Scala with MongoDB

Brendan W. McAdams

Novus Partners, Inc.

New York Scala Enthusiasts
Aug. 8, 2010

NOVUS

Outline

1. Introduction
   - What is MongoDB?
   - A Taste of MongoDB
   - MongoDB + Scala Drivers
2. Scala + MongoDB == Win
   - mongo-scala-driver
   - lift-mongo
   - casbah
   - STM + MongoDB via Akka
3. Interlude: Helping Scala + Java play nice together.
   - Java <-> Scala Basics
   - Implicits and Pimp Hats
4. Closing

NOVUS

Outline

1. Introduction
   - What is MongoDB?
   - A Taste of MongoDB
   - MongoDB + Scala Drivers
2. Scala + MongoDB == Win
   - mongo-scala-driver
   - lift-mongo
   - casbah
   - STM + MongoDB via Akka
3. Interlude: Helping Scala + Java play nice together.
   - Java <-> Scala Basics
   - Implicits and Pimp Hats
4. Closing

NOVUS

Introducing MongoDB

- Categorized as a “Document-Oriented Database”
  - Features of both Key-Value Stores & RDBMS’
  - Rich query interface.
  - Works with JSON-like Documents
  - Favors embedding related data over “foreign key” relationships
- Free license (A-GPL) cross-platform (Packages for Linux, Windows, Mac OS X, Windows, FreeBSD & Solaris)
- Cursor-based query results
- ServerSide Javascript
  - Stored Javascript functions server-side
  - Powerful aggregation - Map/Reduce, Group Commands
  - JS Statements in queries (no indexes though)
- Indexing system is much like RDBMS, includes Geospatial support.
- Scalable file storage with GridFS
- Data scalability with Replica Sets & Autosharding

NOVUS

Introducing MongoDB

- Categorized as a “Document-Oriented Database”
  - Features of both Key-Value Stores & RDBMS’
  - Rich query interface.
  - Works with JSON-like Documents
  - Favors embedding related data over “foreign key” relationships
- Free license (A-GPL) cross-platform (Packages for Linux, Windows, Mac OS X, Windows, FreeBSD & Solaris)
- Cursor-based query results
- ServerSide Javascript
  - Stored Javascript functions server-side
  - Powerful aggregation - Map/Reduce, Group Commands
  - JS Statements in queries (no indexes though)
- Indexing system is much like RDBMS, includes Geospatial support.
- Scalable file storage with GridFS
- Data scalability with Replica Sets & Autosharding

NOVUS

Introducing MongoDB

- Categorized as a “Document-Oriented Database”
  - Features of both Key-Value Stores & RDBMS’
  - Rich query interface.
  - Works with JSON-like Documents
  - Favors embedding related data over “foreign key” relationships
- Free license (A-GPL) cross-platform (Packages for Linux, Windows, Mac OS X, Windows, FreeBSD & Solaris)
- Cursor-based query results
- ServerSide Javascript
  - Stored Javascript functions server-side
  - Powerful aggregation - Map/Reduce, Group Commands
  - JS Statements in queries (no indexes though)
- Indexing system is much like RDBMS, includes Geospatial support.
- Scalable file storage with GridFS
- Data scalability with Replica Sets & Autosharding

NOVUS

Introducing MongoDB

- Categorized as a “Document-Oriented Database”
  - Features of both Key-Value Stores & RDBMS’
  - Rich query interface.
  - Works with JSON-like Documents
  - Favors embedding related data over “foreign key” relationships
- Free license (A-GPL) cross-platform (Packages for Linux, Windows, Mac OS X, Windows, FreeBSD & Solaris)
- Cursor-based query results
- ServerSide Javascript
  - Stored Javascript functions server-side
  - Powerful aggregation - Map/Reduce, Group Commands
  - JS Statements in queries (no indexes though)
- Indexing system is much like RDBMS, includes Geospatial support.
- Scalable file storage with GridFS
- Data scalability with Replica Sets & Autosharding

NOVUS

Introducing MongoDB

- Categorized as a “Document-Oriented Database”
  - Features of both Key-Value Stores & RDBMS’
  - Rich query interface.
  - Works with JSON-like Documents
  - Favors embedding related data over “foreign key” relationships
- Free license (A-GPL) cross-platform (Packages for Linux, Windows, Mac OS X, Windows, FreeBSD & Solaris)
- Cursor-based query results
- ServerSide Javascript
  - Stored Javascript functions server-side
  - Powerful aggregation - Map/Reduce, Group Commands
  - JS Statements in queries (no indexes though)
- Indexing system is much like RDBMS, includes Geospatial support.
- Scalable file storage with GridFS
- Data scalability with Replica Sets & Autosharding

NOVUS

Introducing MongoDB

- Categorized as a “Document-Oriented Database”
  - Features of both Key-Value Stores & RDBMS’
  - Rich query interface.
  - Works with JSON-like Documents
  - Favors embedding related data over “foreign key” relationships
- Free license (A-GPL) cross-platform (Packages for Linux, Windows, Mac OS X, Windows, FreeBSD & Solaris)
- Cursor-based query results
- ServerSide Javascript
  - Stored Javascript functions server-side
  - Powerful aggregation - Map/Reduce, Group Commands
  - JS Statements in queries (no indexes though)
- Indexing system is much like RDBMS, includes Geospatial support.
- Scalable file storage with GridFS
- Data scalability with Replica Sets & Autosharding

NOVUS

Introducing MongoDB

- Categorized as a “Document-Oriented Database”
  - Features of both Key-Value Stores & RDBMS’
  - Rich query interface.
  - Works with JSON-like Documents
  - Favors embedding related data over “foreign key” relationships
- Free license (A-GPL) cross-platform (Packages for Linux, Windows, Mac OS X, Windows, FreeBSD & Solaris)
- Cursor-based query results
- ServerSide Javascript
  - Stored Javascript functions server-side
  - Powerful aggregation - Map/Reduce, Group Commands
  - JS Statements in queries (no indexes though)
- Indexing system is much like RDBMS, includes Geospatial support.
- Scalable file storage with GridFS
- Data scalability with Replica Sets & Autosharding

NOVUS

Introducing MongoDB

- Categorized as a “Document-Oriented Database”
  - Features of both Key-Value Stores & RDBMS’
  - Rich query interface.
  - Works with JSON-like Documents
  - Favors embedding related data over “foreign key” relationships
- Free license (A-GPL) cross-platform (Packages for Linux, Windows, Mac OS X, Windows, FreeBSD & Solaris)
- Cursor-based query results
- ServerSide Javascript
  - Stored Javascript functions server-side
  - Powerful aggregation - Map/Reduce, Group Commands
  - JS Statements in queries (no indexes though)
- Indexing system is much like RDBMS, includes Geospatial support.
- Scalable file storage with GridFS
- Data scalability with Replica Sets & Autosharding

NOVUS

Programming with MongoDB

- Provides a native API which allows interaction to adapt to the programming language (rather than vice versa).
- Official drivers for. . .
  - C
  - C++
  - Java
  - JavaScript
  - Perl
  - PHP
  - Python
  - Ruby
- Community supported drivers include. . .
  - .Net: C# & F#
  - JVM: Clojure, Scala, Groovy
  - Erlang
  - Haskell
  - Go
  - . . . and many more.

NOVUS

Programming with MongoDB

- Provides a native API which allows interaction to adapt to the programming language (rather than vice versa).
- Official drivers for. . .
  - C
  - C++
  - Java
  - JavaScript
  - Perl
  - PHP
  - Python
  - Ruby
- Community supported drivers include. . .
  - .Net: C# & F#
  - JVM: Clojure, Scala, Groovy
  - Erlang
  - Haskell
  - Go
  - . . . and many more.

NOVUS

Programming with MongoDB

- Provides a native API which allows interaction to adapt to the programming language (rather than vice versa).
- Official drivers for...
  - C
  - C++
  - Java
  - JavaScript
  - Perl
  - PHP
  - Python
  - Ruby
- Community supported drivers include...
  - .Net: C# & F#
  - JVM: Clojure, Scala, Groovy
  - Erlang
  - Haskell
  - Go
  - ...and many more.

NOVUS

But is anyone actually *using* it?!?

MongoDB is deployed in production at companies including. . .

- New York Times
- Foursquare
- bit.ly
- SourceForge
- Etsy
- Disqus
- Github
- . . . The Large Hadron Collider.

But is anyone actually *using* it?!?

MongoDB is deployed in production at companies including. . .

- New York Times
- Foursquare
- bit.ly
- SourceForge
- Etsy
- Disqus
- Github
- . . . The Large Hadron Collider.

NOVUS

Outline

1. Introduction
   - What is MongoDB?
   - A Taste of MongoDB
   - MongoDB + Scala Drivers
2. Scala + MongoDB == Win
   - mongo-scala-driver
   - lift-mongo
   - casbah
   - STM + MongoDB via Akka
3. Interlude: Helping Scala + Java play nice together.
   - Java <-> Scala Basics
   - Implicits and Pimp Hats
4. Closing

NOVUS

Core Concepts

- MongoDB's equivalent to "tables" are called "collections";"collections" contain "documents" (individual pieces of data)
- Databases & Collections are lazy - they are created when first inserted into.
- MongoDB's wire format/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.
  - **BSON** implementation being split into its own package in most drivers.
  - Creative Commons licensed **http://bsonspec.org**
- Java driver represents **BSON** with a map-like **DBObject** (Which most Scala drivers use); many dynamic languages (Perl, Ruby, Python, etc) use native dictionary objects.

The basics of Querying I

- Find a single row with *findOne()*; returns the first document found (by natural order).
- You can find all documents matching your query with *find()*. No query means you get the entire collection back.
- Queries are specified as **BSON** documents to match against.
- The *find()* and *findOne()* methods can take an optional second **DBObject** specifying 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 result objects (**DBCursor** in Java-driver land) to adjust your results. These all return a new cursor.

NOVUS

The basics of Querying II

- *distinct()* can be used (on **DBCollection** to find all distinct values for a given key; it returns a list of values.

```
> db.routes.findOne({"route_short_name": "E"})
 {
        "_id" : ObjectId("4c5f755608c3693f59580f8c"),
        "route_id" : "E",
        "agency_id" : "MTA NYCT",
        "route_short_name" : "E",
        "route_long_name" : "8 Avenue Local",
        "route_desc" : "Trains operate between Jamaica Center (Parsons/Archer),
     Queens, and World Trade Center, Manhattan, at all times.",
        "route_type" : 1,
        "route_url" : "http://www.mta.info/nyct/service/pdf/tecur.pdf"
 }

> db.routes.find({"route_long_name": /Local$/},
                 {"route_short_name": 1, "route_long_name": 1})

 { "_id" : ObjectId("4c5f755608c3693f59580f7f"), "route_short_name" : 1,
     "route_long_name" : "Broadway - 7 Avenue Local" }
 { "_id" : ObjectId("4c5f755608c3693f59580f84"), "route_short_name" : 6,
     "route_long_name" : "Lexington Avenue Local" }
 { "_id" : ObjectId("4c5f755608c3693f59580f86"), "route_short_name" : 7,
     "route_long_name" : "Flushing Local" }
 { "_id" : ObjectId("4c5f755608c3693f59580f8a"), "route_short_name" : "C",
     "route_long_name" : "8 Avenue Local" }
```

The basics of Querying III

```
{ "_id" : ObjectId("4c5f755608c3693f59580f8c"), "route_short_name" : "E",
    "route_long_name" : "8 Avenue Local" }
{ "_id" : ObjectId("4c5f755608c3693f59580f8d"), "route_short_name" : "F",
    "route_long_name" : "Queens Blvd Express/ 6 Av Local" }
{ "_id" : ObjectId("4c5f755608c3693f59580f91"), "route_short_name" : "J",
    "route_long_name" : "Nassau St Local" }
{ "_id" : ObjectId("4c5f755608c3693f59580f92"), "route_short_name" : "L",
    "route_long_name" : "14 St-Canarsie Local" }
{ "_id" : ObjectId("4c5f755608c3693f59580f93"), "route_short_name" : "M",
    "route_long_name" : "Nassau St Local" }
{ "_id" : ObjectId("4c5f755608c3693f59580f96"), "route_short_name" : "R",
    "route_long_name" : "Broadway Local" }
{ "_id" : ObjectId("4c5f755608c3693f59580f99"), "route_short_name" : "V",
    "route_long_name" : "Queens Blvd/6 Av Local" }
{ "_id" : ObjectId("4c5f755608c3693f59580f9a"), "route_short_name" : "W",
    "route_long_name" : "Broadway Local" }

> db.routes.distinct("route_short_name")
[
        1,
        2,
        3,
        4,
        5,
        6,
        7,
        "A",
        "B",
        "C",
```

NOVUS

The basics of Querying IV

```
            "D",
            "E",
            "F",
            "G",
            "J",
        /*... */
    ]
```

Query Operators I

- 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**
- Using a query operator requires nested objects...

NOVUS

Query Operators II

```
> db.stops.find({"stop_lat" : {$lt: 40.6}, {"stop_lon": {$gte: -73.8}}})
 { "_id" : ObjectId("4c5f755608c3693f59580ef0"), "stop_lat" : 40.590927, "stop_lon"
      : -73.796924, "stop_id" : "H06", "stop_name" : "BEACH 67TH ST - GASTON",
      "location_type" : 0, "stop_geo" : { "lat" : 40.590927, "lon" : -73.796924 } }
 { "_id" : ObjectId("4c5f755608c3693f59580ef1"), "stop_lat" : 40.592374, "stop_lon"
      : -73.788522, "stop_id" : "H07", "stop_name" : "BEACH 60TH ST - STRAITON AV",
      "location_type" : 0, "stop_geo" : { "lat" : 40.592374, "lon" : -73.788522 } }
 { "_id" : ObjectId("4c5f755608c3693f59580ef2"), "stop_lat" : 40.592943, "stop_lon"
      : -73.776013, "stop_id" : "H08", "stop_name" : "BEACH 44TH ST - FRANK AV",
      "location_type" : 0, "stop_geo" : { "lat" : 40.592943, "lon" : -73.776013 } }
 { "_id" : ObjectId("4c5f755608c3693f59580ef3"), "stop_lat" : 40.595398, "stop_lon"
      : -73.768175, "stop_id" : "H09", "stop_name" : "BEACH 36TH ST - EDGEMERE",
      "location_type" : 0, "stop_geo" : { "lat" : 40.595398, "lon" : -73.768175 } }


> db.trips.findOne({"route_id": {$in: ["E", "4", "5"]}})
 {
         "_id" : ObjectId("4c5f755708c3693f59583400"),
         "route_id" : "E",
         "service_id" : "B20100308W",
         "trip_id" : "B20100308W_001350_E..S04R",
         "trip_headsign" : "To World Trade Ctr",
         "direction_id" : 1,
         "shape_id" : 177710
 }

> db.trips.find({"route_id": {$in: ["E", "4", "5"]}}).count()
928
```

NOVUS

Query Operators III

- No syntactic sugar in Java to make it easier. . .

Insert/Update/Save I

- Objects in MongoDB Collections have an "_id" field, which must be unique.
- Three ways to add/update data in MongoDB...
  - `insert()` always attempts to add a new row. If "_id" is present and contains a value already in the collection, insert fails.
  - `save()` inserts if there is no "_id" field, otherwise it tries to update the document with the specified "_id".
  - `update()` takes a query and the new values to save. By default it updates only the first document matching the query.
  - For `update()` you can specify two booleans whose default is false: *upsert*, which indicates you wish to create a new document if the query doesn't match, and *multi*, which allows updating **all** documents who match the query.

Insert/Update/Save II

```
> db.testData.insert({"userCount": 5})
> x = db.testData.findOne({"userCount": 5})
{ "_id" : ObjectId("4c607f48150c335a4e187f41"), "userCount" : 5 }
> x.userCount
5
> x.userCount = 20
20
> db.testData.save(x)
> db.testData.findOne({_id: x._id})
{ "_id" : ObjectId("4c607f48150c335a4e187f41"), "userCount" : 20 }
> db.testData.update({_id: x._id}, {$inc: {"userCount": 12}})
> db.testData.findOne({_id: x._id})
{ "_id" : ObjectId("4c607f48150c335a4e187f41"), "userCount" : 32 }
// upsert
> db.testData.update({"userCount": 5}, {"userCount": 209}, true)
> db.testData.findOne({"userCount": 209} )
{ "_id" : ObjectId("4c60800e08c3693f5962dda5"), "userCount" : 209 }
```

NOVUS

Geospatial Support I

- MongoDB supports Geospatial indexing and distance based queries
- I loaded all of the NYC Subway data (in Google Transit format) into MongoDB
- Quick python code to index the “Stops” data.

```
connection = Connection()
db = connection['nyct_subway']
print "Indexing the Stops Data."
for row in db.stops.find():
    row['stop_geo'] = {'lat': row['stop_lat'], 'lon': row['stop_lon']}
    db.stops.save(row)

db.stops.ensure_index([('stop_geo', pymongo.GEO2D)])
```

- “stop_geo” field is now Geospatially indexed for each stop.
- How hard is it to find the 5 closest subway stops to Meetup HQ?

Geospatial Support I

```
db.stops.find( { stop_geo: { $near: [40.726021, -73.99617] } }, {'stop_name':
    1}).limit(5);

  { "_id" : ObjectId("4c5f755608c3693f59580e9b"), "stop_name" : "BROADWAY-LAFAYETTE
      " }
  { "_id" : ObjectId("4c5f755608c3693f59580e29"), "stop_name" : "BLEECKER
    STREET-LEXINGTON" }
  { "_id" : ObjectId("4c5f755608c3693f59580f50"), "stop_name" : "PRINCE STREET
      " }
  { "_id" : ObjectId("4c5f755608c3693f59580e2a"), "stop_name" : "SPRING
    STREET-LEXINGTON" }
  { "_id" : ObjectId("4c5f755608c3693f59580f4f"), "stop_name" : "8TH STREET (NYU)
    " }
```

- Further commands exist to define a rectangle or circle radius for the search.

Finally, Data Scalability.

- Traditional master-slave replication
- Replica Sets (new in 1.6)
  - Replaces master-slave setup with 1-7 server clusters
  - Automatic failover and recovery
- AutoSharding (new in 1.6)
  - Horizontal scaling - partition your collections & data across as many nodes as necessary.
  - Multiple nodes can service the same shard, allowing for balancing & failover.
  - Map/Reduce runs across multiple shards, allowing concurrency.

NOVUS

Outline

1. Introduction
   - What is MongoDB?
   - A Taste of MongoDB
   - MongoDB + Scala Drivers
2. Scala + MongoDB == Win
   - mongo-scala-driver
   - lift-mongo
   - casbah
   - STM + MongoDB via Akka
3. Interlude: Helping Scala + Java play nice together.
   - Java <-> Scala Basics
   - Implicits and Pimp Hats
4. Closing

NOVUS

Using Scala with the official Java Driver I

- JVM Object are JVM Objects...

```
import com.mongodb._

val conn = new Mongo()
val db = conn.getDB("test")
val coll = db.getCollection("testData")

val pies = new BasicDBList()
pies.add("cherry")
pies.add("blueberry")
pies.add("apple")
pies.add("rhubarb")
pies.add("3.14")

val doc = new BasicDBObject()
doc.put("foo", "bar")
doc.put("spam", "eggs")
doc.put("up", "down")
doc.put("pie", pies)

coll.insert(doc)
```

- ...Not terribly “Scala-ey”.

NOVUS

Using Scala with the official Java Driver II

- The Java driver works, but doesn't fit well in Scala.
- You need to convert your Scala objects to Java Objects, and get nothing but Java Objects out.
- Gets messy quickly.

NOVUS

The Scala Community Adapted... I

Compare the previous with various Scala drivers.

- mongo-scala-driver wraps & enhances the Java driver:

```
import com.mongodb._
import com.osinka.mongodb._

val conn = new Mongo()
val db = conn.getDB("test")
val coll = db.getCollection("testData").asScala

coll << Map(
  "foo" -> "bar",
  "spam" -> "eggs",
  "up" -> "down",
  "pie" -> List(
    "cherry",
    "blueberry",
    "apple",
    "rhubarb",
    "3.14"
  )
)
```

NOVUS

The Scala Community Adapted... II

- .. Much better, although I was confused initially. Has a object<->MongoDB mapping layer.
- lift-mongodb has more than one way to do it... here's just a taste:

```
import com.mongodb._

import net.liftweb.mongodb._
import net.liftweb.json._
import net.liftweb.json.JsonAST.JObject
import net.liftweb.json.JsonDSL._

implicit val formats = DefaultFormats.lossless

MongoDB.defineDb(DefaultMongoIdentifier,
                 MongoAddress(MongoHost("localhost", 27017)), "test")

val json = JsonParser.parse("""
{ "foo": "bar",
  "spam": "eggs",
  "up": "down",
  "pie": [
    "cherry",
    "blueberry",
    "apple",
    "rhubarb",
    "3.14"
```

The Scala Community Adapted... III

```
  ]
}
""").asInstanceOf[JObject]

MongoDB.useCollection("testData")( coll => {
  coll.save(JObjectParser.parse(json))
})
```

- ...Lift's JS & JSON tools make it very flexible, as we'll see later. Also has an ActiveRecord style Object<->MongoDB Mapping layer.
- Casbah reflects my own attempt at creating a sane interface between Scala & MongoDB. Influenced by pymongo:

The Scala Community Adapted... IV

```
import com.novus.casbah.mongodb.Imports._

val coll = MongoConnection()("test")("testData")

val builder = MongoDBObject.newBuilder
builder += "foo" -> "bar"
builder += "spam" -> "eggs"
builder += "up" -> "down"
builder += "pie" -> List("cherry", "blueberry",
                         "apple", "rhubarb", "3.14")

coll += builder.result
```

- ...The syntax is still growing but is meant to match Scala syntax sanely. Object<->MongoDB Mapping coming soon.
- We're going to cover several tools, although I know Casbah best.

NOVUS

Outline

1. Introduction
   - What is MongoDB?
   - A Taste of MongoDB
   - MongoDB + Scala Drivers
2. Scala + MongoDB == Win
   - mongo-scala-driver
   - lift-mongo
   - casbah
   - STM + MongoDB via Akka
3. Interlude: Helping Scala + Java play nice together.
   - Java <-> Scala Basics
   - Implicits and Pimp Hats
4. Closing

NOVUS

- At its core `mongo-scala-driver` provides a few improvements over the Java driver via wrappers.
- Converting the Java objects to Scala requires explicit conversions, converts `DBCollection` to `DBObjectCollection`.
- `DBObjectCollection` implements `Iterable`, and provides several operators . . .
  - « maps to “insert”
  - «? “insert”s, and checks for errors.
  - += maps to “save”
  - -= maps to “remove”
- `Map` objects can be used in place of `DBObject` (as previously shown) due to implicit conversions.
- Object mapping implemented via “Shapes”; provides Query syntax.

mongo-scala-driver: Shapes for Object Mapping I

- "Shapes" uses Scala code to define Object models.
- Flexible, but syntactically verbose.
- Object representing "stops" on the NY Subway...

```
// Base representation
class SubwayStop extends MongoObject {
  var id: Int = _
  var name: String = _
  var locationType: Option[Int] = None
  var latitude: Double = _
  var longitude: Double = _

  override def toString: String =
    "NYC Subway Stop at %s [Lat: %d, Lon: %d]".
      format(name, latitude, longitude)
}

// Factory object
object SubwayStop extends MongoObjectShape[SubwayStop] {
  lazy val id = Field.scalar("stop_id", _.id,
                        (x: SubwayStop, v: Int) => x.id = v)
  lazy val name = Field.scalar("stop_name", _.name,
                        (x: SubwayStop, v: String) => x.name = v)
  lazy val locationType = Field.optional(
```

NOVUS

mongo-scala-driver: Shapes for Object Mapping II

```
                              "location_type", _.locationType,
                              (x: SubwayStop, v: Option[Int]) =>
                               x.locationType = v)
  lazy val latitude = Field.scalar("stop_lat", _.id,
                           (x: SubwayStop, v: Double) => x.latitude = v)
  lazy val longitude = Field.scalar("stop_lon", _.id,
                           (x: SubwayStop, v: Double) => x.longitude = v)
  // per docs, you must define * with all fields or (de)serialization won't work.
  override lazy val * = id :: name :: locationType ::
                           latitude :: longitude :: Nil

  override def factory(dbo: DBObject) = Some(new SubwayStop)
}

// retrieving items
val conn = new Mongo()
val db = conn.getDB("nyct_subway")
val coll = db.getCollection("stops")

val stopsColl = coll of SubwayStop
/* stopsColl is a ShapedCollection[SubwayStop] */
SubwayStop where {
  SubwayStop.latitude < 40.6,
  SubwayStop.longitude >= -73.8} sortBy SubwayStop.name descending
```

NOVUS

Outline

1. Introduction
   - What is MongoDB?
   - A Taste of MongoDB
   - MongoDB + Scala Drivers
2. Scala + MongoDB == Win
   - mongo-scala-driver
   - lift-mongo
   - casbah
   - STM + MongoDB via Akka
3. Interlude: Helping Scala + Java play nice together.
   - Java <-> Scala Basics
   - Implicits and Pimp Hats
4. Closing

NOVUS

lift-mongo I

- Formerly “scamongo”, integrated with Lift as of 2.0
- Base code provides session wrappers to MongoDB, still utilizes Java driver’s DBObject code.

```
MongoDB.defineDb(DefaultMongoIdentifier,
                 MongoAddress(MongoHost("localhost", 27017)), "test")

MongoDB.useCollection(collectionName) ( coll => {
  val doc = new BasicDBObject
  doc.put("name", "MongoDB")
  doc.put("type", "database")
  doc.put("count", 1)
  // save the doc to the db
  coll.save(doc)
})

// Alternately, do everything in a single thread...
MongoDB.useSession ( db => {
  val coll = db.getCollection("testCollection")
  val doc = new BasicDBObject
  doc.put("name", "MongoSession")
  doc.put("type", "db")
  doc.put("count", 1)
  coll.save(doc)
})
```

NOVUS

lift-mongo II

- “lift-mongo-record” provides object mapping.
- No native query syntax, but Foursquare is working on open sourcing something they use internally.

lift-mongo-record & querying I

- Object definitions are fairly straightforward...

```
class MainDoc extends MongoRecord[MainDoc] with MongoId[MainDoc] {
  def meta = MainDoc
  object name extends StringField(this, 12)
  object cnt extends IntField(this)
  object refdoc extends DBRefField[MainDoc, RefDoc](this, RefDoc)
  object refdocId extends ObjectIdField(this) {
    def fetch = RefDoc.find(value)
  }
}
object MainDoc extends MainDoc with MongoMetaRecord[MainDoc] {
  def createRecord = new MainDoc
}
class RefDoc extends MongoRecord[RefDoc] with MongoId[RefDoc] {
  def meta = RefDoc
}
object RefDoc extends RefDoc with MongoMetaRecord[RefDoc] {
  def createRecord = new RefDoc
}

// Querying appears limited to constructing Mongo DBObjects
val mdq1 = MainDoc.findAll(("name" -> "md1"))
```

NOVUS

lift-mongo-record & querying II

- Foursquare's query library allow for a saner way to query data...

```
// FSMongoRecord extends "MongoRecord" to add a few methods
class Venue extends FSMongoRecord[Venue] {
  def meta = Venue

  object venuename extends FSStringField(this, 255)
  object keywords extends StringField(this, 255)
  object userid extends LongField(this)
  object closed extends BooleanField(this) with AuditableField[Venue]
  object mayor extends LegacyForeignKey(this, User) {
    override def optional_? = true
  }
  object mayor_count extends OptionalIntField(this)
  object aliases extends MongoListField[Venue, String](this)
  object popularity extends MongoListField[Venue, Int](this)
  object popularityUpdated extends OptionalJodaDateTimeField[Venue](this)

  object tags extends MongoListField[Venue, String](this)
  object categories extends MongoForeignObjectIdList(this, Category)
}

object Venue extends Venue with FSMetaRecord[Venue] {
  override def collectionName = "venues"
  def createRecord = new Venue
  override def mongoIdentifier = NamedMongoIdentifier.venue
}
```

NOVUS

lift-mongo-record & querying III

```
// Foursquare's query engine allows for fluid queries in code
Venue where (_.venuename is "Starbucks")
Venue where (_.venuename nin ("Starbucks", "Whole Foods"))
Venue where (_.geolatlng near (40.72, -73.99))
```

- *Thank you* to `@jliszka` for sharing this!

Outline

1. Introduction
   - What is MongoDB?
   - A Taste of MongoDB
   - MongoDB + Scala Drivers
2. Scala + MongoDB == Win
   - mongo-scala-driver
   - lift-mongo
   - casbah
   - STM + MongoDB via Akka
3. Interlude: Helping Scala + Java play nice together.
   - Java <-> Scala Basics
   - Implicits and Pimp Hats
4. Closing

NOVUS

Shameless Self Promotion

- Why Casbah?
- Background in pymongo + MongoKit
- Java driver too. . . “Java-ey”
- Didn’t quite “get” scamongo and mongo-scala-driver early on
- scamongo’s base didn’t fix most of my issues w/ the Java Driver (just helped connection management)
- scamongo’s ORM libraries were dependent on Lift (now scamongo is defunct and has become lift-mongo)
- mongo-scala-driver’s shapes, etc were *very* confusing to me as a newbie w/o much functional background

NOVUS

Casbah is Born

- Borrowed bits I liked/understood from other places and built something that felt comfortable to me
- Early on, very pythonic
- Query DSL, grown from wanting a feel close to the “metal” based on generic MongoDB knowledge
- Heavily influenced in structure by `@jorgeortiz85`’s libraries
- Quickly grew as I used more and more MongoDB with Scala; features have been grown organically from my own needs.

NOVUS

Interacting with DBObjects I

- `DBObject` is far too structurally Java.
- Sought to make them more usable & readable from Scala
- Most recently - match Scala 2.8 collection Factory/Builders
- Implicit conversions of `Product` (base for Tuple), `Map`. Explicit method `asDBObject` for corner cases.
- 'Pimped' version of `DBObject` via `MongoDBObject` - lets DBObject implement Scala's `Map` trait.

NOVUS

Interacting with DBObjects II

```
import com.novus.casbah.mongodb.Imports._  // Only import needed - mongoDB type
      aliases imported too

val coll = MongoConnection()("test")("testData")

// Map
val map: DBObject = Map(
  "foo" -> "bar",
  "spam" -> "eggs",
  "up" -> "down",
  "pie" -> List(
    "cherry",
    "blueberry",
    "apple",
    "rhubarb",
    "3.14"
  )
)

// 'Product'
val product: DBObject =
( "foo" -> "bar",
  "spam" -> "eggs",
  "up" -> "down",
  "pie" -> List(
    "cherry",
    "blueberry",
    "apple",
```

NOVUS

Interacting with DBObjects III

```
    "rhubarb",
    "3.14"
  )
).asDBObject // Explicit conversion method

// "Factory" method
val constructed: DBObject = MongoDBObject(
  "foo" -> "bar",
  "spam" -> "eggs",
  "up" -> "down",
  "pie" -> List(
    "cherry",
    "blueberry",
    "apple",
    "rhubarb",
    "3.14"
  )
)

// We showed the builder before
val builder = MongoDBObject.newBuilder
builder += "foo" -> "bar"
builder += "spam" -> "eggs"
builder += "up" -> "down"
builder += "pie" -> List("cherry", "blueberry",
                         "apple", "rhubarb", "3.14")

val built: DBObject = builder.result
```

Interacting with DBObjects IV

```
// Also responds to the 'Map' methods...
built += "x" -> "y"
built.getOrElse("x", throw new Error("Can't find value for X"))
/* res15: AnyRef = y */
```

- DBCollection behaves as a Scala `Iterable`, but interaction is mostly the same (with addition of methods like +=).

Fluid Query Syntax I

- My thought: Instead of keeping track of **Yet Another API**, MongoDB's Query Objects should "just work".
- Two kinds of Query Operators - 'Bareword' and 'Core'.
- Bareword Operators can be started as 'bare' statements:

```
val setMulti = $set ("foo" -> 5, "bar" -> "N", "spam" -> "eggs")
/* setMulti: DBObject = { "$set" : { "foo" : 5 , "bar" : "N" , "spam" : "eggs"}} */
val pushAll = $pushAll ("foo" -> (5, 10, 15, 20, 25, 38, 12, "bar", "spam", 86,
     "eggs", "omg", 412, "ponies"))
/* pushAll: DBObject = { "$pushAll" : { "foo" : [ 5 , 10 , 15 , 20 , 25 , 38 , 12 ,
     "bar" , "spam" , 86 , "eggs" , "omg" , 412 , "ponies"]}} */
```

NOVUS

Fluid Query Syntax II

- Core Operators need to be anchored to the right of a `DBObject` or a String (typically representing a field name):

```
// Find any documents where "foo" is between 5 and 15
val findFoo: DBObject = "foo" $gte 5 $lte 15
/* findFoo: DBObject = { "foo" : { "$gte" : 5 , "$lte" : 15}} */
// Find any documents where "bar" contains 1, 8 or 12
val findIn: DBObject = "foo" $in (1, 8, 12)
/* findIn: DBObject = { "foo" : { "$in" : [ 1 , 8 , 12]}} */
```

- Just a small taste - all MongoDB Query Objects are supported (For 1.4.x syntax - 1.6.x ($or, etc. soon))

NOVUS

Other Features I

- Custom converter implementations which allow most Scala types to be serialized cleanly to MongoDB. (Joda time serialization/deserialization support).
- Improved GridFS Functionality (loan pattern, support for `scala.io.Source`)
- Wrapper objects for Map/Reduce system (Help parse results to warn of errors, etc)

Coming Soon I

- Max Afonov @max4f working on annotation driven object mapping.
- Investigating ActiveRecord implementation, with fluid query syntax support.
- Support for MongoDB 1.6.x features.

NOVUS

Outline

1. Introduction
   - What is MongoDB?
   - A Taste of MongoDB
   - MongoDB + Scala Drivers
2. Scala + MongoDB == Win
   - mongo-scala-driver
   - lift-mongo
   - casbah
   - STM + MongoDB via Akka
3. Interlude: Helping Scala + Java play nice together.
   - Java <-> Scala Basics
   - Implicits and Pimp Hats
4. Closing

NOVUS

STM + MongoDB via Akka I

- Akka has an implementation of STM inspired by Clojure's; allows datastructures such as Maps and Vectors to become transactional.
- Akka STM supports persistence to several backends including MongoDB.
- Allows you to setup relatively simple, code managed concurrent transactions with state stored safely in MongoDB.
- Supports JTA; not yet distributed (Dependent on Multiverse, which is working on distributed STM)

NOVUS

Outline

1. Introduction
   - What is MongoDB?
   - A Taste of MongoDB
   - MongoDB + Scala Drivers
2. Scala + MongoDB == Win
   - mongo-scala-driver
   - lift-mongo
   - casbah
   - STM + MongoDB via Akka
3. Interlude: Helping Scala + Java play nice together.
   - Java <-> Scala Basics
   - Implicits and Pimp Hats
4. Closing

NOVUS

Helping Java + Scala Interact

- Implicits, “Pimp My Library” and various conversion helper tools simplify the work of interacting with Java.
- Scala and Java have their own completely different collection libraries.
- Some builtins ship with Scala to make this easier.

NOVUS

Helping Java + Scala Interact

- Implicits, “Pimp My Library” and various conversion helper tools simplify the work of interacting with Java.
- Scala and Java have their own completely different collection libraries.
- Some builtins ship with Scala to make this easier.

NOVUS

Helping Java + Scala Interact

- Implicits, “Pimp My Library” and various conversion helper tools simplify the work of interacting with Java.
- Scala and Java have their own completely different collection libraries.
- Some builtins ship with Scala to make this easier.

NOVUS

Interoperability in Scala 2.7.x

- Scala 2.7.x shipped with `scala.collection.jcl`.
- `scala.collection.jcl.Conversions` contained some implicit converters, but only to and from the wrapper versions - no support for “real” Scala collections.
- Neglected useful base interfaces like `Iterator` and `Iterable`
- @jorgeortiz85 provided `scala-javautils`, which used “Pimp My Library” to do a better job.

NOVUS

Interoperability in Scala 2.7.x

- Scala 2.7.x shipped with `scala.collection.jcl`.
- `scala.collection.jcl.Conversions` contained some implicit converters, but only to and from the wrapper versions - no support for “real” Scala collections.
- Neglected useful base interfaces like `Iterator` and `Iterable`
- @jorgeortiz85 provided `scala-javautils`, which used “Pimp My Library” to do a better job.

NOVUS

Interoperability in Scala 2.7.x

- Scala 2.7.x shipped with `scala.collection.jcl`.
- `scala.collection.jcl.Conversions` contained some implicit converters, but only to and from the wrapper versions - no support for “real” Scala collections.
- Neglected useful base interfaces like `Iterator` and `Iterable`
- @jorgeortiz85 provided `scala-javautils`, which used “Pimp My Library” to do a better job.

NOVUS

Interoperability in Scala 2.8.x

- Scala 2.8.x improves the interop game significantly.
- JCL is gone - focus has shifted to proper interoperability w/ built-in types.
- `scala.collection.jcl.Conversions` replaced by `scala.collection.JavaConversions` - provides implicit conversions to & from Scala & Java Collections.
- Includes support for the things missing in 2.7 (`Iterable`, `Iterator`, etc.)
- Great for places where the compiler can guess what you want (implicits); falls short in some cases (like BSON Encoding, as we found in Casbah)
- @jorgeortiz85 has updated `scala-javautils` for 2.8 with `scalaj-collection`
- Explicit `asJava` / `asScala` methods for conversions. Adds `foreach` method to Java collections.

NOVUS

Interoperability in Scala 2.8.x

- Scala 2.8.x improves the interop game significantly.
- JCL is gone - focus has shifted to proper interoperability w/ built-in types.
- `scala.collection.jcl.Conversions` replaced by `scala.collection.JavaConversions` - provides implicit conversions to & from Scala & Java Collections.
- Includes support for the things missing in 2.7 (`Iterable`, `Iterator`, etc.)
- Great for places where the compiler can guess what you want (implicits); falls short in some cases (like BSON Encoding, as we found in Casbah)
- @jorgeortiz85 has updated `scala-javautils` for 2.8 with `scalaj-collection`
- Explicit `asJava` / `asScala` methods for conversions. Adds `foreach` method to Java collections.

Interoperability in Scala 2.8.x

- Scala 2.8.x improves the interop game significantly.
- JCL is gone - focus has shifted to proper interoperability w/ built-in types.
- `scala.collection.jcl.Conversions` replaced by `scala.collection.JavaConversions` - provides implicit conversions to & from Scala & Java Collections.
- Includes support for the things missing in 2.7 (`Iterable`, `Iterator`, etc.)
- Great for places where the compiler can guess what you want (implicits); falls short in some cases (like BSON Encoding, as we found in Casbah)
- @jorgeortiz85 has updated `scala-javautils` for 2.8 with `scalaj-collection`
- Explicit `asJava` / `asScala` methods for conversions. Adds `foreach` method to Java collections.

Interoperability in Scala 2.8.x

- Scala 2.8.x improves the interop game significantly.
- JCL is gone - focus has shifted to proper interoperability w/ built-in types.
- `scala.collection.jcl.Conversions` replaced by `scala.collection.JavaConversions` - provides implicit conversions to & from Scala & Java Collections.
- Includes support for the things missing in 2.7 (`Iterable`, `Iterator`, etc.)
- Great for places where the compiler can guess what you want (implicits); falls short in some cases (like BSON Encoding, as we found in Casbah)
- @jorgeortiz85 has updated `scala-javautils` for 2.8 with `scalaj-collection`
- Explicit `asJava` / `asScala` methods for conversions. Adds `foreach` method to Java collections.

Outline

1. Introduction
   - What is MongoDB?
   - A Taste of MongoDB
   - MongoDB + Scala Drivers
2. Scala + MongoDB == Win
   - mongo-scala-driver
   - lift-mongo
   - casbah
   - STM + MongoDB via Akka
3. Interlude: Helping Scala + Java play nice together.
   - Java <-> Scala Basics
   - Implicits and Pimp Hats
4. Closing

NOVUS

So WTF is an 'Implicit', anyway?

- Implicit Arguments
  - 'Explicit' arguments indicates a method argument you pass, well *explicitly*.
  - 'Implicit' indicates a method argument which is. . . *implied*. (But you can pass them explicitly too.)
  - Implicit arguments are passed in Scala as an additional argument list:

```
import com.mongodb._
import org.bson.types.ObjectId

def query(id: ObjectId)(implicit coll: DBCollection) = coll.findOne(id)

val conn = new Mongo()
val db = conn.getDB("test")
implicit val coll = db.getCollection("testData")

// coll is passed implicitly
query(new ObjectId())

// or we can override the argument
query(new ObjectId())(db.getCollection("testDataExplicit"))
```

  - How does this differ from default arguments?

So WTF is an 'Implicit', anyway?

- Implicit Arguments
  - 'Explicit' arguments indicates a method argument you pass, well *explicitly*.
  - 'Implicit' indicates a method argument which is. . . *implied*. (But you can pass them explicitly too.)
  - Implicit arguments are passed in Scala as an additional argument list:

```
import com.mongodb._
import org.bson.types.ObjectId

def query(id: ObjectId)(implicit coll: DBCollection) = coll.findOne(id)

val conn = new Mongo()
val db = conn.getDB("test")
implicit val coll = db.getCollection("testData")

// coll is passed implicitly
query(new ObjectId())

// or we can override the argument
query(new ObjectId())(db.getCollection("testDataExplicit"))
```

  - How does this differ from default arguments?

So WTF is an 'Implicit', anyway?

- Implicit Methods/Conversions
  - If you try passing a type to a Scala method argument which doesn't match. . .

```
def printNumber(x: Int) = println(x)

printNumber(5)
printNumber("212") // won't compile
```

  - A fast and loose example, but simple. Fails to compile.
  - But with implicit methods, we can provide a conversion path. . .

```
implicit def strToNum(x: String) = x.toInt
def printNumber(x: Int) = println(x)

printNumber(5)
printNumber("212")
```

  - In a dynamic language, this may be called "monkey patching". Unlike Perl, Python, etc. Scala resolves implicits at compile time.

So WTF is an 'Implicit', anyway?

- Implicit Methods/Conversions
  - If you try passing a type to a Scala method argument which doesn't match. . .

```
def printNumber(x: Int) = println(x)

printNumber(5)
printNumber("212") // won't compile
```

  - A fast and loose example, but simple. Fails to compile.
  - But with implicit methods, we can provide a conversion path. . .

```
implicit def strToNum(x: String) = x.toInt
def printNumber(x: Int) = println(x)

printNumber(5)
printNumber("212")
```

  - In a dynamic language, this may be called "monkey patching". Unlike Perl, Python, etc. Scala resolves implicits at compile time.

So WTF is an 'Implicit', anyway?

- Implicit Methods/Conversions
  - If you try passing a type to a Scala method argument which doesn't match...

```
def printNumber(x: Int) = println(x)

printNumber(5)
printNumber("212") // won't compile
```

  - A fast and loose example, but simple. Fails to compile.
  - But with implicit methods, we can provide a conversion path...

```
implicit def strToNum(x: String) = x.toInt
def printNumber(x: Int) = println(x)

printNumber(5)
printNumber("212")
```

  - In a dynamic language, this may be called "monkey patching". Unlike Perl, Python, etc. Scala resolves implicits at compile time.

So WTF is an 'Implicit', anyway?

- Implicit Methods/Conversions
  - If you try passing a type to a Scala method argument which doesn't match. . .

```
def printNumber(x: Int) = println(x)

printNumber(5)
printNumber("212") // won't compile
```

  - A fast and loose example, but simple. Fails to compile.
  - But with implicit methods, we can provide a conversion path. . .

```
implicit def strToNum(x: String) = x.toInt
def printNumber(x: Int) = println(x)

printNumber(5)
printNumber("212")
```

  - In a dynamic language, this may be called "monkey patching". Unlike Perl, Python, etc. Scala resolves implicits at compile time.

Pimp My Library

- Coined by Martin Odersky in a 2006 Blog post. Similar to C# extension methods, Ruby modules.
- Uses implicit conversions to tack on new methods at runtime.
- Either return a new "Rich_" or anonymous class. . .

```
import com.mongodb.gridfs.{GridFSInputFile => MongoGridFSInputFile}

class GridFSInputFile protected[mongodb](override val underlying:
      MongoGridFSInputFile) extends GridFSFile {
  def filename_=(name: String) = underlying.setFilename(name)
  def contentType_=(cT: String) = underlying.setContentType(cT)
}

object PimpMyMongo {
  implicit def mongoConnAsScala(conn: Mongo) = new {
    def asScala = new MongoConnection(conn)
  }

  implicit def enrichGridFSInput(in: MongoGridFSInputFile) =
    new GridFSInputFile(in)
}

import PimpMyMongo._
```

- A note: with regards to type boundaries, `[A <: SomeType]`

Pimp My Library

- Coined by Martin Odersky in a 2006 Blog post. Similar to C# extension methods, Ruby modules.
- Uses implicit conversions to tack on new methods at runtime.
- Either return a new "Rich_" or anonymous class. . .

```
import com.mongodb.gridfs.{GridFSInputFile => MongoGridFSInputFile}

class GridFSInputFile protected[mongodb](override val underlying:
      MongoGridFSInputFile) extends GridFSFile {
  def filename_=(name: String) = underlying.setFilename(name)
  def contentType_=(cT: String) = underlying.setContentType(cT)
}

object PimpMyMongo {
  implicit def mongoConnAsScala(conn: Mongo) = new {
    def asScala = new MongoConnection(conn)
  }

  implicit def enrichGridFSInput(in: MongoGridFSInputFile) =
    new GridFSInputFile(in)
}

import PimpMyMongo._
```

- A note: with regards to type boundaries, `[A <: SomeType]`

Pimp My Library

- Coined by Martin Odersky in a 2006 Blog post. Similar to C# extension methods, Ruby modules.
- Uses implicit conversions to tack on new methods at runtime.
- Either return a new “Rich_” or anonymous class. . .

```
import com.mongodb.gridfs.{GridFSInputFile => MongoGridFSInputFile}

class GridFSInputFile protected[mongodb](override val underlying:
      MongoGridFSInputFile) extends GridFSFile {
  def filename_=(name: String) = underlying.setFilename(name)
  def contentType_=(cT: String) = underlying.setContentType(cT)
}

object PimpMyMongo {
  implicit def mongoConnAsScala(conn: Mongo) = new {
    def asScala = new MongoConnection(conn)
  }

  implicit def enrichGridFSInput(in: MongoGridFSInputFile) =
    new GridFSInputFile(in)
}

import PimpMyMongo._
```

- A note: with regards to type boundaries, `[A <: SomeType]`

Links

- mongo-scala-driver http://github.com/alaz/mongo-scala-driver
- lift-mongo http://www.assembla.com/wiki/show/liftweb/MongoDB
- FourSquare's Lift Mongo DSL Code . . . coming soon? @jliszka
- Casbah http://novus.github.com/docs/casbah
- Jorge Ortiz' (@jorgeortiz85) Libraries
  - scala-javautils (Scala 2.7.x) http://github.com/jorgeortiz85/scala-javautils
  - scalaj-collection (Scala 2.8.x) http://github.com/scalaj/scalaj-collection
- This presentation

  http://github.com/bwmcadams/presentations/tree/master/scala_mongodb/scalany_aug10/

NOVUS

Contact Info

- Twitter: @rit
- Email: bwmcadams@gmail.com
- Github: http://github.com/bwmcadams | http://github.com/novus
- IRC - freenode.net #mongodb
- MongoDB Mailing List http://groups.google.com/group/mongodb-user
- Casbah Mailing List http://groups.google.com/group/mongodb-casbah-user
- Boston MongoDB Conference - Sept. 20 (Cambridge, Mass.)
  http://10gen.com/conferences/mongoboston2010
- MongoDB NY Users Group
  http://www.meetup.com/New-York-MongoDB-User-Group/

NOVUS