Java Development with MongoDB

Brendan W. McAdams

Novus Partners, Inc.
http://novus.com

May 2010 / Mongo NYC

NOVUS

Outline

NOVUS

Outline

NOVUS

Adding the MongoDB Driver To Your Project

Assuming you're using a dependency manager, make setup simple...

Listing 1: Maven Dependency

```
<dependency>
  <groupId>org.mongodb</groupId>
  <artifactId>mongo-java-driver</artifactId>
  <version>1.4</version>
</dependency>
```

Listing 2: Ivy Dependency

```
<dependency org="org.mongodb"
            name="mongo-java-driver"
            rev="1.4"/>
```

NOVUS

Adding the MongoDB Driver To Your Project

Assuming you're using a dependency manager, make setup simple...

Listing 3: Maven Dependency

```
<dependency>
  <groupId>org.mongodb</groupId>
  <artifactId>mongo-java-driver</artifactId>
  <version>1.4</version>
</dependency>
```

Listing 4: Ivy Dependency

```
<dependency org="org.mongodb"
            name="mongo-java-driver"
            rev="1.4"/>
```

NOVUS

Outline

NOVUS

Simple Connection

Getting connected to MongoDB is simple; Connections are pooled, so you only need one...

```
import com.mongodb.Mongo;
import com.mongodb.DB;
Mongo m = new Mongo();
Mongo m = new Mongo("localhost");
Mongo m = new Mongo("localhost", 27017);
```

Fetch a Database Handle (lazy)...

```
DB db = m.getDB("javaDemo");
```

If you need to authenticate...

```
boolean auth = db.authenticate("login", "password");
```

NOVUS

Simple Connection

Getting connected to MongoDB is simple; Connections are pooled, so you only need one...

```
import com.mongodb.Mongo;
import com.mongodb.DB;
Mongo m = new Mongo();
Mongo m = new Mongo("localhost");
Mongo m = new Mongo("localhost", 27017);
```

Fetch a Database Handle (lazy)...

```
DB db = m.getDB("javaDemo");
```

If you need to authenticate...

```
boolean auth = db.authenticate("login", "password");
```

NOVUS

Simple Connection

Getting connected to MongoDB is simple; Connections are pooled, so you only need one...

```
import com.mongodb.Mongo;
import com.mongodb.DB;
Mongo m = new Mongo();
Mongo m = new Mongo("localhost");
Mongo m = new Mongo("localhost", 27017);
```

Fetch a Database Handle (lazy)...

```
DB db = m.getDB("javaDemo");
```

If you need to authenticate...

```
boolean auth = db.authenticate("login", "password");
```

NOVUS

Outline

NOVUS

Working with Collections

Collections are MongoDB "tables"...

- List all of the collections in a database...

```
Set<String> = colls = db.getCollectionNames();
for (String s : colls) {
    System.out.println(s);
}
```

- Get a specific collection (lazy)...

```
DBCollection coll = db.getCollection("testData");
```

- Count the number of documents in a collection...

```
coll.getCount();
```

Working with Collections

Collections are MongoDB "tables"...

- List all of the collections in a database...

```
Set<String> = colls = db.getCollectionNames();
for (String s : colls) {
    System.out.println(s);
}
```

- Get a specific collection (lazy)...

```
DBCollection coll = db.getCollection("testData");
```

- Count the number of documents in a collection...

```
coll.getCount();
```

MongoDB Documents
BSON

- "Documents" are MongoDB's "rows".
- MongoDB's Internal Document representation is '**BSON**'
  - **BSON** is a binary optimized flavor of **JSON**
  - Corrects **JSON**'s inefficiency in string encoding (Base64)
  - Supports extras including Regular Expressions, Byte Arrays, DateTimes & Timestamps, as well as datatypes for Javascript code blocks & functions.
  - Creative Commons licensed.
  - **BSON** implementation being split into its own package in most drivers.
  - **bsonspec.org**

NOVUS

MongoDB Documents
From Java: BasicDBObject

- Java representation of **BSON** is the map-like **DBObject** (Java 2.0 driver has a new base class of **BSONObject** related to the **BSON** split-off)
- Easiest way to work with Mongo Documents is BasicDBObject...
  - **BasicDBObject** implements **java.util.LinkedHashMap<String, Object>**
  - Mutable object
  - Can take a **Map** as a constructor parameter
  - Example:

```
DBObject doc = new BasicDBObject();
doc.put("username", "bwmcadams");
doc.put("password", "MongoNYC");
```

  - *toString* returns a **JSON** serialization.
- Use **BasicDBList** (implements **java.util.ArrayList<Object>**) to represent Arrays.

NOVUS

MongoDB Documents
From Java: BasicDBObject

- Java representation of **BSON** is the map-like **DBObject** (Java 2.0 driver has a new base class of **BSONObject** related to the **BSON** split-off)
- Easiest way to work with Mongo Documents is BasicDBObject. . .
  - **BasicDBObject** implements **java.util.LinkedHashMap<String, Object>**
  - Mutable object
  - Can take a **Map** as a constructor parameter
  - Example:

```
DBObject doc = new BasicDBObject();
doc.put("username", "bwmcadams");
doc.put("password", "MongoNYC");
```

  - *toString* returns a **JSON** serialization.
- Use **BasicDBList** (implements **java.util.ArrayList<Object>**) to represent Arrays.

NOVUS

MongoDB Documents
From Java: BasicDBObject

- Java representation of **BSON** is the map-like **DBObject** (Java 2.0 driver has a new base class of **BSONObject** related to the **BSON** split-off)
- Easiest way to work with Mongo Documents is BasicDBObject. . .
  - **BasicDBObject** implements **java.util.LinkedHashMap<String, Object>**
  - Mutable object
  - Can take a **Map** as a constructor parameter
  - Example:

```
DBObject doc = new BasicDBObject();
doc.put("username", "bwmcadams");
doc.put("password", "MongoNYC");
```

  - *toString* returns a **JSON** serialization.
- Use **BasicDBList** (implements **java.util.ArrayList<Object>**) to represent Arrays.

NOVUS

MongoDB Documents
From Java: BasicDBObject

- Java representation of **BSON** is the map-like **DBObject** (Java 2.0 driver has a new base class of **BSONObject** related to the **BSON** split-off)
- Easiest way to work with Mongo Documents is BasicDBObject...
  - **BasicDBObject** implements **java.util.LinkedHashMap<String, Object>**
  - Mutable object
  - Can take a **Map** as a constructor parameter
  - Example:

```
DBObject doc = new BasicDBObject();
doc.put("username", "bwmcadams");
doc.put("password", "MongoNYC");
```

  - *toString* returns a **JSON** serialization.
- Use **BasicDBList** (implements **java.util.ArrayList<Object>**) to represent Arrays.

NOVUS

MongoDB Documents
From Java: BasicDBObject

- Java representation of **BSON** is the map-like **DBObject** (Java 2.0 driver has a new base class of **BSONObject** related to the **BSON** split-off)
- Easiest way to work with Mongo Documents is BasicDBObject...
  - **BasicDBObject** implements **java.util.LinkedHashMap<String, Object>**
  - Mutable object
  - Can take a **Map** as a constructor parameter
  - Example:

```
DBObject doc = new BasicDBObject();
doc.put("username", "bwmcadams");
doc.put("password", "MongoNYC");
```

  - *toString* returns a **JSON** serialization.
- Use **BasicDBList** (implements **java.util.ArrayList<Object>**) to represent Arrays.

NOVUS

MongoDB Documents
From Java: BasicDBObject

- Java representation of **BSON** is the map-like **DBObject** (Java 2.0 driver has a new base class of **BSONObject** related to the **BSON** split-off)
- Easiest way to work with Mongo Documents is BasicDBObject...
  - **BasicDBObject** implements **java.util.LinkedHashMap<String, Object>**
  - Mutable object
  - Can take a **Map** as a constructor parameter
  - Example:

```
DBObject doc = new BasicDBObject();
doc.put("username", "bwmcadams");
doc.put("password", "MongoNYC");
```

  - *toString* returns a **JSON** serialization.
- Use **BasicDBList** (implements **java.util.ArrayList<Object>**) to represent Arrays.

NOVUS

MongoDB Documents
From Java: BasicDBObject

- Java representation of **BSON** is the map-like **DBObject** (Java 2.0 driver has a new base class of **BSONObject** related to the **BSON** split-off)
- Easiest way to work with Mongo Documents is BasicDBObject...
  - **BasicDBObject** implements **java.util.LinkedHashMap<String, Object>**
  - Mutable object
  - Can take a **Map** as a constructor parameter
  - Example:

```
DBObject doc = new BasicDBObject();
doc.put("username", "bwmcadams");
doc.put("password", "MongoNYC");
```

  - *toString* returns a **JSON** serialization.
- Use **BasicDBList** (implements **java.util.ArrayList<Object>**) to represent Arrays.

NOVUS

MongoDB Documents
From Java: BasicDBObjectBuilder

For those who prefer immutability...

- **BasicDBObjectBuilder** follows the *Builder* pattern.
- *add()* your keys & values:

```
BasicDBObjectBuilder builder = BasicDBObjectBuilder.start();
builder.add("username", "bwmcadams");
builder.add("password", "MongoNYC");
builder.add("presentation", "Java Development with MongoDB");
```

- A **BasicDBObjectBuilder** is not a **DBObject**.
- Call *get()* to return the built-up **DBObject**.
- *add()* returns itself so you can chain calls instead:

```
BasicDBOBjectBuilder.start().add("username", "bwmcadams").add("
    password", "MongoNYC").add("presentation", "Java Development
    with MongoDB").get();
```

NOVUS

MongoDB Documents
From Java: BasicDBObjectBuilder

For those who prefer immutability...

- **BasicDBObjectBuilder** follows the *Builder* pattern.
- *add()* your keys & values:

```
BasicDBObjectBuilder builder = BasicDBObjectBuilder.start();
builder.add("username", "bwmcadams");
builder.add("password", "MongoNYC");
builder.add("presentation", "Java Development with MongoDB");
```

- A **BasicDBObjectBuilder** is not a **DBObject**.
- Call *get()* to return the built-up **DBObject**.
- *add()* returns itself so you can chain calls instead:

```
BasicDBOBjectBuilder.start().add("username", "bwmcadams").add("
    password", "MongoNYC").add("presentation", "Java Development
    with MongoDB").get();
```

NOVUS

MongoDB Documents
From Java: BasicDBObjectBuilder

For those who prefer immutability...

- **BasicDBObjectBuilder** follows the *Builder* pattern.
- *add()* your keys & values:

```
BasicDBObjectBuilder builder = BasicDBObjectBuilder.start();
builder.add("username", "bwmcadams");
builder.add("password", "MongoNYC");
builder.add("presentation", "Java Development with MongoDB");
```

- A **BasicDBObjectBuilder** is not a **DBObject**.
- Call *get()* to return the built-up **DBObject**.
- *add()* returns itself so you can chain calls instead:

```
BasicDBOBjectBuilder.start().add("username", "bwmcadams").add("
    password", "MongoNYC").add("presentation", "Java Development
    with MongoDB").get();
```

NOVUS

MongoDB Documents
From Java: BasicDBObjectBuilder

For those who prefer immutability...

- **BasicDBObjectBuilder** follows the *Builder* pattern.
- *add()* your keys & values:

```
BasicDBObjectBuilder builder = BasicDBObjectBuilder.start();
builder.add("username", "bwmcadams");
builder.add("password", "MongoNYC");
builder.add("presentation", "Java Development with MongoDB");
```

- A **BasicDBObjectBuilder** is not a **DBObject**.
- Call *get()* to return the built-up **DBObject**.
- *add()* returns itself so you can chain calls instead:

```
BasicDBOBjectBuilder.start().add("username", "bwmcadams").add("
    password", "MongoNYC").add("presentation", "Java Development
    with MongoDB").get();
```

NOVUS

MongoDB Documents
Implementation by Extension: DBObject

- If you want to create your own concrete objects, extend & implement **DBObject**
  - Requires you implement a map-like interface including ability to get & set fields by key (even if you don't use them, required to deserialize)
  - Instances of **DBObject** can be saved directly to MongoDB.
- Feeling Fancy? Reflect instead. . .
  - Use **ReflectionDBObject** as a base class for your *Beans*.
  - **ReflectionDBObject** uses reflection to proxy your getters & setters and behave like a DBObject.
  - Downside: With Java's single inheritance you are stuck using this as your base class.
- Existing Object Model? ORM-Like Solutions. . .
  - Morphia (uses JPA-style annotations)
  - Daybreak (annotation based)
  - Mungbean (more Java-ey query syntax & PoJo Mapping)

NOVUS™

MongoDB Documents
Implementation by Extension: DBObject

- If you want to create your own concrete objects, extend & implement **DBObject**
  - Requires you implement a map-like interface including ability to get & set fields by key (even if you don't use them, required to deserialize)
  - Instances of **DBObject** can be saved directly to MongoDB.
- Feeling Fancy? Reflect instead...
  - Use **ReflectionDBObject** as a base class for your *Beans*.
  - **ReflectionDBObject** uses reflection to proxy your getters & setters and behave like a DBObject.
  - Downside: With Java's single inheritance you are stuck using this as your base class.
- Existing Object Model? ORM-Like Solutions...
  - Morphia (uses JPA-style annotations)
  - Daybreak (annotation based)
  - Mungbean (more Java-ey query syntax & PoJo Mapping)

NOVUS

MongoDB Documents
Implementation by Extension: DBObject

- If you want to create your own concrete objects, extend & implement **DBObject**
  - Requires you implement a map-like interface including ability to get & set fields by key (even if you don't use them, required to deserialize)
  - Instances of **DBObject** can be saved directly to MongoDB.
- Feeling Fancy? Reflect instead...
  - Use **ReflectionDBObject** as a base class for your *Beans*.
  - **ReflectionDBObject** uses reflection to proxy your getters & setters and behave like a DBObject.
  - Downside: With Java's single inheritance you are stuck using this as your base class.
- Existing Object Model? ORM-Like Solutions...
  - Morphia (uses JPA-style annotations)
  - Daybreak (annotation based)
  - Mungbean (more Java-ey query syntax & PoJo Mapping)

NOVUS

Briefly: Working with DBObjects

Getting data from DBObjects is relatively simple in Java...

- Check if a field (a.k.a. Key) exists with *containsField()*
- Get a **Set<String>** of a **DBObject**'s field names with *keySet()*
- Get a specific field with *get(String key)*. This returns *Object* so you will need to cast to an expected value.

```
// DBObject doc = <some row fetched from MongoDB>
String username = (String) doc.get("username")
```

- Call *toMap()* to get back a **Map** (String, Object) instead.

NOVUS

Briefly: Working with DBObjects

Getting data from DBObjects is relatively simple in Java...

- Check if a field (a.k.a. Key) exists with *containsField()*
- Get a **Set<String>** of a **DBObject**'s field names with *keySet()*
- Get a specific field with *get(String key)*. This returns *Object* so you will need to cast to an expected value.

```
// DBObject doc = <some row fetched from MongoDB>
String username = (String) doc.get("username")
```

- Call *toMap()* to get back a **Map** (String, Object) instead.

NOVUS

Briefly: Working with DBObjects

Getting data from DBObjects is relatively simple in Java...

- Check if a field (a.k.a. Key) exists with *containsField()*
- Get a **Set<String>** of a **DBObject**'s field names with *keySet()*
- Get a specific field with *get(String key)*. This returns *Object* so you will need to cast to an expected value.

```
// DBObject doc = <some row fetched from MongoDB>
String username = (String) doc.get("username")
```

- Call *toMap()* to get back a **Map** (String, Object) instead.

NOVUS

Briefly: Working with DBObjects

Getting data from DBObjects is relatively simple in Java...

- Check if a field (a.k.a. Key) exists with *containsField()*
- Get a **Set<String>** of a **DBObject**'s field names with *keySet()*
- Get a specific field with *get(String key)*. This returns *Object* so you will need to cast to an expected value.

```
// DBObject doc = <some row fetched from MongoDB>
String username = (String) doc.get("username")
```

- Call *toMap()* to get back a **Map** (String, Object) instead.

NOVUS

Briefly: Working with DBObjects

Getting data from DBObjects is relatively simple in Java...

- Check if a field (a.k.a. Key) exists with *containsField()*
- Get a **Set<String>** of a **DBObject**'s field names with *keySet()*
- Get a specific field with *get(String key)*. This returns *Object* so you will need to cast to an expected value.

```
// DBObject doc = <some row fetched from MongoDB>
String username = (String) doc.get("username")
```

- Call *toMap()* to get back a **Map** (String, Object) instead.

NOVUS

Outline

NOVUS

Inserting Documents

- One at a time:

```
DBObject doc =
    BasicDBOBjectBuilder.start().
        add("username", "bwmcadams").
        add("password", "MongoNYC").
        add("presentation", "Java Development with MongoDB").
        get();
// DBCollection coll
coll.insert(doc);
```

- Got multiple documents? Call *insert()* in a loop, or pass **DBObject[]** or **List<DBObject>**
- Three ways to store your documents:
  - INSERT (*insert()*) always attempts to add a new row.
  - SAVE (*save()*) only attempts to insert unless *_id* is defined. Otherwise, it will attempt to update the identified document.
  - UPDATE (*update()*) Allows you to pass a query to filter by and the fields to change. Boolean option "multi" specifies if multiple documents should be updated. Boolean "upsert" specifies that the object should be inserted if it doesn't exist (e.g. query doesn't match).

Inserting Documents

- One at a time:

```
DBObject doc =
    BasicDBOBjectBuilder.start().
        add("username", "bwmcadams").
        add("password", "MongoNYC").
        add("presentation", "Java Development with MongoDB").
        get();
// DBCollection coll
coll.insert(doc);
```

- Got multiple documents? Call *insert()* in a loop, or pass **DBObject[]** or **List<DBObject>**
- Three ways to store your documents:
  - INSERT (*insert()*) always attempts to add a new row.
  - SAVE (*save()*) only attempts to insert unless *_id* is defined. Otherwise, it will attempt to update the identified document.
  - UPDATE (*update()*) Allows you to pass a query to filter by and the fields to change. Boolean option "multi" specifies if multiple documents should be updated. Boolean "upsert" specifies that the object should be inserted if it doesn't exist (e.g. query doesn't match).

Inserting Documents

- One at a time:

```
DBObject doc =
    BasicDBOBjectBuilder.start().
        add("username", "bwmcadams").
        add("password", "MongoNYC").
        add("presentation", "Java Development with MongoDB").
        get();
// DBCollection coll
coll.insert(doc);
```

- Got multiple documents? Call *insert()* in a loop, or pass **DBObject[]** or **List<DBObject>**
- Three ways to store your documents:
  - INSERT (*insert()*) always attempts to add a new row.
  - SAVE (*save()*) only attempts to insert unless *_id* is defined. Otherwise, it will attempt to update the identified document.
  - UPDATE (*update()*) Allows you to pass a query to filter by and the fields to change. Boolean option "multi" specifies if multiple documents should be updated. Boolean "upsert" specifies that the object should be inserted if it doesn't exist (e.g. query doesn't match).

Inserting Documents

- One at a time:

```
DBObject doc =
    BasicDBOBjectBuilder.start().
        add("username", "bwmcadams").
        add("password", "MongoNYC").
        add("presentation", "Java Development with MongoDB").
        get();
// DBCollection coll
coll.insert(doc);
```

- Got multiple documents? Call *insert()* in a loop, or pass **DBObject[]** or **List<DBObject>**
- Three ways to store your documents:
  - INSERT (*insert()*) always attempts to add a new row.
  - SAVE (*save()*) only attempts to insert unless *_id* is defined. Otherwise, it will attempt to update the identified document.
  - UPDATE (*update()*) Allows you to pass a query to filter by and the fields to change. Boolean option "multi" specifies if multiple documents should be updated. Boolean "upsert" specifies that the object should be inserted if it doesn't exist (e.g. query doesn't match).

Outline

NOVUS

MongoDB Querying
Basics

- Find a single row with *findOne()*. Takes the first row returned.
- Getting a cursor of all documents (*find()* with no query):

```
DBCursor cur = coll.find();
while (DBObject doc : cur) {
    System.out.println(doc);
}
```

- Query for a specific value...

```
DBObject q = new BasicDBObjectBuilder.
                 start().
                 add("username", "bwmcadams").
                 get();
DBObject doc = coll.findOne(q);
```

NOVUS

MongoDB Querying
Basics

- Find a single row with *findOne()*. Takes the first row returned.
- Getting a cursor of all documents (*find()* with no query):

```
DBCursor cur = coll.find();
while (DBObject doc : cur) {
    System.out.println(doc);
}
```

- Query for a specific value...

```
DBObject q = new BasicDBObjectBuilder.
                 start().
                 add("username", "bwmcadams").
                 get();
DBObject doc = coll.findOne(q);
```

NOVUS

MongoDB Querying
Basics

- Find a single row with *findOne()*. Takes the first row returned.
- Getting a cursor of all documents (*find()* with no query):

```
DBCursor cur = coll.find();
while (DBObject doc : cur) {
    System.out.println(doc);
}
```

- Query for a specific value...

```
DBObject q = new BasicDBObjectBuilder.
                 start().
                 add("username", "bwmcadams").
                 get();
DBObject doc = coll.findOne(q);
```

NOVUS

MongoDB Querying
Basics

- You can pass an optional second **DBObject** parameter to *find()* and *findOne()* which specifies the fields to return.
- If you have an embedded object (for example, an address object) you can retrieve it with dot notation in the fields list (e.g. *"address.city"* retrieves just the city value)
- Use *limit()*, *skip()* and *sort()* on **DBCursor** to adjust your results. These all return a new **DBCursor**.
- *distinct()* can be used (on **DBCollection** to find all distinct values for a given key; it returns a list:

```
List values = coll.distinct("postedBy"); // contains all distinct
    values in "postedBy"
/** Or, limit what you're looking for with a query */
DBObject q  = new BasicDBObject("postedBy",
                  new BasicDBObject("$ne", "bwmcadams")
              );
List values = coll.distinct("postedBy", q);
```

MongoDB Querying
Query Operators

MongoDB is no mere Key-Value store. There are myriad powerful operators to enhance your MongoDB queries...

- Conditional Operators: **$gt** (>), **$lt** (<), **$gte (>=)**, **$lte (<=)**
- Negative Equality: **$ne** (!=)
- Array Operators: **$in** (SQL "IN" clause...takes an array), **$nin** (Opposite of "IN"), **$all** (Requires all values in the array match), **$size** (Match the size of an array)
- Field Defined: **$exists** (boolean argument)(Great in a schemaless world)
- Regular Expressions (Language dependent - most drivers support it)
- Pass Arbitrary Javascript with **$where** (No OR statements, so use WHERE for complex range filters)
- Negate any operator with **$not**

NOVUS

MongoDB Querying
Putting Operators to Work

Using a query operator requires nested objects...

- All posts since a particular date:

```
DBObject q = new BasicDBObject("postDate",
                 new BasicDBObject("$gte", new java.util.Date())
             );
DBCursor posts = coll.find(q);
```

- Find all posts NOT by me:

```
DBObject q = new BasicDBObject("postedBy",
                 new BasicDBObject("$ne", "bwmcadams")
             );
DBCursor posts = coll.find(q);
```

- No syntactic sugar in Java to make it easier...
- *Mungbean* provides a more fluid query syntax (but isn't very Mongo-ey), if you use its PoJo mapping.
- You might also consider evaluating other JVM languages for querying....

NOVUS

MongoDB Querying
Putting Operators to Work

Using a query operator requires nested objects...

- All posts since a particular date:

```
DBObject q = new BasicDBObject("postDate",
                 new BasicDBObject("$gte", new java.util.Date())
             );
DBCursor posts = coll.find(q);
```

- Find all posts NOT by me:

```
DBObject q = new BasicDBObject("postedBy",
                 new BasicDBObject("$ne", "bwmcadams")
             );
DBCursor posts = coll.find(q);
```

- No syntactic sugar in Java to make it easier...
- *Mungbean* provides a more fluid query syntax (but isn't very Mongo-ey), if you use its PoJo mapping.
- You might also consider evaluating other JVM languages for querying....

NOVUS

MongoDB Querying
Putting Operators to Work

Using a query operator requires nested objects...

- All posts since a particular date:

```
DBObject q = new BasicDBObject("postDate",
                 new BasicDBObject("$gte", new java.util.Date())
             );
DBCursor posts = coll.find(q);
```

- Find all posts NOT by me:

```
DBObject q = new BasicDBObject("postedBy",
                 new BasicDBObject("$ne", "bwmcadams")
             );
DBCursor posts = coll.find(q);
```

- No syntactic sugar in Java to make it easier...
- *Mungbean* provides a more fluid query syntax (but isn't very Mongo-ey), if you use its PoJo mapping.
- You might also consider evaluating other JVM languages for querying....

NOVUS

MongoDB Querying
Putting Operators to Work

Using a query operator requires nested objects...

- All posts since a particular date:

```
DBObject q = new BasicDBObject("postDate",
                 new BasicDBObject("$gte", new java.util.Date())
             );
DBCursor posts = coll.find(q);
```

- Find all posts NOT by me:

```
DBObject q = new BasicDBObject("postedBy",
                 new BasicDBObject("$ne", "bwmcadams")
             );
DBCursor posts = coll.find(q);
```

- No syntactic sugar in Java to make it easier...
- *Mungbean* provides a more fluid query syntax (but isn't very Mongo-ey), if you use its PoJo mapping.
- You might also consider evaluating other JVM languages for querying....

NOVUS

MongoDB Querying
Putting Operators to Work

Using a query operator requires nested objects...

- All posts since a particular date:

```
DBObject q = new BasicDBObject("postDate",
                 new BasicDBObject("$gte", new java.util.Date())
             );
DBCursor posts = coll.find(q);
```

- Find all posts NOT by me:

```
DBObject q = new BasicDBObject("postedBy",
                 new BasicDBObject("$ne", "bwmcadams")
             );
DBCursor posts = coll.find(q);
```

- No syntactic sugar in Java to make it easier...
- *Mungbean* provides a more fluid query syntax (but isn't very Mongo-ey), if you use its PoJo mapping.
- You might also consider evaluating other JVM languages for querying....

NOVUS

MongoDB Querying
Putting Operators to Work

Using a query operator requires nested objects...

- All posts since a particular date:

```
DBObject q = new BasicDBObject("postDate",
                 new BasicDBObject("$gte", new java.util.Date())
             );
DBCursor posts = coll.find(q);
```

- Find all posts NOT by me:

```
DBObject q = new BasicDBObject("postedBy",
                 new BasicDBObject("$ne", "bwmcadams")
             );
DBCursor posts = coll.find(q);
```

- No syntactic sugar in Java to make it easier...
- *Mungbean* provides a more fluid query syntax (but isn't very Mongo-ey), if you use its PoJo mapping.
- You might also consider evaluating other JVM languages for querying....

NOVUS

Outline

NOVUS

MongoDB Querying
AltJVM Languages - Syntactic Sugar

Listing 5: Groovy Sample

```
def q = [postDate: ["$gte:" new java.util.Date()]
        ] as BasicDBObject
def q = [postedBy: ["$ne": "bwmcadams"]
        ] as BasicDBObject

def posts = coll.find(q)
```

NOVUS

MongoDB Querying
AltJVM Languages - Syntactic Sugar

Listing 6: Scala Sample

```
val q = "postDate" $gte new java.util.Date()
val q = "postedBy" $ne "bwmcadams"

val posts = coll.find(q)
```

NOVUS

MongoDB Querying
AltJVM Languages - Syntactic Sugar

Listing 7: Jython Sample

```
q = {"postDate": {"$gte": datetime.datetime.now()}}
q = {"postedBy": {"$ne": "bwmcadams"}}

posts = coll.find(q)
```

NOVUS

Outline

NOVUS

Indexes

MongoDB Indexes work much like RDBMS indexes...

- Create a single-key index:

```
DBObject idx = new BasicDBObject("postedBy", 1);
coll.createIndex(idx);
```

  This defaults to ascending order (-1 = descending), with a system generated name.

- For a multi-key/compound-key index:

```
DBObject idx = new BasicDBObjectBuilder.start().
                   add("postedBy", 1).
                   add("postDate", -1).
                   get();
coll.createIndex(idx);
```

  This sorts postedBy ascending, but postDate descending.

- If you want to index an embedded field, simply use the query dot notation (e.g. "address.city")

NOVUS

Indexes
Options

MongoDB Indexes work much like RDBMS indexes...

- You can pass a second DBObject of options to change the type of index, name, etc:

```
DBObject idx = new BasicDBObject("slug", 1);
DBObject opts = new BasicDBObject("unique": true);
coll.createIndex(idx, opts);
```

  This will ensure that the "slug" field is unique across all entries. (A list of complete options is available in the MongoDB Documentation)

- Consider using *ensureIndex()* instead of *createIndex()*.

NOVUS

Things We Didn't Cover

Things we didn't cover, but you should spend some time exploring...

- Map/Reduce (great for more complex aggregation)
- Geospatial indexes and queries
- Aggregation queries such as Grouping statements (Which use JavaScript functions)
- GridFS (Efficient storage of large files)
- Lots more at MongoDB.org ...

NOVUS

Contact Info + Where To Learn More

- Contact Me
  - twitter: **@rit**
  - email: **bwmcadams@gmail.com**
- Pressing Questions?
  - IRC - freenode.net **#mongodb**
  - MongoDB Users List - `http://groups.google.com/group/mongodb-user`
- Mongo Java Language Center - `http://mongodb.org/display/DOCS/Java+Language+Center` (Links to Java driver docs, and many of the third party libraries)
- Morphia - `http://code.google.com/p/morphia/`
- Mungbean - `http://github.com/jannehietamaki/mungbean`
- Experimental JDBC Driver - `http://github.com/erh/mongo-jdbc`

NOVUS