# Component – Bean – Action

Người viết: Tô Thị Thuỳ Trang

[1. Định nghĩa 1](#_Toc294171081)

[1.1. Component 1](#_Toc294171082)

[1.2. Bean 1](#_Toc294171083)

[1.2.1. Định nghĩa Bean: 1](#_Toc294171084)

[1.2.2. Các thuộc tính của bean 2](#_Toc294171085)

[1.2.3. Biểu thức gọi bean 2](#_Toc294171086)

[1.2.4. Backing bean 2](#_Toc294171087)

[1.2.5. CDI Beans (CDI – Context and Dependency Injection) 2](#_Toc294171088)

[1.2.6. Bean Scopes 3](#_Toc294171089)

[1.2.7. Cấu hình bean 3](#_Toc294171090)

[1.3. Action 6](#_Toc294171091)

[2. Vòng đời JSF Request: 6](#_Toc294171092)

[3. Vòng đời ICEface request 9](#_Toc294171093)

[4. Ví dụ: 9](#_Toc294171094)

1. Định nghĩa
2. Component

* Các component dùng để hiển thị lên trình duyệt của client do ICEfaces cung cấp.

1. Bean
2. Định nghĩa Bean:

Bean là lớp Java chứa các thuộc tính cần thiết trên view tương ứng. có các hàm getter và setter của các thuộc tính đó và đồng thời chứa các phương thức xử lý các action trên view.

Trong ngữ cảnh JSF thì bean lưu trữ trạng thái của các trang web. Bean được tạo ra thao tác dưới sự quản lý cùa JSF implementation. JSF implementation thực hiện các công việc sau:

* Tạo và huỷ bean khi cần thiết.
* Đọc các thuộc tính của bean khi hiển thị trang web.
* Thay đổi các thuộc tính của bean khi form gửi dữ liệu.

1. Các thuộc tính của bean

Bất kỳ thuộc tính nào trong bean cũng cần được khai báo như sau:

* Tên
* Kiểu dữ liệu
* Phương thức getter và setter của thuộc tính.

Ngoài ra trong lớp bean có thể chứa các phương thức khác như các hàm lấy dữ liệu dưới database, data conversion validation, v.v.

Bên cạnh đó lớp bean phải tạo hàm constructor rỗng.

1. Biểu thức gọi bean

Có thể gọi các lớp bean trên view bằng biểu thức:

Ví dụ: #{user.name} hay như sau <h:inputText value="#{user.name}"/>

1. Backing bean

**Định nghĩa**: backing bean là lớp bean mà có chứa một vài hoặc toàn bộ các đối tượng component của một web form.

Lớp bean này được dùng khi cần validator hoặc xử lý các sự kiện mà cần phải truy suất đến component thực của web form.

Khi sử dụng bakcing bean , cần phải khi báo thuộc tính “binding” cho component. Ví dụ như: <h:inputText binding="#{quizForm.answerComponent}" .../>

1. CDI Beans (CDI – Context and Dependency Injection)

**Định nghĩa:** CDI bean là những bean được bao bởi context như là request hiện tại, session trình duyệt, hoặc sự kiện của chu kỳ ngữ cảnh định nghĩa bởi người dùng.

CDI xác định cụ thể khi nào cần inject bean, ngăn chặn hoặc trang trí các phương thức được gọi hoặc là huỷ hoặc quan sát các sự kiện.

Khi sử dụng CDI bean cũng khai báo giống cách JSF managed bean. Tuy nhiên khai báo nó với từ khoá anotation @Named như sau:

@Named("user")

@SessionScoped

public class UserBean implements Serializable {

...

}

Sau khi khai báo, biểu thức gọi bean này cũng tương tự JSF managed bean như là #{user} hoặc là #{user.name}.

**Lưu ý:**  Cần phải tạo file WEB-INF/beans.xml để kích hoạt CDI bean xử lý.

1. Bean Scopes

Khi bean được định nghĩa, thì cần phải chỉ rõ scopecua3 bean.

Có 3 loại scope phổ biến cho JSF và CDI bean:

* Session scope
* Request scope
* Application scope

JSF 2.0 có thêm 2 scope là view scope và custom scope. Hai scope này thì không được hỗ trợ trong CDI, nhưng mà được dùng như scope chuyển đổi.

Có thể định nghĩa các scope trong bean bằng anotaion:

* @SessionScoped
* @RequestScoped
* @ApplicationScoped

Và lưu ý rằng: Nếu dùng JSF managed bean thì sử dụng các lớp anotation trong gói javax.faces.bean, và trong gói javax.enterprise.context cho CDI bean.

**Session scope:** Browser gửi một yêu cầu đến server, server trả về một response và sau đó browser hoặc server không có bất kỳ nhiệm vụ nào phải giữ lại bất kỳ bộ nhớ của transaction. Ví dụ: Thêm các mặt hàng vào giỏ hàng trong online shopping.

**Request scope:** Tồn tại ngắn, nó bắt đầu khi một HTTP request được submit và kết thúc khi response được gửi trở lại cho client.

**Application scope:** Application scopelưu lại toàn bộ khoảng thời gian mà ứng dụng web tồn tại. Scope đó được chia sẽ giữa tất cả các request và tất cả session.

Khi khai báo anotation cho application scope lưu ý phải khai báo như sau: @ManagedBean(eager=true).

1. Cấu hình bean

Có 2 cách cấu hình bean:

**Cấu hình bằng anotaion:**

* Cách inject CDI bean: cách này thường dùng khi các bean có quan hệ với nhau, chứa lẫn nhau. Ví dụ như, bean EditBean thì chứa UserBean thì cấu hình như sau:

*@Named*

*@SessionScoped*

*public class EditBean {*

*@Inject private UserBean currentUser;*

*...*

*}*

Dùng @Inject đặt trước trường bean hoặc có thể khai báo trong hàm setter hoặc constructor.

* Cách inject managed bean: Dùng anotaion @ManagedProperty để inject bean. Ví dụ khai báo như sau:

*@ManagedBean*

*@SessionScoped*

*public class EditBean implements Serializable {*

*@ManagedProperty(value="#{user}")*

*private UserBean currentUser;*

*public void setCurrentUser(UserBean newValue) { currentUser = newValue; }*

*. . .}*

**Lưu ý**: Phải tạo hàm setter cho bean được inject

Dưới đây là bảng liệt kê scope của các thuộc tính không nhỏ hơn phạm vi cho phép.

|  |  |
| --- | --- |
| Bean scope | Properties scope |
| None | None |
| Application | None, application |
| Session | None, application, session |
| View | None, application, session, view |
| Request | None, application, session, view, request |

* Anotaion chu kỳ vòng đời của bean: dùng anotation @PostConstruct và @PreDestroy, xác định các phương thức được gọi trước khi hoặc sau khi gọi lớp bean. Cấu hình như ví dụ sau:

public class MyBean {

@PostConstruct

public void initialize() {

// initialization code

}

@PreDestroy

public void shutdown() {

// shutdown code

}

// other bean methods

}

Anotaion @PostConstruct thường dùng để lấy dữ liệu để hiển thị lên trang web hoặc thiết lặp kết nối đến database.

Anotaion @PreDestroy và @PostDestruct có thể cấu hình trong JSF managed bean và CDI bean.

**Cấu hình bean bằng file xml**

Tên file cấu hình là: WEB-INF/face-congif.xml.

* Định nghĩa bean: Sử dụng thành phần *managed-bean*, như ví dụ sau:

*<faces-config>*

*<managed-bean>*

*<managed-bean-name>user</managed-bean-name>*

*<managed-bean-class>com.corejsf.UserBean</managed-bean-class>*

*<managed-bean-scope>session</managed-bean-scope>*

*</managed-bean>*

*</faces-config>*

Các loại scope có thể là: *request, view, session, application, none hoặc biểu thức (#{expr})* cho custom scope.

Đối với application scope bean cần khai báo thêm thuộc tính “eager” như:

*<managed-bean eager="true">*

* Thiết đặt các giá trị thuộc tính

Chúng ta bắt đầu bằng ví dụ sau, ở đây chỉ điều chỉnh thể hiện của UserBean:

*<managed-bean>*

*<managed-bean-name>user</managed-bean-name>*

*<managed-bean-class>com.corejsf.UserBean</managed-bean-class>*

*<managed-bean-scope>session</managed-bean-scope>*

*<managed-property>*

*<property-name>name</property-name>*

*<value>me</value>*

*</managed-property>*

*<managed-property>*

*<property-name>password</property-name>*

*<value>secret</value>*

*</managed-property>*

*</managed-bean>*

Khi bean user lần đầu tiên được tìm kiếm, nó được khởi tạo bằng phương thức khởi tạo mặc định UserBean(). Sau đó các phương thức setName và setPassword sẽ được thực thi.

Để khởi tạo một thuộc tính bằng null, hãy sử dụng element null-value.

Ví dụ:

*<managed-property>*

*<property-name>password</property-name>*

*<null-value/>*

*</managed-property>*

1. Action

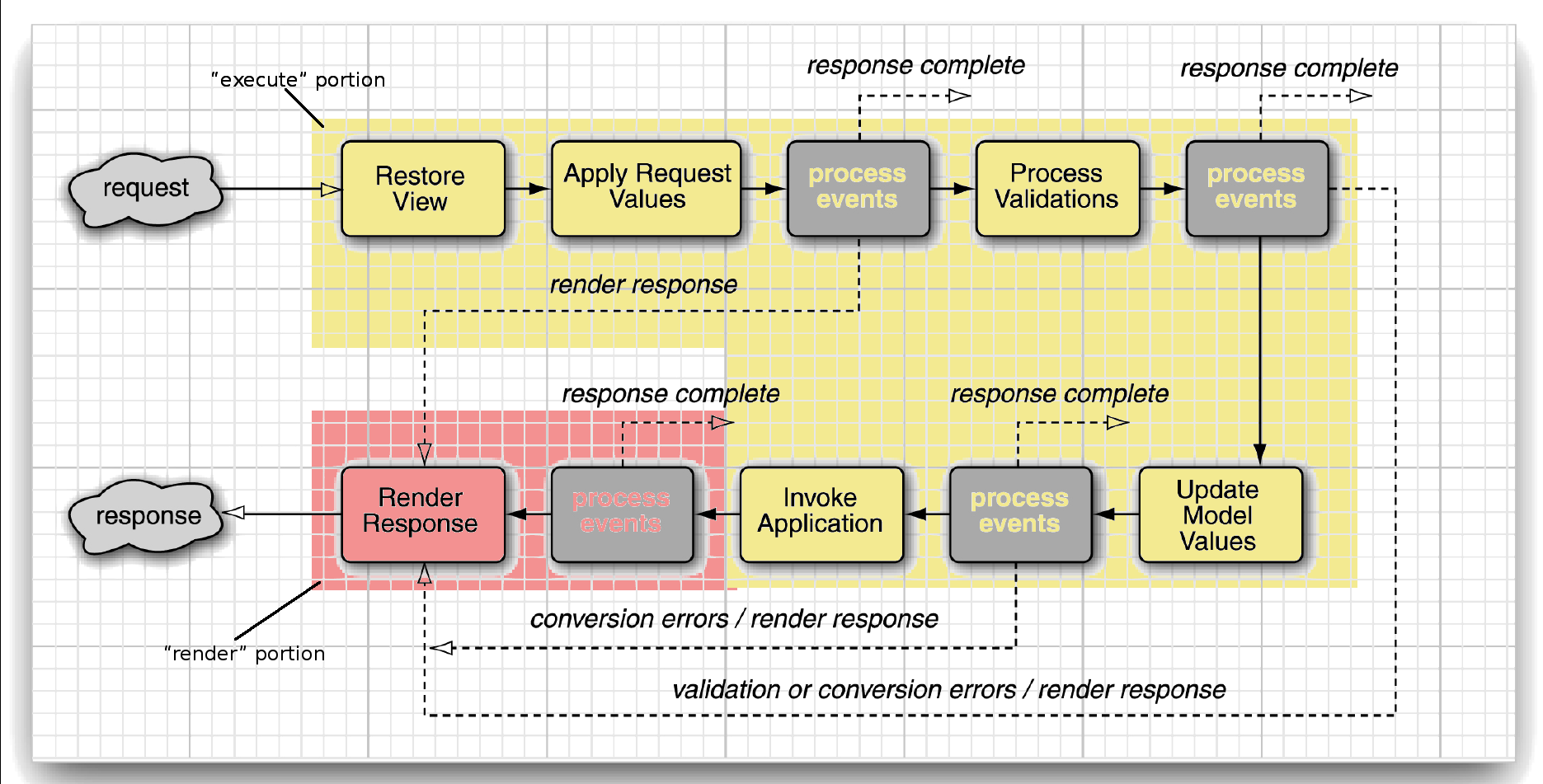
* Action chính là những phương thức xử lý trong bean được component gọi khi cần thực hiện một tác vụ nào đó.

1. Vòng đời JSF Request:

Trong phần này sẽ trình bày về vai trò của component, bean, action, và các xử lý của chúng trong 1 chu kỳ của một request.

1. Vòng đời xử lý của một request

Để xử lý một request JSF xử lý qua 6 bước:



* Giai đoạn 1: **Restore View**:
* Khi có một yêu cầu đi qua FacesServlet. Servlet xem xét yêu cầu và trích ra mã nhận dạng ID của khung nhìn, được xác định bởi tên của trang JSP.
* JSF framework controller sử dụng mã nhận dạng khung nhìn để tìm kiếm các thành phần dùng cho khung nhìn hiện tại. Nếu khung nhìn còn chưa tồn tại, trình điều khiển JSF tạo ra nó. Nếu khung nhìn đã tồn tại, trình điều khiển JSF sử dụng nó. Khung nhìn có chứa tất cả các thành phần GUI.
* Trong bước này JSF đưa ra 3 loại khung nhìn: khung nhìn mới, khung nhìn khởi đầu, và khung nhìn hiện lại (postback), mỗi khung nhìn được xử lý khác nhau. Trong trường hợp khung nhìn mới, JSF xây dựng khung nhìn của trang Faces và nối các trình xử lý sự kiện và trình duyệt tính hợp lệ tới các thành phần. Khung nhìn được lưu trong một đối tượng FacesContext.
* *FacesContext* lưu trữ các thông tin trạng thái mà JSF cần để quản lý trạng thái của thành phần GUI cho yêu cầu hiện tại. *FacesContext* lưu khung nhìn trong thuộc tính *viewRoot* của nó; *viewRoot* chứa tất cả các thành phần JSF của khung nhìn hiện tại dưới dạng cấu trúc hình cây.
* Trong trường hợp của khung nhìn khởi đầu (lần đầu tiên một trang web được nạp), JSF tạo ra một khung nhìn rỗng. Khung nhìn rỗng được điền dần dần khi trang JSP được xử lý. Từ khung nhìn khởi đầu, JSF sẽ chuyển thẳng đến giai đoạn cuối cùng **render response**.
* Trong trường hợp khung nhìn hiện lại (postback - người dùng quay lại một trang đã truy cập trước đó), khung nhìn tương ứng với trang này đã tồn tại, do đó nó chỉ cần được khôi phục lại. Trong trường hợp này, JSF sử dụng thông tin trạng thái của khung nhìn hiện có để xây dựng lại trạng thái của nó.
* Giai đoạn 2: **Apply Request Values**:
* Trong bước này mỗi thành phần lấy ra trạng thái hiện tại của nó. Các thành phần trước tiên phải được lấy ra hoặc được tạo ra từ đối tượng *FacesContext* theo sau là các giá trị của chúng. Các giá trị thành phần thường được lấy ra từ các tham số yêu cầu( tham số của request) sẽ gọi phương thức *decode*, mặc dù chúng cũng có thể được lấy ra từ cookie hoặc các tiêu đề. Giá trị từ tham số yêu cầu cho nhiều thành phần được lưu trong *submittedValue* của thành phần.
* Nếu một thuộc tính xử lý-sự kiện tức thời của thành phần được đặt là true (đúng), các giá trị sẽ được biến đổi sang kiểu đúng và quá trình duyệt tính hợp hợp lệ đúng thì giá trị biến đổi sẽ lưu vào trong thành phần.
* Nếu việc biến đổi giá trị hoặc việc duyệt tính hợp lệ thất bại, một thông báo lỗi được tạo ra và xếp hàng đợi trong *FacesContext,* nơi nó sẽ được hiển thị trong giai đoạn *render response* cùng với bất kỳ lỗi duyệt tính hợp lệ nào khác.
* Như vậy tại giai đoạn này các thành phần (components) sẽ được thay đổi giá trị mới, và các thông báo (messages) cũng như các sự kiện (events) đều được lưu vào trong hàng đợi (queue) của *FacesContext.*
* Giai đoạn 3:**Process Validation:**
* Phép biến đổi và duyệt tính hợp lệ thông thường xảy ra trong giai đoạn này. Vì vậy, nếu trường này được liên kết với một thuộc tính *Số nguyên* (*Integer*) (ví dụ thế), thì giá trị được biến đổi thành *Số nguyên*. Nếu phép biến đổi giá trị không thành công, một thông báo lỗi được tạo ra và xếp hàng đợi trong *FacesContext*, nơi nó sẽ được hiển thị trong giai đoạn render response, cùng với bất kỳ lỗi duyệt tính hợp lệ nào khác.
* Việc xử lý sự kiện đầu tiên của vòng đời diễn ra sau giai đoạn 2. Ở giai đoạn này, các giá trị của mỗi thành phần được duyệt tính hợp lệ theo các quy tắc duyệt tính hợp lệ của ứng dụng. Các quy tắc duyệt tính hợp lệ có thể được định nghĩa trước (được gửi đi cùng với JSF), hoặc được các nhà phát triển xác định. Các giá trị do người sử dụng nhập vào được so với các quy tắc duyệt tính hợp lệ. Nếu một giá trị nhập vào không hợp lệ, một thông báo lỗi sẽ được thêm vào FacesContext, và thành phần này được đánh dấu là không hợp lệ.
* Nếu một thành phần được đánh dấu là không hợp lệ, JSF chuyển tiếp đến giai đoạn render response và bỏ qua các giai đoạn còn lại còn lại. Nó sẽ hiển thị khung nhìn hiện tại với các thông báo lỗi duyệt tính hợp lệ.
* Nếu không có lỗi duyệt tính hợp lệ nào xảy ra, JSF tiến đến giai doạn tiếp theo là update model values.
* Giai đoạn 4: **Update Model Values:**
* Trong giai đoạn này sẽ cập nhật các giá trị thực tại của mô hình phía máy chủ bằng cách cập nhật các thuộc tính của bean. Chỉ có các thuộc tính của bean bị liên kết với một giá trị của thành phần mới được cập nhật.
* Lưu ý rằng pha này xảy ra sau khi duyệt tính hợp lệ, vì vậy bạn có thể chắc chắn rằng các giá trị được sao chép vào các thuộc tính các bean của bạn là hợp lệ (ít nhất là ở mức trường biểu mẫu; chúng có thể vẫn còn không hợp lệ ở mức quy tắc nghiệp vụ).
* Giai đoạn 5**:** **Invoke Application:**
* Trong giai đoạn này JSF sẽ thự thi tất cả các sự kiện đã được đăng của ứng dụng, như là submit form hay link chuyển sang một trang khác.
* Trong giai đoạn này, cần phải chỉ rõ khung nhìn logic tiếp theo đối với một diễn tiến đã cho hoặc một số diễn tiến có khả năng xảy ra. Làm điều này bằng cách định nghĩa một kết quả cụ thể cho một đệ trình biểu mẫu thành công và trả về kết quả đó. *Ví dụ: khi kết quả đúng, di chuyển người dùng đến trang tiếp theo.* Khi xử lý sự kiện này thì mặc định *ActionListener* sẽ tìm đế *outcome, ví dụ như “success”,* là thuộc tính của thành phần action. Sau đó listener sẽ gửi outcome đến NavigationHandler mặc định. Tiếp theo NavigationHandler sẽ ánh xạ *outcome* với nguyên tắc chuyển trang đã được định nghĩa trong tệp tin cấu hình faces-config.xml xem trang cần hiển thị tiếp theo là trang nào. JSF nhận đối tượng trả về từ phương thức action và gọi phương thức toString() của nó. Sau đó, nó sử dụng giá trị này như là kết quả chuyển trang.
* Sau khi chuyển hướng xảy ra, bạn tiến tới giai đoạn render response.
* Giai đoạn 6: **Render Response:**
* Tại thời điểm này, tất cả xử lý bởi framework và ứng dụng đã trải qua. Tất cả chờ được gởi đi trả lời cho người dùng, và đây là mục tiêu chính của bước này.
* Mục tiêu thứ hai là lưu lại trạng thái hiển thị để nó có thể được phục hồi trong bước Restore View nếu user yêu cầu lại. Trạng thái hiển thị lưu lại trong bước này bởi vì thường thì hiển thị được lưu trên client, vì thế nó là một phần của response nhằm gởi trả cho người dùng. Trong trường hợp này, JSF đang lưu trạng thái trên server, vì thế hiển thị hầu hết được lưu trữ trong session của người dùng.

1. Vòng đời ICEface request

Iceface là một framework phát triển các component mở rộng dựa trên nền tảng của JSF. Chính vì vậy vòng đời của ICEface request cũng tương tự như vòng đời của JSF.

1. Ví dụ:

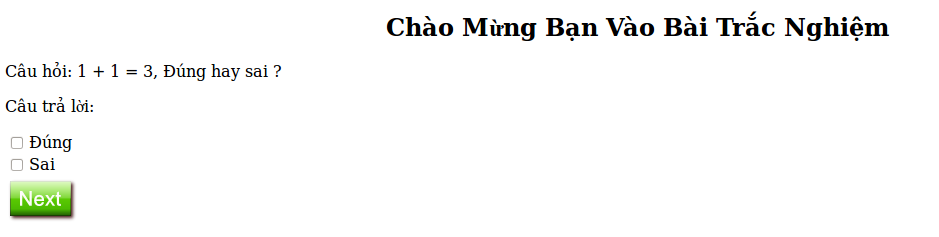
***Bài toán:*** Xây dựng chương trình thi trắc nghiệm. Có các đặc điểm sau

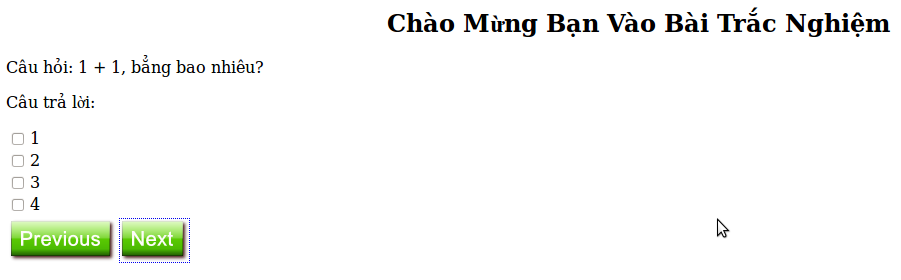
Bắt đầu là màn hình welcome có nút “Start”.

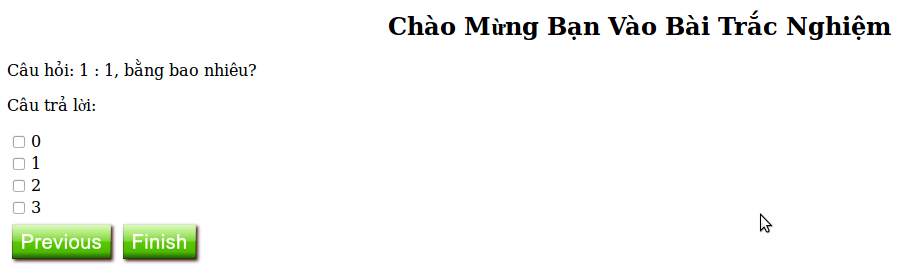
Sau khi click vào nút “Start” thì chuyển sang màn hình xuất hiện từng câu trắc nghiệm và nút “Next”. Sau khi chọn đáp án thì click vào nút “Next” thì chuyển sang màn hình có câu hỏi thứ 2 và 2 nút “Next” và nút “Previous”.Nút previuos để quay về những câu hỏi trước.

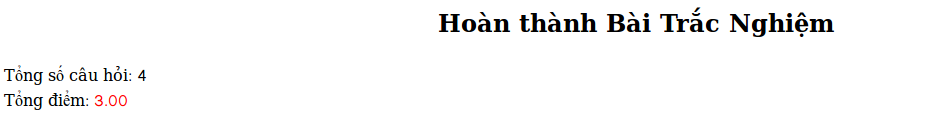
Đến câu hỏi cuối cùng thì chỉ xuất hiện nút previous và nút finish. Khi click vào nút finish thì chương trình xuất hiện màn hình tổng kết số câu hỏi và số điểm đạt được.











***Các bước xây dựng bài toán:***

* Các thành phần cần có:
* Một trang hiển thị các màn hình có tên là: index.xhtml
* Một bean quản lý dữ liệu các câu hỏi. Bean là một lớp java, chứa các thuộc tính cần thiết cho view tương ứng (trong trường hợp này view là index.xhtml) và các phương thức getter, setter của các thuộc tính đó và các hàm xử lý các action trên view.
* Các đối tượng model như sau: CauHoi.java, DapAn.java là các lớp nghiệp vụ chứa các thuộc tính và các hàm getter, setter.
* Cấu hình file web.xml và file faces-config.xml để chạy ứng dụng.
* Nội dung các file trong ứng dụng:
* File: TracNghiem/WebContent/index.xhtml là màn hình chứa các thẻ HTML cùng các thư viện khác như sau:

<html xmlns=*"http://www.w3.org/1999/xhtml"*

xmlns:ui=*"http://java.sun.com/jsf/facelets"*

xmlns:f=*"http://java.sun.com/jsf/core"*

xmlns:h=*"http://java.sun.com/jsf/html"*

xmlns:icecore=*"http://www.icefaces.org/icefaces/core"*

xmlns:ace=*"http://www.icefaces.org/icefaces/components"*

xmlns:ice=*"http://www.icesoft.com/icefaces/component"*>

<h:body>

<h2 align=*"center"*>

<ice:outputText value=*"#{Language['title']}"*

rendered=*"#{not bean.visTitle}"* />

<ice:outputText value=*"#{Language['finish-test']}"*

rendered=*"#{bean.visTitle}"* />

</h2>

<ice:form id=*"form"*>

<!--Button Start -->

<center>

<ice:panelGroup rendered=*"#{bean.imgStart}"*>

<ice:commandLink action=*"#{bean.clickImgStart}"*>

<ice:graphicImage alt=*"#{Language['start']}"*

url=*"/css/images/start.png"* style="align:*center*" />

</ice:commandLink>

</ice:panelGroup>

</center>

<!-- Cau hoi -->

<ice:panelGroup rendered=*"#{bean.visQes}"*>

<p>

<ice:outputLabel value=*"#{Language['question']}: "* />

<ice:outputText value=*"#{bean.questionCurrent.noiDung}"* />

</p>

<p>

<ice:outputLabel value=*"#{Language['answer']}:"* />

</p>

<ice:dataTable value=*"#{bean.questionCurrent.dapAns}"* var=*"dapAn"*>

<ice:column>

<ice:selectBooleanCheckbox value=*"#{dapAn.selected}"* />

</ice:column>

<ice:column>

<ice:outputText value=*"#{dapAn.value}"* />

</ice:column>

</ice:dataTable>

</ice:panelGroup>

<!-- Button Next, Previous -->

<ice:panelGrid columns=*"3"*>

<!-- Button Previous-->

<ice:panelGroup rendered=*"#{bean.imgPrev}"*>

<ice:commandLink action=*"#{bean.clickImgPrev}"*>

<ice:graphicImage alt=*"Previous"* url=*"/css/images/previous.png"* />

</ice:commandLink>

</ice:panelGroup>

<!-- Button Next -->

<ice:panelGroup rendered=*"#{bean.imgNext}"*>

<ice:commandLink partialSubmit=*"true"*

action=*"#{bean.clickImgNext}"*>

<ice:graphicImage alt=*"#{Language.next}"* url=*"/css/images/next.png"* />

</ice:commandLink>

</ice:panelGroup>

<!-- Button Finish -->

<ice:panelGroup rendered=*"#{bean.imgFinish}"*>

<ice:commandLink action=*"#{bean.clickImgFinish}"*>

<ice:graphicImage alt=*"#{Language.finish}"*

url=*"/css/images/finish.png"* />

</ice:commandLink>

</ice:panelGroup>

</ice:panelGrid>

<!-- Button Mark -->

<ice:panelGroup rendered=*"#{bean.visMark}"*>

<ice:outputLabel value=*"#{Language.finalQuestion}: "* />

<ice:outputText value=*"#{bean.sizeQuestion}"* />

<br />

<ice:outputLabel value=*"#{Language.finalMark}: "* />

<ice:outputLabel value=*"#{bean.diem}"* style="color: *red*;">

<f:convertNumber pattern=*"0.00"* />

</ice:outputLabel>

</ice:panelGroup>

</ice:form>

</h:body>

</html>

* File bean chứa tại: src/bean/BeanCauHoi.java

package bean;

import java.io.Serializable;

import java.util.ArrayList;

import java.util.List;

import javax.faces.event.ActionEvent;

import model.CauHoi;

import model.DapAn;

public class BeanCauHoi implements Serializable {

private static final long serialVersionUID = 8342581090247463420L;

private List<CauHoi> cauHois;

private int current = 0;

private CauHoi questionCurrent;

private double diem;

private boolean imgStart = true, imgPrev = false, imgNext = false,

imgFinish = false;

private boolean visQes = false;

private boolean visMark = false;

private boolean visTitle = false;

public Bean() {

cauHois = new ArrayList<CauHoi>();

CauHoi ch1 = new CauHoi();

ch1.setNoiDung("1 + 1 = 3, Đúng hay sai ?");

List<DapAn> da1 = new ArrayList<DapAn>();

da1.add(new DapAn(false, "Đúng"));

da1.add(new DapAn(false, "Sai"));

ch1.setDapAns(da1);

ch1.setDapAnDung(new DapAn(true, "Sai"));

cauHois.add(ch1); //Cau hoi 1

CauHoi ch2 = new CauHoi();

ch2.setNoiDung("1 + 1, bẳng bao nhiêu?");

List<DapAn> da2 = new ArrayList<DapAn>();

da2.add(new DapAn(false, "1"));

da2.add(new DapAn(false, "2"));

da2.add(new DapAn(false, "3"));

da2.add(new DapAn(false, "4"));

ch2.setDapAns(da2);

ch2.setDapAnDung(new DapAn(true, "2"));

cauHois.add(ch2); //Cau hoi 2

CauHoi ch4 = new CauHoi();

ch4.setNoiDung("1 : 1 = 1, Đúng hay sai ?");

List<DapAn> da4 = new ArrayList<DapAn>();

da4.add(new DapAn(false, "Đúng"));

da4.add(new DapAn(false, "Sai"));

ch4.setDapAns(da4);

ch4.setDapAnDung(new DapAn(true, "Đúng"));

cauHois.add(ch4); //Cau hoi 3

CauHoi ch5 = new CauHoi();

ch5.setNoiDung("1 : 1, bằng bao nhiêu?");

List<DapAn> da5 = new ArrayList<DapAn>();

da5.add(new DapAn(false, "0"));

da5.add(new DapAn(false, "1"));

da5.add(new DapAn(false, "2"));

da5.add(new DapAn(false, "3"));

ch5.setDapAns(da5);

ch5.setDapAnDung(new DapAn(true, "1"));

cauHois.add(ch5); //Cau hoi 4

}

public CauHoi getQuestionCurrent() {

this.questionCurrent = this.cauHois.get(current);

return this.questionCurrent;

}

public String clickImgNext() {

current = current + 1;

int step = current + 1;

imgStart = false;

visQes = true;

visMark = false;

// Get answer

if (step == cauHois.size()) {

imgFinish = true;

imgNext = false;

imgPrev = true;

} else if (step < cauHois.size() && step > 1) {

imgNext = true;

imgPrev = true;

imgFinish = false;

} else if (step == 1) {

imgPrev = false;

imgNext = true;

imgFinish = false;

}

return "";

}

public String clickImgStart() {

imgStart = false;

imgFinish = false;

imgNext = true;

imgPrev = false;

visMark = false;

visQes = true;

return "";

}

public String clickImgPrev() {

current = current - 1;

int step = current + 1;

imgStart = false;

visQes = true;

visMark = false;

// Get answer

if (step == cauHois.size()) {

imgFinish = true;

imgNext = false;

imgPrev = true;

} else if (step < cauHois.size() && step > 1) {

imgNext = true;

imgPrev = true;

imgFinish = false;

} else if (step == 1) {

imgPrev = false;

imgNext = true;

imgFinish = false;

}

return "";

}

public String clickImgFinish() {

for (CauHoi ch : cauHois) {

for (DapAn da : ch.getDapAns()) {

double diemMotCau = 0.0;

if (da.isSelected()) {

if (ch.getDapAnDung().equals(da)) {

diemMotCau++;

} else {

diemMotCau--;

}

}

if (diemMotCau > 0) {

diem += diemMotCau;

}

}

}

visQes = false;

imgFinish = false;

imgNext = false;

imgPrev = false;

imgStart = false;

visMark = true;

visTitle = true;

return "";

}

public void setQuestionCurrent(CauHoi questionCurrent) {

this.questionCurrent = questionCurrent;

}

public int getSizeQuestion() {

if (cauHois != null) {

return cauHois.size();

}

return 0;

}

//getter and setter method

}

* File src/model/CauHoi.java:

import java.io.Serializable;

import java.util.List;

public class CauHoi implements Serializable {

private static final long serialVersionUID = 7240174843165059045L;

private String noiDung;

private List<DapAn> dapAns;

private DapAn dapAnDung;

public CauHoi() {

}

//getter and setter method

}

* File src/model/DapAn.java:

package model;

import java.io.Serializable;

public class DapAn implements Serializable {

private static final long serialVersionUID = -2665019127129085506L;

private boolean selected;

private String value;

public DapAn() {

}

public DapAn(boolean selected, String value) {

this.selected = selected;

this.value = value;

}

@Override

public boolean equals(Object obj) {

if (obj != null && obj instanceof DapAn) {

DapAn that = (DapAn) obj;

return this.value.equals(that.value);

}

return false;

}

//Getter and setter method

}

* File WEB-INF/faces-config.xml

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

<faces-config 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/web-facesconfig\_2\_0.xsd"

version="2.0">

<managed-bean>

<managed-bean-name>bean</managed-bean-name>

<managed-bean-class>bean.BeanCauHoi</managed-bean-class>

<managed-bean-scope>session</managed-bean-scope>

</managed-bean>

</faces-config>