretty
```

# Chapter 13. problem solving

*ulimit -an*

```
# 2017-11-02 01:29:06.355 WARN 10993 --- [ restartedMain] org.elasticsearch.env
: [Jackdaw] max file descriptors [10240] for elasticsearch process likely too low,
consider increasing to at least [65536]
# 2017-11-02 01:29:07.074 WARN 10993 --- [ restartedMain]
org.elasticsearch.bootstrap          : JNA not found. native methods will be
disabled.
```

```
launchctl unload /Library/LaunchDaemons/limit.maxfiles.plist
cat <<EOF> /Library/LaunchDaemons/limit.maxfiles.plist
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN"
      "http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
  <dict>
    <key>Label</key>
    <string>limit.maxfiles</string>
    <key>ProgramArguments</key>
    <array>
      <string>launchctl</string>
      <string>limit</string>
      <string>maxfiles</string>
      <string>524288</string>
      <string>524288</string>
    </array>
    <key>RunAtLoad</key>
    <true/>
    <key>ServiceIPC</key>
    <false/>
  </dict>
</plist>
EOF
```

```
launchctl load -w /Library/LaunchDaemons/limit.maxfiles.plist
```

```
launchctl unload /Library/LaunchDaemons/limit.maxproc.plist
cat <<EOF> /Library/LaunchDaemons/limit.maxproc.plist
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN"
      "http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
  <dict>
    <key>Label</key>
    <string>limit.maxproc</string>
    <key>ProgramArguments</key>
    <array>
      <string>launchctl</string>
      <string>limit</string>
```

```
<string>maxproc</string>
<string>2048</string>
<string>2048</string>
</array>
<key>RunAtLoad</key>
<true />
<key>ServiceIPC</key>
<false />
</dict>
</plist>
EOF
launchctl load -w /Library/LaunchDaemons/limit.maxproc.plist

exit
reboot
```

# Chapter 14. resources

1. [documentation: query creation](#)
2. [spring talk](#)
3. [elasticsearch for java dev](#)
4. [another spring search data talk](#)

# Chapter 15. Spring Data Key-Value (webflux / kotlin)

This is a simple spring-boot 5 webflux REST API example using spring-data-keyvalue (Map as database) and kotlin language.

*build and run*

```
bash gradlew clean build
```

using:

1. kotlin
2. spring-data-keyvalue



# Chapter 16. Spring Data Hazelcast

This is a simple spring-mvc REST API example using spring-data-hazelcast and kotlin language.

*build and run*

```
bash gradlew clean build

http :8080/any/ma
[
  {
    "id": "8f0c927a-cf68-430d-afab-cb9f3f9a9253",
    "name": "Max",
    "username": "max"
  }
]

http post :8080 name=Maximus username=xxx
HTTP/1.1 201
Content-Length: 0
Date: Sun, 05 Nov 2017 05:52:16 GMT
Location: /id/d365b264-97de-4458-81da-c99b9f5be1f4

http :8080/id/d365b264-97de-4458-81da-c99b9f5be1f4
{
  "id": "d365b264-97de-4458-81da-c99b9f5be1f4",
  "name": "Maximus",
  "username": "xxx"
}

http :8080/any/Ma
[
  {
    "id": "d365b264-97de-4458-81da-c99b9f5be1f4",
    "name": "Maximus",
    "username": "xxx"
  },
  {
    "id": "8f0c927a-cf68-430d-afab-cb9f3f9a9253",
    "name": "Max",
    "username": "max"
  }
]
```

### *supported query*

```
/*
  1) SIMPLE_PROPERTY("Is", "Equals")
  2)
    TRUE(0, "IsTrue", "True")
    FALSE(0, "IsFalse", "False")
  3)
    LESS_THAN("IsLessThan", "LessThan")
    LESS_THAN_EQUAL("IsLessThanEqual", "LessThanEqual")
    GREATER_THAN("IsGreaterThan", "GreaterThan")
    GREATER_THAN_EQUAL("IsGreaterThanEqual", "GreaterThanEqual")
  4)
    LIKE("IsLike", "Like")
  5)
    IS_NOT_NULL(0, "IsNotNull", "NotNull")
    IS_NULL(0, "IsNull", "Null")
```

### *supported query*

```
AFTER:
BEFORE:
BETWEEN:
CONTAINING:
ENDING_WITH:
EXISTS:
IN:
NEAR:
NEGATING_SIMPLE_PROPERTY:
NOT_CONTAINING:
NOT_IN:
NOT_LIKE:
REGEX:
STARTING_WITH:
WITHIN:
```

using:

1. kotlin
2. spring-data-hazelcast
3. [talk](#)
4. [hazelcast query](#)

# Chapter 17. Spring data reactive (mongo, solr, elastic)

This repository contains examples of usage NOSql databases such elasticsearch, mongodb, solr, couchbase, etc with spring, spring-boot and spring-data

in fucking progress...

```
docker-compose up -d --remove-orphans
gradle clean build
# ...
docker-compose down -v --remove-orphans
```

# Chapter 18. Spring Data (spring-data-rest) advanced audit

This repository contains spring-data audition implementation: object diff history audit

*test*

```
http put :8080/my-entities/1 value=ololo
http put :8080/my-entities/1 value=trololo
http put :8080/my-entities/1 value=ho-ho-ho

http get :8080/my-entities
http get :8080/my-entities-history
```

*build abd run*

```
bash gradlew clean bootRun # bash mvnw clean spring-boot:run

# or in docker:
docker-compose down -v; ./gradlew; docker-compose up --build --force-recreate --remove-orphans

# or using maven:
cp -Rf ./mvn/Dockerfile ./
docker-compose down -v; ./mvnw; docker-compose up --build --force-recreate --remove-orphans
```

# Chapter 19. MapDB | Spring Webflux

- link:<https://github.com/daggerok/spring-5-examples/tree/master/mapdb>

# Chapter 20. links

- [asciidoctor attributes](#)
- 

Enjoy! :)