Understanding Document References

Dan Geabunea
SENIOR SOFTWARE DEVELOPER

@romaniancoder www.romaniancoder.com

Overview

Mongo document references

Creating document references in Spring using @DbRef

Spring Mongo lifecycle events

Demo: Implementing document references & cascading

Mongo Document References

Document Reference

A way to link together multiple documents that are related

For most cases, the denormalized model where data is stored in a single document is optimal

A Simple Example

Bank Transaction 1

```
{
    “_id”: 1,
    “amount”: 3000,
    “customer”: {
        “name”: “John Doe”,
        “nationalIdNb”:”AZ456734”
    }
}
```

Bank Transaction 2

```
{
    “_id”: 2,
    “amount”: 1000,
    “customer”: {
        “name”: “John Doe”,
        “nationalIdNb”:”AZ456734”
    }
}
```

The Problem

Bank Transaction 1

```
{
    “_id”: 1,
    “amount”: 3000,
    “customer”: {
        “name”: “John Doe”,
        “nationalIdNb”:”BG90865”
    }
}
```

Bank Transaction 2

```
{
    “_id”: 2,
    “amount”: 1000,
    “customer”: {
        “name”: “John Doe”,
        “nationalIdNb”:”AZ456734”
    }
}
```

Relationships

Customer

```
{
    “_id”: 234,
    “name”: “John Doe”,
    “nationalIdNb”:”BG90865”
}
```

Bank Transaction

```
{
    “_id”: 849,
    “amount”: 1000,
    “customerId”: 234
}
```

Relating Documents

Manual

Saving the '_id' of a document in another document

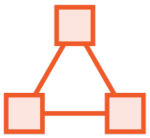
DBRefs

Link documents together by using the '_id' field, collection name and database name.

DBRef

Can link documents across collections or across different databases

To resolve DBRefs, the application must perform additional queries



Cascading is not supported

DBRef Format

$ref

The name of the collection where the linked document resides

$id

The value of the _id field of the referenced document

$db

The name of the database where the referenced document resides

Aircraft

```
{
    “_id”: ObjectId(“4e08277f-2660-43b6-ae71-ef2f5a4c9ce2”)
    “name”: 737
    “capacity”: 234
    “engine” : {
        “$ref”: “engines”,
        “$id”: ObjectId(“3e3df0ff-5531-4e9c-8a90-10e2a6faef8a”),
        “$db”: “atm”    // optional
    }
}
```

The order of fields in the DBRef matters and must be used in the correct sequence

DBRefs are not “smart”

Creating Document References in Spring Using @DBRef

Aircraft

```java
@Document
public class Aircraft {
    @Id private String id;
    private String model;
    private int maxPassengers;
    private Engine engine; // embedded document
}
```

Aircraft

```
@Document
public class Aircraft {
    @Id private String id;
    private String model;
    private int maxPassengers;
    @DBRef private Engine engine;   // document reference
}
```

Engine

```java
@Document
public class Engine {
    @Id private String id;
    private String type;
    private int maxPower;
}
```

Spring Data MongoDB fetches documents annotated with @DBRef for us automatically

```
Aircraft a = new Aircraft(“737”);
a.setEngine(new Engine(“GM”,1000));

aircraftRepositoy.insert(a);
```

Insert

You won’t get an error

But the engine is not getting saved in the Engine collection

Cascading does not work with DBRefs on save by default

```
Engine e = new Engine(“GM”,1000));
engineRepository.insert(e);

Aircraft a = new Aircraft(“737”);
a.setEngine(e);
aircraftRepositoy.insert(a);
```

The Correct Way

Save the (child) object first

Save the (parent/root) after that

```java
@Document
public class Aircraft {
    @Id private String id;
    @DBRef(lazy = true) private Engine engine;
}
```

Lazy Loading

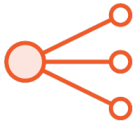
By default, @DBRef instructs the framework to perform an eager load

This can be changed by modifying the lazy property to “true”

- The sub-document is loaded when one of its fields/methods is accessed
- Useful for bi-directional references to avoid a cyclic dependency

@DBRef

@DBRef to creates document references



Eager loading by default but can be changed. The Spring framework fetches the referenced document for you and populates the POJO

Cascading on save is not available

You can hook into Mongo lifecycle events and implement cascading

Pay attention to queries on related documents

Filtering

Aircraft

```
@Document
public class Aircraft {
    @Id private String id;
    private Engine engine;
}
```

Repository

```
@Query("{'engine.maxPower': ?0}")
List<Aircraft> byPower(int pow);

// Will work
```

Filtering

Aircraft

```
@Document
public class Aircraft {
    @Id private String id;
    @DBRef private Engine engine;
}
```

Repository

```
@Query("{'engine.maxPower': ?0}")
List<Aircraft> byPower(int pow);

// Won't work
```

Spring Mongo Lifecycle Events

Lifecycle Events

Events that your application can respond to by registering special beans in the Spring application context.

“Before” Events

onBeforeConvert

onBeforeSave

onBeforeDelete

“After” Events

onAfterConvert

onAfterSave

onAfterLoad

onAfterDelete

Save / Update Document Events

Call insert or save in MongoTemplate

onBeforeConvert is called before the Java object is converted to a Document by the MongoConverter

onBeforeSave is called before inserting or saving the document in the database

onAfterSave is called after the document was inserted or saved in the database

Load Document Events

Call find, findOne, findAndRemove in MongoTemplate

onAfterLoad is called after the Document has been retrieved from the database

onAfterConvert is called after the Document has been converted into a POJO

Delete Document Events

Call remove on MongoTemplate

onBeforeDelete is called just before the document gets deleted from the database

onAfterDelete is called after the Document has been deleted from the database

Event Behavior

Lifecycle events are emitted only for root level types. Sub-documents are not subject to event publication unless they are annotated with @DBRef

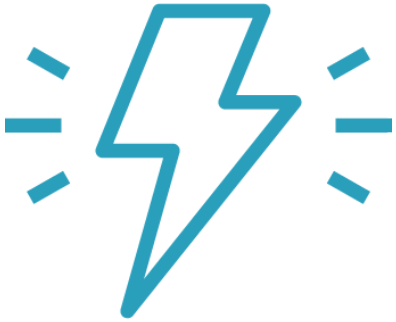
It is all happening in an async fashion. We have no guarantees to when an event is processed.

“Why would I ever want to hook into these events?”

You

Use Cases

Implement cascade on save



Trigger some job/action in different systems

Security audits

“All right, so how can I hook into these events and execute my custom logic”

You

AbstractMongoEventListener

```
public abstract class AbstractMongoEventListener<E>
    implements ApplicationListener<MongoMappingEvent<?>> {

    public void onAfterConvert(AfterConvertEvent<E> event) {…}
    public void onAfterDelete(AfterDeleteEvent<E> event) {…}
    public void onAfterLoad(AfterLoadEvent<E> event) {…}

    //…
}
```

BeforeSaveListener

```
public class BeforeConvertListener
    extends AbstractMongoEventListener<Flight> {

    @Override
    public void onBeforeConvert(BeforeConvertEvent<Flight> event) {
        // your custom logic…
    }
}
```

BeforeSaveListener

```
@Component
public class BeforeSaveListener
    extends AbstractMongoEventListener<Flight> {

    @Override
    public void onBeforeConvert(BeforeConvertEvent<Flight> event) {
        // your custom logic…
    }
}
```

Create an event listener per feature

Demo

Demo: Implementing Document References

- @DBRef
- Implement cascading and auditing using lifecycle events

Summary

Documents that are related can be linked via manual references or by DBRefs

The Spring Data Mongo framework does things under the hood to make your experience easier with @DBRef

Cascading on save is not implemented for DBRefs, but you can achieve this by hooking into lifecycle events

Mongo lifecycle events are a great way to add valuable logic in key persistence moments

For nearly every case where you want to store a relationship between two documents, use manual references. Your application can resolve references as needed.

MongoDB official documentation

However, if you need to reference documents from multiple collections/databases, consider using DBRefs.

MongoDB official documentation

From My Experience Using Mongo with Spring