Группировка компонентов

Рассмотрим файл `index.xhtml` и тег `<h:panelGroup>`. Как правило, он используется для группировки нескольких компонентов таким образом, чтобы они заняли одну ячейку в `<h:panelGrid>`. Этого можно добиться путем добавления компонентов внутрь тега `<h:panelGroup>` и последующего его добавления к тегу `<h:panelGrid>`. Как видно из примера, у нашего экземпляра `<h:panelGroup>` нет никаких дочерних компонентов. В данном случае тег `<h:panelGroup>` предназначен для получения «пустой» ячейки и для того, чтобы заставить компонент `<h:commandButton>`

Отправка формы

Отображением тега `<h:commandButton>` на код HTML является кнопка отправки (*Submit*), отображаемая в обозревателе. Точно так же, как и в случае со стандартным HTML, ее целью является отправка формы. Ее атрибут *value* просто устанавливает метку кнопки. Атрибут *action* данного тега используется для навигации. Следующая страница для отображения основана на значении этого атрибута. У атрибута *action* могут быть строковая константа или выражение метода связывания (method binding expression), означающее, что он может указывать на метод в управляемом бине, который возвращает строку.

Если базовое имя страницы в нашем приложении соответствует значению атрибута *action* тега `<h:commandButton>`, то при нажатии кнопки мы перемещаемся к этой странице. В нашем примере страница подтверждения имеет название confirmation.xhtml. Поэтому, в соответствии с соглашением, эта страница будет показана при щелчке по кнопке, поскольку значение ее атрибута action соответствует базовому имени страницы (”confirmation”). Даже при том, что кнопка называется Сохранить (Save), в нашем простом примере щелчок по кнопке фактически не будет сохранять данные.

Управляемые бины

Следующий код является управляемым бином для нашего примера:

LearnJSF\src\java\beans\Customer.java

```
package beans;
import javax.enterprise.context.RequestScoped;
import javax.inject.Named;

//Превращает  bean в именованный
//Область видимости запроса
@Named
@RequestScoped
public class Customer {
   private String firstName;
   private String lastName;
   private String email;

   public String getEmail() {
      return email;
   }
   public void setEmail(String email) {
      this.email = email;
   }
   public String getFirstName() {
      return firstName;
   }
   public void setFirstName(String firstName) {
      this.firstName = firstName;
   }
   public String getLastName() {
      return lastName;
   }
   public void setLastName(String lastName) {
      this.lastName = lastName;
   }
}
```

Контексты управляемых бинов

У управляемых бинов всегда есть контекст. Контекст управляемых бинов определяет продолжительность их жизни. Он определяется аннотацией уровня класса. В следующей таблице приводятся все допустимые контексты управляемых бинов:

Аннотации контекстов управ - ляемых бинов	Описание
@ApplicationScoped	Один и тот же экземпляр управляемого бина в контексте приложения доступен всем клиентам нашего приложения. Если один из клиентов изменит значение управляемого бина в контексте приложения, то это изменение отразится на всех клиентах
@SessionScoped	Каждый экземпляр управляемого бина в контексте сеанса присваивается каждому из клиентов нашего приложения. Управляемый бин в контексте сеанса используется для хранения специфических данных клиента между запросами
@RequestScoped	Управляемый бин в контексте запроса «живет» только в течение одиночного HTTP-запроса
@ViewScoped	Управляемый бин в контексте представления ассоциируется с определенным представлением (страницей). Он будет уничтожен, как только пользователь перейдет к другому представлению
@NoneScoped	Управляемый бин без контекста создается, когда к нему получает доступ другой управляемый бин, обычно как к управляемому свойству
@CustomScoped	В JSF 2.0 у нас появилась возможность создавать пользовательские контексты для наших управляемых бинов. Атрибут value аннотации @CustomScoped должен разрешаться на карте контекста сеанса

Если никакой контекст не указывается в управляемом бине (как в нашем примере), то по умолчанию используется контекст запроса.

Am new to programming with basic knowledge and I've taken a liken to Java.
I wanted to write a code that calculates a number to the nth power without using loops. I've been trying to use the repeat method from "commons lang" which i came to know about, about 4 days ago. I Found a lot of info in this site and others that helped me in understanding how to use this packed.
So far I downloaded commons-lang3-3.1 then kept the folder in the same folder as my project and added the jar file to my project's library by:- right clicking on libraries
1 then Add JAR/Folder
2 then i opened the commons-lang3-3.1 folder
3 and selected "commons-lang3-3.1.jar" from a number of 4 selections:

- commons-lang3-3.1.jar
  - commons-lang3-3.1-javadoc.jar
  - commons-lang3-3.1-sources.jar
  - commons-lang3-3.1-tests.jar

Пользовательская проверка допустимости данных

JSF не только предоставляет нам стандартные блоки проверки допустимости, но и позволяет создавать нестандартные (пользовательские) элементы верификации.
Для этого предусмотрены два способа: создание класса нестандартного элемента верификации и добавление метода проверки допустимости в наши управляемые бины.

Создание нестандартных элементов верификации

В дополнение к стандартным блокам проверки допустимости JSF позволяет нам создавать нестандартные (пользовательские) элементы верификации путем создания класса Java, реализующего интерфейс javax.faces.validator.Validator. Следующий класс реализует блок проверки допустимости адреса электронной почты на нашей странице ввода информации о клиентах:

LearnJSF\src\java\beans\EmailValidator.java

```
package beans;

import javax.faces.component.UIComponent;
import javax.faces.component.html.HtmlInputText;
import javax.faces.application.FacesMessage;
import javax.faces.context.FacesContext;
import javax.faces.validator.FacesValidator;
import javax.faces.validator.Validator;
import javax.faces.validator.ValidatorException;
import org.apache.commons.lang.StringUtils;

/**
 *
 * @author Computer School
 */

@FacesValidator(value = "emailValidator")
public class EmailValidator implements Validator {

    @Override
    public void validate(FacesContext context, UIComponent component, Object value) throws
ValidatorException {
        org.apache.commons.validator.routines.EmailValidator emailValidator =
org.apache.commons.validator.routines.EmailValidator.getInstance();
//        org.apache.commons.validator.EmailValidator emailValidator =
org.apache.commons.validator.EmailValidator.getInstance();
        HtmlInputText htmlInputText = (HtmlInputText) component;
        String email = (String) value;
        if (!StringUtils.isEmpty(email)) {
            if (!emailValidator.isValid(email)) {
                FacesMessage facesMessage = new FacesMessage(htmlInputText.getLabel() + ": задан неверный формат
email");
                throw new ValidatorException(facesMessage);
            }
        }
    }

}
```

Аннотация @FacesValidator регистрирует наш класс как класс нестандартного элемента верификации JSF. Значением его атрибута value является логическое имя, которое страницы JSF могут использовать для обращения к нему.
Как видно из приведенного примера, единственным методом, который мы должны реализовать при реализации интерфейса Validator, является метод validate() . Он принимает три параметра: экземпляр javax.faces.context.FacesContext, экземпляр javax.faces.component.UIComponent и объект. Как правило, разработчики приложений должны

беспокоиться только о последних двух. Второй параметр является компонентом, данные которого мы проверяем; третий представляет собой фактическое значение. В примере мы приводим uiComponent к типу javax.faces.component.html.HtmlInputText. Таким образом, мы получаем доступ к его методу getLabel() , который сможем использовать в качестве части сообщения об ошибке. Если введенное значение имеет недопустимый формат адреса электронной почты, создается новый экземпляр javax.faces.application.FacesMessage, передающий сообщение об ошибке в качестве параметра его конструктора, которое будет выведено на экран в обозревателе. Затем мы вызываем новое исключение javax.faces.validator.ValidatorException. После этого сообщение об ошибке выводится на экран в обозревателе. То, как оно туда попадает, происходит «за кулисами» API JSF.

Общий блок проверки допустимости Apache

Предыдущий блок проверки допустимости использует общий блок проверки допустимости Apache для выполнения фактической проверки допустимости. Эта библиотека включает много общих проверок допустимости, таких как даты, номера кредитных карт, ISBN и электронные письма. При реализации нестандартного элемента верификации вначале стоит посмотреть, нет ли в этой библиотеке такого блока проверки допустимости, который подходит для нашего случая.

Чтобы использовать наш блок проверки допустимости в странице, мы должны воспользоваться тегом JSF <f:validator>. Следующая страница фэйслета является модифицированной версией экрана ввода данных о клиентах. Эта версия использует тег <f:validator> для проверки поля электронной почты:

LearnJSF\web\index.xhtml

```
<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://java.sun.com/jsf/html"
      xmlns:f="http://java.sun.com/jsf/core">
    <h:head>
        <title>Facelet Title</title>
        <h:outputStylesheet library="css" name="styles.css"/>
    </h:head>
    <h:body>

        <h:form>
            <h:messages></h:messages>
            <h:panelGrid columns="2" columnClasses="rightAlign,leftAlign">
                <h:outputText value="Имя:"></h:outputText>
                <h:inputText label="Имя" value="#{customer.firstName}"
                             required="true">
                    <f:validateLength minimum="2" maximum="30">
                    </f:validateLength>
                </h:inputText>
                <h:outputText value="Фамилия:"></h:outputText>
                <h:inputText label="Фамилия" value="#{customer.lastName}"
                             required="true">
                    <f:validateLength minimum="2" maximum="30">
                    </f:validateLength>
                </h:inputText>
                <h:outputText value="Email:"></h:outputText>
                <h:inputText label="Email" value="#{customer.email}">
                    <f:validator validatorId="emailValidator"/>
                </h:inputText>
                <!-- <h:inputText label="Email" value="#{customer.email}">
                    <f:validateLength minimum="3" maximum="30">
                    </f:validateLength>
                </h:inputText>-->
```

```
            <h:panelGroup></h:panelGroup>
            <h:commandButton action="confirmation" value="Сохранить">
            </h:commandButton>
        </h:panelGrid>
    </h:form>
  </h:body>
</html>
```

После написания нашего нестандартного элемента верификации и изменения нашей страницы для использования его возможностей мы сможем увидеть наш блок проверки допустимости в действии:

Если желаемого результата не увидели, то необходимо подключить недостающие библиотеки.