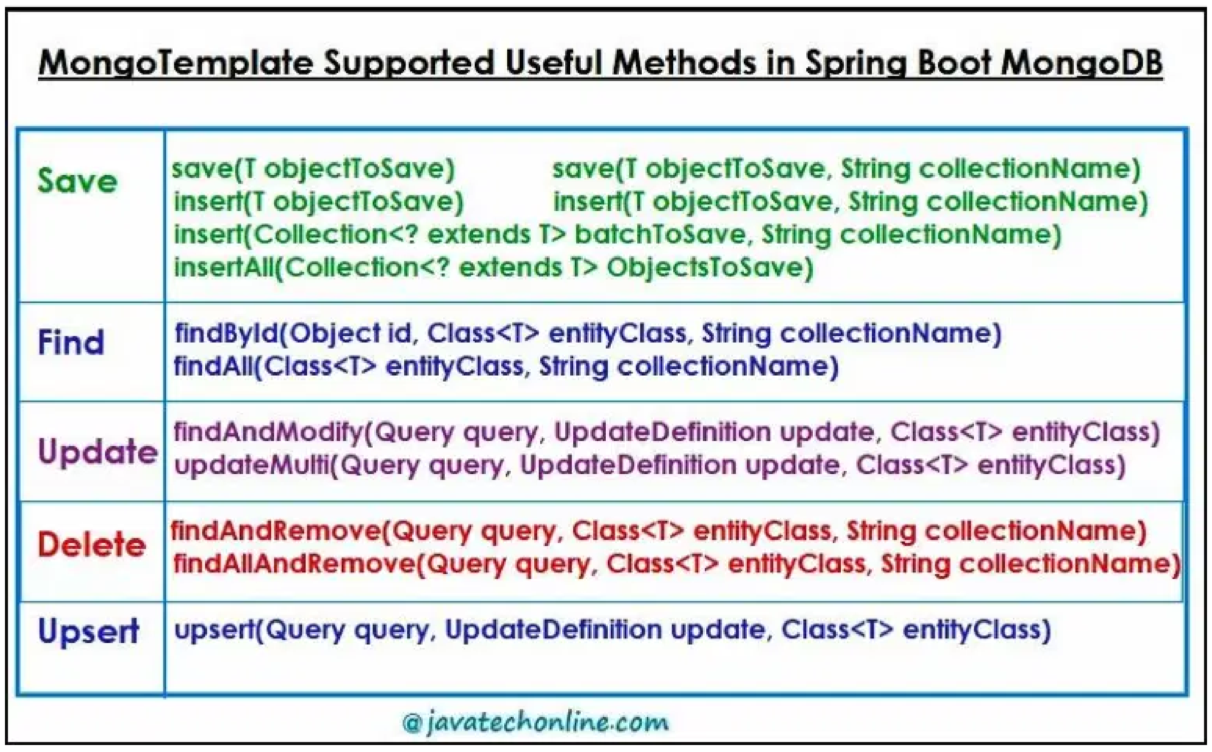
# Spring Boot MongoDB Using MongoTemplate Examples

java MongoDB Spring Boot *by devs5003 - May 7, 2021* 0

In



MongoTemplate Supported Useful Methods in Spring Boot MongoDB

| | |
|---|---|
| Save | save(T objectToSave) save(T objectToSave, String collectionName) insert(T objectToSave) insert(T objectToSave, String collectionName) insert(Collection<? extends T> batchToSave, String collectionName) insertAll(Collection<? extends T> ObjectsToSave) |
| Find | findById(Object id, Class<T> entityClass, String collectionName) findAll(Class<T> entityClass, String collectionName) |
| Update | findAndModify(Query query, UpdateDefinition update, Class<T> entityClass) updateMulti(Query query, UpdateDefinition update, Class<T> entityClass) |
| Delete | findAndRemove(Query query, Class<T> entityClass, String collectionName) findAllAndRemove(Query query, Class<T> entityClass, String collectionName) |
| Upsert | upsert(Query query, UpdateDefinition update, Class<T> entityClass) |

@javatechonline.com

continuation to series of tutorials on Spring Boot with MongoDB, we have already discussed various examples using MongoRepository interface. Now in this article we are going to discuss a different way of accessing data from MongoDB. This is nothing but MongoTemplate. MongoTemplate is defined by Spring Data MongoDB, used to perform DB operations without using any repository interface. If you are reading this article in context of Spring Boot, you must have heard about the term JDBCTemplate. Like JDBCTemplate, MongoTemplate is a class which provides us different handy methods to work on various DB operations. However, we generally use it to retrieve data from MongoDB using different Criteria queries. Hence, our topic here is to discuss is 'Spring Boot MongoDB using MongoTemplate Examples'.

Of course, MongoRepository are more convenient than MongoTemplate but MongoTemplate offers you closer control over what you request from MongoDB. We can even utilize both of them in our programming practice as per our need and for performance enhancements. Moreover, MongoTemplate can offer an easier step to write a complex query than MongoRepository. Even some people in the industry also finds that MongoTemplate is the good choice to write a complex query easily. Let's start discussing our topic 'Spring Boot MongoDB using MongoTemplate Examples'.

Table of Contents (Click on links below to navigate) [hide]

## What is MongoTemplate?

MongoTemplate is nothing but a Java class that comes under org.springframework.data.mongodb.core package. It provides a set of rich features for interacting with MongoDB and acts as a central class for Spring's MongoDB support. Moreover, MongoTemplate is thread-safe and can be reused across multiple instances. The MongoTemplate class implements the interface MongoOperations. Furthermore, MongoOperations has fluent APIs for Query, Criteria, and Update operations. Also, It offers methods using Generics. We can directly use MongoTemplate by auto-wiring it as a property in a class.

## Coding Steps to develop MongoTemplate based Examples

Learn more

Replay

In order to develop and test Query examples, we will first write codes and insert some records into DB. Thereafter, we will start testing Query Examples step by step.

### Step#0 : Setup MongoDB with Spring Boot

Make sure you already have installed MongoDB in your system. If not, kindly visit 'How to install MongoDB in your system ?'. Here, you will also get some handy commands to work with MongoDB. In order to get hold on Spring Boot with MongoDB, kindly visit 'How to work with MongoDB in Spring Boot?'.

### Step#1 : Create a Spring Boot Project using STS(Spring Tool Suite)

Here, we will use STS(Spring Tool Suite) to create our Spring Boot Project. If you are new to Spring Boot, visit Internal Link to create a sample project in spring boot using STS. While creating a project in STS, add starters 'Spring Data MongoDB', and 'Lombok' in order to get features of MongoDB. Furthermore, if you are new to 'Lombok', kindly visit 'How to configure Lombok' and to know all about it in detail.

TOP

## Step#2 : Update application.properties

```
spring.data.mongodb.host=localhost
spring.data.mongodb.port=27017
spring.data.mongodb.database=springBootMongoDB
```

If you are using MongoDB in authentication mode, include username and password of it accordingly.

## Step#3 : Create Entity class

Here we will use Book as an entity to illustrate the examples. Hence, create a Book.java class as below. However, we will consider collection name as 'Books' in MongoDB. It is just to illustrate the use of element 'collection' in @Document annotation. Moreover, we are using 'Lombok' in order to reduce the boilerplate code.

```
import org.springframework.data.annotation.Id;
import org.springframework.data.mongodb.core.mapping.Document;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

@Data
@NoArgsConstructor
@AllArgsConstructor
@Document
public class Book {

      @Id
      private Integer id;

      private String name;
      private Integer pages;
      private String author;
      private Double cost;
}
```

In this example, we have taken id as integer. If you are looking for an example where id as a String, kindly visit our previous tutorial on Spring Boot Mongo DB CRUD Example.

## save(object) *OR* save(object, collectionName)

Both methods save records into DB. If we intentionally want to change the collection name, we can use save(object, collectionName). However, if we use save(object) method, it will automatically insert records into default collection. Here default name of collection is ‘book’.

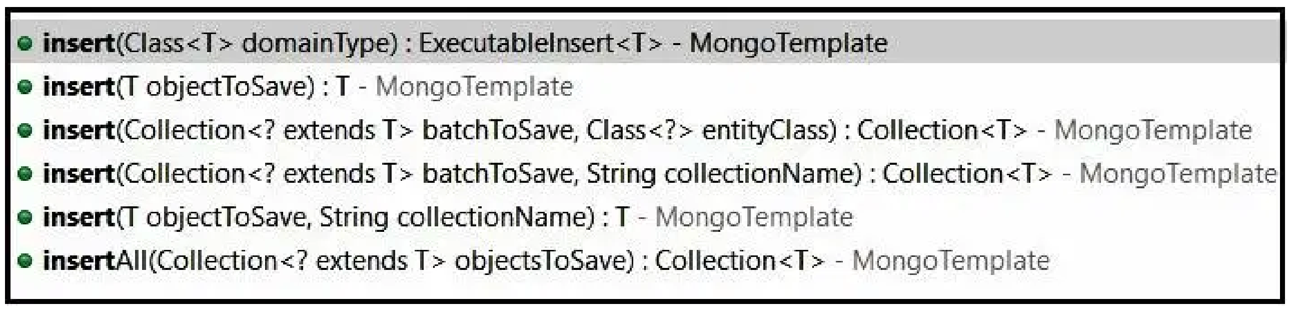
‘book’.

```
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.CommandLineRunner;
import org.springframework.data.mongodb.core.MongoTemplate;
import org.springframework.stereotype.Component;
import com.dev.springboot.model.Book;



@Component
public class BookOperations implements CommandLineRunner {

@Autowired
private MongoTemplate mt;

@Override
public void run(String... args) throws Exception {

    mt.save(new Book(500, "Core Java", 200, "Kathy Sierra", 1065.5));
    mt.save(new Book(501, "JSP & Servlets", 350, "Kathy Sierra", 1749.0));
 // mt.save(new Book(501, "JSP & Servlets", 350, "Kathy Sierra", 1749.0),"Bo
    mt.save(new Book(502, "Spring in Action", 480, "Craig Walls", 940.75));
    mt.save(new Book(503, "Pro Angular", 260, "Freeman", 1949.25));
    mt.save(new Book(504, "HTML CSS", 100, "Thomas Powell", 2317.09));
    mt.save(new Book(505, "Hibernate in Action", 180, "Gavin King", 889.25))
    mt.save(new Book(506, "Practical MongoDB", 180, "Shakuntala Gupta", 785.
    mt.save(new Book(507, "Pro Spring Boot", 850, "Felipe Gutierrez", 2167.9
    mt.save(new Book(508, "Beginning jQuery", 180, "Franklin", 1500.00));
    mt.save(new Book(509, "Java Design Patterns", 114, "Devendra Singh", 919

    System.out.println("------All records has been saved successfully-------
}

}
```

In the above example, we have illustrated one record with collection name of our choice ‘Book’ and commented it. If you uncomment it, a new record in a new collection ‘Book’ will be created.

## insert( ) *AND* insertAll( )

However, these methods are also used to save records into DB. Generally, we use these methods if we want to insert multiple records( batch records) into DB. insert() comes with a parameter as collection name while insertAll() doesn’t come with the same. Below screenshot shows the various flavors of these methods.


● **insert**(Class<T> domainType) : ExecutableInsert<T> - MongoTemplate
● **insert**(T objectToSave) : T - MongoTemplate
● **insert**(Collection<? extends T> batchToSave, Class<?> entityClass) : Collection<T> - MongoTemplate
● **insert**(Collection<? extends T> batchToSave, String collectionName) : Collection<T> - MongoTemplate
● **insert**(T objectToSave, String collectionName) : T - MongoTemplate
● **insert**All(Collection<? extends T> objectsToSave) : Collection<T> - MongoTemplate

For example, let’s use these methods to create same number of records as we did in previous example of save() operation. Please note that we are using List.of() method to save multiple records in DB. However, It will work on JDK9 and above versions. You can also

Learn more

## Using insert(Collection<? extends T> batchToSave, String collectionName)

```
mt.insert(List.of(
                 new Book(500, "Core Java", 200, "Kathy Sierra", 1065.5),
                 new Book(501, "JSP & Servlets", 350, "Kathy Sierra", 1749.0
                 new Book(502, "Spring in Action", 480, "Craig Walls", 940.7
                 new Book(503, "Pro Angular", 260, "Freeman", 1949.25),
                 new Book(504, "HTML CSS", 100, "Thomas Powell", 2317.09),
                 new Book(505, "Hibernate in Action", 180, "Gavin King", 889
                 new Book(506, "Practical MongoDB", 180, "Shakuntala Gupta",
                 new Book(507, "Pro Spring Boot", 850, "Felipe Gutierrez", 2
                 new Book(508, "Beginning jQuery", 180, "Franklin", 1500.00)
                 new Book(509, "Java Design Patterns", 114, "Devendra Singh"
                 ),
           "Book"
         );
```

## Using insertAll(Collection<? extends T> ObjectsToSave)

```
mt.insertAll(List.of(
             new Book(500, "Core Java", 200, "Kathy Sierra", 1065.5),
             new Book(501, "JSP & Servlets", 350, "Kathy Sierra", 1749.0),
             new Book(502, "Spring in Action", 480, "Craig Walls", 940.75),
             new Book(503, "Pro Angular", 260, "Freeman", 1949.25),
             new Book(504, "HTML CSS", 100, "Thomas Powell", 2317.09),
             new Book(505, "Hibernate in Action", 180, "Gavin King", 889.25)
             new Book(506, "Practical MongoDB", 180, "Shakuntala Gupta", 785
             new Book(507, "Pro Spring Boot", 850, "Felipe Gutierrez", 2167.
             new Book(508, "Beginning jQuery", 180, "Franklin", 1500.00),
             new Book(509, "Java Design Patterns", 114, "Devendra Singh", 91
            ));
```

TOP

Learn more

Ultimately, same records will get inserted into DB using any of methods.

Learn more

## findAll(className) *OR* findAll(className, collectionName)

Above both methods gets data from DB collection as List<T> format. Here T is Class name. If class name and collection name both are same then use findAll(T.class), else use findAll(T.class, "collectionName"). Let's retrieve the records saved in the previous save() example. For example, below code will retrieve the records from DB.

```
List<Book> list = mt.findAll(Book.class);
//List<Book> list = mt.findAll(Book.class,"Book");  //If collection name & t
list.forEach(System.out::println);
```

## Output

If we use the code 'List<Book> list = mt.findAll(Book.class)', we will get the below result.

```
Book(id=500, name=Core Java, pages=200, author=Kathy Sierra, cost=1065.5)
Book(id=501, name=JSP & Servlets, pages=350, author=Kathy Sierra, cost=1749.
Book(id=502, name=Spring in Action, pages=480, author=Craig Walls, cost=940.
Book(id=503, name=Pro Angular, pages=260, author=Freeman, cost=1949.25)
Book(id=504, name=HTML CSS, pages=100, author=Thomas Powell, cost=2317.09)
Book(id=505, name=Hibernate in Action, pages=180, author=Gavin King, cost=88
Book(id=506, name=Practical MongoDB, pages=180, author=Shakuntala Gupta, cos
Book(id=507, name=Pro Spring Boot, pages=850, author=Felipe Gutierrez, cost=
Book(id=508, name=Beginning jQuery, pages=180, author=Franklin, cost=1500.0)
Book(id=509, name=Java Design Patterns, pages=114, author=Devendra Singh, co
```

## findById(id, entityClass) *OR* findById(id, entityClass, collectionName)

TOP

We use this method to fetch data from DB collection using PK(ID). If Id exist then JSON

```
Book book = mt.findById(504, Book.class);
//Book book = mt.findById(501, Book.class,"Book");
System.out.println(book);
```

If we run the above statements, We will get the required output.

## findAndModify(query, update, entityClassName)

We use this method to fetch data based on given condition (query) and then modify data for given update type. We will require two objects to complete the whole process.

1)'Query' object to fetch data from Collection that needs to be modified
2)'Update' to specify which key=value combination need to be modified for the provided Result by Query.

```
Query query= new Query();
query.addCriteria(Criteria.where("id").is(501));

Update update = new Update();
update.set("cost", 1065.25);
update.set("name", "Core Java");

mt.findAndModify(query, update, Book.class);
System.out.println("Data Modified");
```

If you check the result from DB also, you will find the updated values of the record.

## updateMulti(query, update, entityClassName)

This method updates all the records of the DB that matches the given criteria. For the example, in below code we are trying to update the cost of all Books by 999.0 which have pages count less than or equals to 180.

```
Query query= new Query();
query.addCriteria(Criteria.where("pages").lte(180));
Update update = new Update();
update.set("cost", 999.0);
mt.updateMulti(query, update, Book.class);
```

## findAndRemove(query, entityClassName)

We use this method to remove data which are actually fetched from a given condition.

```
Query query= new Query();
query.addCriteria(Criteria.where("cost").is(1749.0));
mt.findAndRemove(query, Book.class);
```

Here in the above example, we are deleting a Book whose cost is equal to 1749.0.

## findAllAndRemove(query, entityClassName)

We use this method to remove multiple records which are actually fetched from a given condition.

```
Query query= new Query();
query.addCriteria(Criteria.where("cost").gte(1000.0));
mt.findAllAndRemove(query, Book.class);
```

Here in the above example, we are deleting all the Books whose cost is greater than or equal to 1000.0.

## upsert(query, update, entityClassName)

This method should be new to many of us. When the given criteria matches any record in DB, upsert() will update that record. If it doesn't find any record of the given criteria, then it will insert a new record.

```
Query query= new Query();
query.addCriteria(Criteria.where("id").is(510));
Update update = new Update();
update.set("cost", 1065.25);
update.set("name", "Core Java");
mt.upsert(query, update, Book.class);
```

Here in the above example, we are trying to update a record which doesn't exist in the DB. Hence upsert() will create a new record with updated value leaving other fields as null. Below is the record which will get inserted after executing the above code.

```
Book(id=510, name=Core Java, pages=null, author=null, cost=1065.25)
```

## Conclusion

We have covered a bit of theoretical knowledge and enough amount of examples on

the important methods which are used by the people in the industry in most of the cases. Should you need any further assistance in this area, you may visit one of the official documentation.

## Links to Other tutorials on MongoDB with Spring Boot

Below are the links to learn MongoDB deeply with Spring Boot.

**MongoDB Basics & How to work with Spring Boot**
**Spring Boot MongoDB CRUD Examples**
**Spring Boot MongoDB @Query Examples**

| | | |
|---|---|---|
| Spring Boot MongoDB @Query Examples | Spring Boot MongoDB CRUD Example | What is Spring Boot ? |
| April 29, 2021 | April 23, 2021 | June 1, 2021 |
| In "java" | In "java" | In "java" |

Like

Tagged mongotemplate mongotemplate example spring boot mongotemplate in spring boot mongotemplate spring boot mongotemplate spring boot configuration mongotemplate spring boot example query mongodb spring Spring Boot + MongoTemplate spring boot mongodb mongotemplate example Spring Boot MongoDB using MongoTemplate Examples spring boot mongotemplate crud example Spring Boot MongoTemplate CRUD Examples spring boot mongotemplate example Spring Data mongodb bulk upsert example spring data mongodb complex query Spring Data MongoTemplate Example spring mongotemplate spring mongotemplate example

< Previous article
Spring Boot MongoDB @Query Examples

Next article >
JVM Architecture & Class Loaders Java

### Leave a Reply

Name *

Email Address *

Website

Comment *

Post Comment

TOP

☐ Yes, add me