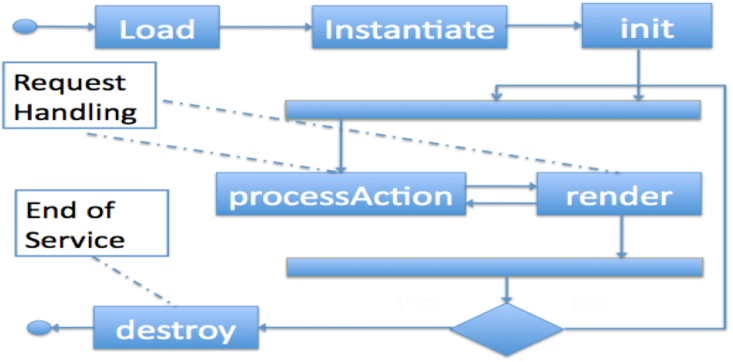
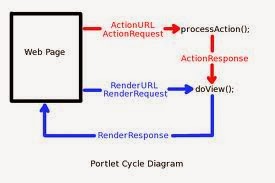
### PortletLifeCycle

init(PortletConfig config)  
  
            It  is called once, immediately after a new portlet instance is created. It can be used to perform startup tasks and is akin to a servlets init method. PortletConfig represents read-only configuration data, specified in a portlet's descriptor file, portlet.xml (more on this file later). For example, PortletConfig provides access to initialization parameters.  
  
  
processAction(ActionRequest request, ActionResponse response)  
  
              It is called in response to a user action such as clicking a hyperlink or submitting a form. In this method, a portlet may invoke business logic components, such as JavaBeans, to accomplish its goal. The ActionRequest and ActionResponse Interfaces are sub interfaces of PortletRequest and PortalRequest. In processAction, a portlet may modify its own state as well as persistent information about a portlet.  
  
  
render(RenderRequest request, RenderResponse response)  
  
         This method follows processAction in the chain of lifecycle methods. Render generates the markup that will be made accessible to the portal user. RenderRequest and RenderResponse methods, also sub interfaces of PortletRequest and PortletResponse, are available during the rendering of a portlet. The way in which the render method generates output may depend on the portlet's current state.  
  
destroy()  
  
        This method is the last lifecycle method, called just before a portlet is garbage collected and provides a last chance to free up portlet resources.

[](http://4.bp.blogspot.com/-AWP__D8ALY0/UwtqgeF2O2I/AAAAAAAACnM/6zYmLbMynu0/s1600/portlet+lifecycle.png)

[](http://3.bp.blogspot.com/-bQzk_vLLHYU/UwtqPo-Ie2I/AAAAAAAACnE/kQxiRvIFblM/s1600/flow+digram.jpg)

AddtionalLifeCycle methods  
  
An event is a life cycle operation that occurs before the rendering phase. Events can be described as a loosely coupled, brokered means of communication between portlets. Events allow portlets to respond on actions or state changes not directly related to an interaction of the user with the portlet.  
  
A portlet can declare events in its deployment descriptor by using the event-definition element in the portlet application section. In the portlet section, each portlet specifies the events it would like to publish through the supported-publishing-event element and the events it would like to process through the supported-processing-event element.  
  
The supported-publishing-event and supported-processing-event elements must reference the event name defined in the portlet application section in a event-definition element.  
  
The portlet creates events using the setEvent() method during action processing. This will be processed by the portlet container after the action processing has finished. Portlets can also create events during event phase by calling setEvent() method on EventResponse.  
  
To receive events, the portlet must implement the javax.Portlet.EventPortlet interface. The portlet container calls the processEvent() method for each event targeted to the portlet with an EventRequest and EventResponse object. The portlet can access the event that triggered the current process event call by using the EventRequest.getEvent() method. This method returns an object of type Event encapsulating the current event name and value.

[](http://2.bp.blogspot.com/-0haP0vEbN1c/Uwtq8ODDaCI/AAAAAAAACnU/4JyqROMJucg/s1600/Liferay_method_flow.png)

serveResource()  
  
The serveResource method can be used to implement Ajax use cases by invoking the resource URL through the XMLHttpRequest (or XMLPortletRequest) in client-side JavaScript code.

**……………………………….**

**Inter Portlet Communication**

Portlet  to  Portlet  Communication

Introduction

The first version of the portlet specification**, JSR-168/portlet1.0**, did not include any support for Inter Portlet Communication. The second version, JSR-286/ portlet2.0, which is supported for IPC  Mechanism.

IPC  is made easy  with JSR-286 to share the data between two portlets. Using IPC mechanisms, we can share the data from **ACTION** to **VIEW** phase and  **VIEW-VIEW** Phase.

There are 3  ways  to  share  the  data   between  2  portlets.

1. **Portlet session**
2. **IPC  Mechanisms**
   1. Public Render Parameters
   2. Event
   3. Client-Side IPC
3. **Cookies**

**1. Portlet  Session**

By default , Each war has its own session and will not be shared with other  wars. Liferay provides a mechanism by which Portlets can share session attributes across WARs.

A **PortletSession** is created for each user per portlet application. This makes the **PortletSession** useful for communicating all user related information among different portlets in the same portal application.

**Step 1:  set below attributes in Portlet1**

liferay-portlet.xml :

<portlet>

<private-session-attributes>false</private-session-attributes>

</portlet>

Step 2:  To set the Session:

PortletSession session = renderRequest.getPortletSession();

session.setAttribute("sessionValue",some-value , PortletSession.APPLICATION\_SCOPE);

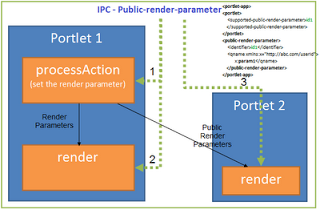
Step 3 : Get the Session Value in Portlet2

PortletSession ps = renderRequest.getPortletSession();

String tabNames = (String)ps.getAttribute("sessionValue ",ps.APPLICATION\_SCOPE);

**2. IPC Mechanism**

**2.1 Public Render Parameter** :   **IPC** ( Inter Portlet Communication)  :

[](http://4.bp.blogspot.com/-AO95eT72tt0/Tdo7MFfw2tI/AAAAAAAAACQ/gIiWP_Q0hzo/s1600/IPC.png)

In JSR 168, the render parameters set in ***processAction*** is only available in the ***render*** of the same portlet. With the **Public Render Parameters** feature, the **render parameters** set in the ***processAction*** of one portlet will be available in ***render*** of other portlets also.

By adding the following property in **portlet-ext**, we can enable portlets to share render states with other portlets that are on different pages:

**portlet.public.render.parameter.distribution**=ALL\_PORTLETS

**Step 1:  Add below attribute in “Sender-Portlet”**

**<portlet-app>**

**<portlet>**

            <supported-public-render-parameter>

**id1**

</supported-public-render-parameter>  
**</portlet>**

**<public-render-parameter>**

  <identifier>**id1**</identifier>

<qnamexmlns:x="http://abc.com/userId">x:**param1**</qname>

**</public-render-parameter>**

**</portlet-app>**

Note: We  can declare a list of **public parameters** for a portlet application.

**Step 2**:

 We can set render parameter in the **processAction**() method by using the defined public render parameter identifier as the key.

**response.setRenderParameter("id1", “someIdValue”);**

e.g.  
  
public void processAction(ActionRequest  request, ActionResponse response)  
  
throws IOException, PortletException  { ........

response.setRenderParameter("id1", “someIdValue”); ........  
  
}

**Step  3 : Receiver PortletPortlet  “portlet.xml”**

**Specify the render parameter the portlet would like to share in the portlet section.**

**<portlet-app>**  
<portlet>

<portlet-name >PortletB< /portlet-name >

< supported-public-render-parameter >**id1**< /supported-public-render-parameter >

< /portlet>

**<public-render-parameter>**

  <identifier>**id1**</identifier>

<qnamexmlns:x="http://abc.com/userId">x:**param1**</qname>

**</public-render-parameter>**

**</portlet-app>**

**Step 4** :

A portlet can read public render parameter using following method

**request.getPublicParameterMap()**

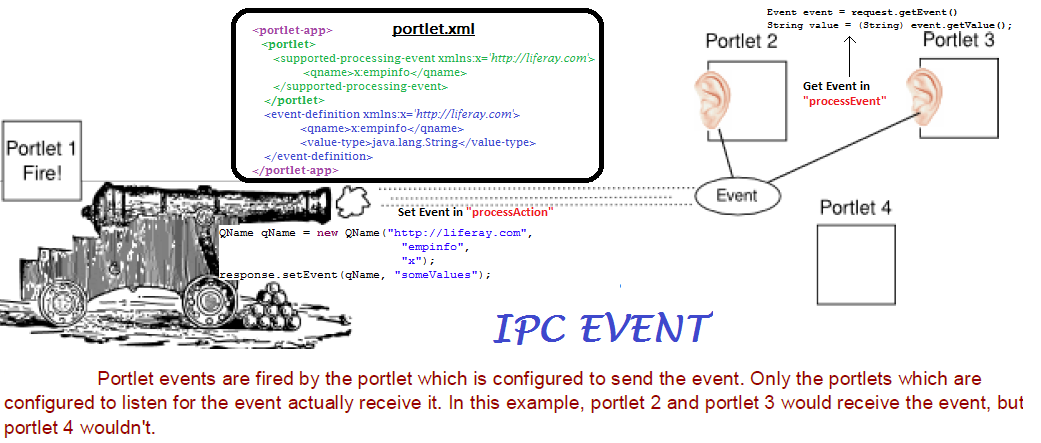
Note: Public render parameters are merged with regular parameters so can also be read using

**request.getParameter(“id1”);**

**Step 5:**

A portlet can remove a public render parameter by invoking following methods.  
  
response.removePublicRenderParameter(“**id1**”)

**3         Event  :   IPC ( Inter Portlet Communication)  :**

[](http://cdn.www.liferay.com/image/image_gallery?uuid=9cf047bd-d939-42f2-a477-af826b2fd0dd&groupId=14&t=1290961988382)

Portlet events that a portlet can receive and send.

**In JSR-168 :**  
The only way to achiveeventing was through portlet session.  
Limitation :Portlet has to be in the same web application.  
  
**In JSR-286 :**  
JSR 286 (Portlet 2.0) defines a lifecycle for events, so that eventing is possible between portlets that are in different web applications.

By adding the following property in portal-ext, we can enable portlets to send and receive events from other portlets that are on different pages

            portlet.event.distribution=ALL\_PORTLETS

**Step 1: Sender Portlet**

portlet.xml

-----------

The portlet standard defines a way of telling the portlet container  whichportlet is responsible for sending an event.

Add this inside **<portlet>** tag:

         <portlet-app>

<portlet>

                        <supported-publishing-eventxmlns:x=*'http://liferay.com'*>

<qname>x:empinfo</qname>

</supported-publishing-event>

</portlet>

<event-definitionxmlns:x=*'http://liferay.com'*>

<qname>x:empinfo</qname>

<value-type>java.lang.String</value-type>

</event-definition>

         </portlet-app>

**Step 3 : Set the event in process action:**

 javax.xml.namespace.QNameqName =

**new**QName("http://liferay.com", "empinfo", "x");

response.setEvent(

qName,

"HaiYou have received Event Data sent from Sender Portlet");

**Step 4: ListnerPortlet**

portlet.xml:

-----------

<portlet-app>

<portlet>

<supported-processing-eventxmlns:x=*'http://liferay.com'*>

                                    <qname>x:empinfo</qname>

                        </supported-processing-event>

</portlet>

<event-definitionxmlns:x=*'http://liferay.com'*>

<qname>x:empinfo</qname>

            <value-type>java.lang.String</value-type>

</event-definition>

</portlet-app>

**Step 5: get the EVENT:**

This Even will be called after **processAction** as shown in the picture:

**Lifecycle for IPC Event:**

**[](http://cdn.www.liferay.com/image/image_gallery?uuid=084d7a05-94b4-4125-bad8-572e95753d1e&groupId=14&t=1290962128191)**

@javax.portlet.ProcessEvent(qname = "{http://liferay.com}empinfo")

**publicvoid**handleProcessempinfoEvent(

javax.portlet.EventRequest request, javax.portlet.EventResponse response)

**throws**javax.portlet.PortletException, java.io.IOException {

        javax.portlet.Event event = request.getEvent();

        String value = (String) event.getValue();

            System.*out*.print("value in process event>>>>>>>>>" + value);

            response.setRenderParameter("empInfo", value);

 }

**2.3 Client-Side IPC  :**

There are 2 APIs for client side IPC.   
  
Event generation (call from portlet A):

Liferay.fire('<eventName>', {  
            name : value    
});  
  
**e.g.**  
Liferay.fire('planTravel', {  
            origin : 'pune',  
            destination : 'mumbai'  
        });

Event Listener ((call from portlet B):

Liferay.on('<eventName>', function(event) {  
          
    });  
  
**e.g.**  
Liferay.on('planTravel', function(event) {  
        showNews('', event.origin);  
        showNews('', event.destination);  
    });

**3.  Cookies**

Other than the **IPC mechanism,**  There is an easiest way to get the data between portlets on different pages called **COOKIES**.

But there are some limitations for cookies  that it will not accept more than 4KB size datas and the biggest limitation is, the 20 cookies per server limit, and so it is not a good idea to use a different cookie for each variable that has to be saved

**Portlet 1    :**

**To Set the Cookies through jQuery** :

<script src="/html/js/jquery/cookie.js" type="text/javascript" ></script>

functionsetCookie(docURL) {

jQuery.cookie("cookieParam",docURL);

}

**To Set the Cookies through java / jsp:**

HttpServletResponse response = PortalUtil.getHttpServletResponse(

                                    actionResponse);

Cookie cookieParam = new Cookie("cookieParam ", password);

response.addCookie(cookieParam);

**Portlet 2:**

**To get the Cookies through jQuery :**

jQuery.cookie("cookieParam ");

**To get the Cookie through java/ jsp :**

String sessionid = "";

Cookie[] cookies = request.getCookies();

    if (cookies != null) {

      for (int i = 0; i <cookies.length; i++) {

        if (cookies[i].getName().equals("cookieParam ")) {

          sessionid = cookies[i].getValue();

          break;

        }

      }

    }

# [JSR 286 vs JSR 168](http://techportals.blogspot.com/2008/12/jsr-286-vs-jsr-168.html)

|  |  |  |
| --- | --- | --- |
| **Features** | **JSR 168 Specification** | **JSR 286 Specification** |
| Inter Portlet Communication | * Only supported within the same portlet application using session attributes * Target portlets will only "see" messages during next render request. | Add additional coordination capabilities   * Limited only to String Values. * Sharing of session data beyond the current Portlet application. * Sharing of render parameters across portlets. |
| Life cycle | Portlets cannot update their state during a render request: "event" handling not really possible | New 3rd life cycle phase before rendering |
| Portlet Filters | Doesn't Support | * Supports Allow on the fly transformations of information in both the request to and the response from the portlet * Defined in portlet.xml |
| Caching |  | * Extended Cache support. * Allow public cached content for multiple users |
| Common Web Frameworks | * Servlet dispatching not supported from process Action. * Needs Portals Bridges or similar solutions. * JSTL support very limited | * Extended Cache support. * Allow public cached content for multiple users. * Improved support for web frameworks (Struts, JSF, Spring) Allow servlet dispatching during all lifecycle calls: processAction, processEvent, render, serverResource. * Extended JSP tag library <defineObjects/>, support for JSF |
| Non HTML Resources(pdf, doc, images etc.) | * A portlet can only render html fragments. * Have to fallback/delegate to the servlet container. * Requires coordination between portlet and servlet. |  |

**JSR 168 & 286 Specifications**

JSR 168 & 286 SPECIFICATIONS: These specifications are nothing but set of rules, which are used for the development of portlets and these are the members of the liferay portal technology.

DIFFERENCE BETWEEN JSR 168 & 286:

**JSR 168**:

     Portlets communication will be through portlet session, with in the same application.

     Doesn’t support the Ajax Implementation.

     Action URL & Render URL is given by the JSR 168.

**JSR 286**:

     Portlets communication will be through events and public render parameters. (Support IPC).

     Support the Ajax implementation, portlet filters & listeners.

     Resource URL is given by the JSR 286.

|  |  |  |
| --- | --- | --- |
|  | **JSR 168** | **JSR 286** |
| **1** | ActionURL | ActionURL |
|  | RenderURL | RenderURL |
|  |  | ResourceURL |
| **2** | Window States | Window States |
|  | (min) | (min) |
|  | (max) | (max) |
|  | (normal) | (normal) |
| **3** | Portlet Modes | Portlet Modes |
|  | (view) | (view) |
|  | (help) | (help) |
|  | (edit) | (edit) |
| **4** |  | IPC (Inter Portlet Communication) |

**URLs..**

1.  ActionURL

a.  <portlet:actionURL………/>

Always points to processActon() method.

Uses (ActionRequest, ActionResponse)

2.  RenderURL

a.  <portlet:renderURL………/>

Always points to render() method

Uses (RenderRequest, RenderResponse)

3.  ResourceURL

a.  <portlet:resourceURL………/>

Always points to serveResource() method.

Uses (ResourceRequest, ResourceResponse)

[Servlets VsPortlets](http://fanatech.wordpress.com/servlets-vs-portlets/)

**Similarities**

* Servlets and Portlets are web based components which use Java for their implementation.
* Portlets are managed by a portlet container just like servlet is managed by servlet container.
* Both static and dynamic content can be generated by Portlets and Servlets.
* The life cycle of portlets and servlets is controlled by the container
* The client/server model is used for both servlets and portlets
* The packaging and deployment are essentially the same, WAR/EARs.

**Dissimilarities**

* Servlets can render complete web pages, whereas portlets renders html fragments. These fragments are aggregated by the portal into a complete web page.
* The content type of JSR 168 portlets can be only cHTML, XHTML, WML. It does not support other content types.
* Portlets are not allowed to generate HTML code that contains tags such as body, frame, frameset, head, html, or title.
* A Portlet unlike a servlet doesn’t have URL attached to it so it cannot be accessed directly. Access is only through the portal page which holds the portlet.
* Portlets can be provided with controls to manipulate its window states or portlet modes.
* Multiple instances of a single portlet can be placed onto the same page.
* Portlets support persistent configuration and customization, profile information.
* Portlets can have two types of request viz. render request and action request.
* Portlets have two scopes within session; application scope for communication across portlets and portlet scope for intra portlet communication.
* Portlet cannot set the character set encoding of the response nor can it set the HTTP response headers.
* Portlets doesn’t have access to request URL. So it cannot access the query parameters appended to the URL.
* Portlets cannot set cookies.
* Typical methods of Portlet API are doView(), doEdit(), doHelp() and processAction() while those of servlet are doService(), doPost(), doGet().

### Similarities and Differences between Portlets and Servlets..

This post is about to understand the differences and similarities of portlet and servlet.

**The differences between the portlet and servlet are:**  
1. Servlets can provide the content to complete web pages, whereas portlets only provide fragments. And these fragments are then aggregated to form a complete web page by the portal.  
2. The user cannot access a portlet directly using a URL in the way that a servlet is accessed. Instead, the URL points to the page containing all of the portlets on single page.  
3. Portlets aren’t allowed to generated HTML code that contains tags such as base, body, frame, frameset, head, title or html . Note: Here the iframe tag can be used with caution.  
4. The communication between the web client and the portlets is performed through the portal.  
5. Portlet allows to manipulate the portlets’ window states or portlet modes by providing the buttons or controls.

6. Portlets support two scopes within the session; application scope and portlet scope but servlets not.  
  
**The similarities between the servlet and portlet:**  
1. Servlets and portlets are web based components that utilize Java for their implementation.  
2. Portlets are managed by a portlet container similar to a servlet container.  
3. Both of these components generate content, which can be static or dynamic.  
4. Both portlets and servlets have a lifecycle that is controlled by the container.  
5. The client/server model is used for both servlets and portlets.  
6. The packaging and deployment are essentially the same.  
7. The manner in which the classes are loaded and the class loaders that perform the work are also the same.  
8. Lifecycle management is similar.  
9. The Request and Response semantics are also similar.

### Creating Filter in custom portlet

Here we will do how access the portlet filter. These portlet filter which introduced in JSR-286.  
  
We will go through the how to create filter in customportlet.  
  
**STEP:1**  
  
Here i am create the portlet filter when an action takes place i.e whenever there is an Action Request.  
Create the class the "**ActionJSONFilter**" which implements "**ActionFilter**"

public class **ActionJSONFilter**implements **ActionFilter**

Also we can implement the **RenderFilter**also **ResourceFilter**. All these are in the **javax.portlet.filter.\*;** package.

public class ActionJSONFilter implements ActionFilter {  
public void destroy() {  
System.out.println("destroy");  
}  
public void init(FilterConfig arg0) throws PortletException {  
System.out.println("init");  
}  
public void doFilter(ActionRequest arg0, ActionResponse arg1,  
FilterChain arg2) throws IOException, PortletException {  
System.out.println("doFilter");  
}  
}

**STEP:2**  
 Define the below entries in the **portlet.xml ,** Here we need to define the filter class and lifecycle and portletname.  
<filter>  
 <filter-name>Action</filter-name>  
 <filter-class>com.test.filter.json.ActionJSONFilter</filter-class>  
 <lifecycle>ACTION\_PHASE</lifecycle>  
</filter>  
<filter-mapping>  
  <filter-name>Action</filter-name>  
  <portlet-name>JSON</portlet-name>  
</filter-mapping>

Deploy the portlet .we will observe the SOP of init() during the deployment.  
Whenever we perform the Actonrequest we can able to see the SOP of doFilter().

Hooks are used to override/extends liferay core functionalities.   
Hooks are hot-deployable.   
Hooks are created within liferay-plug-in-sdk/hooks folder.   
Hooks are preferred over Ext-plugins whenever possible.   
-------------------------------------------------------------------------  
Uses of Hook:   
Performing custom actions on portal startup or user login   
Overriding or extend portal JSPs   
Modifying portal properties   
Replacing portal services with your own implementation   
----------------------------------------------------------------------------   
Example :  
Some custom hook available for download   
[By Liferay Community](http://www.liferay.com/downloads/liferay-portal/community-plugins/-/software_catalog/products?_98_tabs1TabsScroll=&_98_keywords=&_98_type=hook&_98_itemsPerPage=20&_98_page=1)   
[By Liferay Official :](http://www.liferay.com/downloads/liferay-portal/official-plugins/-/software_catalog/products?_98_tabs1TabsScroll=&_98_keywords=&_98_type=hook&_98_itemsPerPage=20&_98_page=1)  
[From Liferay Source :](https://github.com/liferay/liferay-plugins/tree/6eb0c53bfb128dd07daf1fa430d815a1be2a008f/hooks)  
----------------------------------------------------------------------------   
Creating Hooks :   
Hooks are created in plugin-sdk/hook folder.   
Open plugin-sdk/hook folder from command prompt.(Recommended to use drop-to-dos utility.)   
Write the command :  
  
create .bat myfirsthook “My First Hook”  
----------------------------------------------------------------------------   
Command Prompt Console :  
  
D:\liferay-plugins-sdk-6.1\hooks>create.bat myfirsthook "My First Hook"   
Unable to locate tools.jar. Expected to find it in C:\Program Files\Java\jre6\lib\tools.jar   
Buildfile: D: \liferay-plugins-sdk-6.1\hooks\build.xml create:   
[copy] Copying 3 files to D: \liferay-plugins-sdk-6.1\hooks\myfirsthook-hook   
BUILD SUCCESSFUL  
Total time: 1 second  
  
--------------------------------------------------------------------   
  
Now Plugin-sdk/hook folder has new myfirshthook-hook available.  
  
Plugin-SDK append –hook to name of hook automatically.   
----------------------------------------------------------------------------   
  
Deploy myfirsthook-hook :  
Open folder plugin-sdk/hooks/myfirsthook-hook using command prompt.   
Enter command "ant deploy"  
  
command Prompt Console :  
-----------------------------------------------------------------------------    
D:\liferay-plugins-sdk-6.1\hooks\myfirsthook-hook>ant deploy   
  
Unable to locate tools.jar. Expected to find it in C:\Program Files\Java\jre6\lib\tools.jar  
Buildfile: D:\Liferay Portal 6.1 CE Plugin SDK\liferay-plugins-sdk-6.1\hooks\myfirsthook-hook\build.xml compile: merge: [mkdir] Created dir: D:\liferay-plugins-sdk-6.1\hooks\myfirsthook-hook\docroot\WEB-INF\classes   
  
[mkdir] Created dir: D: \liferay-plugins-sdk-6.1\hooks\myfirsthook-hook\docroot\WEB-INF\lib   
  
merge: war:   
  
clean-portal -dependencies:   
  
[zip] Building zip: D: \liferay-plugins-sdk-6.1\dist\myfirsthook-hook-6.1.0.1.war deploy: [copy] Copying 1 file to D:\Liferay Portal 6.1 CE tomcat Bundle\liferay-portal-6.1.0-ce-ga1\deploy BUILD SUCCESSFUL   
  
Total time: 1 second  
----------------------------------------------------------------------------  
  
Once Hook get deployed successfully tomcat-console show following messages :   
  
ñ 07:42:25,575 INFO [AutoDeployDir:167] Processing myfirsthook-hook-6.1.0.1.war   
  
ñ 07:42:25,598 INFO [HookAutoDeployListener:43] Copying hook plugin for   
  
ñ D:\Liferay Portal 6.1 CE tomcat Bundle\liferay-portal-6.1.0-ce-ga1\deploy\myfirsthook-hook 6.1.0.1.war   
  
ñ Expanding: D:\Liferay Portal 6.1 CE tomcat Bundle\liferay-portal-6.1.0-ce ga1\deploy\myfirsthook-hook-6.1.0.1.war into D:\Liferay Portal 6.1 CE tomcat Bundle\liferay-portal-6.1.0-ce-ga1\tomcat-7.0.23\temp\20120703074225674   
  
ñ Copying 1 file to D: \Liferay Portal 6.1 CE tomcat Bundle\liferay-portal-6.1.0-ce-ga1\tomcat-7.0.23\temp\20120703074225674\WEB-INF  
  
ñ Copying 1 file to D:\Liferay Portal 6.1 CE tomcat Bundle\liferay-portal-6.1.0-ce-ga1\tomcat-7.0.23\temp\20120703074225674\WEB-INF\classes Copying 1 file to D: \Liferay Portal 6.1 CE tomcat Bundle\liferay-portal-6.1.0-ce-ga1\tomcat-7.0.23\temp\20120703074225674\WEB-INF\classes   
  
ñ Copying 1 file to D: \Liferay Portal 6.1 CE tomcat Bundle\liferay-portal-6.1.0-ce-ga1\tomcat-7.0.23\temp\20120703074225674\WEB-INF   
  
ñ Copying 1 file to D: \Liferay Portal 6.1 CE tomcat Bundle\liferay-portal-6.1.0-ce-ga1\tomcat-7.0.23\temp\20120703074225674\META-INF   
  
ñ Warning: WEB-INF\liferay-plugin-package.properties modified in the future.   
  
ñ Copying 10 files to D: \Liferay Portal 6.1 CE tomcat Bundle\liferay-portal-6.1.0-ce-ga1\tomcat-7.0.23\webapps\myfirsthook-hook   
  
ñ Copying 1 file to D: \Liferay Portal 6.1 CE tomcat Bundle\liferay-portal-6.1.0-ce-ga1\tomcat-7.0.23\webapps\myfirsthook-hook   
  
ñ Deleting directory D: \Liferay Portal 6.1 CE tomcat Bundle\liferay-portal-6.1.0-ce-ga1\tomcat-7.0.23\temp\20120703074225674 07:42:28,298 INFO [HookAutoDeployListener:49] Hook for D: \Liferay Portal 6.1 CE tomcat Bundle\liferay-portal-6.1.0-ce-ga1\deploy\myfirsthook-hook-6.1.0.1.war copied successfully.   
  
ñ Deployment will start in a few seconds. Jul 3, 2012 7:42:35 AM org.apache.catalina.startup.HostConfigdeployDirectory  
  
ñ INFO: Deploying web application directory D:\Liferay Portal 6.1 CE tomcat Bundle\liferay-portal-6.1.0-ce-ga1\tomcat-7.0.23\webapps\myfirsthook-hook 07:42:36,261 INFO [PluginPackageUtil:1099] Reading plugin package for myfirsthook-hook   
  
ñ 07:42:36,337 INFO [HookHotDeployListener:1901] Registering hook for myfirsthook-hook 07:42:36,339 INFO [HookHotDeployListener:978] Hook for myfirsthook-hook is available for use  
----------------------------------------------------------------------------   
  
What is happing during deployment :   
ant clean deploy   
  
- Clean the source files  
- Compiles updated files  
- Create a war file of Hook  
- Copy war file into plugin-sdk/dist folder for distribution purpose.  
- Copy war file into tomcat/deploy folder of liferay.

HookHotDeployListener  
  
- It detects war file for hook in tomcat/deploy folder.  
- It extracts war file.  
- It removes /update old hook .  
- It deploys hook.  
  
Deployed Hook is get available within liferay-portal-6.1.0-ce-ga1\tomcat-7.0.23\webapps.   
This hook is just empty hook not overriding/extending anything in liferay.   
----------------------------------------------------------------------------     
How to undeploy hook.   
There is no ant command is available to remove hook.   
Remove it manually by just deleting it from liferay-portal-6.1.0-ce-ga1\tomcat-7.0.23\webapps.   
Hook will get unregister.   
Overriding /extending functionality will get removed.   
----------------------------------------------------------------------------      
Summary :  
What are Hooks in Liferay ?   
Hooks creation   
Hooks deployment   
Hooks updeploymet

For anyone who may be wondering how to create custom role types (sub-types to be exact). This may be used to search for role types if you have a large number of roles.  
  
Here are the steps:  
  
1) add the following to portal-ext.properties:

# Specify subtypes of roles if you want to be able to search for roles   
# using your custom criteria.   
#

roles.community.subtypes=  
roles.organization.subtypes=   
roles.regular.subtypes=subtype1,subtype2,subtype3