(/)

Last modified: November 6, 2018

by baeldung (https://www.baeldung.com/author/baeldung/)

Spring Data (https://www.baeldung.com/category/persistence/spring-persistence/spring-data/)

MongoDB (https://www.baeldung.com/tag/mongodb/)

Spring Annotations (https://www.baeldung.com/tag/spring-annotations/)

1. Overview

This tutorial will explore some of the core features of Spring Data MongoDB – indexing, common annotations and converters.

2. Indexes

2.1. *@Indexed*

This annotation **marks the field as indexed** in MongoDB:

```
@QueryEntity
@Document
public class User {
    @Indexed
    private String name;

    ...
}
```

Now that the *name* field is indexed – let's have a look at the indexes in MongoDB:

```
db.user.getIndexes();
```

Here's what we have at the database level:

```
[
    {
        "v" : 1,
        "key" : {
            "_id" : 1
         },
        "name" : "_id_",
        "ns" : "test.user"
    },
    {
         "v" : 1,
         "key" : {
             "name" : 1
          },
          "name" : "name",
          "ns" : "test.user"
     }
]
```

As you can see, we have two indexes – one of them is *_id* – which was created by default due to the *@Id* annotation and **the second one is our *name* field.**

2.2. Create an Index Programmatically

We can also create an index programmatically:

```
mongoOps.indexOps(User.class).
  ensureIndex(new Index().on("name", Direction.ASC));
```

We've now created an index for the field *name* and the result will be the same as in the previous section.

2.3. Compound Indexes

MongoDB supports compound indexes, where a single index structure holds references to multiple fields.

Let's see a quick example using compound indexes:

```
@QueryEntity
@Document
@CompoundIndexes({
    @CompoundIndex(name = "email_age", def = "{'email.id' : 1, 'age': 1}")
})
public class User {
    //
}
```

We created a compound index with the *email* and *age* fields. Let's now check out the actual indexes:

```
{
    "v" : 1,
    "key" : {
        "email.id" : 1,
        "age" : 1
    },
    "name" : "email_age",
    "ns" : "test.user"
}
```

Note that a *DBRef* field cannot be marked with *@Index* – that field can only be part of a compound index.

3. Common Annotations

3.1 *@Transient*

As you would expect, this simple annotation excludes the field from being persisted in the database:

```
public class User {

    @Transient
    private Integer yearOfBirth;

    // standard getter and setter

}
```

Let's insert user with the setting field *yearOfBirth*:

```
User user = new User();
user.setName("Alex");
user.setYearOfBirth(1985);
mongoTemplate.insert(user);
```

Now if we look the state of database, we see that the filed *yearOfBirth* was not saved:

```
{
    "_id" : ObjectId("55d8b30f758fd3c9f374499b"),
    "name" : "Alex",
    "age" : null
}
```

So if we query and check:

```
mongoTemplate.findOne(Query.query(Criteria.where("name").is("Alex")), User.class).getYearOfBirth()
```

The result will be *null*.

3.2. *@Field*

@Field indicates the key to be used for the field in the JSON document:

```
@Field("email")
private EmailAddress emailAddress;
```

Now *emailAddress* will be saved in the database using the key *email:*

```
User user = new User();
user.setName("Brendan");
EmailAddress emailAddress = new EmailAddress();
emailAddress.setValue("a@gmail.com");
user.setEmailAddress(emailAddress);
mongoTemplate.insert(user);
```

And the state of the database:

```
{
    "_id" : ObjectId("55d076d80bad441ed114419d"),
    "name" : "Brendan",
    "age" : null,
    "email" : {
        "value" : "a@gmail.com"
    }
}
```

3.3. *@PersistenceConstructor* and *@Value*

@PersistenceConstructor marks a constructor, even one that's package protected, to be the primary constructor used by the persistence logic. The constructor arguments are mapped by name to the key values in the retrieved *DBObject*.

Let's look at this constructor for our *User* class:

```
@PersistenceConstructor
public User(String name, @Value("#root.age ?: 0") Integer age, EmailAddress emailAddress) {
    this.name =  name;
    this.age = age;
    this.emailAddress =  emailAddress;
}
```

Notice the use of the standard Spring *@Value* annotation here. It's with the help of this annotation that we can use the Spring Expressions to transform a key's value retrieved from the database before it is used to construct a domain object. That is a very powerful and highly useful feature here.

In our example if *age* is not set that it will be set to *0* by default.

Let's now see how it works:

```
User user = new User();
user.setName("Alex");
mongoTemplate.insert(user);
```

Our database will look:

```
{
    "_id" : ObjectId("55d074ca0bad45f744a71318"),
    "name" : "Alex",
    "age" : null
}
```

So the *age* field is *null*, but when we query the document and retrieve *age*:

```
mongoTemplate.findOne(Query.query(Criteria.where("name").is("Alex")), User.class).getAge();
```

The result will be 0.

4. Converters

Let's now take a look at another very useful feature in Spring Data MongoDB – converters, and specifically at the *MongoConverter*.

This is used to handle the mapping of all Java types to *DBObjects* when storing and querying these objects.

We have two options – we can either work with *MappingMongoConverter* – or *SimpleMongoConverter* in earlier versions (this was deprecated in Spring Data MongoDB M3 and its functionality has been moved into *MappingMongoConverter*).

Or we can write our own custom converter. To do that, we would need to implement the *Converter* interface and register the implementation in *MongoConfig*.

Let's look at **a quick example**. As you've seen in some of the JSON output here, all objects saved in a database have the field *_class* which is saved automatically. If however we'd like to skip that particular field during persistence, we can do that using a *MappingMongoConverter*.

First – here's the custom converter implementation:

```
@Component
public class UserWriterConverter implements Converter<User, DBObject> {
    @Override
    public DBObject convert(User user) {
        DBObject dbObject = new BasicDBObject();
        dbObject.put("name", user.getName());
        dbObject.put("age", user.getAge());
        if (user.getEmailAddress() != null) {
            DBObject emailDbObject = new BasicDBObject();
            emailDbObject.put("value", user.getEmailAddress().getValue());
            dbObject.put("email", emailDbObject);
        }
        dbObject.removeField("_class");
        return dbObject;
    }
}
```

Notice how we can easily hit the goal of not persisting *_class* by specifically removing the field directly here.

Now we need to register the custom converter:

```
private List<Converter<?,?>> converters = new ArrayList<Converter<?,?>>();

@Override
public MongoCustomConversions customConversions() {
    converters.add(new UserWriterConverter());
    return new MongoCustomConversions(converters);
}
```

We can of course achieve the same result with XML configuration as well, if we need to:

```
<bean id="mongoTemplate"
  class="org.springframework.data.mongodb.core.MongoTemplate">
    <constructor-arg name="mongo" ref="mongo"/>
    <constructor-arg ref="mongoConverter" />
    <constructor-arg name="databaseName" value="test"/>
</bean>

<mongo:mapping-converter id="mongoConverter" base-package="org.baeldung.converter">
    <mongo:custom-converters base-package="com.baeldung.converter" />
</mongo:mapping-converter>
```

Now, when we save a new user:

```
User user = new User();
user.setName("Chris");
mongoOps.insert(user);
```

The resulting document in the database no longer contains the class information:

```
{
    "_id" : ObjectId("55cf09790bad4394db84b853"),
    "name" : "Chris",
    "age" : null
}
```

5. Conclusion

In this tutorial we've covered some core concepts of working with Spring Data MongoDB – indexing, common annotations and converters.

The implementation of all these examples and code snippets **can be found in my github project** (https://github.com/eugenp/tutorials/tree/master/persistence-modules/spring-data-mongodb) – this is an Eclipse based project, so it should be easy to import and run as it is.

▲ newest ▲ oldest ▲ most voted

Guest

Kumar Sambhav Jain

How to register custom converters while using java config & spring boot?

+ 0 −

3 years ago

Guest

Eugen Paraschiv (https://www.baeldung.com/)

Hey Kumar – that's covered in the article, but if you need a link to the full config, here it is (https://github.com/eugenp/tutorials/blob/master/persistence-modules/spring-data-mongodb/src/main/java/com/baeldung/config/MongoConfig.java). Cheers,
Eugen.

+ 0 −

3 years ago

Guest

Kumar Sambhav Jain

Thanks Eugen for pointing out. ' extends AbstractMongoConfiguration' somehow hurts my eyes as it forces mt to override 'getDatabaseName()' & 'mongo()' method – something that I already have nicely placed in my application.yml file (spring.data.mongodb.database). I know I can inject those values in the Config class but doesn't look good to me. I managed by injecting 'MappingMongoConverter' and is seems to be working fine. You can have look :-
https://github.com/ksambhav/trueyes/blob/master/trueyes-crm/trueyes-crm-repo/src/main/java/com/samsoft/trueyes/crm/repo/CRMMongoRepositoryConfig.java (https://github.com/ksambhav/trueyes/blob/master/trueyes-crm/trueyes-crm-repo/src/main/java/com/samsoft/trueyes/crm/repo/CRMMongoRepositoryConfig.java)

+ 1 −

3 years ago

Guest

Eugen Paraschiv (https://www.baeldung.com/)Looks like an interesting, clean alternative. Maybe worthwhile for a future article in this series – thanks for the link. Cheers, Eugen.

+ 0 −

3 years ago

Guest

dali

i dont really understand the @index annotation and her role

+ 0 −

2 years ago

Guest

Grzegorz Piwowarek

Dali, you do not understand the concept of the index or just the annotation usage?

+ 0 −

2 years ago

Guest

dali

i do not understand the concept , i tried to understand it .. but failed

+ 0 −

2 years ago

Guest

Grzegorz Piwowarek

Simply put, db index is something that allows it to perform certain operations faster. Have you tried https://en.wikipedia.org/wiki/Database_index (https://en.wikipedia.org/wiki/Database_index) ?

+ 0 −

2 years ago

Guest

dali

its more clear now , thks , i search for an example to more understand

+ 0 −

2 years ago

Guest

Grzegorz Piwowarek

It's hard to show an actual example because the example usage in practice is sometimes just a few keywords or even one annotation. If you google, you should be able to find performance comparisons easily for tables with and without indexes

+ 0 −

2 years ago

Comments are closed on this article!

CATEGORIES

SPRING (HTTPS://WWW.BAELDUNG.COM/CATEGORY/SPRING/)
REST (HTTPS://WWW.BAELDUNG.COM/CATEGORY/REST/)
JAVA (HTTPS://WWW.BAELDUNG.COM/CATEGORY/JAVA/)
SECURITY (HTTPS://WWW.BAELDUNG.COM/CATEGORY/SECURITY-2/)
PERSISTENCE (HTTPS://WWW.BAELDUNG.COM/CATEGORY/PERSISTENCE/)
JACKSON (HTTPS://WWW.BAELDUNG.COM/CATEGORY/JSON/JACKSON/)
HTTP CLIENT-SIDE (HTTPS://WWW.BAELDUNG.COM/CATEGORY/HTTP/)
KOTLIN (HTTPS://WWW.BAELDUNG.COM/CATEGORY/KOTLIN/)

SERIES

JAVA "BACK TO BASICS" TUTORIAL (/JAVA-TUTORIAL)
JACKSON JSON TUTORIAL (/JACKSON)
HTTPCLIENT 4 TUTORIAL (/HTTPCLIENT-GUIDE)
REST WITH SPRING TUTORIAL (/REST-WITH-SPRING-SERIES)
SPRING PERSISTENCE TUTORIAL (/PERSISTENCE-WITH-SPRING-SERIES)
SECURITY WITH SPRING (/SECURITY-SPRING)

ABOUT

ABOUT BAELDUNG (/ABOUT)
THE COURSES (HTTPS://COURSES.BAELDUNG.COM)
CONSULTING WORK (/CONSULTING)
META BAELDUNG (HTTP://META.BAELDUNG.COM/)
THE FULL ARCHIVE (/FULL_ARCHIVE)
WRITE FOR BAELDUNG (/CONTRIBUTION-GUIDELINES)
EDITORS (/EDITORS)
OUR PARTNERS (/PARTNERS)
ADVERTISE ON BAELDUNG (/ADVERTISE)

TERMS OF SERVICE (/TERMS-OF-SERVICE)
PRIVACY POLICY (/PRIVACY-POLICY)
COMPANY INFO (/BAELDUNG-COMPANY-INFO)
CONTACT (/CONTACT)