JSF 2 fu.: Часть 1. Упрощаем разработку Web-приложений

Упрощение навигации, устранение XML-конфигурации и упрощение обращения к ресурсам с помощью JSF 2

Дэвид Джири
президент
Clarity Training, Inc.

14.01.2011

С помощью инфраструктуры Java™Server Faces (JSF) версии 2.0 можно легко создавать надежные Web-приложения, использующие Ajax. В этой первой из трех статей цикла, написанного членом экспертной группы JSF 2.0 Дэвидом Джири, мы рассмотрим, как с помощью новых возможностей JSF 2 можно оттачивать свои навыки, как мастер кунг-фу. В этой статье вы узнаете, как с помощью JSF 2 упростить разработку, заменив XML-конфигурацию аннотациями и соглашениями, упростив навигацию и обеспечив простой доступ к ресурсам. Также вы увидите, как можно использовать в JSF-приложениях Groovy.

Больше статей из этой серии

Идет непрекращающийся спор о том, где лучше создаются инфраструктуры Web-приложений: в академических кругах силами теоретиков или в реальном мире, где инфраструктуры рождаются в суровых условиях, когда необходимо быстро разработать решение для практических задач. Интуитивно понятно, что практика выигрывает у теории, и, я думаю, при более внимательном рассмотрении окажется, что интуиция права.

Инфраструктура JSF 1 создавалась теоретиками, и нельзя сказать, что результат был очень убедительным. Однако кое-что в JSF было сделано правильно — она позволяла участникам рынка разрабатывать на ее основе множество инноваций. Сначала появилась инфраструктура Facelets, предлагающая замену технологии JavaServer Pages (JSP). Затем появились Rich Faces - мощная JSF-библиотека поддержки Ajax; ICEFaces – новый подход к работе с Ajax в JSF; Seam; Spring Faces; компоненты Woodstock; JSF Templating и т.д. Все эти проекты с открытым кодом для JSF были созданы разработчиками, нуждавшимися в соответствующей функциональности.

Экспертная группа JSF 2.0 в итоге стандартизировала самое лучшее из этих проектов с открытым кодом. Хотя спецификация JSF 2 фактически создавалась теоретиками, в ней также учитывались и инновации из реального мира. Оглядываясь назад, можно сказать, что работа экспертной группы была относительно проста, так как мы «стояли на плечах» таких гигантов, как Гевин Кинг (Gavin King, Seam), Александр Смирнов (Rich Faces), Тед Годдард (Ted Goddard, ICEFaces) и Кен Полсон (Ken Paulson, JSF Templating). Фактически все эти люди входили в экспертную группу JSF 2. Поэтому, во многих отношениях JSF 2 сочетает в себе лучшие аспекты академичности и практичности. JSF 2 является огромным шагом вперед по сравнению с предшествующей версией.

Эта статья - первая в цикле из трех частей. В этом цикле мы преследуем две цели: рассмотреть замечательные возможности JSF 2 и показать, как наилучшим образом их использовать. Я буду перемежать изложение советами по наилучшему использованию JSF. В данной статье мы рассмотрим следующие советы:

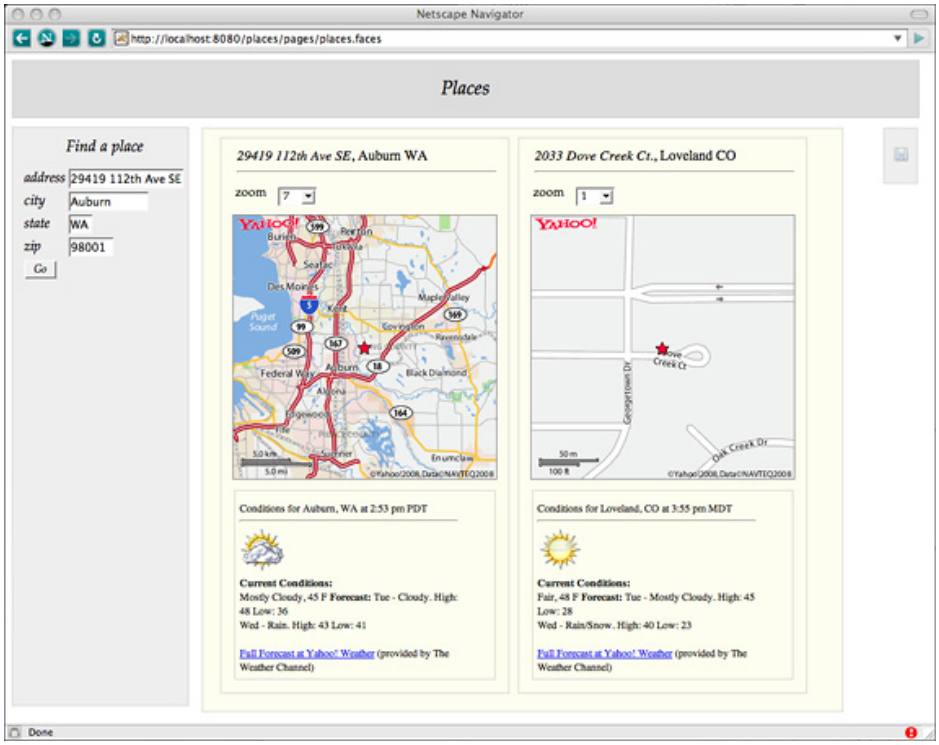
- Совет 1: избавляемся от XML-конфигурации
- Совет 2: упрощаем навигацию
- Совет 3: применяем Groovy
- Совет 4: пользуемся обработкой ресурсов

Однако сначала мы познакомимся с приложением-примером, которое мы будем использовать в этой серии статей. Вы можете загрузить исходный код этого приложения..

Пример основанного на картах приложения, использующего набор Web-сервисов

На рисунке 1 показан JSF-интерфейс — я буду его называть приложением places (места), в котором с помощью Web-сервисов Yahoo! адреса конвертируются в карты различных масштабов и прогнозы погоды:

Рисунок 1. Просматриваем карты и информацию о погоде, полученные от Web-сервисов Yahoo!

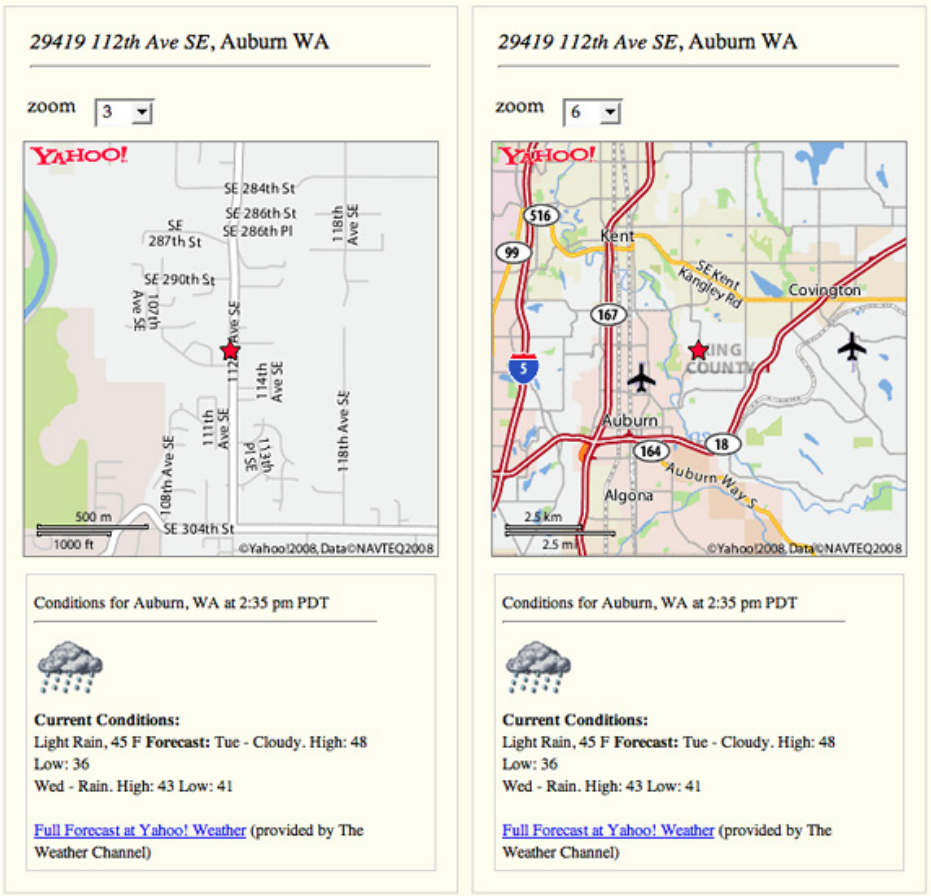


Чтобы добавить новое место, нужно заполнить форму адреса и нажать на кнопку Go. Приложение передает введенный адрес двум Web-сервисам: Yahoo! Maps и Yahoo! Weather.

Сервис Map возвращает для указанного адреса 11 URL-адресов карт различного масштаба, хранящихся на сервере Yahoo!. Сервис Weather возвращает фрагмент разметки HTML. URL-адреса изображений и фрагменты HTML можно легко отображать в представлениях JSF с помощью тегов `<h:graphicImage>` и `<h:outputText>`, соответственно.

Приложение places позволяет вводить сколько угодно адресов. Можно даже использовать один и тот же адрес несколько раз, как показано на рисунке 2, иллюстрирующем карты разных масштабов:

Рисунок 2. Карты разных масштабов



Структура приложения

В приложении places имеется четыре управляемых bean-компонента, перечисленных в таблице 1:

Таблица 1. Управляемые bean-компоненты приложения places

Имя управляемого bean-компонента	Класс	Область видимости
`mapService`	`com.clarity.MapService`	Приложение
`weatherService`	`com.clarity.WeatherService`	Приложение
`places`	`com.clarity.Places`	Сеанс
`place`	`com.clarity.Place`	Запрос

Запускаем приложение

Для запуска приложения places вам необходимо получить так называемый ID приложения по адресу developer.yahoo.com/maps/ajax, который нужен для использования Web-сервисов Yahoo!. Нажмите на этой странице кнопку **Get an App ID**. Получив свой ID, замените значение переменной `YOUR_ID_HERE` в файлах `MapService.java` и `WeatherService.java`.

Приложение хранит список объектов `Place`, изображенный на рисунке 1, в области видимости сеанса и поддерживает объект `Place` в области видимости запроса. Также

приложение предоставляет простые API к Web-сервисам Yahoo! map и weather с помощью управляемых компонентов `mapService` и `weatherService` соответственно, которые находятся в области видимости всего приложения.

Создать новое место очень просто. В листинге 1 приведен код формы адреса из представления, показанного на рисунке 1:

Листинг 1. Форма адреса

```
<h:form>
  <h:panelGrid columns="2">
    #{msgs.streetAddress} <h:inputText value="#{place.streetAddress}" size="15"/>
    #{msgs.city}          <h:inputText value="#{place.city}"          size="10"/>
    #{msgs.state}         <h:inputText value="#{place.state}"         size="2"/>
    #{msgs.zip}           <h:inputText value="#{place.zip}"           size="5"/>

    <h:commandButton value="#{msgs.goButtonText}"
        style="font-family:Palatino;font-style:italic"
        action="#{place.fetch}"/>

  </h:panelGrid>
</h:form>
```

Когда пользователь нажимает кнопку Go и отправляет форму, JSF вызывает метод действия кнопки: `place.fetch()`. Этот метод посылает информацию от Web-сервисов методу `Place.addPlace()`, который создает новый экземпляр класса `Place`, инициализирует его данными, переданными в метод, и сохраняет в его в области видимости запроса.

В листинге 2 показан метод `Place.fetch()`:

Листинг 2. Метод `Place.fetch()`

```
public class Place {
  ...
  private String[] mapUrls
  private String weather
  ...
  public String fetch() {
    FacesContext fc = FacesContext.getCurrentInstance()
    ELResolver elResolver = fc.getApplication().getELResolver()

    // Получаем карты

    MapService ms = elResolver.getValue(
      fc.getELContext(), null, "mapService")

    mapUrls = ms.getMap(streetAddress, city, state)

    // Получаем погоду

    WeatherService ws = elResolver.getValue(
      fc.getELContext(), null, "weatherService")

    weather = ws.getWeatherForZip(zip, true)

    // Получаем список мест

    Places places = elResolver.getValue(
      fc.getELContext(), null, "places")

    // Добавляем в список новое место
```

```
    places.addPlace(streetAddress, city, state, mapUrls, weather)

    return null
  }
}
```

Метод `Place.fetch()` с помощью имеющегося в JSF механизма разрешения имен переменных находит управляемые bean-компоненты `mapService` и `weatherService` и использует их для получения карты и информации о погоде от Web-сервисов Yahoo!. Затем в методе `fetch()` вызвается метод `places.addPlace()`, который, используя информацию об адресе, картах и погоде, создает новый объект `Place` в области видимости запроса.

Обратите внимание, что метод `fetch()` возвращает `null`. Так как `fetch()` – это метод действия, ассоциированный с кнопкой, то возвращаемое значение `null` заставляет JSF перезагрузить то же самое представление. Как показано в листинге 3, в этом представлении отображаются все места, видимые в этом сеансе пользователя:

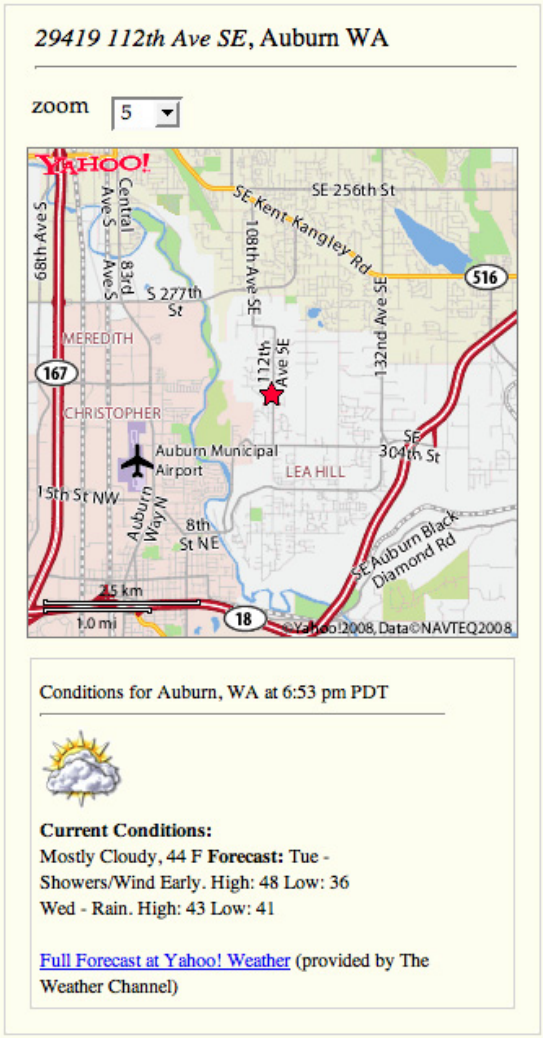
Листинг 3. Отображаем места в представлении

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

  <h:form>
    <!-- Обходим список мест -->
    <ui:repeat value="#{places.placesList}" var="place">

      <div class="placeHeading">
        <h:panelGrid columns="1">

          <!-- Адрес -->
          <h:panelGroup>
            <div style="padding-left: 5px;">
              <i><h:outputText value="#{place.streetAddress}"/></i>,
              <h:outputText value="#{place.city}"/>
              <h:outputText value="#{place.state}"/>
              <hr/>
            </div>
          </h:panelGroup>

          <!-- надпись и выпадающий список со значениями масштаба -->
          <h:panelGrid columns="2">
            <!-- надпись -->
            <div style="padding-right: 10px;margin-bottom: 10px;font-size:14px">
              #{msgs.zoomPrompt}
            </div>

            <!-- выпадающий список -->
            <h:selectOneMenu onchange="submit()"
                 value="#{place.zoomIndex}"
                 valueChangeListener="#{place.zoomChanged}"
                 style="font-size:13px;font-family:Palatino">

              <f:selectItems value="#{places.zoomLevelItems}"/>

            </h:selectOneMenu>
          </h:panelGrid>

          <!-- Карта -->
```

```
          <h:graphicImage url="#{place.mapUrl}" style="border: thin solid gray"/>

        </h:panelGrid>

        <!-- Погода -->
        <div class="placeMap">
          <div style="margin-top: 10px;width:250px;">
            <h:outputText style="font-size: 12px;"
              value="#{place.weather}"
              escape="false"/>
          </div>
        </div>
      </div>

    </ui:repeat>
  </h:form>

</ui:composition>
```

В коде из Листинга 3 для обхода списка хранящихся в сеансе пользователя мест используется тег Facelets `<ui:repeat>`. Для каждого места результат выглядит так, как показано на рисунке 3:

Рисунок 3. Место, отображаемое представлением



Изменяем масштаб

У меню zoom (см. Рисунок 3 и Листинг 3) имеется атрибут `onchange="submit()"`, поэтому JavaScript-функция `submit()` отправляет окружающую меню форму, когда пользователь выбирает значение масштаба. В результате этого JSF вызывает слушатель, связанный с изменением значения меню — метод `Place.zoomChanged()`, показанный в листинге 4:

Листинг 4. Метод `Place.zoomChanged()`

```
public void zoomChanged(ValueChangeEvent e) {
  String value = e.getComponent().getValue()
  zoomIndex = (new Integer(value)).intValue()
}
```

Метод `Place.zoomChanged()` помещает значение масштаба в переменную `zoomIndex` члена класса `Place`. Мы остаемся на этой же странице, поэтому JSF перезагружает ту же самую страницу с картой нового масштаба: `<h:graphicImage url="#{place.mapUrl}..."/>`. Когда изображение отрисовано, JSF вызывает метод `Place.getMapUrl()`, который возвращает URL-адрес карты с текущим масштабом, как показано в листинге 5:

Листинг 5. Метод `Place.getMapUrl()`

```
public String getMapUrl() {
  return mapUrls == null ? "" : mapUrls[zoomIndex]
}
```

Сила Facelets

Если вы ранее работали с JSF 1, возможно, в коде примеров для этой статьи вы заметили небольшие отличия, появившиеся в JSF 2. Во-первых, мы используем появившуюся в JSF 2 новую технологию отображения Facelets вместо JSP. Как вы увидите в следующих статьях этого цикла, Facelets предоставляет много мощных возможностей, помогающих создавать надежные, гибкие и расширяемые пользовательские интерфейсы. Однако в предыдущих листингах мы не вдавались в них подробно. Одно из множества небольших улучшений, которые технология Facelets привнесла в JSF, является возможность помещать значения выражений JSF непосредственно в страницу XHTML. Например, в листинге 1, я разместил непосредственно на странице такие выражения, как `#{msgs.city}`. В JSF 1 эти выражения нужно было оборачивать в элемент `<h:outputText>`, следующим образом: `<h:outputText value="#{msgs.city}"/>`. Однако помните, что в целях безопасности нужно всегда экранировать поступающий от пользователя текст. Поэтому, например, в листинге 3 для отображения информации о месте я использую `<h:outputText>`, который экранирует текст по умолчанию.

Также, говоря о Facelets, следует упомянуть тег `<ui:composition>`, использованный в листинге 3. Этот тег обозначает, что XHTML-страницу из листинга 3 будут включать в другие XHTML-страницы. Композиции Facelets являются центральным компонентом концепции *шаблонов* Facelets, похожей на популярную инфраструктуру Apache Tiles. В следующей статье этого цикла мы расскажем о шаблонах Facelets и покажем, как структурировать свои представления в соответствии с шаблоном *Composed Method* языка Smalltalk.

Код, который мы видели до сих пор, за исключением использования Facelets не сильно отличается от JSF 1. Теперь мы рассмотрим более существенные отличия. Первым существенным отличием является количество конфигурационной информации в XML, нужной для приложений JSF 2.

Совет 1: избавляемся от XML-конфигурации

Конфигурирование Web-приложений с помощью XML всегда была сомнительным компонентом из-за трудоемкости и подверженности ошибкам. Лучше всего делегировать конфигурацию инфраструктуре, например, с помощью аннотаций, соглашений или предметно-ориентированных языков. Разработчику следует концентрироваться на выполнении задания, а не возиться с деталями подробной XML-конфигурации.

В качестве примера в листинге 6 показаны 20 строк XML-конфигурации, необходимые для декларирования управляемых bean-компонентов в приложении places для JSF 1:

Листинг 6. Декларации управляемых bean-компонентов для JSF 1

```
<managed-bean>
```

```
  <managed-bean-class>com.clarity.MapService</managed-bean-class>
  <managed-bean-name>mapService</managed-bean-name>
  <managed-bean-scope>application</managed-bean-scope>
</managed-bean>

<managed-bean>
  <managed-bean-class>com.clarity.WeatherService</managed-bean-class>
  <managed-bean-name>weatherService</managed-bean-name>
  <managed-bean-scope>application</managed-bean-scope>
</managed-bean>

<managed-bean>
  <managed-bean-class>com.clarity.Places</managed-bean-class>
  <managed-bean-name>places</managed-bean-name>
  <managed-bean-scope>session</managed-bean-scope>
</managed-bean>

<managed-bean>
  <managed-bean-class>com.clarity.Place</managed-bean-class>
  <managed-bean-name>place</managed-bean-name>
  <managed-bean-scope>request</managed-bean-scope>
</managed-bean>
```

В JSF 2 вместо XML-конфигурации используются аннотации классов, как показано в листинге 7:

Листинг 7. Аннотации управляемых bean-компонентов в JSF 2

```
@ManagedBean(eager=true)
public class MapService {
  ...
}

@ManagedBean(eager=true)
public class WeatherService {
  ...
}

@ManagedBean()
@SessionScoped
public class Places {
  ...
}

@ManagedBean()
@RequestScoped
public class Place {
  ...
}
```

По соглашению имя управляемого bean-компонента совпадает с именем класса, за исключением того, что первая буква приводится к нижнему регистру. Поэтому, например, в листинге 7, будут созданы следующие управляемые bean-компоненты (сверху вниз): `mapService`, `weatherService`, `places`, и `place`. Имя управляемого bean-компонента также можно указать явно с помощью атрибута `name` аннотации `ManagedBean`, например: `@ManagedBean(name = "place")`.

В листинге 7, для управляемых bean-компонентов `mapService` и `webService` используется атрибут `eager`. Если атрибут `eager` имеет значение `true`, JSF создает этот управляемый bean-компонент при старте и помещает его в область видимости приложения.

Также с помощью аннотации `@ManagedProperty` можно задать свойства управляемого bean-компонента. В таблице 2 показан полный список имеющихся в JSF 2 аннотаций управляемых bean-компонентов:

Таблица 2. Аннотации управляемых bean-компонентов JSF 2 (аннотации `@...Scoped` допустимы только совместно с аннотациями `@ManagedBean`)

Аннотация управляемого bean-компонента	Описание	Атрибуты
`@ManagedBean`	Регистрирует экземпляр этого класса в качестве управляемого bean-компонента и помещает его в область видимости, указанную одной из аннотаций `@...Scoped`. Если область видимости не указана, JSF помещает bean-компонент в область видимости запроса, а если не указано имя, JSF генерирует имя, конвертируя первую букву имени класса в нижний регистр; например, для класса с именем `UserBean` JSF создает управляемый bean-компонент с именем `userBean`. Атрибуты `eager` и `name` являются необязательными. Эту аннотацию можно использовать только для Java-классов, реализующих конструктор без аргументов.	`eager`, `name`
`@ManagedProperty`	Задает свойства управляемого bean-компонента. Эту аннотацию необходимо помещать перед декларацией переменных-членов класса. Атрибут `name` указывает имя свойства, по умолчанию оно совпадает с именем переменной-члена. Атрибут `value` является значением свойства, им может быть либо строка, либо JSF-выражение, такое как `#{...}`.	`value`, `name`
`@ApplicationScoped`	Помещает управляемый bean-компонент в область видимости приложения.	
`@SessionScoped`	Помещает управляемый bean-компонент в область видимости сеанса.	
`@RequestScoped`	Помещает управляемый bean-компонент в область видимости запроса.	
`@ViewScoped`	Помещает управляемый bean-компонент в область видимости представления.	
`@NoneScoped`	Указывает, что управляемый bean-компонент не имеет области видимости. Управляемые bean-компоненты без области видимости полезны, когда на них ссылаются другие bean-компоненты.	
`@CustomScoped`	Помещает управляемый bean-компонент в специальную область видимости. Специальная область видимости представляет собой карту, доступную авторам страницы. Специальные области видимости позволяют программно задавать видимость и время жизни находящихся в ней bean-компонентов. Атрибут `value` указывает на эту карту.	`value`

Удаление деклараций управляемых bean-компонентов из файла faces-config.xml существенно сокращает размер используемого XML. В JSF 2 от него можно избавиться практически полностью благодаря использованию рассмотренных здесь аннотаций

управляемых bean-компонентов, а также соглашений, как в случае имеющейся в JSF 2 упрощенной обработки навигации.

Совет 2: упрощаем навигацию

В JSF 1 правила перехода описывались с помощью XML. Например, перейти со страницы login.xhtml на страницу places.xhtml можно с помощью правила перехода, указанного в листинге 8:

Листинг 8. Правила перехода в JSF 1

```
<navigation-rule>
  <navigation-case>
    <from-view-id>/pages/login.xhtml</from-view-id>
    <outcome>places</outcome>
    <to-view-id>/pages/places.xhtml</to-view-id>
  </navigation-case>
</navigation-rule>
```

Чтобы избавиться от XML-конфигурации из листинга 8, можно воспользоваться имеющимися в JSF 2 соглашениями по переходу: JSF добавляет xhtml к концу имени действия кнопки и загружает соответствующий файл. Это означает, что можно полностью избавиться от написания правил навигации, не используя при этом аннотации или что-нибудь еще, – достаточно просто следовать соглашению. В листинге 9 действие кнопки называется `places`, поэтому JSF загружает файл places.xhtml:

Листинг 9. Переход в соответствии с соглашением

```
<h:commandButton id="loginButton"
  value="#{msgs.loginButtonText}"
  action="places"/>
```

Для листинга 9 не требуется настраивать навигацию посредством XML. По нажатию на кнопку в листинге 9 загружается файл places.xhtml, но только если этот файл находится в той же директории, что и файл, в котором находится кнопка. Если имя действия не начинается с прямого слеша (`/`), JSF полагает, что это относительный путь. При желании можно указать абсолютный путь, как показано в листинге 10:

Листинг 10. Навигация с абсолютными путями

```
<h:commandButton id="loginButton"
  value="#{msgs.loginButtonText}"
  action="/pages/places"/>
```

Когда пользователь нажимает кнопку из листинга 10, JSF загружает файл /pages/places.xhtml.

По умолчанию JSF при переходе с одной XHTML-страницы на другую использует переадресацию (forward), однако с помощью параметра `faces-redirect`, показанного в листинге 11, можно делать перенаправление с помощью redirect:

Листинг 11. Переход с помощью redirect

```
<h:commandButton id="loginButton"
  value="#{msgs.loginButtonText}"
  action="places?faces-redirect=true"/>
```

Совет 3: используйте Groovy

Самая замечательная вещь в технологии Java – это не язык Java, а виртуальная Java-машина (JVM). В JVM работают такие мощные новые языки, как Scala, JRuby и Groovy, предоставляющие новое измерение, в котором можно писать код. Одним из наиболее популярных таких языков является Groovy — несмотря на странное название, очень мощный язык, представляющий смесь Ruby, Smalltalk и Java (см Ресурсы).

Есть множество причин использовать Groovy, начиная с того, что он гораздо лаконичнее и мощнее, чем Java. Еще две причины – это отсутствие необходимости использовать точки с запятой и приведение типов.

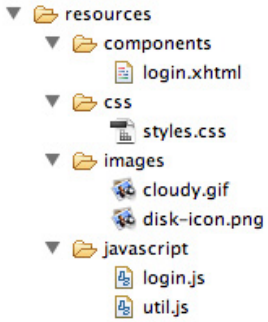
Возможно, вы не заметили, но код листинга 2, для класса `Place` написан на Groovy. Подсказкой может служить отсутствие точек с запятыми, однако следует обратить внимание и на следующую строку кода: `MapService ms = elResolver.getValue(...)`. В Java результат выполнения метода `ElResolver.getValue()`, пришлось бы явно приводить к типу `MapService`, так как этот метод возвращает значение типа `Object`. Groovy же это приведение выполняет самостоятельно.

Groovy можно использовать для любых артефактов JSF, которые обычно пишутся на Java — таких как компоненты, рендереры, валидаторы и конвертеры. На самом деле это не является новинкой JSF 2 — исходные файлы Groovy компилируются в байт-код Java, поэтому .class-файлы, сгенерированные с помощью Groovy, можно использовать так же, как и файлы, скомпилированные с помощью `javac`. Конечно же, как только у вас это заработает, вы захотите узнать, как быстро разворачивать исходные файлы кода Groovy. Для пользователей Eclipse ответ довольно прост: нужно загрузить и установить модуль расширения Groovy Eclipse (см. Ресурсы). В Mojarra - реализации JSF от Sun - имеется явная поддержка Groovy начиная с версии 1.2_09 (см. Ресурсы).

Совет 4: используйте обработчики ресурсов

JSF 2 предоставляет стандартный механизм определения ресурсов и доступа к ним. Ресурсы размещаются в директории верхнего уровня с именем resources, а доступ к ним из представлений осуществляется с помощью специальных тегов. Например, на рисунке 4 показаны ресурсы приложения places:

Рисунок 4. Ресурсы приложения places



Единственным требованием к ресурсу является то, чтобы он находился либо в директории resources, либо в ее поддиректории. Поддиректории директории resources можно называть как угодно.

В коде представления доступ к ресурсам осуществляется с помощью двух тегов JSF 2: `<h:outputScript>` и `<h:outputStylesheet>`. Эти теги взаимодействуют с тегами JSF 2 `<h:head>` и `<h:body>`, как показано в листинге 12:

Листинг 12. Доступ к ресурсам из XHTML

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

  <h:head>
    ...
  </h:head>

  <h:body>
    <h:outputStylesheet library="css" name="styles.css" target="body"/>
    <h:outputScript library="javascript" name="util.js" target="head"/>
    ...
  </h:body>
</html>
```

В тегах `<h:outputScript>` и `<h:outputStylesheet>` имеется два атрибута, идентифицирующих сценарий или таблицу стилей: `library` и `name`. Атрибут `library` соответствует поддиректории директории resources, в которой находится этот ресурс. Например, если у вас есть таблица стилей, которая находится в директории resources/css/en directory, атрибут `library` будет иметь значение `css/en`. Атрибут `name` – это имя самого ресурса.

Перемещаемые ресурсы

Важно заметить, что разработчики могут указывать, где именно на странице должен появиться ресурс. Например, если сценарий JavaScript, помещен в тело страницы, браузер выполнит его сразу после загрузки страницы. С другой стороны, если поместить код JavaScript в заголовок (тег head) страницы, то он будет выполнен только при вызове. Так как местонахождение ресурса может влиять на то, как он будет использоваться, необходимо иметь возможность указать, где вы хотите поместить данный ресурс.

В JSF 2 ресурсы являются *перемещаемыми* в том смысле, что можно указать, где именно на странице вы хотите их разместить. Местонахождение ресурса указывается с помощью атрибута `target`; например, в листинге 12 я поместил таблицу стилей CSS в тело, а сценарий JavaScript – в заголовок страницы.

Иногда бывает нужно получить доступ к ресурсу с помощью языка выражений JSF (JSF expression language или EL). Например, в листинге 13 показано, как с помощью тега `<h:graphicImage>` можно получить доступ к изображению:

Листинг 13. Получаем доступ к ресурсу с помощью языка выражений JSF

```
<h:graphicImage value="#{resource['images:cloudy.gif']}"/>
```

Альтернатива листингу 13, не использующая EL

Признаем, что синтаксис из листинга 13 не является интуитивно понятным. Фактически там осуществляется доступ к карте, которую JSF создала для хранения ресурсов, поэтому вам редко нужно будет использовать этот синтаксис. В действительности посредством тега `<h:graphicImage/>` можно получить доступ к изображению и без использования EL, например, следующим образом: `<h:graphicImage library="images" name="cloudy.gif"/>`

Синтаксис доступа к ресурсам в EL выглядит так: `resource['`*`LIBRARY`*`:`*`NAME`*`']`, где *`LIBRARY`* и *`NAME`* соответствует атрибутам `library` и `name` тегов `<h:outputScript>` and `<h:outputStylesheet>`.

А дальше?

В этой статье мы рассмотрели лишь малую часть функциональности JSF 2: аннотации управляемых bean-компонентов, упрощенную переход по страницам и поддержку ресурсов. В двух других статьях этого цикла мы рассмотрим Facelets, составные компоненты JSF 2 и встроенную поддержку Ajax.

Загрузка

Описание	Имя	Размер
Исходный код	jsf2fu1.zip	1.9 МБ

Ресурсы

Научиться

- Познакомьтесь с оригиналом статьи: JSF 2 fu, Part 1: Streamline Web application development (developerWorks, май 2010 г.).(EN)
- Домашняя страница JSF: здесь можно найти множество ресурсов о разработке с помощью JSF.(EN)
- Groovy+Mojarra (блог Райан Лубке, апрель 2008г.): Лубке, инженер компании Sun, рассуждает об использовании Groovy с Mojarra 1.2_09.(EN)
- Public Access to JSF 2.0 JSR-314-EG Discussions Now Available (блог Эда Бёрна, java.net, март 2009г.): узнайте, как подписаться на рассылку экспертной группы JSF 2.(EN)
- Блог Роджера Китейна: Роджер Китейн (Roger Kitain) и Эд Бёрнс (Ed Burns) являются ведущими разработчиками спецификации JSF 2.0.(EN)
- В блоге Джима Дрисколла (Jim Driscoll) имеется много заметок о JSF 2.(EN)
- Блог Райана Лубке: Райан Лубке (Ryan Lubke) является автором первого образца реализации JSF 2.(EN)
- Научитесь работать с Groovy с помощью серии статей *Practically Groovy* (Эндрю Гловер и Скотт Дэвис, developerWorks).(EN)

Получить продукты и технологии

- JSF: загрузите JSF 2.0.(EN)
- Используйте модуль расширения Groovy Eclipse для упрощения разработки с помощью Groovy.(EN)

Обсудить

- Читайте блоги developerWorks и участвуйте в жизни сообщества developerWorks.(EN)

Об авторе

Дэвид Джири

Автор, лектор и консультант Дэвид Джири является президентом компании Clarity Training, Inc., Он обучает разработчиков создавать Web-приложения с использованием JSF и Google Web Toolkit (GWT). Он участвовал в экспертных группах JSTL 1.0 и JSF 1.0/2.0, был соавтором сертификационного экзамена Web Developer Certification Exam компании Sun, а также принимал участие в проектах с открытым кодом, в том числе Apache Struts и Apache Shale. Книга Дэвида *Graphic Java Swing* стала одной из самых продаваемых книг о Java, а *Core JSF* (в соавторстве с Кэем Хорстманом) - одна из самых продаваемых книг о JSF. Дэвид регулярно выступает на конференциях и встречах пользовательских групп. Он является регулярным участником конференций NFJS с 2003 года, проводил курсы в университете Java и дважды удостаивался звания JavaOne rock star.