WebRock (A Web Services Framework)

Description

WebRock is a Web Services Framework that gives facilities to Programmers to avoid burden of writting thousands of Servlet classes for each module in their web application instead of which what a Backend Java Programmer can do is they can create a package structure inside classes folder of tomcat's WEB-INF Folder and specify package Prefix in tomcat's web.xml File with param-name as mentioned in the below example's A developer must use appropriate annotations (Some of them are class level,field,method and property) to use this framework properly which will be analyzed by our framework on server Startup.

Features

. Custom Annotations

. Overcomes the headache of writing monotonous Servlet Code.

. User can Use Scope(Container) provided by tomcat Server.

. Single JAR Availability So that you just have to include in your classpath and see Magic.

Installation

. Install latest JDK on your System which supports streams and lambda

. Download TMWebRock.jar from lib folder inside WEB-INF folder

. Download apache tomcat and place it on your C Drive’s root (ex : C:\tomcat9)

. Create a new folder named as lib in WEB-INF on your system

. Paste TMWebRock.jar inside lib folder

. Paste google gson inside lib folder from this repo’s lib folder

. Create a new file named as web.xml inside WEB-INF Folder on your system

. Copy and Paste web.xml contents from this repo to your local web.xml file

. Don’t Change any Class packaging, or class name in web.xml

. Create necessary folder structure and and necessary classes according to your requirement and need with necessary annotations included

User Manual

. Assume I have create a folder structure as com/demo/project inside classes folder

web.xml File Content

```
<?xml version="1.0" encoding="UTF-8"?>

<web-app version="2.5" xmlns="http://java.sun.com/xml/ns/javaee"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/webapp_

2_5.xsd">
```

```
<servlet>
   <servlet-name>TMWebRock</servlet-name>
   <servlet-
class>com.thinking.machines.webrock.TMWebRock</servlet-class>
  </servlet>
   <servlet-mapping>
   <servlet-name>TMWebRock</servlet-name>
   <url-pattern>/anything/*</url-pattern>
   </servlet-mapping>

<servlet>
   <servlet-name>TMWebRockStarter</servlet-name>
   <init-param>
   <param-name>SERVICE_PACKAGE_PREFIX</param-name>
   <param-value>com</param-value>
   </init-param>
   <load-on-startup>1</load-on-startup>
   <servlet-
class>com.thinking.machines.webrock.TMWebRockStarter</servlet-
class>
</servlet>
</web-app>
```

. Inside com/demo/project Create a Service Named as StockService to test functionalities of Get Path OnStartup and Forward Annotation

. Customize request URI service Path using Path annotation (Applicable for Both method and Class)

. For Get type request Use Get Annotation and vice versa for Post this annotation is applicable at both class and method level if no annotation is provided at class and at method then it will be applicable for Both get and post.dont conflict by applying post on class and get and method otherwise it will show method not allowed.

. Forward annotation forwards request to any static contents like jsp,css,html etc or any service url like /stock/service-path

. OnStartup(priority=num) specify that this service will be invoked on Server Startup based on priority number since there can be multiple services in Priority Queue

index.jsp file

```
<html>

<body>

<h1>Stocks Index JSP Page</h1>

<% out.print(2*5); %>

</body>

</html>
```

StockService.java File

```
package com.project.demo;
```

```
import com.thinking.machines.webrock.annotations.*; //import all
annotations
@Path("/stock")
public class StockService
{
@Path("/addStock")
@Get
@Forward("/index.jsp") // a file named as index.jsp is necessary
@OnStartup(priority=1)
public void addStock()
{
System.out.println("Add Stock Invoked");
}
@Path("/updateStock")
@Get
@Forward("/stock/addStock")
public void updateStock()
{
System.out.println("Update Stock Invoked");
}
}
```

//to compile this while staying in your services folder(demo folder) open command prompt and type

```
javac -classpath c:\tomcat9\lib;c:\tomcat9\webapps\yourproject\WEB-INF\lib\*; c:\tomcat9\lib;c:\tomcat9\webapps\yourproject\WEB-INF\classes ;. StockService.java
```

. Testcases For Various Scopes,AutoWired and Post Requests

➔ There are 3 class level annotations for Scopes which are as under

1) InjectApplicationScope (Used for storing & maintaining objects throughout the server lifecycle)
2) InjectRequestScope (Used for storing & managing objects for a particular request-response pair)
3) InjectSessionScope (Used for storing & maintaining objects through the Http session)
4) InjectApplicationDirectory (Used for storing & maintaining application level directory of server)

   When providing any of these annotations developer must have to declare a parameter for particular type and provide appropriate setters according to parameter name

➔ AutoWired(key) (Used to Initialize Objects values if present inside session or application or request scope)

Example : Let's Create a Pojo named as Employee.java

```
package com.project.demo;
public class Employee
{
```

```
private Integer code;
private String name;
public Employee()
{
this.code=0;
this.name=null;
}
public void setCode(Integer code)
{
this.code=code;
}
public Integer getCode()
{
return this.code;
}
public void setName(String name)
{
this.name=name;
}
public String getName()
{
return this.name;
}
}
```

EmployeeService.java

```
package com.project.demo;
import com.thinking.machines.webrock.annotations.*;
import com.thinking.machines.webrock.injections.*; //for scopes
```

```
@Path("/employee")
@InjectApplicationScope
@InjectRequestScope
public class EmployeeService
{
private ApplicationScope applicationScope;
public void setApplicationScope(ApplicationScope
applicationScope) //setter injection
{
this.applicationScope=applicationScope;
System.out.println("Application Scope invoked for employee
service "+applicationScope);
}
@AutoWired(name="xyz")
private Employee secondEmployee; //if no key with xyz name
exist in any of the scope then this will be null
@AutoWired(name="test")
private Employee employee;
@Path("/addEmployee")
@Get
@Forward("/member/addMember")
public void addEmployee()
{
System.out.println("Add Employee Invoked");
}
@Path("/getSecondEmployee")
@Get
public void getSecondEmployee()
{
Employee emp=new Employee();
```

```
emp.setCode(101);
emp.setName("Raju");
if(this.applicationScope.getAttribute("test")!=null)
{
Employee
e=(Employee)this.applicationScope.getAttribute("test");
System.out.println(e.getName());
}
this.applicationScope.setAttribute("test",emp);//putting in
application scope same for request for session
System.out.println("Get Second Employee Chala ");
}
@Path("/printSecondEmployee")
@Get
@Forward("/index.jsp")
public void printSecondEmployee()
{
System.out.println("Print Second Employee Invoked");
if(this.employee!=null)
System.out.println(this.employee.getCode()+","+this.employee.ge
tName());
}

@Path("/getThirdEmployee")
@Get
public void getThirdEmployee()
{
Employee emp=new Employee();
emp.setCode(102);
emp.setName("Ram");
```

```
        if(this.applicationScope.getAttribute("xyz")!=null)
        {
        Employee e=(Employee)this.applicationScope.getAttribute("xyz");
        System.out.println(e.getName());
        }
        this.applicationScope.setAttribute("xyz",emp);
        System.out.println("Get Third Employee Chala ");
        }
        @Path("/printThirdEmployee")
        @Get
        @Forward("/index.jsp")
        public void printThirdEmployee()
        {
        System.out.println("Print Third Employee Invoked");
        if(employee==null)
        {
        System.out.println("First if is null");
        }
        if(this.secondEmployee!=null)
        System.out.println(this.secondEmployee.getCode()+","+this.secon
        dEmployee.getName());
        }


}
```

//Compilation will be same as mentioned above for StockService.java

. Testcases For RequestParameter,PathVariable and InjectRequestParameter Annotation

- ➔ RequestParameter(key) Object any To get any key of specific type from query String in GET Request we can use this annotation
  Example : my request URI :
  localhost:8090/project/webservice/serviceClass/serviceMethod?code=101&name=Suresh

  So for that if I want this values directly in my parameter I can use this annotation like this

```
package com.demo.project.*;
import com.thinking.machines.webrock.*;
@Path("/serviceClass")
public class serviceClass
{
@Path("/serviceMethod")
@Get
public Object testForParams(@RequestParameter("code") Integer
code,@RequestParameter("name") String name)
{
System.out.println(code+""+name);
return new Object();
}
}
//compilation will be same as mentioned above

//so as per the above example code and name values will be
injected directly as a method parameters if they is improper this
values will be either null or default values of data type
```

➔ PathVariable annotation

If a developer wants to accumulate some data in requestURI then it can be done with the help of PathVariable annotation

Suppose my requestURI be like this
localhost:8090/project/webservice/serviceClass/serviceMethod/101/Mohan/1002/true

to get all four values like 101,mohan,1002 and true one must have to use PathVariable annotation at method parameter level

example :

```
package com.demo.project.*;
import com.thinking.machines.webrock.*;
@Path("/serviceClass")
public class serviceClass
{
@Path("/serviceMethod")
@Get
public Object testForParams(@PathVariable Integer
c1,@PathVariable String name,@PathVariable Integer
c2,@PathVariable Boolean flag)
{
System.out.println(c1); //it will print 101
System.out.println(name); //it will print mohan
System.out.println(c2); //it will print 102
System.out.println(flag); //it will print true
return new Object();
}
}
```

➔ InjectRequestParameter(key)
Whenever developer will use this annotation on class properties webrock will initialize that property value with whatever is arriving against the particular key in query string which is matching with annotation key ,condition is property type must be compatible with the value arriving in query string

Suppose my requestURI be like this
localhost:8090/project/webservice/serviceClass/serviceMethod?xyz=Saksham

example :

```
package com.demo.project.*;
import com.thinking.machines.webrock.*;
@Path("/serviceClass")
public class serviceClass
{
@InjectRequestParameters("xyz")
private String name;
@Path("/serviceMethod")
@Get
public Object testForInjectRequestParams()
{
System.out.println(name); //it will print Saksham
```

```
          return new Object();
         }
        }
```

. Testcases For Post Request

```
//pojo Class Student.java
package com.project.demo;
public class Student
{
private int rollNumber;
private String name;
//necessary setters and getters
}

// StudentService.java File
package com.project.demo;
import com.thinking.machines.webrock.annotations.*;
import com.thinking.machines.webrock.injections.*;
@Path("/studentService")
@Post
public class StudentService
```

```
{
@Path("/testStudent")
public Student testStudent(Student student)
{
System.out.println(student.getRollNumber()); // will print 101 (see
below example)
System.out.println(student.getName()); // will print Saksham (see
below example)
return student;
}
}
```

When user will hit this url and send Json in response Body from REST Tool

POST Request
localhost:8090/project/webservice/studentService/testStudent

Json:

```
{
"rollNumber":101
"name":"Saksham"
}
```

The above service will be invoked and will return student object in wrapper with success as true and this object will be arrived against the property named as message

```
{
Success:true,
isException:false,
message:{
“rollNumber:”101,
“name”:”Saksham”
}
}
```

So each time when a method will return something then the response of method will be wrapped inside this message property.