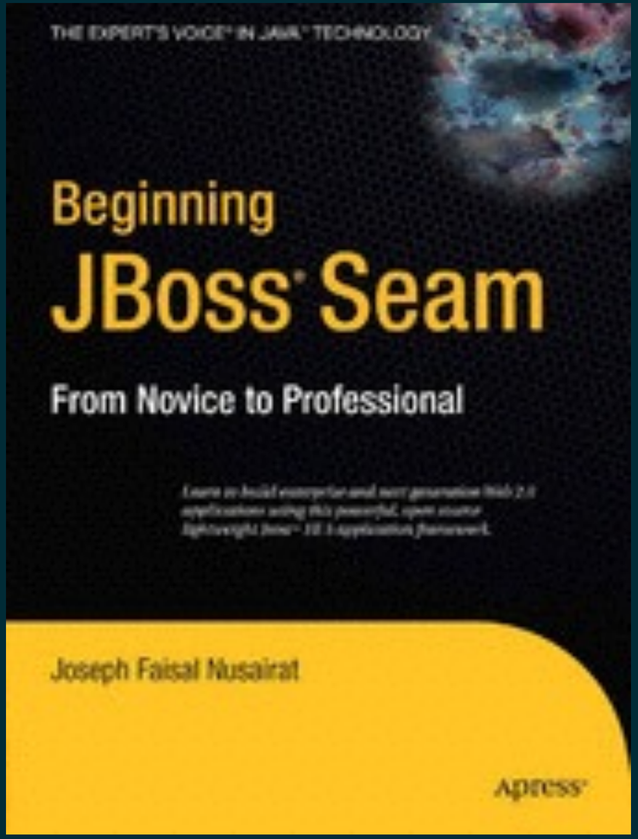
NOSQL With Grails

Joseph Nusairat
Groovy Sage
@nusairat

Integrallis
Ideas, Implemented.

About Me

- Java Developer since 1997
- Groovy / Grails Developer since 2007
- Scottsdale Groovy Brigade co-founder



THE EXPERT'S VOICE® IN JAVA™ TECHNOLOGY

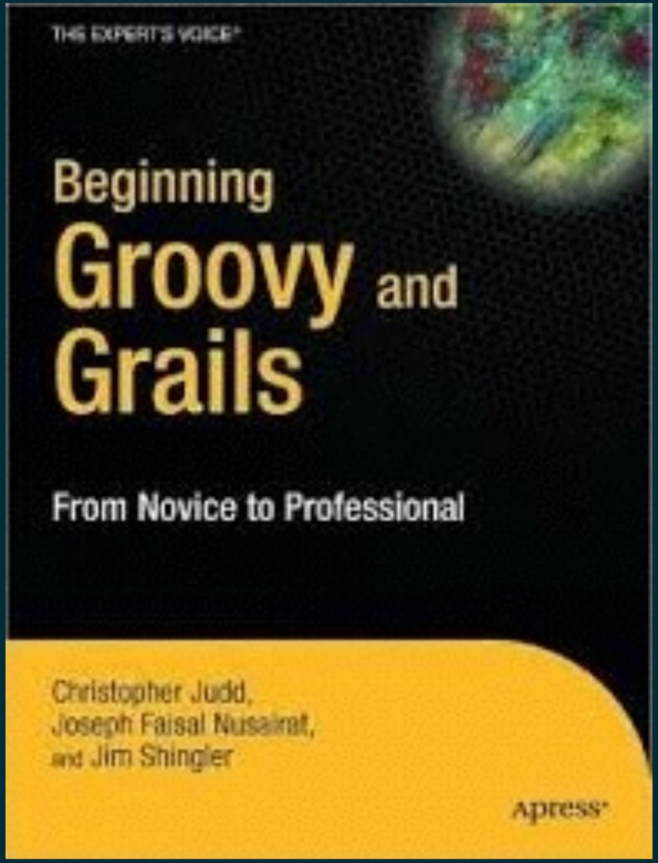
Beginning JBoss® Seam

From Novice to Professional

Learn to build enterprise and next generation Web 2.0 applications using this powerful, open source lightweight Java™ EE 5 application framework.

Joseph Faisal Nusairat

Apress®



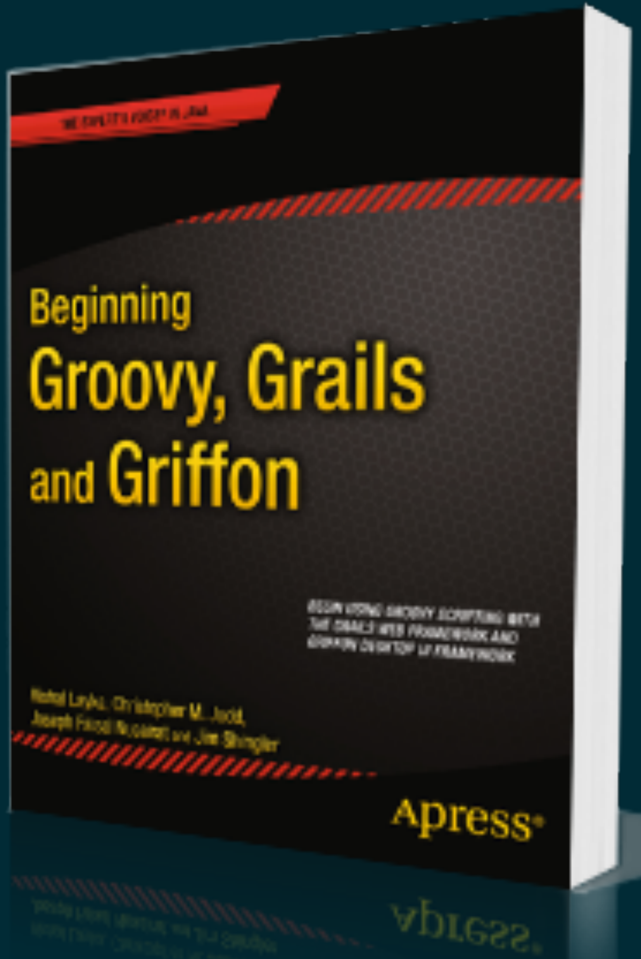
THE EXPERT'S VOICE®

Beginning Groovy and Grails

From Novice to Professional

Christopher Judd, Joseph Faisal Nusairat, and Jim Shingler

Apress®



THE EXPERT'S VOICE® IN JAVA

Beginning Groovy, Grails and Griffon

BEGIN USING GROOVY SCRIPTING WITH THE GRAILS WEB FRAMEWORK AND GRIFFON DESKTOP UI FRAMEWORK

Vishal Layka, Christopher M. Judd, Joseph Faisal Nusairat and Jim Shingler

Apress®

Abstract

- What is NoSQL
- How to use NoSQL
- How to call MongoDB
  - How to set up Mongo Documents
  - Querying Mongo
- Using Mongo with Grails
- Examples

NoSql

- Term originally coined in 1998 by Carlo Strozzi
- Reintroduced in 2009 by Johan Oskarsson
- Is used to refer to databases that do not use the standard SQL interface

NoSql Types

- Column: HBase, Accumulo
- Document: MarkLogic, MongoDB, Couchbase
- Key-value: Dynamo, Riak, Redis, Cache, Voldemort
- Graph: Neo4J, Allegro, Virtuoso

Design

- Transactions are fast because there are none. No overhead of transaction managers
- There are no tables or joins in the traditional sense. Meaning access to multiple tables with one query is not possible

In Memory

- Part of the speed performance is that NoSql often uses in memory storage
- This is generally a temporary store till it flushes to the file system, but this can lead to volatility if the server goes down.

Using NoSQL

General Use

- RDBS systems and traditional database design in mid to late 2000s started to show their age
- Many developers started switching ad hoc to NoSQL systems to speed up the applications
- Became the defacto toy to the shiny object crowd

Issues

- Sacrificed normalized design for performance
- No transactions meaning you’d have to make sure you were not doing multiple saves
- Potential for over duplication of data

Speed vs Design

Not Only SQL

- A more accurate representation of the name in how to use it
- NoSQL is used for high performance and throughput of data
- Any part of your site that gets used by spiders or heavy searching by people
- Increase performance on what needs performance

Where to Use

- Front end facing features
- Pages linked on the front end
- If its a a sales site, the items you are selling
- Huge non transactional calls
- Huge amounts of data

When Not to Use

- Highly complexed structured relations that you want to preserve or change
- Inheritance where updating a sub item effects others
- Rarely used data
- Small data sets

Combination

- Sometimes you have complex data that needs to be accessed quickly
- Data that needs back end complexity to set up and manage, but needs to be easily accessed on the front end
- Duplication is O K.

Example

- Deal Site
  - A user creates a complex deal with locations, versions of the deal, multiple deals
  - Store the inheritance for the user but then create a NoSQL record for the front end to run and query on

Threshold?

- Depends on hardware
- Depends on database size
- Also consider with document based NoSQL the taxing of the database is much less

Possible Exceptions

- Multiple database servers can require multiple expertise and great amounts of money
- If you have only a few collections / tables that you’d want to store in RDBMS then it may not make sense from a cost view

Mongo DB

What is it

- Document type database originally written in C++
- Data is stored in Binary JSON (BSON) which allows for for binary data type
- Data is arranged in collections which also allows for collections embedded in collections

What Makes it Special

- Document databases allow for more complex storage of data in an easy to read way (reads like JSON)
- Can also store binary files on the file system with a reference to them
- Allows for location based queries

Everything's Dynamic

- Traditional databases you define your databases, tables, etc ahead of time
- With Mongo it can all be defined on the fly
  - Databases
  - Collections

Collections

- Collections will be stored with BSON
- Collection by default contains a unique "_id"
- Collections can contain embedded documents or even lists of embedded documents
- Collections do not have a distinct set of columns
- Collections store documents

A Document

An Entire Collection

```
{
        "username" : "bob",
        "street" : "123 Main Street",
        "city" : "Springfield",
        "state" : "NY"
}
```

Collections in Documents

- When you need to reference collections from a document you can handle in 2 ways
- Embedded - having all the objects in the collection
- Reference - having the documents in one collection then referencing them in another

Embedded

- You can embed documents either in a list format or just one document at a time

```
{
     "username" : "bob",
     "address" : {
        "street" : "123 Main Street",
        "city" : "Springfield",
        "state" : "NY"
     }
}
```

Contains only one

Embedded

- A list will look the same but have an [] around the items.

```
{
     "username" : "bob",
     "address" : [ {
        "street" : "123 Main Street"
     },
     {
        "street" : "234 High Street"
     } ]
}
```

Contains many addresses

Reference

- You can also reference the collections from another collection, this will look more like a SQL type collection.
  - For collections with data stored outside
  - For extremely large collections
  - Rarely used data

Reference

- References can be reverenced either with the _id directly or with DBRef object

```
{
  "_id" : ObjectId(“52a848d8 … b0b3”),
  "name" : "My Item 1”,
  "locations" : [
    DBRef("locations", ObjectId(“5"))
  ]
}
```

Inserting a Document

- If you want to insert the previous document into a collection called “people”

```
db.people.insert({
     "firstName" : "joseph",
     “lastName”  : “nusairat”,
     "address" : {
        "street" : "123 Main Street",
        "state" : "NY"
     }
})
```

Inserting a Document

- The result in the database is creation of an object with the id “_id”

```
{
     “_id” :
          ObjectId("4cda8571b5da950b52727746")
     "firstName" : "joseph",
     “lastName”  : “nusairat”,
     "address" : {
        "street" : "123 Main Street",
        "state" : "NY"
     }
}
```

Updating Records

- Record updates in Mongo is much different than in normal RDBS systems.
  - Default is to update entire record
  - Can do an update where it will insert if the record does nto exist

Default Update

- Default call will take in a set of parameters that will be the “find” part of the update
- The second item is what to update
- This will update people to having just the column first name

```
db.people.update(
  {_id :  ObjectId(“4cda…7746”)},
  { firstName : 'joseph'}
)
```

Setting one Column

- If you only want to set one column you can use the $set call
- Using the set will update the column if it exists, or add it if it does not exist

```
db.people.update(
  { **condition** },
  { $set :
   { firstName : ‘joseph’}
  }
)
```

Other Choices

- Upsert : You can choose whether to add a record if it doesn't exist
- You can choose whether to update all records or the first match

```
db.people.update(
{}, { $set : {middleName : 'Javier'},
true, true } )
```

If all documents matched in the criteria are to get updated

Upsert : if the record does not exist we should insert it

Removing a Record

- Removing record is a direct call
- The call will remove all records that match a given set of indicators

```
db.people.remove({ firstName : 'joe'})
```

Defines the syntax to search for on removals.

File Storage - Grid FS

- GridFS is used for storing files. This is used more specifically to store files that are over 4MB
- This allows for safely storing large files by dividing them up among multiple documents
- Stores the files in different buckets the default bucket is called “fs”

Example with Groovy

- Becomes a bit more complex with direct calls
- Returns an id for correlation

```
def gridfs = new GridFS(db)
def f = new File('/var/in/struts.png')
def inputFile = gridfs.createFile()
inputFile.save()
```

Querying Mongo

Querying

- Querying records is more similar conceptually to those using GORM.
- You can query and find one record or find all

```
db.people.find({ firstName : 'joe'})
```

Find all records that

```
db.people.findOne({ firstName : 'joe'})
```

Find the first match

Map - Reduce

- Is used for more complex querying from the database
- This helps with the aggregation of data
- Map reduce can be used in a situation where you would normally have used group bys in SQL.
- Written in JavaScript
- Data outputted to temporary table

How it Works - Map

- Mapping is designed to take a large input and then divide up into smaller pieces
- The map explicitly defines the item that is going to be aggregated on

How it Works - Reduce

- The reduce aggregates the map outputs.
- Taking the smaller pieces and bringing them back into one for the final item
- The aggregation is based on the key passed in

Example

```
// emit calls the next result
// first: differentiator / second: aggregator
function map() {
  emit(1, {count : this.amount});
},

function reduce(key, values) {
  var count = 0
        for (var i = 0; i < values.length; i+
+)
            count += values[i].count
        return {count: count}
    }
```

Geo Spacial Querying

- Has the ability to set up a per collection field that is geo spatial specific
- These fields need to be defined with an index “2d”
- Can query records based on location

Geo Querying Types

- Near - find the items with the closest locations to you in order of closest first
- Box - define boundaries of a box and find all matching items in that box
- Circle - define boundaries of a circle and find all matching items in that circle

Mongo With Grails

Stand Alone or GORM

Mongo with Grails

- Mongo can integrate with Grails either via Mongo API calls directly or by a GORM syntax
- Both syntax’s can be used in the same application
- Will depend what you are comfortable with

Inline

- Inline syntax will use Mongo directly
- All creates / removes / queries will be with Mongo syntax and have to convert to local objects
- The syntax this way can be more precise when creating complex queries

GORM

- Makes use of the Grails sponsored plugin : mongodb
- This plugin wraps a GORM syntax around calls to Mongo
- Obscures the actual calls to Mongo
- Allows for embedding or referencing collections and domains

Adding Mongo Gorm

- Need to add the plugin
- Define the data source for Mongo
- If you are using a mix mode system must include:

  ```
  static mapWith = "mongo"
  ```

Identifier

- nothing
  - By default will use a sequence number storing a sequence in Mongo
  - I wouldn’t recommend
- String id
  - Sets a UUid
- ObjectId *recommended*
  - Uses the Mongo Object id

Querying with GORM

- Querying with Mongo uses the same Hibernate style syntax
- There is custom syntax for the Geo Queries

```
findByLocationNear
findByLocationWithinBox
findByLocationWithinCircle
```

Issues With GORM

- When changing the object on the embedded collection, GORM will not automatically update the embedded objects
- Limited to geo querying on fist levels of the collection
- 2D on embedded lists

Examples

THANK YOU - Q&A

https://github.com/nusairat/mongo-groovy-example