RESTful and JSON remoting
Table of Contents

Introduction

1. Installation
   1.1 Linux setup
   1.2 Windows setup

2. Creating an app
   2.1 Testing remoting
   2.2 Step by step creation

3. Configuring JSON remoting
RESTful and JSON remoting
Installation
Installation

In this chapter you will install and test the Spring ROO meta-framework, either from a Linux or Windows box. For Mac OS X systems, follow the Linux instructions.

Linux setup

Prerequisites

Java 6
mvn 2.0.9 or higher

Install meta-framework

Install roo

```
$ wget http://spring-roo-repository.springsource.org/spring-roo-1.2.5.RELEASE.zip
$ unzip spring-roo-1.2.5.RELEASE.zip
$ export ROO_HOME=`pwd`/spring-roo-1.2.5.RELEASE
$ export PATH=$PATH:$ROO_HOME/bin
$ sudo ln -s $ROO_HOME/bin/roo.sh /usr/bin/roo
$ export ROO_OPTS="-Droo.bright=true"
$ export JAVA_OPTS="-Xms512m -Xmx1024m -XX:PermSize=64m -XX:MaxPermSize=1024m"
$ MAVEN_OPTS="-XX:MaxPermSize=1024m"
```

Test roo

Run from the ROO console:

```
$ mkdir hello
$ cd hello
$ roo
roo> hint
roo> project --topLevelPackage com.foo
roo> jpa setup --provider HIBERNATE --database HYPersonic_In_MEMORY
roo> entity jpa --class ~.Timer --testAutomatically
roo> field string --fieldName message --notNull
roo> hint web mvc
roo> web mvc setup
roo> web mvc all --package ~.web
roo> selenium test --controller ~.web.TimerController
roo> web gwt setup
roo> web gwt all --proxyPackage ~.client.proxy --requestPackage ~.client.request
roo> perform tests
roo> quit
```
Alternatively, download hello.roo and run:

```
$ mkdir hello
$ cd hello
roo> script --file hello.roo
roo> quit
```

Run the tomcat application server:

```
$ mvn tomcat:run
```

Test the application:

```
$ firefox http://localhost:8080/tenminutes
$ firefox http://localhost:8080/hello
```

## Windows setup

### Prerequisites

- Java 6
- mvn 2.0.9 or higher

### Install meta-framework

#### Install roo

Download Spring ROO and unzip in a local_folder:

```
http://spring-roo-repository.springsource.org/spring-roo-1.2.5.RELEASE.zip
```

Define environment variables:

```
> set ROO_HOME=<local_folder>/spring-roo-1.2.5.RELEASE
> set PATH=%PATH%;%ROO_HOME/bin%
> set JAVA_OPTS="-Xms512m -Xmx1024m -XX:PermSize=64m -XX:MaxPermSize=1024m"
> set MAVEN_OPTS="-Xmx512m -XX:MaxPermSize=1024m"
```

#### Test roo

Run from the ROO console:

```
> md hello
> cd hello
> roo
roo> hint
roo> project --topLevelPackage com.foo
roo> jpa setup --provider HIBERNATE --database HYPersonic_IN_MEMORY
```
roo> entity jpa --class ~.Timer --testAutomatically
roo> field string --fieldName message --notNull
roo> hint web mvc
roo> web mvc setup
roo> web mvc all --package ~.web
roo> selenium test --controller ~.web.TimerController
roo> web gwt setup
roo> web gwt all --proxyPackage ~.client.proxy --requestPackage ~.client.request
roo> perform tests
roo> quit

Alternatively, download hello.roo and run:

> mkdir hello
> cd hello
roo> script --sile hello.roo
roo> quit

Run the tomcat application server:

> mvn tomcat:run

Test the application:

- Open http://localhost:8080/tenminutes in a web browser
- Open http://localhost:8080/hello in a web browser
Creating an app
Creating an app

Application model

The application model follows the MVC architecture. The generated application to be generated by the ROO meta-framework will consist of a domain Model (M) and a web tier containing Views (V) and Controllers (C).

Class domain model

This is the class diagram of the domain model for the application we are going to build:

Web tier

The web tier usually consists of a number of controllers and a number of HTML views to have access to the controller functions from regular web browsers.

To know about creating the web tier: hint web mvc

Creating RESTful Spring MVC web controllers is quick and easy using Roo. Controllers can be made that automatically expose an entity. Alternately, we can create a stub, empty controller for you to finish off.

For the former, type 'web mvc setup' and hit ENTER followed by 'web mvc scaffold' and hit TAB three times. The --class is the controller name; it need not reflect an entity name. We suggest putting controllers under a '~.web' package to improve maintenance. You can also specify the --backingType the controller should expose.

Generating the full app

Firstly we will generate the complete PizzaShop application just to test it. Later we will partly re-generate the application step-by-step with ROO, only to have a remoting layer that consists of a set of JSON-based RESTful web services.

From the command line interface, run:

```
$ mkdir sample
$ cd sample
$ roo
roo> script --file pizzashop.boo
roo> quit
$ mvn tomcat:run
```

(If running in Windows, use similar commands)

Testing remoting
First install Advanced REST Client Chrome extension or use `curl` (for Unix-like systems) from the command line.

Using curl

If using curl, for commodity add a line containing `-w "\n"` to your `~/.curlrc` file

It is also useful installing `jq` command-line JSON processor to pretty-print the JSON output.

Example scripts with `curl`:

Run the following scripts from the command line:

**Creates (POST) a new Thin Crust pizza base:**


**Creates (POST) two new pizza bases (Cheesy Crust and Thick Crust):**


**Creates (POST) four new toppings: Fresh Tomato, Prawns, Mozarella (the z typo is deliberate) and Bogus:**


**Removes (DELETE) the Bogus (id=7) topping:**


**Retrieves (GET) the info about topping 6 (check self-generated version number):**


**Updates (PUT) the name and version values for Mozarella (id=6) topping (in JSON data, previous version number must be included; id=6 is not required here):**


**Retrieves (GET) the list of toppings:**
Retrieves (GET) the info about topping 6:


Creates (POST) a new *Napolitana* thin-crust pizza:

```sh
```

Creates (POST) a new pizza order:

This does not work :-?

```sh
curl -i -X POST -H "Content-Type: application/json" -H "Accept: application/json" -d '{"name": "Stefan", "total": 7.5, "address": "Sydney, AU", "deliveryDate": 1314595427866, "id": {"shopCountry": "AU", "shopCity": "Sydney", "shopName": "Pizza Pan 1"}, "pizzas": [{"id": 8, "version": 1}]}' http://localhost:8080/pizzashop/pizzaorders
```

Using a REST client

GET request: all toppings

![REST client screenshot](attachment:image.png)

**POST request: create new Pizza (Achtung! POST is not safe - idempotence)**
Step by step creation

Create project

Create a new project that uses JPA persistence using EclipseLink and H2. Notice that the H2 database is in memory, so you will lose all data after exiting the application.

From the roo console, run:

```
project --topLevelPackage com.springsource.pizzashop
jpa setup --provider ECLIPSELINK --database H2_IN_MEMORY
```

Create domain entities

Generate all classes needed to build the PizzaShop domain model:

```
entity jpa --class ~.domain.Base --activeRecord false --testAutomatically
  field string --fieldName name --sizeMin 2 --NotNull

entity jpa --class ~.domain.Topping --activeRecord false --testAutomatically
  field string --fieldName name --sizeMin 2 --NotNull
```
Define a repository layer for persistence

Generate a repository layer that manages the persistence. It is a **DataAccessObject** (DAO) alternative to the default **ActiveRecord** pattern to deal with persistence.
Configuring JSON remoting
Configuring JSON remoting

Define a service/façade layer:

Create a service layer that acts as a façade that exposes all the PizzaShop functionalities. This is required if you want to integrate a RESTful remoting web service. Other common use case for service layers is e.g. security.

```
service type --interface ~.service.ToppingService --entity ~.domain.Topping
service type --interface ~.service.BaseService --entity ~.domain.Base
service type --interface ~.service.PizzaService --entity ~.domain.Pizza
service type --interface ~.service.PizzaOrderService --entity ~.domain.PizzaOrder
```

Adding JSON remoting

Configure JSON-based MVC controllers for all domain types:

```
json all --deepSerialize
web mvc json setup
web mvc json all --package ~.web
```

Configure regular web-based MVC controllers for all domain types:

```
web mvc setup
web mvc all --package ~.web
```

Due to a still unsolved issue in Spring ROO, generating JSON controllers with `web mvc json setup` does not work anymor after running once `web mvc setup` to generate regular web controllers:

```
roo> web mvc json setup
Command 'web mvc json setup' was found but is not currently available
```

To solve that issue, the `@RooWebJson` tag can be added to the controller (e.g. PizzaController), and ROO will automatically generate the Aspect that adds the JSON implementation:

```
@RooWebJson(jsonObject = Pizza.class)
```

Testing

*Test again* the JSON remoting services against the newly created application

...and you're done!