Reading and Writing to MongoDB from Crafter Engine via Groovy

This article will provide a basic, PoC model that demonstrates how to create a data access object that reads and writes from MongoDB and then leverage that DAO from a service or a page/component controller

For background and basics on scripting Crafter Engine with Groovy read the following article.

Create a Data Access Object (DAO)

In this example we will create a data access object for a game leader board. There are two data operations, create a new leader board entry and get the sorted list of existing entries.

Basic Class Structure

In the example below you will note the following:

- We have declared the package we want our class to live in: "scripts.libs"
- We're using the Grapes GRAB capability to declare a dependency on MongoDB's drive and then to import the GMongo driver class
- We have a set of class wide global objects and a method to initialize them
- We have our 2 relevant CRUD operations WRITE and READ
- The name of the class file "MyDAO.groovy" is the same as the name of the class MyDAO
- Init assumes a MongoDB collection "leaders" exists in the MongoDB location specified in the driver configuration (assumes default ports / credentials if left unmodified)

How you construct and initialize your objects will depend on your use cases. In our example we're going to instantiate this class each time we need it. This may not be the most efficient way to connect but it keeps the code simple by removing the need to handle concurrency.

/SCRIPTS/lib/MyDAO.groovy

```
package scripts.libs

@Grab('com.gmongo:gmongo:1.2')
import com.gmongo.GMongo

class MyDAO {

    def mongo = null;
    def db = null

    /**
     * initialize our object
     */
    def init() {
        mongo = new GMongo()
        db = mongo.getDB('groovy-book')
    }

    /**
     * create a leaderboard record
     * @param name name of the player
     * @param moved moves made during the game
     * @param count number of cards in the game
     */
    def createLeader(name, moves, cardCount) {
       ...
    }

    /**
     * return sorted list of game leaders
     */
    def getLeaders() {
       ...
       return leaders;
    }
}
```

Method Implementations

Note that there is no need to create a db object, it is global to the class and was set up by INIT

createLeader

```
 def createLeader(name, moves, cardCount) {
    def pairs = cardCount / 2;
    def score = new Double((pairs / moves) * 100);
    db.scores.insert([name: name, moves: moves, cardCount: cardCount, score: score])
}
```

Note that we use the MongoDB FIND operation to pull all of the results and then iterate over each result to create the kind of object we want to return.

This code might be simplified by simply returning the result object but it's worth demonstrating that it's easy to transform or limit the scope of what a reader can see.

getLeaders

```
def getLeaders() {
       def leaders = [];
       def leaderResults = db.scores.find().sort([ score: -1 ]);

       def index = 0;
       leaderResults.each { record ->
          def leader = [:];
          leader.name = record.name;
          leader.moves = record.moves;
          leader.cardCount = record.cardCount;
          leader.score = record.score;
          leaders[index++] = leader;
       }

       return leaders;
    }
```

Using the DAO Object

This section will show how to use the DAO object once you have it in different contexts like a page controller or a RESTful service

RESTful Service

Note the following

- The script name create-leader.post.groovy indicates it's HTTP method
- The folder after /SCRIPTS/rest/... are arbitrary. You can organize your services however you like
- We import our DAO class (and others) with an import. Note that to import we simply say **import** and then specify both the package followed by the class name
- We instantiate the class with a **new** statement and then initialize it
- We parse incoming JSON in to values for the leader's name and score components and then pass those values to our DAO

/SCRIPTS/rest/game/leader/create-leader.post.groovy

```
import scripts.libs.MyDAO;
import scripts.libs.MyHTTP;
import groovy.json.JsonSlurper;

def dao = new MyDAO();
dao.init();
def status = [success:true, msg:""];

try {
    def postData = new MyHTTP().postToJSON(request);

    def name = postData.name;
    def moves = postData.moves;
    def pairs = postData.cardCount;

    dao.createLeader(name, moves, pairs);
}
catch(err) {
    status.msg = ""+err;
    success: false;
}

return status;
```

Page or Component Controller

The Controller

- We import our DAO class (and others) with an import. Note that to import we simply say **import** and then specify both the package followed by the class name
- We instantiate the class with a **new** statement and then initialize it
- We return the objects from the getLeaders call to the model so that the template can use it (via a variable named leaders)

/SCRIPTS/page/leaderboard.groovy

```
import scripts.libs.MyDAO;
def dao = new MyDAO();
dao.init();
model.leaders = dao.getLeaders();
```

The View

- The view simply renders a table of leaders by iterating over the **leaders** variable and creating/outputting a row for each one

/templates/web/leaderboard.ftl

```
<#include "/templates/system/common/cstudio-support.ftl" />
<!doctype html>
<html ng-app="memoryGameApp">
<head>
  <meta charset="utf-8">
  <title>Memory Game</title>
  <link rel="stylesheet" href="/static-assets/css/app.css">
</head>
<body>
   <h1>Leader Board</h1>
   <table border='1'>
       <tr>
         <th>Player Name</th>
         <th>Score</th>
       </tr>
     <#if leaders??>
         <#list leaders as leader>
           <tr>
             <td>${leader.name}</td>
             <td>${leader.score}</td>
           </tr>
         </#list>
      </#if>
   </table>

   <@cstudioOverlaySupport/>
</body>
</html>
```