

# Java Development with MongoDB

Brendan W. McAdams

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

May 2010 / Mongo NYC



# Outline

## 1 MongoDB + Java Basics

- Setting up your Java environment
- Connecting to MongoDB

## 2 Working with MongoDB

- Collections + Documents
- Inserting Documents to MongoDB
- Querying MongoDB

# Outline

- 1 MongoDB + Java Basics
  - Setting up your Java environment
  - Connecting to MongoDB
- 2 Working with MongoDB
  - Collections + Documents
  - Inserting Documents to MongoDB
  - Querying MongoDB

# Adding the MongoDB Driver To Your Project

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

- Maven Dependency

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

- Ivy Dependency

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

# Adding the MongoDB Driver To Your Project

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

- **Maven Dependency**

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

- **Ivy Dependency**

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

# Outline

- 1 MongoDB + Java Basics
  - Setting up your Java environment
  - Connecting to MongoDB
- 2 Working with MongoDB
  - Collections + Documents
  - Inserting Documents to MongoDB
  - Querying MongoDB

# 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");
```

# 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");
```





# 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");
```



# 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");
```



# Outline

## 1 MongoDB + Java Basics

- Setting up your Java environment
- Connecting to MongoDB

## 2 Working with MongoDB

- Collections + Documents
- Inserting Documents to MongoDB
- Querying MongoDB

# 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**

# 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.



# 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.





# 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.



# 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.



# 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.



# 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.



# 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.



# 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();
```



# 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();
```



# 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();
```





# 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();
```



# 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 (*Detailed later*)...
  - Daybreak
  - Morphia



# 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 (*Detailed later*). . .
  - Daybreak
  - Morphia



# 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 (*Detailed later*). . .
  - Daybreak
  - Morphia



## Briefly: Working with DBObjects

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



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



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

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



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

# Outline

## 1 MongoDB + Java Basics

- Setting up your Java environment
- Connecting to MongoDB

## 2 Working with MongoDB

- Collections + Documents
- **Inserting Documents to MongoDB**
- Querying MongoDB

# 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

## 1 MongoDB + Java Basics

- Setting up your Java environment
- Connecting to MongoDB

## 2 Working with MongoDB

- Collections + Documents
- Inserting Documents to MongoDB
- **Querying MongoDB**



# 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);
```

- You can pass an optional second **DBObject** parameter to *find()* and *findOne()* which specifies the fields to return.





# 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);
```

- You can pass an optional second **DBObject** parameter to *find()* and *findOne()* which specifies the fields to return.



# 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);
```

- You can pass an optional second **DBObject** parameter to *find()* and *findOne()* which specifies the fields to return.



# 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);
```

- You can pass an optional second **DBObject** parameter to *find()* and *findOne()* which specifies the fields to return.



# 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**



# 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);
```



# 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);
```



# 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);
```

