Salat in fifteen minutes

...or less

Rose Toomey
Novus Partners

Salat in fifteen minutes... or less

- What is Salat?
  - Getting started
- Demonstration
- How does it work?
  - Moving parts
  - What Scala types can it handle?
- SalatDAO
- More information

NOVUS

What is Salat?

Salat is a bi-directional Scala case class serialization library that leverages MongoDB's DBObject (which uses BSON underneath) as its target format.

Salat has dependencies on the latest releases of:

- **scalap, a Scala library that provides functionality for parsing Scala-specific information out of classfiles**
- **mongo-java-driver, the official Java driver for MongoDB**
- **casbah-core, the official Scala toolkit for MongoDB**

NOVUS

Availability

The latest release, Salat 0.0.7, is available for Scala 2.8.1.

The latest snapshot, Salat 0.0.8-SNAPSHOT, is available for Scala 2.8.1 and 2.9.0-1. This snapshot is compatible with Play 1.2.2RC1+.

Salat is *not* available for Scala 2.7.7 because pickled Scala signatures were introduced in Scala 2.8.0.

Salat is *not* compatible with Java classes for the same reason.

NOVUS

Getting started

Add the Novus repos and the salat-core dependency to your sbt project

```
val novusRepo = "Novus Release Repository" at "http://repo.novus.com/releases/"
val novusSnapsRepo = "Novus Snapshots Repository" at "http://repo.novus.com/snapshots/"

val salat = "com.novus" %% "salat-core" % "0.0.8-SNAPSHOT"
```

Import Salat implicits and default context

```
import com.novus.salat._
import com.novus.salat.annotations._
import com.novus.salat.global._
```

Demonstration: there and back again

```
case class Book(@Key("_id") id: ObjectId = new ObjectId,
          title: String)

scala> val b = Book(title = "Great Expectations")
b: prasinous.model.Book = Book(4dfa8327cf55223e5c15a200,Great Expectations)

scala> val dbo = grater[Book].asDBObject(b)
dbo: com.mongodb.DBObject = { "_typeHint" : "prasinous.model.Book" ,
  "_id" : { "$oid" : "4dfa8327cf55223e5c15a200"} , "title" : "Great Expectations"}

scala> val b_* = grater[Book].asObject(dbo)
b_*: prasinous.model.Book = Book(4dfa8327cf55223e5c15a200,Great Expectations)
```

NOVUS

How does it work?

A case class instance extends Scala's Product trait, which provides a product iterator over its elements.

Salat used pickled Scala signatures to turn case classes into indexed fields with associated type information.

These fields are then serialized or deserialized using the memoized indexed fields with type information.

For more information about pickled Scala signatures, see

`scala.tools.scalap.scalax.rules.scalasig.ScalaSigParser`

In addition, refer to this brief paper:

SID # 10 (draft) - Storage of pickled Scala signatures in class files

NOVUS

Moving parts

- a `Context` has global serialization behavior including:
  - how type hinting is handled (always, when necessary or never) - default is always
  - what the type hint is - default is `_typeHint`
  - how enums are handled (by value or by id) - default is by value
  - math context used for deserializing BigDecimal (default precision is 17)
- a `Grater` can serialize and deserialize an individual case class

NOVUS

Keeping things in scope

The context is an implicit supplied by importing Salat's `global` package object (or your own package object).

```
import com.novus.salat.global._
```

Graters are created on first request. Use the `grater` method supplied in Salat's top level package object:

```
import com.novus.salat._

grater[Book].asObject(dbo)
```

NOVUS

Try it out!

The sample code shown in this presentation is available at:
rktoomey/scalathon-presentation.
If you need to install MongoDB, see the Quick Start

You can build and run the project using simple-build-tool.

The quickest way to get started experimenting is to clone the project and run `sbt console` to use a Scala interpreter with a classpath that includes compiled sources and managed libs:

```
~ $ git clone git://github.com/rktoomey/scalathon-presentation.git
~ $ cd scalathon-presentation
~/scalathon-presentation $ sbt update
~/scalathon-presentation $ sbt console
```

NOVUS

What Scala types can Salat handle?

- case classes
  - embedded case classes
- embedded case classes typed to a trait or abstract superclass annotated with `@Salat`
- Scala enums
- Options
- collections

Collections

Maps are represented as `DBObject`; all other collections turn into `DBList`.

NOVUS

In detail: Salat collection support

Salat 0.0.7 and below support the following immutable collections:

- **Map**
- **List**
- **Seq**

Salat 0.0.8-SNAPSHOT and above support the following mutable and immutable collections:

- **Map**
- **Lists and linked lists**
- **Seqs and indexed seqs**
- **Set**
- **Buffer**
- **Vector**

NOVUS

What does Casbah's BSON encoding hooks handle?

- **org.joda.time.DateTime**
- **BSON types - see BSON specs for more information**

For more information on how to write and use BSON encoding hooks, see the Casbah API docs:
Briefly: Automatic Type Conversions

```
com.mongodb.casbah.commons.conversions.scala.RegisterConversionHelpers()
com.mongodb.casbah.commons.conversions.scala.RegisterJodaTimeConversionHelpers()
```

NOVUS

Unsupported types

Salat can't support any of these types right now:

- **Nested inner classes (as used in Cake pattern)**
- **see this interesting scala-internals discussion about developing the Scala reflection API**
- **A class typed at the top-level to a trait or an abstract superclass**
- `com.mongodb.DBRef`

Salat can't support these types because the mongo-java-driver doesn't support them:

- **Any type of Map whose key is not a String**
  - **any type of map whose key is a String containing `.` or `$`**

NOVUS

Annotations

Salat offers the following annotations to customize serialization behavior:

- `@Salat` **to support polymorphic instances of a trait or abstract superclass**
- `@Key` **to change the name of a field**
- `@Persist` **to serialize a value outside the case class constructor**
- `@Ignore` **to ignore a field in the case class constructor**
- `@EnumAs` **to customize the behavior of a particular enum**

To use these annotations, import the types:

```
import com.novus.salat.annotations._
```

NOVUS

SalatDAO: just add water

`SalatDAO` **makes it simple to start working with your case class objects. Use it as is or as the basis for your own DAO implementation.**

By extending `SalatDAO`**, you can do the following out of box:**

- **insert and get back an Option with the id**
- **findOne and get back an Option typed to your case class**
- **find and get back a Mongo cursor typed to your class**
- **iterate, limit, skip and sort**
- **update with a query and a case class**
- **save and remove case classes**

NOVUS

SalatDAO: getting started

```
import com.novus.salat._
import com.novus.salat.global._
import com.novus.salat.annotations._

case class Author(@Key("_id") id: ObjectId = new ObjectId,
            lastName: String,
            firstName: String,
            middleName: Option[String] = None,
            nationality: Option[String] = None,
            yearOfBirth: Option[Int] = None)  {

  // contrived example of using @Persist to allow a value outside the case class contructor to be
  /// serialized when an instance of Author is changed into a DBObject
  @Persist lazy val displayName = "%s %s".format(firstName, lastName)

}

object AuthorDAO extends SalatDAO[Author, ObjectId](collection = MongoConnection()("library")("author"))
```

NOVUS

SalatDAO: insert and find

```
scala> val a = Author(firstName = "Charles", lastName = "Dickens")
a: prasinous.model.Author = Author(4dfa8853cf55223e5c15a203,Dickens,Charles,None,
  None,None)

scala> val _id = AuthorDAO.insert(a)
54 [Thread-51] INFO prasinous.DB$ - loaded config from file:
  /home/rose/workspace/scala-orms/salat/target/scala_2.9.0-1/test-resources/phase.conf
197 [Thread-51] INFO prasinous.collections$ - unique index: author by: { "firstName" : 1 ,
  "lastName" : 1}
217 [Thread-51] INFO prasinous.collections$ - unique index: books by: { "title" : 1}
219 [Thread-51] INFO prasinous.collections$ - unique index: book_author by: { "bookId" : 1 ,
  "authorId" : 1}
220 [Thread-51] INFO prasinous.collections$ - unique index: borrowal by: { "bookId" : 1 ,
  "borrowerId" : 1 , "scheduledToReturnOn" : 1}
_id: Option[com.mongodb.casbah.Imports.ObjectId] = Some(4dfa8853cf55223e5c15a203)

scala> val a_* = AuthorDAO.findOneByID(new ObjectId("4dfa8853cf55223e5c15a203"))
a_*: Option[prasinous.model.Author] = Some(Author(4dfa8853cf55223e5c15a203,Dickens,
  Charles,None,None,None))

scala> val a_* = AuthorDAO.findOne(MongoDBObject("lastName" -> "Dickens"))
a_*: Option[prasinous.model.Author] = Some(Author(4dfa8853cf55223e5c15a203,Dickens,
  Charles,None,None,None))
```

SalatDAO: update and save

```
scala> AuthorDAO.update(MongoDBObject("_id" -> a.id), a.copy(yearOfBirth = Some(1813)),
  upsert = false, multi = false, new WriteConcern)

scala> AuthorDAO.findOneByID(new ObjectId("4dfa8853cf55223e5c15a203"))
res12: Option[prasinous.model.Author] = Some(Author(4dfa8853cf55223e5c15a203,
  Dickens,Charles,None,None,Some(1813)))

scala> AuthorDAO.save(a.copy(yearOfBirth = Some(1812)))

scala> AuthorDAO.findOneByID(new ObjectId("4dfa8853cf55223e5c15a203"))
res3: Option[prasinous.model.Author] = Some(Author(4dfa8853cf55223e5c15a203,Dickens,
  Charles,None,None,Some(1812)))
```

SalatDAO: remove

```
scala> AuthorDAO.remove(a)

scala> AuthorDAO.findOneByID(new ObjectId("4dfa8853cf55223e5c15a203"))
res14: Option[prasinous.model.Author] = None
```

NOVUS

SalatDAO: define relationships between collections

So we can serialize and deserialize `Author` and `Book` - but how can we find what books an author has written or how many authors a book has?

Disclaimer: This approach is not particularly idiomatic to MongoDB, but was shown to demonstrate how using SalatDAO with child collections can ease a traditional SQL approach shown in the `bookauthor` table definition below.

```
//  CREATE TABLE bookauthor (
//    book_id BIGINT NOT NULL,
//    author_id BIGINT NOT NULL,
//
//    PRIMARY KEY (book_id, author_id),
//    FOREIGN KEY (author_id) REFERENCES author(id),
//    FOREIGN KEY (book_id) REFERENCES book(id)
//  );

case class BookAuthor(@Key("_id") id: ObjectId = new ObjectId,
                      bookId: ObjectId,
                      authorId: ObjectId)
```

NOVUS

Now add a child collection to AuthorDAO

```
object AuthorDAO extends SalatDAO[Author, ObjectId](collection = db.author) {

  class BookAuthorCollection(collection: MongoCollection, parentIdField: String)
    extends ChildCollection[BookAuthor, ObjectId](collection, parentIdField)

  val bookAuthor = new BookAuthorCollection(collection = db.bookAuthor, parentIdField = "authorId")

}
```

NOVUS

What can you do with a child collection?

```
scala> val b = Book(title = "L'Assommoir")
b: prasinous.model.Book = Book(4dfa93decf55946e59a52864,L'Assommoir)

scala> BookDAO.insert(b)
res2: Option[com.mongodb.casbah.Imports.ObjectId] = Some(4dfa93decf55946e59a52864)

scala> val ab = BookAuthor(authorId = a.id, bookId = b.id)
ab: prasinous.model.BookAuthor = BookAuthor(4dfa93fccf55946e59a52865,
  4dfa93decf55946e59a52864,4dfa90a3cf55946e59a52862)

scala> AuthorDAO.bookAuthor.insert(ab)
res3: Option[com.mongodb.casbah.Imports.ObjectId] = Some(4dfa93fccf55946e59a52865)
```

NOVUS

OK, how about something more useful?

```
def addBook(a: Author, b: Book) = {
  BookDAO.insert(b)
  bookAuthor.insert(BookAuthor(bookId = b.id, authorId = a.id))
}

scala> val b = Book(title = "La Bête humaine")
b: prasinous.model.Book = Book(4dfa9558cf553f152cb73f7f,La Bête humaine)

scala> AuthorDAO.addBook(a, b)
res1: Option[com.mongodb.casbah.Imports.ObjectId] = Some(4dfa956acf553f152cb73f80)

scala> AuthorDAO.bookAuthor.findByParentId(a.id).toList
res5: List[prasinous.model.BookAuthor] = List(BookAuthor(4dfa956acf553f152cb73f80,
  4dfa9558cf553f152cb73f7f,4dfa9516cf553f152cb73f7e))
```

NOVUS

Projections

```
scala> BookDAO.primitiveProjection[String](MongoDBObject(), "title")
  res14: Option[String] = Some(The Demon-Haunted World: Science as a Candle in the Dark)

scala> BookDAO.primitiveProjections[String](MongoDBObject(), "title").sorted
res11: List[String] = List(Cosmos, J'Accuse, L'Assommoir, La Bête humaine,
  Programming in Scala, The Demon-Haunted World: Science as a Candle in the Dark)
```

Salat supports two types of projections:

- **case classes**
- **"primitive" types where deserialization is handled by BSON**

NOVUS

Projections + child collections

```
def booksByAuthor(author: Author): List[Book] = {
  val bookIds = bookAuthor.primitiveProjectionsByParentId[ObjectId](parentId = author.id, field = "bookId")
     BookDAO.find(ref = MongoDBObject("_id" -> MongoDBObject("$in" -> MongoDBList(bookIds: _*))))
       .sort(MongoDBObject("title" -> 1))
       .toList
}

scala> AuthorDAO.addBook(a, Book(title = "J'Accuse"))
res6: Option[com.mongodb.casbah.Imports.ObjectId] = Some(4dfa9654cf553f152cb73f82)

scala> AuthorDAO.addBook(a, Book(title = "L'Assommoir"))
res7: Option[com.mongodb.casbah.Imports.ObjectId] = Some(4dfa9668cf553f152cb73f84)

scala> AuthorDAO.booksByAuthor(a)
res13: List[prasinous.model.Book] = List(Book(4dfa9654cf553f152cb73f81,J'Accuse),
  Book(4dfa9668cf553f152cb73f83,L'Assommoir), Book(4dfa9558cf553f152cb73f7f,La Bête humaine))
```

NOVUS

Does a child collection really have to be a child collection?

No! "Child collection" implies a relationship from the perspective of the outer class: you can map the `bookAuthor` collection multiple times: once from `AuthorDAO` using `Author`'s `_id` as the parent id, and again from `BookDAO` using the `Book`'s `_id` as the parent id.

```
object BookDAO extends SalatDAO[Book, ObjectId](collection = db.book) {

  class BookAuthorCollection(collection: MongoCollection, parentIdField: String)
    extends ChildCollection[BookAuthor, ObjectId](collection, parentIdField)

  val bookAuthor = new BookAuthorCollection(collection = db.bookAuthor, parentIdField = "bookId")

  def authorsForBook(book: Book): List[Author] = {
    val bookIds = bookAuthor.primitiveProjectionsByParentId[ObjectId](parentId = book.id, field = "authorId")
    AuthorDAO.find(ref = MongoDBObject("_id" -> MongoDBObject("$in" -> MongoDBList(bookIds: _*))))
      .sort(MongoDBObject("lastName" -> 1))
      .toList
  }
}
```

NOVUS

What happens next?

We're working to make the code in Salat more modular and general purpose.

- **we're renovating our current transformer framework to use chained partial functions instead**
- **add your own custom transformers!**
- **proof of concept in this gist**
- **better support for abstract classes and traits**
- **our tools for working with pickled Scala signatures will be moved to `salat-util`, a standalone module without dependencies**
- **the current `salat-core` module will contain a generic framework for managing contexts and transformers**
  - **`salat-core` will have core JSON and BSON transformers**
  - **submodules will provide additional `Grater` capabilities by providing additional transformer implementations**
- **the Casbah dependencies will be moved out to `salat-casbah` in preparation for adding...**
- **a new Salat module for using Brendan McAdams' Hammersmith project**

plus...

Chris Lewis is joining Novus with a lot of great ideas! We've been discussing the possibility of a query DSL like the homage to rogue DSL that Chris recently added to highchair.

NOVUS

Finding out more

Slides and sample code for this presentation are posted at rktoomey/scalathon-presentation

- **The specs in the Salat source code provide many usage examples.**

  Github
  https://github.com/novus/salat

  Wiki
  https://github.com/novus/salat/wiki

  Mailing List
  http://groups.google.com/group/scala-salat

  Twitter
  @prasinous

NOVUS

Thank you

- Novus supports the development of Salat
- PHASE for planning and hosting Scalathon
- the June PHASE meetup on Scala ORMs should not be missed! scala-phase/scala-orms
- 10gen for supporting the ongoing development of Casbah
- Eric Torreborre for specs2, which I use to write specifications for Salat
  - see slides for specs2: What's new in the Scala BDD world?, my recent ny-scala meetup presentation on specs2
- Brendan McAdams for Casbah and Hammersmith
  - and for being a wellspring of constructive inspiration on how open source projects can make things better...
- Oliver Dodd for submitting Salat's first pull request, to provide better `Traversable` support
- @softprops for picture-show, a simple elegant way to make slides with Markdown
- finally, thank you to all the mailing list members who keep the discussion going: your participation helps to make better tools!

NOVUS