

Table of Contents

JVx already has [REST services](#) for action calls, data access and administration. Sometimes it's necessary to offer custom REST services for an application. Sure, you could create a simple action in your application and an action can be called with standard REST services, but this requires application authentication. If you have a different use-case for your REST service it's also possible to create your own REST service without using different libraries.

The easiest solution is to create a service class which is a servlet context listener. Why a listener? because we need an entry point to register our custom service. If you don't prefer a listener, it would also be possible to create a [server plugin](#) for JVx' server. But in this article, we use a servlet context listener.

Our class looks like:

```
package com.sibvisions.jvx;

import java.util.List;
import java.util.Map;

import javax.servlet.ServletContextEvent;
import javax.servlet.ServletContextListener;

import com.sibvisions.rad.server.config.Configuration;
import com.sibvisions.rad.server.config.Configuration.ApplicationListOption;
import com.sibvisions.rad.server.http.rest.service.AbstractCustomService;
import com.sibvisions.rad.server.http.rest.service.ICustomServiceGetDelegate;
import com.sibvisions.rad.server.http.rest.service.ICustomServicePostDelegate;
import com.sibvisions.rad.server.http.rest.service.UserService;
import com.sibvisions.util.OrderedHashtable;

@WebListener
public class CustomRESTServices implements ServletContextListener,
                                           ICustomServiceGetDelegate,
                                           ICustomServicePostDelegate
{
    /** the health check command. */
    private static final String CMD_HEALTHCHECK = "healthCheck";

    public void contextInitialized(ServletContextEvent pEvent)
    {
        UserService.register(getApplicationName(), CMD_HEALTHCHECK, this);
    }

    public void contextDestroyed(ServletContextEvent pEvent)
    {
        UserService.unregister(getApplicationName(), CMD_HEALTHCHECK, this);
    }

    public Object call(AbstractCustomService pService, String
```

```
pApplicationName, String pAction, String pParameter)throws Throwable
{
    //GET request

    OrderedHashtable<String, String> ohtResult = new
OrderedHashtable<String, String>();

    ohtResult.put("code", "SUCCESS");
    ohtResult.put("message", "GET is working!");

    return ohtResult;
}

public Object call(AbstractCustomService pService, String
pApplicationName, String pAction, String pParameter, Map<String, Object>
pInput) throws Throwable
{
    //POST request

    OrderedHashtable<String, Object> ohtResult = new
OrderedHashtable<String, Object>();

    ohtResult.put("code", "SUCCESS");
    ohtResult.put("message", "POST is working!");

    if (pInput != null)
    {
        //adds the input to the output
        ohtResult.putAll(pInput);
    }

    return ohtResult;
}

private String getApplicationName()
{
    List<String> list =
Configuration.listApplicationNames(ApplicationListOption.All);

    if (list.size() == 1)
    {
        return list.get(0);
    }
    else
    {
        return "demo";
    }
}
}
```

If you won't define the context listener as `@WebListener`, simply add:

```
<listener>
  <listener-class>com.sibvisions.jvx.CustomRESTServices</listener-class>
</listener>
```

to your deployment descriptor (web.xml).

To use your services, simply send GET or POST requests, e.g.

```
@Test
public void testGet() throws Exception
{
    ClientResource cres = new ClientResource(getBaseURL() +
    "_user/healthCheck");
    cres.get();

    HashMap<String, Object> hmpResult = (HashMap<String,
    Object>)JSONUtil.getObject(cres.getResponse().getEntity());

    Assert.assertEquals(200, cres.getStatus().getCode());
    Assert.assertEquals("SUCCESS", hmpResult.get("code"));
    Assert.assertEquals("GET is working!", hmpResult.get("message"));
    Assert.assertEquals(2, hmpResult.size());
}

@Test
public void testPost() throws Exception
{
    IBean bean = new Bean();
    bean.put("username", "@john.doe");
    bean.put("firstName", "John");
    bean.put("lastName", "Doe");

    ClientResource cres = new ClientResource(getBaseURL() +
    "_user/healthCheck");
    cres.post(bean);

    HashMap<String, Object> hmpResult = (HashMap<String,
    Object>)JSONUtil.getObject(cres.getResponse().getEntity());

    Assert.assertEquals(200, cres.getStatus().getCode());
    Assert.assertEquals("SUCCESS", hmpResult.get("code"));
    Assert.assertEquals("POST is working!", hmpResult.get("message"));
    Assert.assertEquals(5, hmpResult.size());
    Assert.assertEquals(bean.get("username"), hmpResult.get("username"));
}
```

The user-zone is per default `_zone` (e.g.

http://localhost/AppContext/services/rest/_user/healthCheck). If you want another zone name, simply add a parameter to the servlet mapping in deployment descriptor (web.xml):

```
<servlet>
```

```
<servlet-name>RestletServlet</servlet-name>
<servlet-class>com.sibvisions.rad.server.http.rest.RESTServlet</servlet-
class>

<init-param>
  <param-name>zone.user</param-name>
  <param-value>uzone</param-value>
</init-param>
</servlet>
```

The zone name is now uzone (e.g. <http://localhost/AppContext/services/rest/uzone/healthCheck>).

The complete source code is available [here](#).

In this document, we registered User services. It's also possible to register custom services as [Admin services](#). The difference is that it's possible to [en/disable admin services](#) by name.

From:
<http://doc.sibvisions.com/> - **Documentation**

Permanent link:
http://doc.sibvisions.com/jvx/communication/rest_customservice



Last update: **2022/11/22 10:06**