Улучшения и упрощения в JSF 2.0

Управляемые бины

Аннотации

В JSF 2.0 стало возможным использование аннотаций для объявления управляемых бинов, благодаря чему управляемые бины более не требуется описывать в конфигурационном файле `faces-config.xml`.

Все поддерживаемые аннотационные типы определены в пакете `javax.faces.bean`.

Для регистрации класса управляемого бина предназначена аннотация `@ManagedBean`, элемент `name` которой определяет наименование управляемого бина. Например,

```
@ManagedBean(name="GuessMe")
public class GuessBean {...}
```

Если элемент `name` не указан, то наименованием управляемого бина становится простое имя класса, первая буква которого переводится в нижний регистр. Если в приведенном примере элемент `name` не был бы указан, то управляемый бин назывался бы **`guessBean`**.

Сканирование классов веб-приложения на наличие аннотации `@ManagedBean` выполняется при запуске приложения, до обработки им первого запроса.

Также для класса управляемого бина необходимо указать контекст, в котором будет храниться соответствующий ему бин, с помощью одной из следующих аннотаций:

- `@ApplicationScoped` для контекста приложения
- `@SessionScoped` для контекста сессии
- `@ViewScoped` для контекста представления (новый контекст в JSF 2.0)
- `@RequestScoped` для контекста запроса
- `@NoneScoped` для неопределенного контекста (подобные управляемые бины могут использоваться только при инициализации свойств других бинов, к ним невозможно обратиться из JSF-страниц)

Для инициализации свойств управляемого бина можно использовать аннотацию `@ManagedProperty`, элемент `value` которой содержит выражение для присваивания (в том числе EL-выражение). Аннотация `@ManagedProperty` указывается для поля, но при этом для поля обязательно должен существовать set-метод.

Пример определения управляемого бина с инициализацией свойства:

```
@ManagedBean(name="GuessBean")
@SessionScoped
public class GuessBean implements Serializable {
    @ManagedProperty(value="1")
    private int count;

    public int getCount() { return count; }

    public void setCount(int count) { this.count = count; }
}
```

Также в JSF 2.0 появилась возможность описывать валидаторы, конвертеры, компоненты и рендереры при помощи аннотаций, что, в совокупности с описываемой далее прямой (неявной) навигацией, позволяет отказаться от конфигурационного файла `faces-config.xml`.

Контекст представления

В JSF 2.0 появился новый контекст для управляемых бинов – так называемый View Scope, – а также возможность определять собственные контексты, жизненным циклом объектов в которых управляет разработчик класса контекста.

Контекст представления (View Scope) привязан к текущему представлению (дереву компонентов) и существует, пока с ним идет взаимодействие. Когда в результате

выполнения приложения в соответствии с правилами навигации изменяется текущее представление, прежний контекст представления удаляется и создается новый. Таким образом, по времени жизни контекст представления занимает промежуточное положение между контекстами запроса и сессии. Он может быть полезен для предоставления пользователю возможности выполнения одних и тех же функций приложения в различных вкладках браузера. В данном случае контекст сессии затруднительно использовать для хранения промежуточных данных, так как запросы к приложению от разных вкладок браузера относятся к одной и той же сессии.

Валидация

В JSF 2.0 поддерживается валидация управляемых бинов средствами Bean Validation API. Особенность последнего заключается в том, что ограничения на значения полей объектов задаются с помощью аннотаций. В Bean Validation API определено несколько стандартных валидаторов и имеются средства для определения собственных валидаторов.

В библиотеке базовых действий JSF 1.2 имелись действия для валидации полей ввода, но поддержка Bean Validation API поднимает валидацию на качественно иной уровень. Она позволяет проверять правильность вводимых данных независимо от типа интерфейса пользователя, так как ведется проверка данных объекта предметной области.

В JSF 2.0 добавлено действие `<f:validateBean>`, которое запрашивает валидацию свойств управляемых бинов средствами Bean Validation API на шаге «Выполнить валидацию» в цикле обработки JSF-запроса. Пример использования этого действия для нескольких свойств одновременно (также можно вложить действие `<f:validateBean>` в конкретное поле ввода):

```
<f:validateBean>
    Try <h:outputText id="tryCount" value="#{GuessBean.count}"/>:
    <h:inputText type="text" id="guessField" value="#{GuessBean.guess}"/>
</f:validateBean>
```

В классе управляемого бина определены следующие ограничения:

```
import javax.validation.constraints.Max;
import javax.validation.constraints.Min;

@ManagedBean(name="GuessBean")
@SessionScoped
public class GuessBean implements Serializable {
    private int number;
    @Min(value=0, message="Число больше или равно 0") @Max(9)
    private int guess;
    @ManagedProperty(value="1")
    private int count;
…
}
```

Обратите внимание, что ошибки валидации собираются в FacesContext вместе со всеми остальными сообщениями. Для их вывода необходимо использовать действие `<h:messages>` или `<h:message>`.

Навигация

В JSF 2.0 появилась возможность напрямую использовать идентификатор представления для навигации, не определяя для этого правила в конфигурационном файле `faces-config.xml`. Для этого в атрибуте `action` либо при возврате из action-метода нужно указать путь к следующей JSF-странице от корня веб-приложения. Пример подобной неявной навигации (implicit navigation) в action-методе управляемого бина:

```
    public String check() {
        System.out.println("check() called");
        count++;
        if (guess == number)
```

```
            return "/checkguess.xhtml";
        else
            return "/guess.xhtml";
    }
```

Пример неявной навигации на JSF-странице:

```
<h:commandButton action="/checkguess_1.xhtml" value="Check"/>
```

Второе улучшение состоит в добавлении вычисляемых условий в правила навигации. Таким образом, для определения следующего представления теперь можно использовать не только комбинацию текущего представления и результата action-метода (либо значения атрибута `action`), но и выражение логического типа на языке EL.

Третье улучшение заключается в возможности отправки GET-запросов к JSF-страницам. Для этого в библиотеку Basic HTML RenderKit добавлены действия `<h:link>` и `<h:button>`. Использование GET-запросов позволяет получить т.н. bookmarkable URLs (адреса, которые можно использовать для закладок) и индексировать соответствующие страницы в поисковых машинах.

В отличие от обычной процедуры обработки запроса, где следующее представление определяется на предпоследнем шаге, а URL запроса соответствует текущему представлению, действия `<h:link>` и `<h:button>` уже при формировании страницы ответа определяют следующее представление с помощью атрибута `outcome`.

Для передачи параметров через GET-запрос используются параметры, вложенные в действия `<h:link>` и `<h:button>`. Пример формирования ссылки для отправки GET-запроса:

```
<h:link outcome="/checkguess.xhtml" includeViewParams="true" value="GET it">
    <f:param name="param" value="5"/>
</h:link>
```

Результирующая ссылка должна иметь следующий вид:

```
<a href="/app/faces/checkguess.xhtml?param=5">GET it</a>
```

Для обработки параметров, переданных через GET-запрос, можно использовать параметры представления (view parameters):

```
<f:metadata>
    <f:viewParam name="param" value="#{GuessBean.number}"/>
</f:metadata>
```

Facelets

Facelets – это альтернативная технология управления представлением для JSF, которая требует для функционирования валидные XML-документы. Это означает, что веб-страницы должны быть созданы с использованием языка разметки XHTML. Facelets поддерживает все компоненты JSF и создает собственное дерево компонент.

В настоящее время технология Facelets считается более предпочтительной, чем JSP, для формирования представления в рамках JSF. Возможно, основная причина – изначальное отсутствие скриптовых элементов, в результате чего на страницах отсутствует Java-код. Кроме того, Facelets поддерживают механизм шаблонов и обеспечивают простую разработку компонентов пользовательского интерфейса для использования в JSF.

Важным отличием Facelets от JSP с точки зрения обработки является то, что Facelets не компилируются.

Помимо XML-элементов в Facelet можно непосредственно использовать EL-выражения для вывода информации.

Для того, чтобы у разработчика была возможность использовать для редактирования Facelets традиционные XHTML-редакторы, не знающие о существовании действий JSF, был добавлен особый атрибут `jsfc`. Он используется в элементах XHTML и указывает название действия JSF, которое на самом деле будет использовано. Таким образом, перед исполнением Facelet-а происходит его трансформация в JSF-страницу.

Далее приведено два примера Facelet-ов, которые эквивалентны с точки зрения их выполнения, но второй из них гораздо более удобен для редактирования, так как содержит только стандартные элементы XHTML.

Пример 1:

```
<?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">
    <h:head>
        <title>Facelet Title</title>
    </h:head>
    <h:body>
        <h:form>
            Hello from Facelets<p/>#{GuessBean.number}<p/>
            <h:inputText value="#{GuessBean.guess}"/>
            <h:commandButton action="/checkguess_1.xhtml" value="Check"/>
        </h:form>
    </h:body>
</html>
```

Пример 2:

```
<?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">
    <head>
        <title>Facelet Title</title>
    </head>
    <body>
        <form jsfc="h:form">
            Hello from Facelets<p/>#{GuessBean.number}<p/>
            <input type="text" jsfc="h:inputText"
                    value="#{GuessBean.guess}"/>
            <input type="submit" jsfc="h:commandButton"
                    action="/checkguess_1.xhtml" value="Check"/>
        </form>
    </body>
</html>
```

Обратите внимание, что для использования библиотеки действий достаточно определить префикс для соответствующего ей пространства имен:

```
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://java.sun.com/jsf/html">
```

Шаблоны в Facelets

Шаблон – это разметка страницы, которую можно повторно использовать. Например, любая страница некоторого веб-приложения должна содержать неизменные «шапку», боковое меню и «подвал».

В JSP можно включить JSP-фрагменты или другие JSP-страницы, содержащие «шапку», боковое меню и «подвал», в текущую JSP-страницу, но на каждой странице придется повторять эту общую структуру. Очевидно, что технология JSP не имеет встроенной поддержки шаблонов, поэтому при изменении общего принципа верстки (например, отказе от «подвала») придется изменить все основные JSP-страницы приложения, что очень неэффективно. Для поддержки шаблонов в рамках технологии JSP были разработаны сторонние библиотеки, например, Apache Tiles.

Технология Facelets обеспечивает встроенную поддержку шаблонов. Соответствующие действия определены в пространстве имен `http://java.sun.com/jsf/facelets`, которому обычно назначается префикс `ui`.

В странице-шаблоне (template) используются действия <ui:insert>, которые отмечают места, в которые будет вставляться содержимое, изменяющееся от страницы к странице.

В страницу-клиент шаблона (template client) вставляется действие <ui:composition>, атрибут template которого определяет используемый шаблон. JSF игнорирует на странице-клиенте шаблона любую разметку, находящуюся вне действия <ui:composition>, поэтому страница-клиент может содержать элементы <html>, <body>, необходимые для ее корректного редактирования в XHTML-редакторе.

Действиям <ui:insert> из страницы-шаблона в странице-клиенте соответствуют действия <ui:define>, содержащие разметку, специфичную для конкретной страницы приложения. Для установки соответствия между действиями <ui:insert> и <ui:define> применяется атрибут name.

Пример страницы-шаблона, содержащей три точки вставки содержимого:

```
<?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:ui="http://java.sun.com/jsf/facelets"
      xmlns:h="http://java.sun.com/jsf/html"
      xmlns:f="http://java.sun.com/jsf/core">
<f:view>
    <h:head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
        <link href="./resources/css/default.css" rel="stylesheet"
                type="text/css" />
        <link href="./resources/css/cssLayout.css" rel="stylesheet"
                type="text/css" />
        <title><ui:insert name="top">Facelets Template</ui:insert></title>
    </h:head>

    <h:body>
        <div id="content" class="center_content">
            <ui:insert name="content">Content</ui:insert>
        </div>

        <div id="bottom">
            <ui:insert name="bottom">Bottom</ui:insert>
        </div>
    </h:body>
</f:view>
</html>
```

Пример страницы-клиента приведенного выше шаблона:

```
<?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:ui="http://java.sun.com/jsf/facelets"
      xmlns:h="http://java.sun.com/jsf/html"
      xmlns:f="http://java.sun.com/jsf/core">
    <body>

        <ui:composition template="./GuessSiteTemplate.xhtml">

            <ui:define name="top">
                Guess Number Title
            </ui:define>

            <ui:define name="content">
                <form jsfc="h:form">
Try <span id="tryCount" jsfc="h:outputText" value="#{GuessBean.count}"/>:
```

```
<input type="text" id="guessField" jsfc="h:inputText"
value="#{GuessBean.guess}"/>
<input type="submit" jsfc="h:commandButton" action="#{GuessBean.check}"
value="Check"/>
                </form>
            </ui:define>

            <ui:define name="bottom">
             </ui:define>

        </ui:composition>

    </body>
</html>
```

Разработка компонентов интерфейса пользователя в Facelets

В JSF 1.2 для разработки компонента интерфейса пользователя требовалось создать 1) класс компонента, 2) класс рендерера, 3) класс обработчика действия для JSP, 4) дескриптор библиотеки действий и выполнить настройку в конфигурационном файле `faces-config.xml`. При этом формированием выходной разметки компонента (например, в виде элемента HTML) занимался класс рендерера, написанный на Java.

Технология Facelets значительно упростила этот процесс: теперь достаточно создать один Facelet, формирующий выходную разметку, все остальное работает по принципу «Convention over configuration», что можно вольно перевести как «Соблюдение соглашений лучше, чем явная настройка». Так как Facelet представляет собой, как правило, XHTML-документ, это открывает широкие возможности для повторного использования фрагментов существующих страниц веб-приложения.

Компоненты пользовательского интерфейса, разрабатываемые с помощью Facelet, ничем не отличаются по своему использованию от стандартных компонентов JSF. Они вызываются так же – в виде custom actions (действий, определяемых программистом), у них могут быть атрибуты, к ним можно присоединять валидаторы, конвертеры и обработчики событий.

Как и для традиционных компонентов интерфейса пользователя JSF 1.2, для компонентов Facelets необходима библиотека действий, описывающая, как эти компоненты можно использовать в JSF-представлениях (JSF-страницах). Отличие состоит в том, что для компонентов Facelets библиотеки действий формируются динамически, на основании определения интерфейса компонента, содержащемся в самом Facelet. Кроме того, файлы Facelet'ов должны размещаться в каталоге ресурсов, чтобы считаться компонентами пользовательского интерфейса. Каждому подкаталогу каталога ресурсов (например, `/resources/`**`component`**) соответствует отдельная библиотека действий, пространство имен которой состоит из `http://java.sun.com/jsf/composite/` и имени подкаталога (например, `http://java.sun.com/jsf/composite/`**`component`**).

Для использования такой библиотеки действий с компонентами пользовательского интерфейса достаточно в Facelet объявить соответствующее пространство имен, например:

```
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:ui="http://java.sun.com/jsf/facelets"
      xmlns:h="http://java.sun.com/jsf/html"
      xmlns:f="http://java.sun.com/jsf/core"
      xmlns:mycomp="http://java.sun.com/jsf/composite/component">
```

Название действия совпадает с именем файла Facelet, содержащего определение компонента интерфейса пользователя. Например, если полное имя такого файла в проекте /resources/component/sample.xhtml, то компонент добавляется на страницу с помощью следующего действия:

```
<mycomp:sample/>
```

Каждый компонент пользовательского интерфейса JSF, создаваемый с помощью Facelets, состоит из двух частей: интерфейса и реализации. Интерфейс предоставляет автору страницы, на которой используется компонент, все необходимые сведения для использования компонента. Реализация содержит разметку, которая выводится в результирующую страницу.

Интерфейс компонента определяется в действии `<cc:interface>`, где префикс `cc` по соглашению назначается пространству имен `http://java.sun.com/jsf/composite`.

Автор страницы, содержащей компонент, может передать компоненту необходимые ему данные с помощью атрибутов, которые описываются в интерфейсе компонента с помощью действий `<cc:attribute>`. У атрибута компонента должно быть имя (`name`), может быть указан конкретный тип значений (`type`), а также обязательность атрибута при использовании компонента (`required`). В качестве имени атрибута можно указать одно из специальных значений: `action` – для передачи компоненту action-метода, `actionListener` – для обработчика действия пользователя, `validator` - для валидатора и `valueChangeListener` – для обработчика изменения значения. При использовании одного из специальных значений требуется в атрибуте `targets` указать перечень идентификаторов вложенных компонентов, к которым относится переданный метод.

Кстати говоря, подразумевается, что компонент интерфейса пользователя, разрабатываемый с помощью Facelets, состоит из разметки и вложенных компонентов, именно поэтому данные компоненты называются составными (Composite Component). Но с точки зрения использования составной компонент выглядит как стандартный, поэтому он может выступать в роли источника действия (аналога кнопки или ссылки) и/или компонента для отображения/редактирования значения. Так как составной компонент может включать несколько вложенных компонентов, необходимо в интерфейсе составного компонента установить соответствие между его наблюдаемыми свойствами и внутренней реализацией. Для этого предназначены действия `<cc:actionSource>`, `<cc:valueHolder>` и `<cc:editableValueHolder>`, каждое из которых позволяет задать внешнее имя свойства (атрибут `name`) и его отображение на реализацию (атрибут `targets`).

Реализация компонента определяется в действии `<cc:implementation>`. Для доступа к атрибутам компонента используется EL-выражение `#{cc.attrs}`.

Рассмотрим пример составного компонента, представляющего собой HTML-форму с полем ввода и кнопкой:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://java.sun.com/jsf/html"
      xmlns:cc="http://java.sun.com/jsf/composite">
    <cc:interface>
        <cc:attribute name="guess" required="true"/>
        <cc:attribute name="action" required="true"
                targets="guessForm:checkButton"/>
        <cc:editableValueHolder name="inputField"
                targets="guessForm:guessField"/>
    </cc:interface>
    <cc:implementation>
        <form jsfc="h:form" id="guessForm">
            <h:message for="guessField"/><p/>
            Your guess:
            <input type="text" label="guess" id="guessField"
                    jsfc="h:inputText" value="#{cc.attrs.guess}"/>
            <input type="submit" id="checkButton" jsfc="h:commandButton"
                    value="Check"/>
        </form>
    </cc:implementation>
```

```
</html>
```

У компонента два обязательных атрибута: 1) `guess` – представляет редактируемое значение, 2) `action` – метод, вызываемый при нажатии кнопки Check. Кроме того, компонент предоставляет доступ извне к редактируемому значению, названному `inputField`.

Так как HTML-форма определена в самом компоненте, то его можно использовать в любом месте вызывающей страницы, но из-за этого приходится использовать составные идентификаторы в интерфейсе компонента.

В реализации компонента в качестве значения поля ввода используется значение атрибута `guess`.

Далее показан пример использования компонента в Facelet:

```
<mycomp:sample id="guessComponent" guess="#{GuessBean.guess}"
               action="#{GuessBean.check}">
    <f:validateLongRange for="inputField" minimum="0" maximum="9"/>
</mycomp:sample>
```

В качестве атрибута `guess` передается отложенное EL-выражение, связанное со свойством управляемого бина, а для атрибута `action` – отложенное EL-выражение, связанное с методом управляемого бина. Так как отложенное EL-выражение, по сути, указывает на свойство объекта, а не содержит конкретное значение, то поле ввода `guessForm:guessField` в компоненте `<mycomp:sample>` становится связанным со свойством `GuessBean.guess`. Кроме того, для этого поля определено ограничение на диапазон допустимых значений (вложенное действие `<f:validateLongRange>`), так как к нему предоставлен доступ по внешнему имени `inputField`.

Поддержка AJAX в технологии Facelets

AJAX (от англ. Asynchronous Javascript and XML — «асинхронный JavaScript и XML») — подход к построению интерактивных пользовательских интерфейсов веб-приложений, заключающийся в «фоновом» обмене данными браузера с веб-сервером. В результате, при обновлении данных, веб-страница не перезагружается полностью и веб-приложения становятся более быстрыми и удобными.

AJAX — не самостоятельная технология, а концепция использования нескольких смежных технологий. AJAX базируется на двух основных принципах:

1) использование технологии динамического обращения к серверу «на лету», без перезагрузки всей страницы полностью, например:
- с использованием XMLHttpRequest (основной объект);
- через динамическое создание дочерних фреймов;
- через динамическое создание тега <script>.

2) использование DHTML для динамического изменения содержания страницы.

В качестве формата передачи данных обычно используются JSON или XML.

Впервые термин AJAX был публично использован 18 февраля 2005 года в статье Джесси Джеймса Гарретта (Jesse James Garrett) «Новый подход к веб-приложениям», выдержки из которой приведены далее.

Классическая модель веб-приложения действует следующим образом: большинство действий пользователя отправляют обратно на сервер HTTP-запрос. Сервер производит необходимую обработку - получает данные, обрабатывает числа, взаимодействует с различными унаследованными системами и затем выдаёт HTML-страницу клиенту. Эта модель заимствована из первоначального применения веба как гипертекстовой среды.

Приложение Ajax исключает взаимодействие типа старт-стоп-старт-стоп путём введения механизма Ajax как промежуточного слоя между пользователем и сервером. Может показаться что добавляя новый уровень в приложение можно только замедлить его реакцию, но в действительности наоборот.

Вместо того чтобы загружать страницу в начале пользовательской сессии браузер загружает движок Ajax, написанный на JavaScript и обычно спрятанный в скрытый фрейм. Этот движок отвечает за формирование пользовательского интерфейса и взаимодействие с сервером от имени пользователя. Движок Ajax позволяет производить взаимодействие с пользователем асинхронно, то есть независимо от взаимодействия с сервером. Таким образом пользователю больше не нужно наблюдать пустое окно браузера и курсор в виде песочных часов в ожидании действий сервера.

Рисунок 1: Сравнение традиционной модели веб-приложений (слева) с моделью Ajax (справа).

Каждое действие пользователя, которое обычно производит HTTP-запрос, теперь вместо этого принимает форму JavaScript-вызова движка Ajax. Каждый ответ на действие пользователя, не требующее обращения к серверу, как то простая проверка данных, редактирование данных в памяти, и даже некоторая навигация, выполняется движком самостоятельно. Если же для ответа требуется информация с сервера, например загрузка дополнительного интерфейсного кода, передача данных для обработки, или получение новых данных, то движок производит необходимые запросы асинхронно, обычно при помощи XML, не прерывая взаимодействия пользователя с приложением.

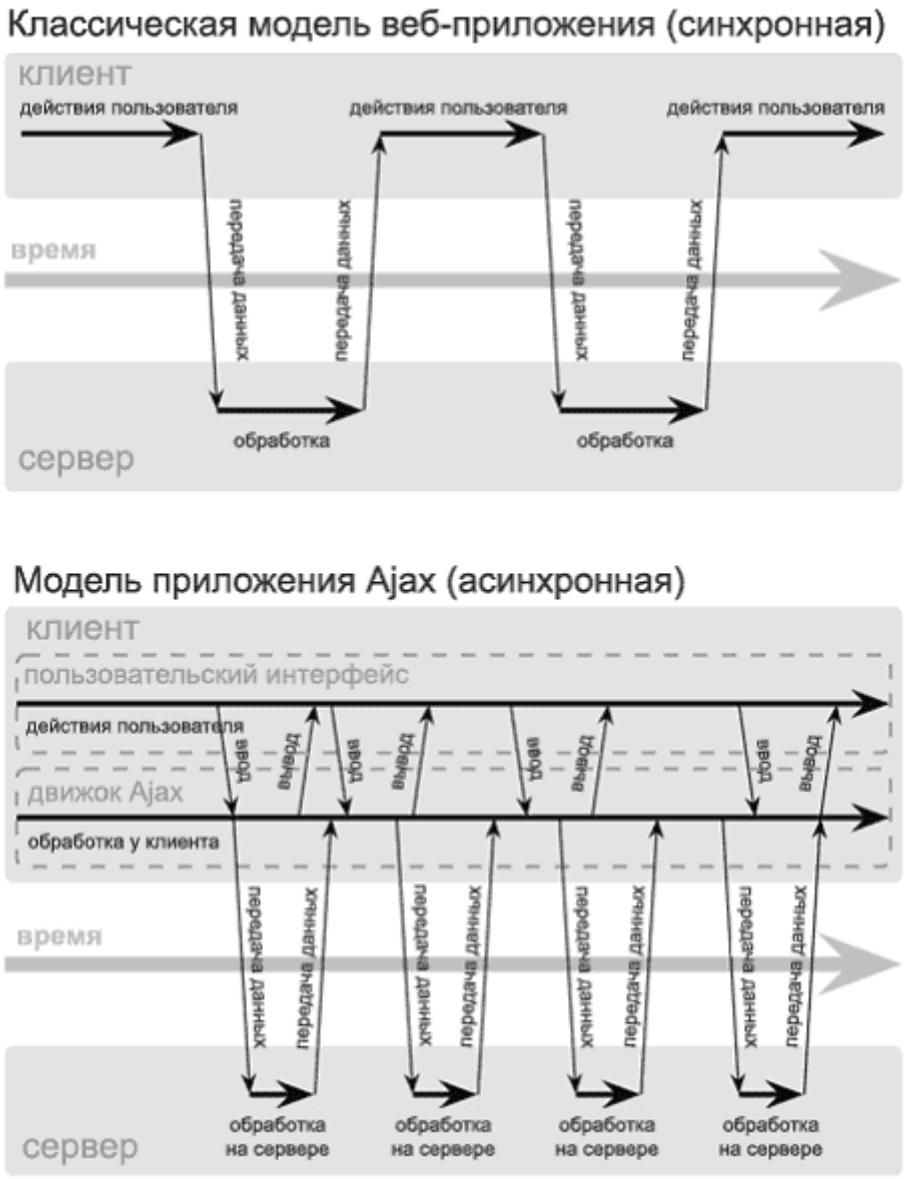

Рисунок 2: Сравнение диаграмм взаимодействия традиционного веб-приложения (сверху) и асинхронного приложения Ajax (снизу).

Технология Facelets предоставляет два способа добавить в приложение возможности AJAX: 1) декларативный, с помощью действия `<f:ajax>` и 2) с использованием JavaScript API, закрепленного в спецификации JSF.

Действие `<f:ajax>` можно применять двояко: как вложенное в один компонент интерфейса пользователя, либо как охватывающее группу компонентов. В любом случае это действие обеспечивает отправку запроса на сервер при возникновении события, указываемого в атрибуте `event`. Если атрибут `event` не указан, то ожидается «событие по умолчанию» для компонента (например, для ссылок и кнопок это действие `action`). Запрос, отправляемый на сервер, содержит значения лишь тех компонентов, которые указаны (через пробел) в атрибуте `execute`. Последний может содержать также специальное значение `@this`, `@form`, `@all` или `@none`. В результате обработки запроса на сервере формируется «частичный ответ», который содержит только компоненты пользовательского интерфейса, перечисленные в атрибуте `render`, для которого действуют такие же специальные значения, что и для `execute`.

Рассмотрим пример добавления возможностей AJAX в форму для отгадывания загаданного числа:

```
<form jsfc="h:form">
```

```
    <h:messages id="msgs"/>
    <f:validateBean>
        Try <span id="tryCount" jsfc="h:outputText"
                value="#{GuessBean.count}"/>:
        <input type="text" id="guessField" jsfc="h:inputText"
                value="#{GuessBean.guess}"/>
    </f:validateBean>
    <input type="submit" jsfc="h:commandButton" action="#{GuessBean.check}"
value="Check">
        <f:ajax execute="guessField" render="tryCount msgs"/>
    </input>
</form>
```

При нажатии на кнопку Check на сервер отправляется запрос, содержащий значение поле ввода `guessField`. На сервере выполняется action-метод `GuessBean.check`. Если этот метод возвращает идентификатор представления, соответствующий странице, с которой пришел запрос, то в браузер возвращается частичный ответ, содержащий только данные для компонентов пользовательского интерфейса с идентификаторами `tryCount` (счетчик попыток) и `msgs` (сообщения). Обновлять поле ввода `guessField` не имеет смысла, так как значение связанного с ним свойства `GuessBean.guess` не изменяется в ходе проверке числа. Обновление компонента `msgs` в данном случае необходимо, так как он будет содержать ошибки валидации введенного значения свойства `GuessBean.guess`.