Filters Builders

https://www.mongodb.com/docs/drivers/java/sync/current/fundamentals/builders/filters/

Overview

In this guide, you can learn how to use **builders** to specify **filters** for your queries in the MongoDB Java driver.

Builders are classes provided by the MongoDB Java driver that help you construct BSON objects. To learn more, see our guide on builders.

Filters are the operations MongoDB uses to limit your results to what you want to see.

Some places we use filters are:

- `find()`
- match stage of aggregation
- `deleteOne()`/`deleteMany()`
- `updateOne()`/`updateMany()`

Some examples of results from queries with filters are:

- Items that cost $0 to $25
- A hotel with amenities that include an indoor swimming pool and free parking
- A food critic review that mentions "spicy"

This guide shows you how to use builders with examples of the following types of operators:

- Comparison
- Logical
- Arrays
- Elements
- Evaluation
- Bitwise
- Geospatial

The `Filters` class provides static factory methods for all the MongoDB query operators. Each method returns an instance of the BSON type, which you can pass to any method that expects a query filter.

TIP

For brevity, you can choose to import all methods of the Filters class statically:

```
import static com.mongodb.client.model.Filters.*;
```

The following examples assume this static import.

The Filter examples in this guide use the following sample collections:

Collection: `paint_purchases`

```
{ "_id": 1, "color": "red", "qty": 5, "vendor": ["A"] }
{ "_id": 2, "color": "purple", "qty": 10, "vendor": ["C", "D"] }
{ "_id": 3, "color": "blue", "qty": 8, "vendor": ["B", "A"] }
{ "_id": 4, "color": "white", "qty": 6, "vendor": ["D"] }
{ "_id": 5, "color": "yellow", "qty": 11, "vendor": ["A", "B"] }
```

{ "_id": 6, "color": "pink", "qty": 5, "vendor": ["C"] }
{ "_id": 7, "color": "green", "qty": 8,"vendor": ["B", "C"] }
{ "_id": 8, "color": "orange", "qty": 7, "vendor": ["A", "D"] }

Collection: `binary_numbers`

{ "_id": 9, "a": 54, "binaryValue": "00110110" }
{ "_id": 10, "a": 20, "binaryValue": "00010100" }
{ "_id": 11, "a": 68, "binaryValue": "1000100" }
{ "_id": 12, "a": 102, "binaryValue": "01100110" }

Collection: `geo_points`

{ "_id": 13, "coordinates": { "type": "Point", "coordinates": [2.0, 2.0] } }
{ "_id": 14, "coordinates": { "type": "Point", "coordinates": [5.0, 6.0] } }
{ "_id": 15, "coordinates": { "type": "Point", "coordinates": [1.0, 3.0] } }
{ "_id": 16, "coordinates": { "type": "Point", "coordinates": [4.0, 7.0] } }

Comparison

The comparison filters include all operators that compare the value in a document to a specified value.

The comparison operator methods include:

Comparison Method	Matches
eq()	values equal to a specified value.
gt()	values greater than a specified value.
gte()	values greater than or equal to a specified value.
lt()	values less than a specified value.
lte()	values less than or equal to a specified value.

Comparison Method	Matches
ne()	values not equal to a specified value.
in()	any of the values specified in an array.
nin()	none of the values specified in an array.
empty()	all the documents.

The following example creates a filter that matches all documents where the value of the `qty` field equals "5" in the `paint_purchases` collection:

```
Bson equalComparison = eq("qty", 5);
collection.find(equalComparison).forEach(doc -> System.out.println(doc.toJson()));
```

The following shows the output of the preceding query:

```
{ "_id": 1, "color": "red", "qty": 5, "vendor": ["A"] }
{ "_id": 6, "color": "pink", "qty": 5, "vendor": ["C"] }
```

The following example creates a filter that matches all documents where the value of the `qty` field is greater than or equal to "10" in the `paint_purchases` collection:

```
Bson gteComparison = gte("qty", 10);
collection.find(gteComparison).forEach(doc -> System.out.println(doc.toJson()));
```

The following shows the output of the preceding query:

```
{ "_id": 2, "color": "purple", "qty": 10, "vendor": ["C", "D"] }
{ "_id": 5, "color": "yellow", "qty": 11, "vendor": ["A", "B"] }
```

The following example creates a filter that matches all documents in the `paint_purchases` collection because the predicate is empty:

```
Bson emptyComparison = empty();
collection.find(emptyComparison).forEach(doc -> System.out.println(doc.toJson()));
```

The output of the preceding query consists of all the documents in the collection.

{ "_id": 1, "color": "red", "qty": 5, "vendor": ["A"] }
{ "_id": 2, "color": "purple", "qty": 10, "vendor": ["C", "D"] }
{ "_id": 3, "color": "blue", "qty": 8, "vendor": ["B", "A"] }
...

Logical

The logical operators perform logical operations based on the conditions of the specified method.

The logical operator methods include:

Logical Method	Matches
and()	documents with the conditions of all the filters. This operator joins filters with a logical `AND`.
or()	documents with the conditions of either filter. This operator joins filters with a logical `OR`.
not()	documents that do not match the filter.
nor()	documents that fail to match both filters. This operator joins filters with a logical `NOR`.

The following example creates a filter that matches documents where the value of the `qty` field is greater than "8" or the value of the `color` field equals "pink" in the `paint_purchases` collection:

```
Bson orComparison = or(gt("qty", 8), eq("color", "pink"));
collection.find(orComparison).forEach(doc -> System.out.println(doc.toJson()));
```

The following shows the output of the preceding query:

{ "_id": 2, "color": "purple", "qty": 10, "vendor": ["C", "D"] }
{ "_id": 5, "color": "yellow", "qty": 11, "vendor": ["A", "B"] }
{ "_id": 6, "color": "pink", "qty": 5, "vendor": ["C"] }

Arrays

The array operators evaluate the array field in a document.

The array operator methods include:

Array Method	Matches
all()	documents if the array field contains every element specified in the query.
elemMatch()	documents if an element in the array field matches all the specified conditions.
size()	documents if the array field is a specified number of elements.

The following example matches documents with a `vendors` array containing both "A" and "D" in the `paint_purchases` collection:

```
List<String> search = Arrays.asList("A", "D");
Bson allComparison = all("vendor", search);
collection.find(allComparison).forEach(doc -> System.out.println(doc.toJson()));
```

The following shows the output of the preceding query:

```
{ "_id": 8, "color": "orange", "qty": 7, "vendor": ["A", "D"] }
```

Elements

The elements operators evaluate the nature of a specified field.

The elements operator methods include:

Elements Method	Matches
exists()	documents that have the specified field.
type()	documents if a field is of the specified type.

The following example matches documents that have a `qty` field and its value does not equal "5" or "8" in the `paint_purchases` collection:

```
Bson existsComparison = and(exists("qty"), nin("qty", 5, 8));
collection.find(existsComparison).forEach(doc -> System.out.println(doc.toJson()));
```

The following shows the output of the preceding query:

```
{ "_id": 2, "color": "purple", "qty": 10, "vendor": ["C", "D"] }
{ "_id": 4, "color": "white", "qty": 6, "vendor": ["D"]}
{ "_id": 5, "color": "yellow", "qty": 11, "vendor": ["A", "B"] }
{ "_id": 8, "color": "orange", "qty": 7, "vendor": ["A", "D"] }
```

Evaluation

The evaluation operators evaluate the value of any field in a document.

The evaluation operator methods include:

Evaluation Method	Matches
mod()	documents where a modulo operation on a field value produces a specified result.
regex()	documents where values contain a specified regular expression.
text()	documents which contain a specified full-text search expression.
where()	documents which contain a specified JavaScript expression.

The following example matches documents that have a `color` field starting with the letter "p" in the `paint_purchases` collection:

```
Bson regexComparison = regex("color", "^p");
collection.find(regexComparison).forEach(doc -> System.out.println(doc.toJson()));
```

The following shows the output of the preceding query:

```
{ "_id": 2, "color": "purple", "qty": 10, "vendor": ["C", "D"] }
{ "_id": 6, "color": "pink", "qty": 5, "vendor": ["C"] }
```

Bitwise

The bitwise operators convert a number into its binary value to evaluate its bits.

The bitwise operator methods include:

Bitwise Method	Matches
bitsAllSet()	documents where the specified bits of a field are set (i.e. "1").
bitsAllClear()	documents where the specified bits of a field are clear (i.e. "0").
bitsAnySet()	documents where at least one of the specified bits of a field are set (i.e. "1").
bitsAnyClear()	documents where at least one of the specified bits of a field are clear (i.e. "0").

The following example matches documents that have a `bitField` field with bits set at positions of the corresponding bitmask "34" (i.e. "00100010") in the `binary_numbers` collection:

```
Bson bitsComparison = bitsAllSet("a", 34);
collection.find(bitsComparison).forEach(doc -> System.out.println(doc.toJson()));
```

The following shows the output of the preceding query:

{ "_id": 9, "a": 54, "binaryValue": "00110110" }
{ "_id": 12, "a": 102, "binaryValue": "01100110" }

Geospatial

The geospatial operators evaluate a specified coordinate and its relation to a shape or location.

The geospatial operator methods include:

Geospatial Method	Matches
geoWithin()	documents containing a GeoJSON geometry value that falls within a bounding GeoJSON geometry.
geoWithinBox()	documents containing a coordinates value that exist within the specified box.
geoWithinPolygon()	documents containing a coordinates value that exist within the specified polygon.
geoWithinCenter()	documents containing a coordinates value that exist within the specified circle.
geoWithinCenterSphere()	geometries containing a geospatial data value (GeoJSON or legacy coordinate pairs) that exist within the specified circle, using spherical geometry.
geoIntersects()	geometries that intersect with a GeoJSON geometry. The `2dsphere` index supports `$geoIntersects`.
near()	geospatial objects in proximity to a point. Requires a geospatial index. The `2dsphere` and `2d` indexes support `$near`.
nearSphere()	geospatial objects in proximity to a point on a sphere. Requires a geospatial index. The `2dsphere` and `2d` indexes support `$nearSphere`.

The following example creates a filter that matches documents in which the `point` field contains a GeoJSON geometry that falls within the given Polygon in the `geo_points` collection:

```
Polygon square = new Polygon(Arrays.asList(new Position(0, 0),
```

```
new Position(4, 0),
new Position(4, 4),
new Position(0, 4),
new Position(0, 0)));
// Prints documents that contain "coordinates" values that are within the bounds of the polygon passed as the filter parameter
Bson geoWithinComparison = geoWithin("coordinates", square);
collection.find(geoWithinComparison).forEach(doc -> System.out.println(doc.toJson()));
```

The following shows the output of the preceding query:

```
{ "_id": 13, "coordinates": {"type": "Point", "coordinates": [2.0, 2.0]} }
{ "_id": 15, "coordinates": {"type": "Point", "coordinates": [1.0, 3.0]} }
```