Groovy, Smashing. Yay capitalism!

Posted At : March 27, 2009 3:56 PM | Posted By : Jon Messer
Related Categories: Groovy

Although capitalism does seem to be falling apart at the moment, at least we still have Groovy. And boy is it groovy baby, yeah.

Thanks to the (super hero like even) efforts of this guy, it is ridiculously easy to use Groovy within ColdFusion. Why would you want to use another language within a CF application? Speed, elegance, flexibility or just for giggles. Let's start with the speed.

ColdFusion is a fantastic RAD platform and has a very low bar of entry for building database powered HTML applications. But it's single biggest failing (in my mind) is it's dog slow performance in terms of object creation.

This becomes especially apparent when you start to build complex domain models. You should never have to consider the performance impact of creating a new class if it's required in your domain, but unfortunately in CF you do.

And while Transfer is the bees knees in terms of CF persistence frameworks, it is still constrained by the Object Instantiation Penalty (OIP) of CF itself. Transfer (and all other CF "ORM"s for that matter) also by default is fairly tightly bound to the relational model, not completly, it (they) does let you use composition to define relationships in a few different ways. But it doesn't do inheritence, bi-directional relationships, classes that span tables, or multiple classes persisted within one table, or any of the more exotic mapping strategies that Hibernate allows.

So what are we to do? We want to use CF for what it is great at namely, integration, front controllers, service layer, and HTML UI generation, but we also want to be able to build a domain model unconstrained by the relational model or instantiaion penalties.

Well Java combined with grandaddy of all ORMs are very performant and CF does run on top of the JVM after all so why not Java? Java is too verbose and ceremonious for my taste, plus I prefer dynamic typing, oh and I want an elegant implementation of closures at the language level. JPA annotated Groovy to the rescue, and again thanks to Barney it is a completely painless install. Now you might not even realise how expensive CFCs are to create vs POGOs or POJOs, but they are at least an order of magnitude slower. I'm talking heap sort vs bubble sort here ($O(n^2)$ vs O(n log n)) kinda difference (OK I totally made that up but it sure feels like that big of a difference).

So let's install it and build a domain model already!

I was using the 1.0 branch until a few days ago , and if you were going to deploy code using CFGroovy any time soon that might be a better choice, but the 1.1 branch doesn't require you to drop any jar file in the CF class path, and it uses Groovy 1.6.

So that's what I'm playing with now. First off you can download the project files that I'll be using in this post and place the into a blank webroot.

If you're going to follow along with my project files you need to have Coldspring installed, but other than that the only configuration that you have to do is set up a CF data source called cfgroovydemo and point it at a blank database of your choosing {yes i said **blank** we are modelling objects here we don't need to create no stinkin tables :) }

WARNING if you do run this demo on your machine, make sure that you are pointing at a **blank** database I don't want to be responsible for wiping out your tables.

Most of the plumbing code in my demo is just adapted from the demos included with CFGroovy, I highly recommend playing around with them to see some basics. Just grab the BER of CFGroovy and place it into a web root (https://ssl.barneyb.com/svn/barneyb/cfgroovy/trunk/) and you can start playing with the demos.

I'm not going to go into any detail about the Application.cfc stuff, it is pretty striaght forward, it just spins up a coldspring applicationContext to manage the CFGroovy runtime and hibernate factory. You don't have to manage the CFGroovy runtime, but it is quite costly to spin up so you really should, unless you are just playing around.

Now let's create us some Groovy models...

I'm going to create a (intentionally) flawed model of Employers and Employees (very database table oriented). And then as we refine it you'll see the beauty of not having to worry about the relational model or having too many classes. So here are our Groovy model objects

Employee.groovy

```
package model

import model.*
import javax.persistence.*

@Entity
@Table(name="EMPLOYEES")
class Employee
{
 @Id
 @GeneratedValue
 Long employeeId

 String surname
 String givenName
 String mainPhone
 String mainFax
 String address
 String city
 String stateCode
 String zipCode

 @OneToMany(mappedBy="employee",cascade=[CascadeType.ALL])
 Set<EmploymentRelationship> employers

}
```

Employer.groovy

```
package model
import javax.persistence.*
import model.base.*

@Entity
@Table(name="EMPLOYERS")
class Employer
{
 @Id
 @GeneratedValue
 Long employerId

 String employerName
 String mainPhone
 String mainFax
 String address
 String city
 String stateCode
 String zipCode

 @OneToMany(mappedBy="employer",cascade=[CascadeType.ALL])
 Set<EmploymentRelationship> employees
}
```

EmploymentRelationship.groovy

```
package model import javax.persistence.*
import model.*
```

```
@Entity
@Table(name="EMPLOYEE_EMPLOYER")
class EmploymentRelationship
{
 @Id
 @GeneratedValue
 Long employmentId

 @ManyToOne(cascade = [ CascadeType.ALL ])
 @JoinColumn(name = "EmployeeId")
 Employee employee

 @ManyToOne(cascade = [ CascadeType.ALL ])
 @JoinColumn(name = "EmployerId")
 Employer employer

 String jobTitle
 Boolean isCurrent
 Date fromDate
 Date toDate
}
```

Now, let's see how easy it is to work with these in CF. But WAIT you say, what about our database don't we need to set up some tables or something? Nope, hibernate as configured with CFGroovy will create and update our database for us based on all those funky @nnotations. So here is some trivially simple CF code that will create and display our objects.

```
<cfimport prefix="g" taglib="/engine/tags" />

<cfscript>
  bf = application.beanFactory;
  local=structNew();
</cfscript>

<cfif not application.modelsCreated>

  Let's create some Groovy Models...<br>

  <g:script variables="#local#">
     import model.*
     variables.employer = new Employer(employees : new HashSet())
     variables.employee = new Employee(surname:'Cartman', givenName:'Eric')
     variables.employmentrecord = new EmploymentRelationship(
          employer:variables.employer,
          employee:variables.employee)      variables.employer.employees.add(variables.employmentrecord)
  </g:script>

  Now let's jump back to CF and manipulate them... <br>

  <cfscript>
    local.employer.employerName = 'Groovy Models Inc.';
    local.employee.setCity('South Park');
    request.hibernateSession.save(local.employmentrecord);
    application.modelsCreated = true;

    employeeArray = local.employer.employees.toArray();
```

```
  </cfscript>

<cfelse>

  Let's load our Groovy Models...<br>

  <cfquery datasource="#application.dsn#" name="local.q" maxrows="1">
    select employerId from EMPLOYERS
  </cfquery>

  <g:script variables="#local#">
    <cfoutput>
      import model.*
      variables.employer = request.hibernateSession.get(
            Employer.class,
            Long.parseLong('#local.q.employerId#'))
    </cfoutput>
  </g:script>

  <cfscript>
      employeeArray = local.employer.employees.toArray();
  </cfscript>
</cfif>

<cfoutput>
  We would suspect that
  #employeeArray[1].employee.surname# #employeeArray[1].employee.getGivenName()#
  works for
  #local.employer.employerName#<br>
  Let's do a query and see...<br>
</cfoutput>

  <cfquery datasource="#application.dsn#" name="seeTheyGotSaved">
    select employerName, surname, givenname
    from employers e1
      join employee_employer ee
        on e1.employerid = ee.employerId
      join employees e2
        on ee.employeeId = e2.employeeId
  </cfquery>

<cfoutput>
  And we find that
  #seeTheyGotSaved.givenname# #seeTheyGotSaved.surname#
  does indeed work for
  #seeTheyGotSaved.employerName#
</cfoutput>


<br>
  <cfdump expand="false" var="#local#">
<br>
```

Sweet! We just created our model, had our tables auto generated, saved some data, displayed it back, we were hopping between cf and groovy like there wasn't a difference! What more could you ask for?

Well as I am want to do, I have waxed on for far too long, I'll have to continue this in another post. We'll refine our fine young model to a more granular class structure and we'll map it to a more normalized db schema next time...