How to start working with Jersey in grails

The following easy steps will help you to start with Jersey web service( RESTful ) using grails:

1. Install and configure grails. Please make sure to read http://grails.org/doc/latest/guide/gettingStarted.html
2. If you face any difficulty installing grails refer to my other tutorial (TechnicalInstruction-Grails\_V-1.0.pdf) or contact with me for further clarification. You must have basic grails knowledge to follow this guide.
3. Open command prompt and type "grails create-app TestApp-WS" to create an application using grails
4. Install jaxrs plug-in using "grails install-plugin jaxrs" , this will download the latest released version of the plugin from the [Grails Plugin Repository](http://www.grails.org/plugin/home). For further installation options, such as installing a development snapshot, refer to the [installation instructions](http://code.google.com/p/grails-jaxrs/wiki/InstallationInstructions).
5. **To create a JAX-RS resource named test enter**

grails create-resource test

This will create a TestResource.groovy file under grails-app/resources and a TestResourceTests.groovy file under test/unit. The TestResourceTests.groovy file is a unit test template. The TestResource.groovy file is the generated JAX-RS resource. Both files are in the testapp.ws package.

package testapp.ws  
  
import javax.ws.rs.GET  
import javax.ws.rs.Path  
import javax.ws.rs.Produces  
  
  
@Path('/api/test')  
class TestResource {  
  
    @GET  
    @Produces('text/plain')  
    String getTestRepresentation() {  
        'Test'  
    }  
      
}

It defines a single method that responds to HTTP GET operations. The HTTP response contains the return value of this method, Test in this example. The content type of the response (Content-Type header) is text/plain. The created resource is ready to use as shown in the next section.

Creating resources via the command line is only one option. An alternative is to create resource files by hand. Any \*Resource.groovy file created under grails-app/resources is assumed to be a JAX-RS resource and auto-detected by the grails-jaxrs plugin. These resources are checked for the presence of JAX-RS annotations as defined by JAX-RS 1.1 specification, section 3.1. Resources that aren't properly annotated are ignored by the plugin.

1. **Run the application**

To start the application enter

grails run-app

on the comamnd line. Then open a browser window and go to http://localhost:8080/TestApp-WS/api/test. The browser should now display "Test".

1. **Change the code**

The grails-jaxrs plugin also support code changes at runtime i.e. without restarting the server. To demonstrate that we let add a name parameter to the getTestRepresentation method and bind it to a name query parameter using the JAX-RS @QueryParam annotation. The HTTP response entity will vary depending on the name query parameter. Here's the modified source code.

package testapp.ws  
  
import javax.ws.rs.GET  
import javax.ws.rs.Path  
import javax.ws.rs.Produces  
import javax.ws.rs.QueryParam  
  
  
@Path('/api/test')  
class TestResource {  
  
    @GET  
    @Produces('text/plain')  
    String getTestRepresentation(@QueryParam('name') String name) {  
        "Hello ${name ? name : 'unknown'}"  
    }  
      
}

When you save the changes the plugin re-initializes the JAX-RS runtime. Go to http://localhost:8080/TestApp-WS/api/test?name=Afzal and you should see Hello Afzal in the browser window. If you additionally want to factor out the greeting logic into a Grails service, refer to the [service injection](http://code.google.com/p/grails-jaxrs/wiki/AdvancedFeatures#Service_injection) section for instructions.

## Generate WADL

**Available in version 0.4 or higher**. A WADL document for resources managed by the plugin can be generated by sending a GET request to http://localhost:8080/TestApp-WS/application.wadl. The result should look like

<application xmlns="http://research.sun.com/wadl/2006/10">  
  <doc xmlns:jersey="http://jersey.dev.java.net/" jersey:generatedBy="Jersey: 1.1.4.1 11/24/2009 01:30 AM"/>  
  <resources base="http://localhost:8080/ TestApp-WS/">  
    <resource path="/api/test">  
      <method name="GET" id="getTestRepresentation">  
        <request>  
          <param xmlns:xs="http://www.w3.org/2001/XMLSchema" type="xs:string" style="query" name="name"/>  
        </request>  
        <response>  
          <representation mediaType="text/plain"/>  
        </response>  
      </method>  
    </resource>  
  </resources>  
</application>

Generating WADL documents only works when the plugin is [configured](http://code.google.com/p/grails-jaxrs/wiki/AdvancedFeatures#JAX-RS_implementation) to use Jersey as JAX-RS implementation.

## Create a domain class

To create a Person domain class go to the project's root directory and enter

grails create-domain-class person

Open the generated Person.groovy file (under grails-app/domain) and add two properties, firstName and lastName.

package testapp.ws  
  
class Person {  
  
    static constraints = {  
    }  
      
    String firstName  
      
    String lastName  
      
}

1. **Generate the REST API**

To generate JAX-RS resources that implement the RESTful service interface for that domain class enter

grails generate-resources testapp.ws.Person

This will generate two resource classes, PersonCollectionResource.groovy and PersonResource.groovy (in the testapp.ws package) that support HTTP POST, GET, PUT and DELETE operations for creating, reading, updating and deleting Person objects, respectively. PersonCollectionResource.groovy is related to Person lists, PersonResource.groovy is related to individual Person instances. Let's take a look at how to use the generated RESTful service interface.

1. **Use the REST API**

Start the TestApp-WS application with

grails run-app

New person objects can be created by POSTing to http://localhost:8080/TestApp-WS/api/person. The following request POSTs an XML representation of a person object.

POST /TestApp-WS/api/person HTTP/1.1  
Content-Type: application/xml  
Accept: application/xml  
Host: localhost:8080  
Content-Length: 82  
  
<person>  
  <firstName>Afzalur</firstName>  
  <lastName>Rashid</lastName>  
</person>

The Content-Type header must be set either to application/xml. After sending the request, the server creates a new person object in the database and returns an XML representation of it.

HTTP/1.1 201 Created  
Content-Type: application/xml  
Location: http://localhost:8080/TestApp-WS/api/person/1  
Transfer-Encoding: chunked  
Server: Jetty(6.1.14)  
  
<?xml version="1.0" encoding="UTF-8"?>  
<person id="1">  
  <firstName>Afzalur</firstName>  
  <lastName>Rashid</lastName>  
</person>

The client explicitly requested an XML representation via the Accept request header. Note that the returned representation differs from the submitted representation by an id attribute in the <person> element. This id is also contained in the Location response header, the URL of the created resource. The response code is 201 (CREATED). Let's create another person object using a JSON representation. Here's the request

POST /TestApp-WS/api/person HTTP/1.1  
Content-Type: application/json  
Accept: application/json  
Host: localhost:8080  
Content-Length: 58  
  
{"class":"Person","firstName":"Fabien","lastName":"Barel"}

The response also contains a JSON representation of the created person (see Accept request header). The id of the created person object is 2.

HTTP/1.1 201 Created  
Content-Type: application/json  
Location: http://localhost:8080/TestApp-WS/api/person/2  
Transfer-Encoding: chunked  
Server: Jetty(6.1.14)  
  
{"class":"Person","id":"2","firstName":"Fabien","lastName":"Barel"}

Content negotiation via Content-Type and Accept headers works for other HTTP methods as well. To GET a list of created persons, open a browser (Firefox in our example) and enter the URL http://localhost:8080/TestApp-WS/api/person. This returns an XML representation of the list of persons stored in the database.

To learn more about this please visit

http://code.google.com/p/grails-jaxrs/wiki/AdvancedFeatures

http://code.google.com/p/grails-jaxrs/wiki/AdvancedFeatures#Using\_GORM

http://jersey.java.net/use/getting-started.html

http://docs.oracle.com/javaee/6/tutorial/doc/giepu.html