**Navigation (Trong 1 Portlet) Và Xử Lý Sự Kiện**

Người viết: Nguyễn Thị Bích Thủy

**Mục Lục**

[1. Navigation (trong 1 portlet) 2](#_Toc293838222)

[1.1. Static navigation 5](#_Toc293838223)

[1.2. Dynamic navigation 5](#_Toc293838224)

[2. Xử lý sự kiện (thay đổi giá trị, action event, event listener) 6](#_Toc293838225)

[2.1. Giới thiệu 6](#_Toc293838226)

[2.2. Action event 6](#_Toc293838227)

[2.2.1. Sử dụng thuộc tính “actionListener” của các thẻ. 6](#_Toc293838228)

[2.2.2. Sử dụng thẻ <f:actionListener> 7](#_Toc293838229)

[2.3. Thay đổi giá trị 7](#_Toc293838230)

[2.4. Event Listener 8](#_Toc293838231)

[2.5. Phase Events 8](#_Toc293838232)

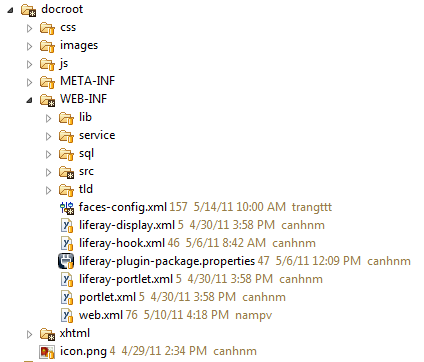
1. Navigation (trong 1 portlet)

Navigation trong một portlet được dùng để chuyển trang trong cùng một portlet, được cấu hình trong file face-config.xml.

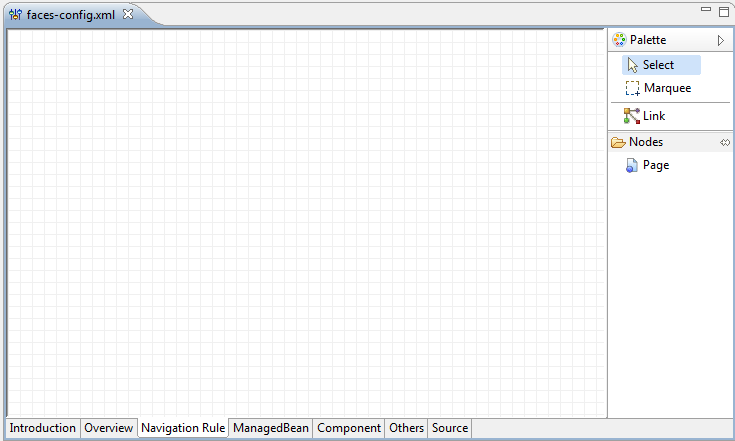
Các bước thực hiện tạo outcome cho navigation trong portlet.

**Khai báo trên giao diện**

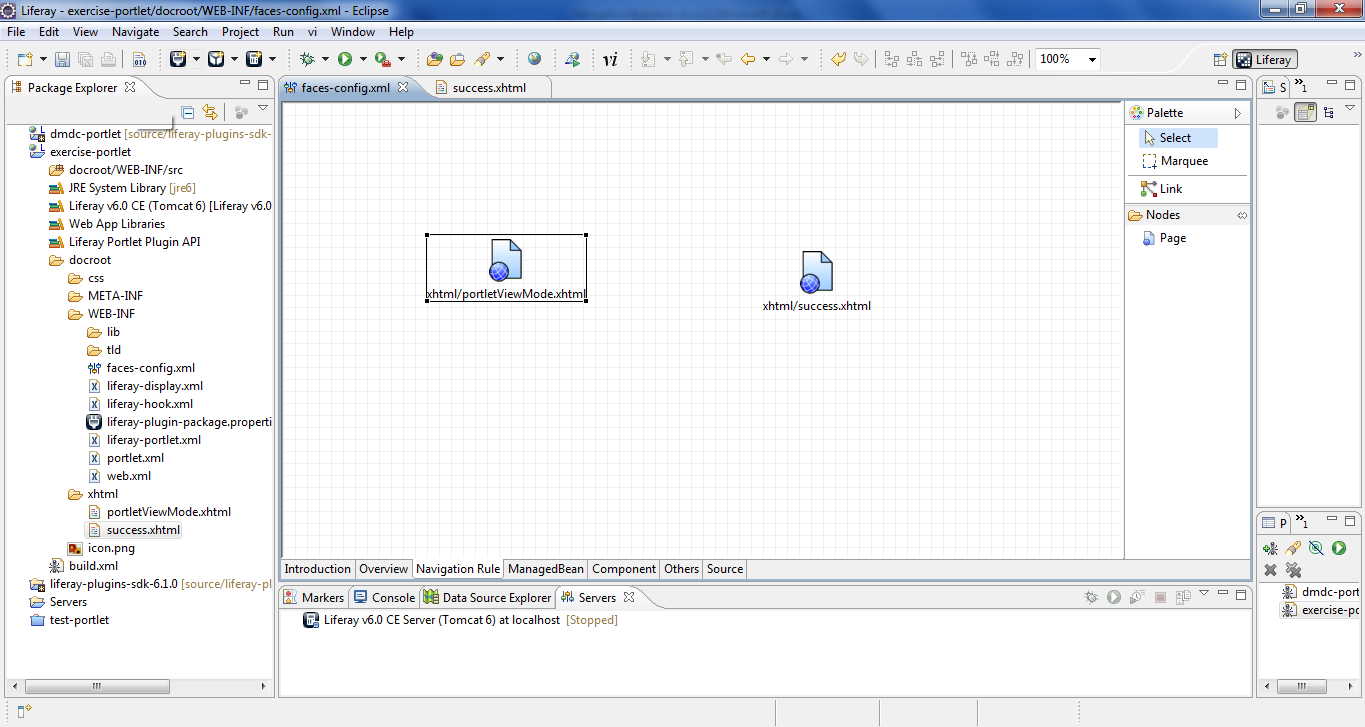
* Bước 1: mở file **face-config.xml** trong thư mục /docroot/WEB-INF/face-config.xml.



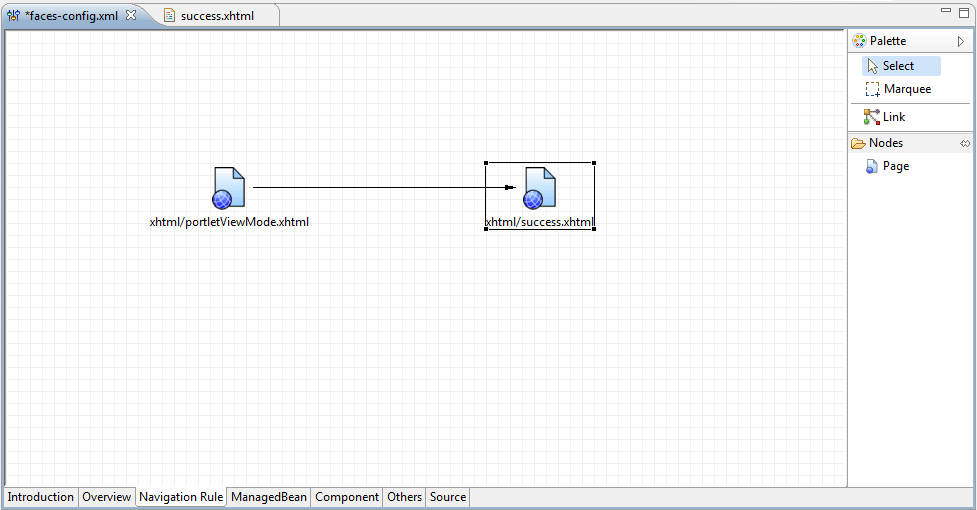
* Bước 2: Chọn tab **Navigation Rule** trong file face-config.xml



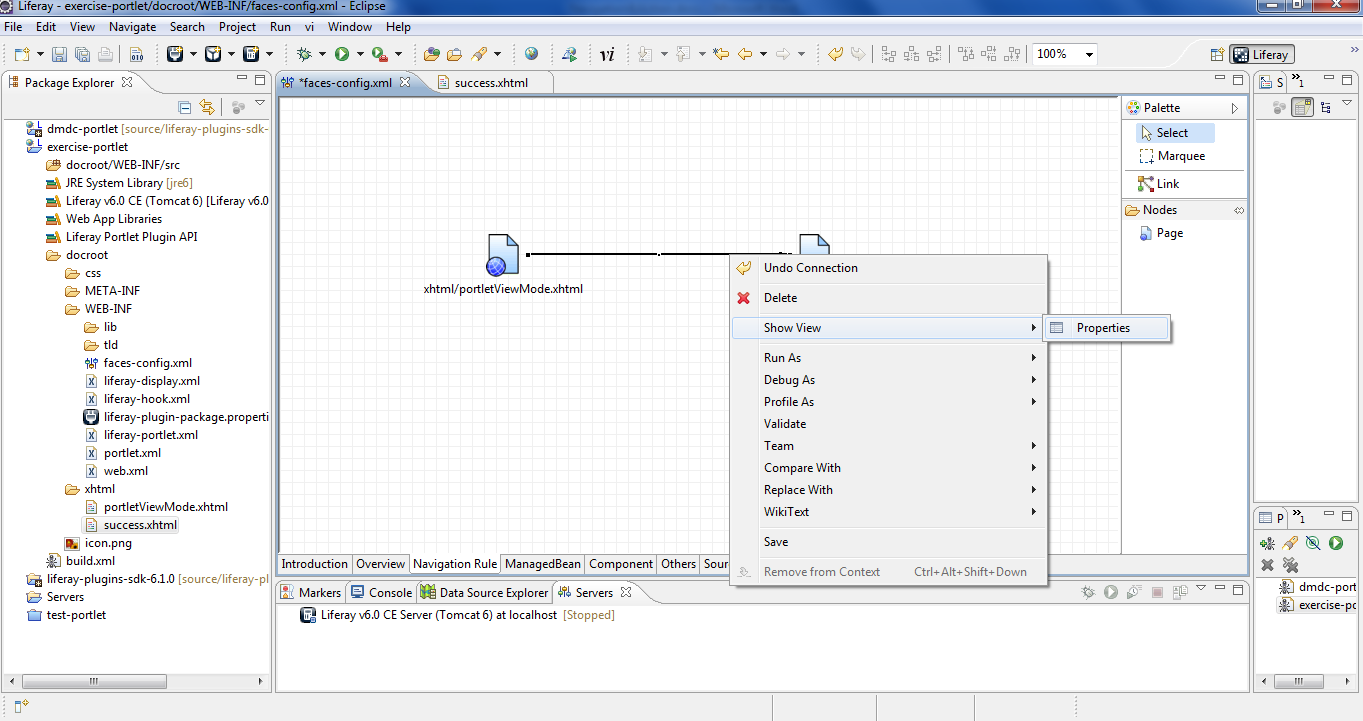
* Bước 3: Đưa file .xhtml cần chuyển trang vào trong giao diện “Navigation Rule” (kéo file .xhtml vào giao diện).



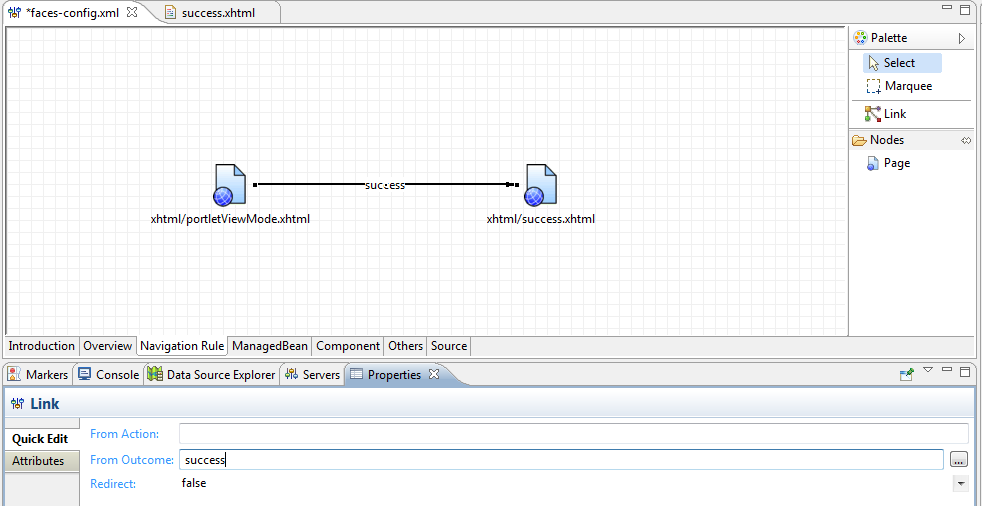
* Bước 4: Chọn biểu tượng Link trong thanh menu bên phải để tạo link chuyển trang.



* Bước 5: Tạo outcome, click chuột vào link chọn phải chuột, chọn Show view 🡪 Properties.



* Bước 6: Nhập giá trị cho outcome trong giao diện properties.



* From Action: định nghĩa action gọi phương thức từ lớp bean.
* From outcome: định nghĩa outcome để chuyển trang
* Redirect: có 2 giá trị “true” or “false”. Nếu “true” thực hiện chuyển trang trực tiếp, nếu là “false” thì thực hiện chuyển trang và gửi biến.

**Khai báo dưới dạng xml**

Tạo outcome để chuyển trang từ index.xhtml đến trang success.xhtml, bạn khai báo trong file face-config.xml

Page chuyển đến

Định nghĩa outcome

Page cần chuyển

|  |
| --- |
| <navigation-rule>  <from-view-id>/index.xhtml</from-view-id>  <navigation-case>  <from-outcome>success</from-outcome>  <to-view-id>/success.xhtml</to-view-id>  </navigation-case>  </navigation-rule> |

* Định nghĩa action cho trong file xml ta khai báo thêm

Lớp bean

Phương thức xử lý trong bean

<from-action>#{bean.method}</from-action>

* 1. Static navigation

Outcome chuyển trang từ index.xhtml🡪success.xhtml

Từ outcome vừa tạo, thực hiện chuyển trang trực tiếp trên html

1.9.PNG

* 1. Dynamic navigation

Hầu hết ứng dụng web thì thường sử dụng dynamic navigation. Luồng của trang không phụ thuộc vào nút nào mà bạn nhấn mà còn tùy vào input mà bạn cung cấp.

Ví dụ: submit một trang login có hai outcome: success và failure.

* Nếu username và password không chính xác thì sẽ thực hiện outcome “failure”.
* Nếu username và password chính xác thì sẽ thực hiện outcome “success”.

Để cài đặt dynamic navigation, nút submit phải có một phương thức tham chiếu:  
    <ice:commandButton label="Login" action="#{loginController.verifyUser}"/>

Một phương thức tham chiếu trong thuộc tính action không có tham số và trả về kiểu String. Ví dụ phương thức verifyUser nên như thế này:  
        String verifyUser() {  
               if (...)  
                  return "success";  
               else  
                  return "failure";  
        }  
  
Phương thức trả về một chuỗi outcome như "success" hoặc "failure". Navigation handler sử dụng chuỗi trả về để tìm navigation rule phù hợp.

1. Xử lý sự kiện (thay đổi giá trị, action event, event listener)
   1. Giới thiệu

Sự kiện trong JSF bao gồm 4 loại chính:

* Action Events
* Value Change Events
* Event Listener
* Phase Events
  1. Action event

Action event thường được dùng cho các đối tượng giao diện như: command button hoặc là một hyper-link. Khi người dùng nhấn chuột vào button hoặc click vào một hyper-link để tạo ra các sự kiện. Có thể gắn Listener vào một tài nguyên.

* + 1. Sử dụng thuộc tính “actionListener” của các thẻ.

Trang jsp

|  |
| --- |
| <ice:commandButton id="cb1" value="#{ActionBean.buttonText}" immediate="true"  **actionListener="#{ActionBean.processMyAction}"**/> |

Class bean:

|  |
| --- |
| /\*Action listener method\*/  public void processMyAction(ActionEvent event){       disable = !disable;      System.out.println("disable: "+ disable);       if(buttonText.equals("Disable")){            buttonText = "Enable";             System.out.println("buttonText: "+ buttonText);      }      else{            buttonText = "Disable";            System.out.println("buttonText: "+ buttonText);      } } |

* + 1. Sử dụng thẻ <f:actionListener>

Các interface ActionListener được thực thi để thực hiện các sự kiện. Interface sẽ cung cấp phương thức processAction() được gọi khi ActionEvent thực thi ActionListener đã được đăng ký.

Code trên jsp

|  |
| --- |
| <ice:commandLink id="France" immediate="true" value="Set Locale(France)">             **<f:actionListener type="roseindia.MyActionBean" />** </ice:commandLink> |

Code lớp bean:

|  |
| --- |
| package roseindia;  import java.util.\*; import javax.faces.event.\*; import javax.faces.context.FacesContext;  public class MyActionBean extends Object implements **ActionListener** {       **public void processAction(ActionEvent event) throws AbortProcessingException**{                       Map locales = null;              locales = new HashMap();              locales.put("America", new Locale("en", "US"));               locales.put("France", new Locale("fr", "FR"));               String current = event.getComponent().getId();              FacesContext context = FacesContext.getCurrentInstance();              context.getViewRoot().setLocale((Locale) locales.get(current));      } } |

* 1. Thay đổi giá trị

Sự kiện thay đổi giá trị thường được dùng cho các thành phần như: Text Field, Check-Box, List và RadioButton. Sự kiện thay đổi giá trị được thực hiện ngay sau khi giá trị của các thành phần được chỉnh sửa. Litener được gắn vào sự kiện thay đổi giá trị thường được sử dụng để gắn với phần validator để kiểm tra đầu vào giá trị nhập mới được chấp nhận hay không.

Ví dụ: kiểm tra giá trị đầu vào của một Combobox

Code jsp:

|  |
| --- |
| <ice:selectOneMenu value="#{addressBean.country}"  valueChangeListener="#{addressBean.populatePinNumber}">  </ice:selectOneMenu> |

Lưu ý việc sử dụng các thuộc tính 'valueChangeListener. Trong đoạn mã trên, bất cứ khi nào giá trị của tên quốc gia được thay đổi, thì giá trị PIN cho tên nước là tự động dân cư trong lĩnh vực văn bản có liên quan sẽ thay đổi.

Code java:

|  |
| --- |
| package net.javabeat.articles.jsf.events.actions;  import javax.faces.event.AbortProcessingException;  import javax.faces.event.ValueChangeEvent;  import javax.faces.event.ValueChangeListener;  public class CountryValueChangeListener{  public CountryValueChangeListener(){}  public void populatePinNumber(ValueChangeEvent event){  Object oldValue = event.getOldValue();  Object newValue = event.getNewValue();  if (oldValue != newValue){  // Get the country name and populate the pin value.  String countryName = (String)newValue;  }  }  } |

* 1. Event Listener

Khi người dùng click chuột vào button hoặc link, thay đổi giá trị của field, hoặc lựa chọn thành phần trong một combobox, ứng dụng có những thay đổi khi sự kiện này xảy ra, sự kiện được xử lý bằng mã ứng dụng đã được đăng ký khi có sự thay đổi các thành phần trên giao diện sẽ hiển thị thông báo cho sự kiện.

Hai event listener phổ biến nhất là action và thay đổi giá trị. Listener có thể được khởi tạo trong một lớp bean. Đặt Listener trong một class riêng biệt cho phép tái sử dụng trên nhiều trang. Nếu tạo ra một class Listener thì phải thực thi interface ActionListener hoặc ValueChangeListener.

* 1. Phase Events

Như chúng ta đều biết life-cycle của việc xử lí REQUEST trong JSF bao gồm 6 giai đoạn và bất kì việc thực hiện JSF nào cũng sẽ thực hiện giai đoạn EVENTS trong suốt từ lúc bắt đầu đến lúc kết thúc mỗi giai đoạn. Nếu chúng ta muốn nắm bắt được sự kiện trong từng giai đoạn, chúng ta phả định nghĩa lớp PHASE LISTENER như sau:

Code java :

CustomPhaseListener.java

|  |
| --- |
| package net.javabeat.articles.jsf.events.actions;  import javax.faces.event.PhaseEvent;  import javax.faces.event.PhaseId;  import javax.faces.event.PhaseListener;  public class CustomPhaseListener implements PhaseListener {    public CustomPhaseListener() {}  public void afterPhase(PhaseEvent event) {  System.out.println("After Phase->" + event.getPhaseId());  }  public void beforePhase(PhaseEvent event) {  System.out.println("Before Phase->" + event.getPhaseId());  }  public PhaseId getPhaseId() {  return PhaseId.ANY\_PHASE;  }  } |

Đê thực hiện custom lại sự kiện trong giai đoạn cần một class implement interface PhaseListener. Phương thức afterPhase()sẽ được gọi khi kết thúc một giai đoạn. Phương thức beforePhase() được gọi trước khi bắt đầu một giai đoạn. Để đăng ký sự kiện trên vào ứng dụng của chúng ta, cần định nghĩa trong file face-config.xml nội dung sau:

|  |
| --- |
| <lifecycle>  <phase-listener>  net.javabeat.articles.jsf.events.actions.TestActionListener  </phase-listener>  </lifecycle> |

Bạn có thể đăng ký nhiều sự kiện ***Phase Listeners*** trong ứng dụng.

|  |
| --- |
| <lifecycle>  <phase-listener>PhaseListener1</phase-listener>  <phase-listener>PhaseListener2</phase-listener>  </lifecycle> |

Để lấy tất cả các Phase Listener được định nghĩa trong file face-config.xml ta thực hiện code như sau:

|  |
| --- |
| public void listAllPhaseListeners() {  LifecycleFactory lifecycleFactory = (LifecycleFactory) FactoryFinder.getFactory(FactoryFinder.LIFECYCLE\_FACTORY);  Lifecycle applicationLifecycle = lifecycleFactory.getLifecycle(LifecycleFactory.DEFAULT\_LIFECYCLE);  PhaseListener phaseListeners[] = applicationLifecycle.getPhaseListeners();  for (PhaseListener phaseListener : phaseListeners) {  System.out.println(phaseListener.getPhaseId());  }  } |