Relational Data Mapping with GORM

Fall Forecast 2009

CHARIOT SOLUTIONS

Agenda

- Creating an Application
- Grails Domain Classes
- Defining Constraints
- Defining Relationships
- The Grails Console

Creating a Grails App

- Issue command from the command line

```
grails create-app confreg
```

Grails Project Structure

- **grails-app** – where all grails application code resides
- **web-app** – Static content, CSS, Javascript, etc... Also Spring MVC and web.xml files
- **src** – place non-framework Java or Groovy code here
- **lib** – drop your Java library in this directory to add to the class path
- **test** – unit and integration JUnit tests live here

The grails-app Directory

- **conf** – configuration files
- **controllers** – Groovy MVC controllers
- **domain** – Domain classes
- **services** – Service classes
- **views** – GSP views and templates

Grails Domain Classes

- POGOs - Plain Old Groovy Objects
  - Each member variable represents a column in a table
  - Implied version, primary key fields
  - Can define relationships with other domains, validation constraints

```
grails create-domain-class com.chariot.customer
```

Example Domain Class

```
package com.chariot

class Customer {

   String firstName
   String lastName
   Date registrationDate
   Boolean preferredCustomer
   BigDecimal awardsBalance
}
```

- Grails creates a table with appropriate column names and types

```
mysql> describe customer;
+---------------------+---------------+------+-----+---------+----------------+
| Field               | Type          | Null | Key | Default | Extra          |
+---------------------+---------------+------+-----+---------+----------------+
| id                  | bigint(20)    | NO   | PRI | NULL    | auto_increment |
| version             | bigint(20)    | NO   |     | NULL    |                |
| awards_balance      | decimal(19,2) | NO   |     | NULL    |                |
| first_name          | varchar(255)  | NO   |     | NULL    |                |
| last_name           | varchar(255)  | NO   |     | NULL    |                |
| preferred_customer  | bit(1)        | NO   |     | NULL    |                |
| registration_date   | datetime      | NO   |     | NULL    |                |
+---------------------+---------------+------+-----+---------+----------------+
```

CHARIOT SOLUTIONS

Specifying Database Details

```
package com.chariot

class Customer {

    static mapping = {
        table “tb_customer”
        id column: “cust_id”
     }

    String firstName
    String lastName
    Date registrationDate
    Boolean preferredCustomer
    BigDecimal awardsBalance
}
```

CHARIOT SOLUTIONS

Configure DataSource

- Data is persisted based on the configuration file:

```
grails-app/conf/DataSource.groovy
```

Defining Constraints

- Added via the constraints closure
- Checked automatically when saved or when `validate()` is called

```
class Customer {

   static constraints = {
      firstName(maxLength: 15, blank:false)
      lastName(maxLength: 20, blank:false)
      registrationDate(nullable:false)
      preferredCustomer(default: true)
      awardsBalance(range: 0.0..5000)
   }

   String firstName
   String lastName
   Date registrationDate
   Boolean preferredCustomer
   BigDecimal awardsBalance
}
```

CHARIOT SOLUTIONS

Well-known Constraints: String

- **size: 2..5** // 2 to 5 characters
- **blank: false** // allow blank (not null) values
- **unique: true** // ensure uniqueness
- **matches: "[0-9]+"** // regular expression
- **email:true** // validate against email semantics
- **inList: ['One', 'Two', 'Three']** // validate against a list of values
- **notEqual: "Y"** // anything but 'Y'
- **creditCard: true** // matches if a valid credit card #

CHARIOT SOLUTIONS

Well-known Constraints: Numeric

- Fairly self explanatory
  - min: 234
  - max: 532423
  - range: 2..190034
  - nullable: true
  - scale: 2 // for decimal #s, how many digits to the right
  - Use max + scale or range + scale to define the size of a NUMERIC in the database

Date based constraints

- // allow null date
  nullable: true
- // Dates later than Jan 1, 2000
  min: new Date("01/01/2000")
- // no later than Jan 1, 2050
  max: new Date("01/01/2000")
- // Only in the 20th Century
  range: new Date("01/01/1900")..new Date("01/01/2000")

What is GORM?

- Grails Object Relational Model
- Common methods for Domain classes:
  - load, save, update, query, etc
- Backed by Hibernate
  - can use existing Hibernate mappings

GORM API – Saving the Domain

- Create a new row by 'newing' the instance, then calling save()
  - –You may call validate() to check for errors
  - –If validate fails, check errors for all error objects

Key GORM method calls

- **save()** - will persist new GORM instances or update new ones
- **delete()** – will delete the row associated with a Domain instance (by id)
- ***DomainClass*.get(id)** – fetches the domain object by the referred-to ID
- ***DomainClass*.count()** – Executes a select count (*) on the class and returns the number of rows
- ***DomainClass*.list()** – loads a list of all instances. Can be passed parameters.

CHARIOT SOLUTIONS

Testing in Grails Console

- Bootstraps Grails environment
- Creates tables if configured
- You can test your Domain classes against a live database

```
grails console

//(sample script)
import com.chariot.*

Customer c = new Customer(
    firstName: "Joe",
    lastName:"Musbecrazy",
    registrationDate: new Date("01/15/2009"),
    preferredCustomer: true,
    awardsBalance: 200.0)

if (c.validate()) {
        c.save()
} else {
        println c.errors
}
```

CHARIOT SOLUTIONS

Relating Domains

```
package com.chariot

class Customer {
  String firstName
  String lastName
  Date registrationDate
  Boolean preferredCustomer
  BigDecimal awardsBalance

  static hasMany = [courses: Course]

  static constraints = {
 ...
  }
}
```

```
package com.chariot

class Course {
   String courseName
   Date courseStartDate
   Integer numberOfDays
   Integer maxStudents

   static constraints = {
   }
}
```

CHARIOT SOLUTIONS

Defining Relationships

- One to One
- One to Many
- Many to Many
- Parent/Child (Inheritence)

One to Many Relationship

- Parent Side defines ref to child with hasMany
- Account owns
- Order relates back

```
class Account {
   String accountNumber

   static hasMany = [orders:Order]
}
```

```
class Order {

   // note 'order' is db keyword
   static mapping = { table 'orders' }
   static belongsTo =
[account:Account]

   Date orderDate
   BigDecimal amount

}
```

CHARIOT SOLUTIONS

Working with the Relationship

```
// create the 'one' side
def a = new Account(accountNumber:"234-234")
a.save()

// create a few orders
def orders = [
   new Order(orderDate: new Date("01/23/2009"),
                  amount: 234234.33),
   new Order(orderDate: new Date("03/15/2009"), amount:234.3) ]

a.orders = orders
a.save()

// print them out
a.orders.each {
  println "Order ${it.id} on ${it.orderDate} @ ${it.amount}"
}
```

CHARIOT SOLUTIONS

Introducing Dynamic Finders

- What if we want to see all orders where the customer spent more than $100?
- Use a dynamic finder
  - Call the finder with the following syntax:

    DomainClass.findBy*column*(*value*)

Dynamic Finder examples

```
// Given :
class Person {
    String state
    String lastName
}

// find all rows
def results = Person.findAll()

// find all who live in 'PA'
def results = Person.findAllByState("PA")

// find all who live in PA but also have a last name starting with Smi
def results = Person.findAllByStateAndLastNameLike("PA", "Smi%")

// finders have 'single row' versions that bring back the first row:
def result = Person.findByLastNameLike("Smi%")
```

CHARIOT SOLUTIONS

Summary

- Domains define relational content in terms of Groovy classes
- The Grails GORM framework wraps and simplifies Hibernate
- Validation is made possible via a series of constraints
- Mapping settings can be defined to adjust key physical model attributes and settings