Groovy On The Trading Desk:

Best Practices Developed From Distributed Polyglot Programming

Jonathan Felch
jonathan.felch@gmail.com
JonathanFelch on Twiiter

Agenda

- Groovy Main Point
  - Groovy Manifesto
  - Major Language Features
- Computational Finance and Distributed Computing
  - Finance Specific: Math / Data / Business / Languages
  - Groovy Lessons: Use Cases
- Smart Grid and Dynamic Programming
  - Data Grid: Moving Data Around
  - Computational Grid: Moving Work and Code Around
  - Operator Overloading and Dependency Graphs
  - Groovy Lessons: Groovy Types, Dynamic Methods, GPars
- Functional Programming and The Problem of State
  - Objects Versus “Smart Tuples”
  - Closures, Operators, Currying and Chaining
  - Groovy Lessons: Groovy Uses Or Groovy Is ?
  - Groovy Type System: Friend or Foe?

Introduction

Jonathan Felch

- NASDAQ Software Architect 1997-99
- Lehman Brothers Global e-Commerce Architect 1999-2000
- Venture Capital Associate @ GS / BCG JV 2000-2001
- Quantitative Systems @ Syntax Capital 2005-2006
- VP Quantitative Prop Trading @ Credit Suisse 2006-2009
- Quant Trader @ E.H. Smith Jacobs High Frequency 2009+

jonathan.felch@gmail.com

JonathanFelch On Twitter and LinkedIn

Groovy Manifesto

- is an agile and dynamic language for the Java Virtual Machine
- builds upon the strengths of Java but has additional power features inspired by languages like Python, Ruby and Smalltalk
- makes modern programming features available to Java developers with almost-zero learning curve
- supports Domain-Specific Languages and other compact syntax so your code becomes easy to read and maintain
- makes writing shell and build scripts easy with its powerful processing primitives, OO abilities and an Ant DSL
- increases developer productivity by reducing scaffolding code when developing web, GUI, database or console applications
- simplifies testing by supporting unit testing and mocking out-of-the-box
- seamlessly integrates with all existing Java objects and libraries
- compiles straight to Java bytecode so you can use it anywhere you can use Java

Groovy Use Cases

- Super Glue
- Half Baked Ideas
- Cookie Cutter Apps For Really Good Cookies
- Meta-Programming, Builders, And DSLs

Super Glue Example

Combine GUI Library (Swing), Network Library, and XML Parser to make RSS Feed

```
def url ='http://www.groovyblogs.org/feed/rss'
def items = new XmlParser().parse(url).channel.item
def cols = 'pubDate title description'.tokenize()

groovy.swing.SwingBuilder.build {
 frame(id:'f', title: 'Groovy RSS', visible:true) {
   scrollPane {
     table {
       tableModel(list: items) {
         cols.each { col →
           closureColumn header: col,
            read: { it[col].text() }

     } } } } }
   f.pack()
 }
```

Groovy Performance: Numeric Collections

Operator Overloading Creates Implicit Dependency Graph That Optimizes Evaluation

- Only Re-calculate Values That Change
  - Overloading operators in numeric collections allow numeric operations to only-recalculate variations in the dependency graph
  - JIT / Optimizers will load partial expressions into CPU registered
- Closures as formulas
  - Rather than using loops for executing an expression many times, the collections can be mixed with numeric values and constants in a single expression

Groovy Performance: Numeric Grid

```
//  Monte Carlo Simulation For European Put Option in 10 Lines Or Less



def px = 100, r = 0.05, vol = 0.15, t = 1.0
def strikes = [80, 90, 100, 110, 120 ]

def w = RandomNumbers.getNormDist(1000,1000)

def S = px * Math.E ** ((r - ½ * vol * vol) * t + sqrt(t) * vol * w)

strikes.each { K →
  def optionValue = Math.max(0, S - K)
  def df = exp(-rate * time)
  println "${strike} : ${df * optionValue as Number}"
}


// In Java or C You Would Have To Loop
```

Why Groovy ?

- Pith, Speedy Development Cycle
  - Made for half baked ideas
- Learning Curve
  - Familiar To Java Programmers, Java Syntax is (Mostly) Groovy Syntax
- Dynamic Programming
  - Meta-Programming, DSL Support
- Java / JEE / Enterprise
  - Easy Stuff Is Actually Easy
- Community

What is Quant Finance ?

A quant designs and implements software and mathematical models for the pricing of derivatives, assessment of risk, or predicting market movements

THE
SECRET FORMULA
That Destroyed Wall Street

$$\mathbf{P=\phi(A,B,\gamma)}$$

$$dV = \left(\mu S\frac{\partial V}{\partial S} + \frac{\partial V}{\partial t} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 V}{\partial S^2}\right) dt + \sigma S \frac{\partial V}{\partial S} dW.$$

$$S_t = S_0 e^{(\mu - \frac{1}{2}\sigma^2)t + \sqrt{(t)}\sigma\omega}$$

What's The Problem: The Math

- Quant Finance Models are Wrong
  - Even The Best Models Fail, Failure Is Expensive
- Assumption of Quantitative Finance Are Wrong
  - Market Are Social Phenomena
  - Not Random Walks, Not Natural Systems
- Quant Finance Models Change
  - In Bad Times, They Change A Lot
- Coding Almost everything in C++ takes forever
- Coding Everything Else in VBA doesn't scale

What's The Problem: The Market

- Market Structures Drive Financial Data
  - Different Calendars, Different Measures
  - Equities and Government Auctions are Transparent
    - Also options, some bonds, some preferred
  - Exotic and Credit Markets are Illiquid, No Transparency
    - Some of products are not really 'securities'
- Identifiers are ridiculous, don't work, unclear
  - ISIN, CUSIP, SEDOL, Tickers, ADRs, …
  - Lifecycle of a Bond's ISIN (144a, Reg S, Registered)

What's The Problem: The Data

- Lots of Data, Lots of Math, Lots of Products
  - Credit Market Subset
    - 1500 Companies / 2500 Curves / 10 Indices & Tranches
    - 10,000 Liquid Bonds / 2,000 Liquid Converts / 2,000 Loans
    - 1500 Liquid Equities / 169 point vol surface to start
  - Derivatives and Quant strategies have many metrics for each time series observation
    - Securities can't be compared on price
    - Relative values metrics are complex and there are many

What's The Problem: The Traders

- Great Trades Come From Half-Baked Ideas
  - Fully Baked Ideas Have Already Been Priced In
- Traders Do Not Know What They Want
  - Good traders ride the cusp of intuition and logic
- Whatever They Think They Want, They Wanted It Yesterday
- Whatever They Want Today, They Will Never Use Again
  - Downside of the half baked idea

The Evils Of Financial Databases I

Date	Price	SMA_3
3	102	101
4	103	102
5	104	103
6	105	104

Date	Price	Ticker
1	100	ABC
2	101	ABC
3	102	ABC
4	103	ABC

WRONG WAY:

```
SELECT DATE, PRICE, (TS1.PRICE+
TS2.PRICE+TS3.PRICE) / 3 AS SMA_3

FROM TIMESERIES TS1,
  TIMESERIES TS2, TIMESERIES TS3

WHERE TS1.TICKER = TS2.TICKER   |
  AND TS2.TICKER = TS3.TICKER AND
  TS2.DATE = (TS1.DATE-1) AND
  TS3.DATE = (TS2.DATE-1) AND
  TS1.TICKER = 'ABC'
```

Languages of Quant Finance

- Commonly used languages of Quant Finance
  - C++ (The Dominant Industrial Strength Language)
  - VBA
  - Matlab, SAS, STATA, S+, and R
  - C#
  - Java (Most limited to Fixed Income and Web)

- Up and Coming / Research Languages of Interest to Quant Finance
  - Fortress, Scala, Groovy, Python, F#, and Erlang

Where Should We Go

- Polyglot Coding:
  - Use C++ or Java Where You Need To
  - Extend That Foundations With Python, Groovy, Lua, Ruby, Scala, or some other dynamic language with support for closures, meta-programming, and high-level operations
- Post-SQL Data Management
  - Combine Column Oriented and Row Oriented Database Features In Cache
  - Use Cache and Workspace and Integration Space
  - Allow “Objects” to Evolve Dynamically
  - Naturally Order Data Is Ordered In Cache

Groovy Performance: Bad News

Overhead if NumericGrid Had Been Written in Groovy Rather than Groovy-Aware Java

- Type System:
  - Groovy Really Likes Java Collections, But Not Array
  - Groovy Really Likes BigDecimal, But Not Primatives
  - Groovy Really Likes Duck Typing
- Method Invocation
- Gparallelizer (Now Gpars)
  - DSL For the JSR 166y ParallelArray Would Have Invoked Many Copies of Groovy Collections Into Primative Maps

Databases versus Caching

- Traditional Model: Hibernate
  - Data Model = Database plus Cache of POJOs
    - All Objects of the name class share structure
    - No (Persistent) Dynamic Properties on 1st class objects
    - All first class objects (query-able) lived in the database

- Our Model: All POJOs → TupleMaps or Nodes
  - Tuples of same class may 'grow' existing structure
  - Tuples do not all have to come from data
    - Questions about what does and does not belong in database
    - Query Language = Gpath / Xpath + Hibernate
    - Includes dynamic properties and calculated values

Distributed Cache and MetaProgramming I

- Terracotta for the shared memory and synchronization
  - Integration point for Hibernate and Hibernate Cache
  - Integration point for Groovy Data Adapters
- All First Class Objects are decomposed from Java or Groovy objects to a 'Tuple'
  - Not perfectly named, but a simple data structure than implements Map and List
  - Usable with XPATH
  - Small Set of Primitives optimized for Terracotta

Distributed Cache and Meta-Programming II

- Everything is a Property
  - Data and methods
  - Behavior follows a mathematical model
  - Property listeners manage invalidation

- Missing Methods / Missing Properties
  - Widely used calculations and method results stored as property values so avoid redundant calculation
  - Calculated values are never stored in the database

Distributed Cache and Meta Programming III

- Tuple Class
  - Much like a Groovy Class
  - Joins objects like associations / relations in Hibernate
  - Defines raw types / names / converters
  - Defines property finders / chained finders / methods
- Missing Methods / Missing Properties
  - Widely used calculations and method results stored as property values so avoid redundant calculation
  - Calculated values are never stored in the database

Distributed Cache and Meta Programming IV

- Do We Even Want A Database ??
  - Sometimes Accessing Data Remotely Works Just As Well
  - Sometimes Pulling Data from Flat Files On Demand works Just As Well
  - Sometimes Calculating from Values from old inputs makes more sense than persisting it (normal forms)
- 'Active' Cache As a Integration Space
  -

Distributed Cache and Meta Programming IV

- Did TupleMap and Tuple Class Simply Re-Create The Object ?
  - Functional Closures != Methods
  - State Is Never Shared
  - Curried Closures Can Move Large Tasks to Distributed Work Queues or Thread Pools
- Data + Computational Grid = Scheduling Fun
  - Move Work Requests To Where Data Lives
  - Send Curried Closures To Computational Engines

Grails As A Integration Hub

- Controller requests arrive via JSON, XML, JMS
  - R Language Client: JSON → Grails
  - Excel Client: JSON → Grails Over HTTP
  - Excel RTD: JSON → Grails over JMS
  - SwingX Table (Real-Time) JSON → Grails via JMS
  - SwingX Table JSON → Grails via HTTP

- Affiliated Business Units:
  - XML Web Services from dot Not
  - Matlab dot Net

Cache Logic: Reporting

```
def r = builder.reportWith(Bond.class, ”bond.cdsBasis < 0”) {
   attr {
      expression = 'bond.ticker'
      name = 'Tkr'
   }
   attr {
      expression = 'bond.coupon'
      name = 'Tkr'
   }
   attr {
      expression = 'bond.maturity'
      name = 'Tkr'
   }
   attr {
      expression = 'bond.spot?.price?.last'
      name = 'Px'
   }
   attr {
      expression = 'bond.spot?.yield?.last'
      name = 'Px'
   }
   attr {
      expression = 'bond.spot?.zspread?.last'
      name = 'Px'
   }
   attr {
      expression = 'bond.spot?.cdsBasis?.last'
      name = 'Px'
   }
}
```

Grails to Excel I: DASL(Ticker,Exp)

Ticker / Expression	JAVA Equity	ORCL Equity	GM Equity	IBM Equity
it.spot.px	9.0	18.42	1.09	101.37
it.volSurface.find(delta : 50, expiry : 365).spot.iVol	22	44	180	39
it.cds.spot.spread	65.31	63	23730	60
it.refBond.spot.zspread	230	55	18700	57
it.cds.spot.basis	-164.7	8	1030	3
it.fin.mrq.netDebt	-300	10000	28846	21000
it.fin.mrq.totalDebt / it.fin.mrq.ebitda	N/A	5.5	N/A	6.3

Grails to Excel II: DSLH(Ticker,Exp,Start,End)

Ticker / Expression	JAVA Equity	ORCL Equity	GM Equity	IBM Equity
it.spot.px	9.0	18.42	1.09	101.37
15 May 2009	9.0	18.42	1.09	101.37
14 May 2009	9.0	18.46	1.15	101.05
13 May 2009	8.95	18.07	1.21	102.26
12 May 2009	9.05	18.38	1.15	103.94
11 May 2009	8.91	18.56	1.44	99.83
8 May 2009	8.71	18.32	1.61	101.49

- Expressions can be complex, traverse related objects, join disparate data sources

Grails to Excel III: DSLR(Report[Optional Para])

=DSLR('SUNMA')	EqyPx	CDS	Bond	Basis	Debt	Leverage
JAVA Equity	9.0	63	230	-164.7	-300	N/A
ORCL Equity	18.42	63	55	8	10000	5.5
IBM Equity	101.37	60	57	3	21000	6.3

Dasel:
A DSL for Financial Data

- Injectable Closures Into Tuples for “Column” Definitions
- Simple Reporting / Excel Grammar
- GRAILS Rendered Everything Into Web Pages or JSON / XML Services
- Component Library For Quantitative Analysis
- Massively Scalable Time Series Data Cache

The Revolution I Technology And Finance

- The Network Is The Computer
  - We Can't Agree On Which End Of The Byte Comes First (Big Endian / Little Endian)
  - We Can't Agree On Character Set and Line Delimiters (EBCIDEC, ASCII, Unicode)
  - We Can't Agree How to Share Files
  - We Can't Agree How To Share Code

Groovy Gotchas

- Pimping my library → Not Always Helping
  - GPars: Copies are expensive, the syntax is great
- Language Gotchas
- Dynamic Method Invocation
  - More Expensive than it should be
- 'Groovy' Can Be Expensive: Abusing Each
  - Anonymous Closures Versus loops (list.each {})