Основные паттерны J2EE
Composite View

Ситуация

Сложные Web-страницы представляют контент из многочисленных источников данных, используя для этого подвиды (subview), образующие общую страницу. Кроме того, в разработке и поддержке таких страниц участвует целая команда специалистов из разных направлений.

Задача

Вместо создания механизма, позволяющего собирать страницы из модульных, мелких частей, их конструируют путем внедрения кода форматирования в каждый вид напрямую.

Из-за дублирования кода изменять разметку в большом количестве видов очень сложно, при этом могут возникать ошибки.

Требования

- Мелкие части контента вида часто изменяются.
- Многие составные виды используют схожие подвиды. Эти составляющие оформляются при помощи различных блоков шаблонного текста или появляются в различных точках страницы.
- Когда подвиды встроены напрямую и дублируются во множестве видов, изменять разметку и поддерживать код становится гораздо сложнее.
- Внедрение напрямую в виды часто изменяющихся блоков шаблонного текста также оказывает потенциальное воздействие на доступность и администрирование системы. Возможно потребуется перезагрузка сервера, прежде чем клиенты увидят изменения или обновления этих шаблонных компонентов.

Решение

Используйте составные виды, включающие в себя множество мелких подвидов. Каждый компонент шаблона может быть динамически включен в общую страницу, а ее разметкой можно управлять независимо от контента.

Данное решение предусматривает создание составного вида, которое базируется на включении и замене модульных динамических и статических фрагментов шаблона. Повторное использование отдельных частей вида поддерживается посредством модульной конструкции. Составной вид подходит для генерации страниц, содержащих видимые компоненты, которые комбинируются различными способами. Такой сценарий подходит, например, при создании порталов, содержащих на одной странице большое число независимых подвидов (поля новостей, информация о погоде и т.п.). Управление разметкой страницы ведется независимо от контента подвида.

Другим преимуществом данного паттерна является то, что Web-дизайнеры могут создавать макет разметки сайта, вставляя статическое содержимое в каждую шаблонную область. По мере продвижения разработки сайта на место «заполнителей» ставится реальное содержимое.

На рисунке 7.17 показан скриншот официальной домашней страницы Java *java.sun.com*. Обозначим на нем четыре области: навигация, поиск, тематическая статья и заголовки. И хотя содержимое каждого компонента подвида может быть получено из отдельного независимого источника, все они «бесшовно» соединены в одну составную страницу.

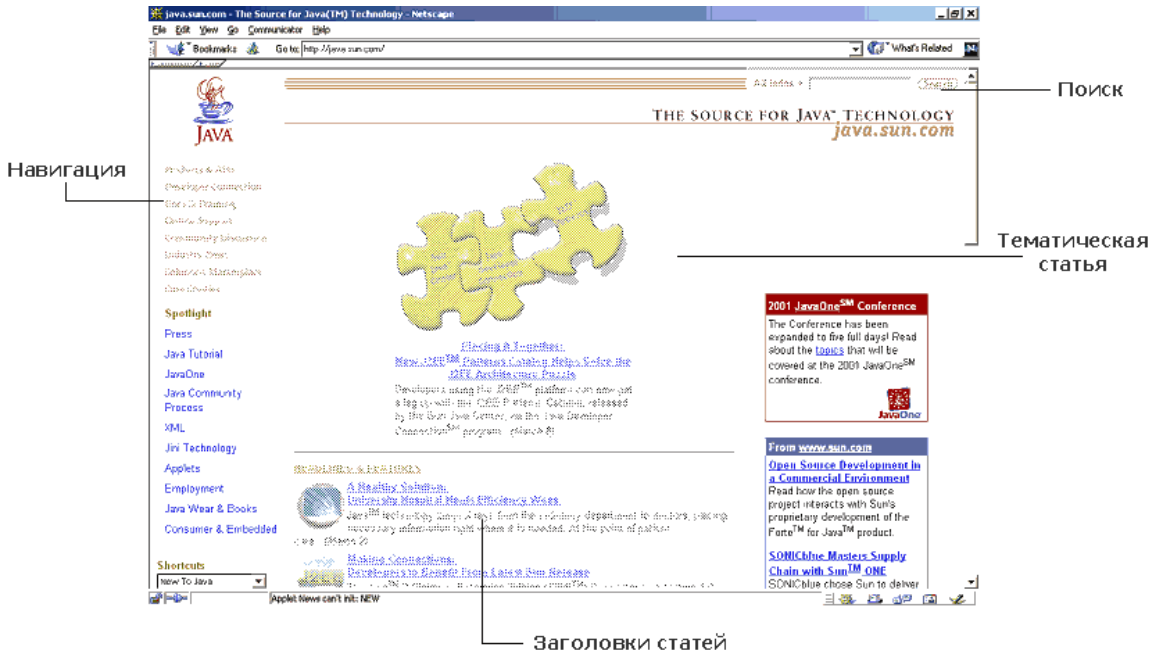

Рисунок 7.17 Скриншот модульной страницы, включающей области поиска, навигации, тематической статьи и заголовков

Этот паттерн не лишен недостатков. Увеличение гибкости приводит к более сложному рабочему циклу. Кроме этого, использование более сложного механизма разметки усложняет управление и разработку, так как требуется поддерживать большее количество артефактов, а уровень реализации недостаточен для понимания.

Структура

На рисунке 7.18 показана классовая диаграмма паттерна Composite View.

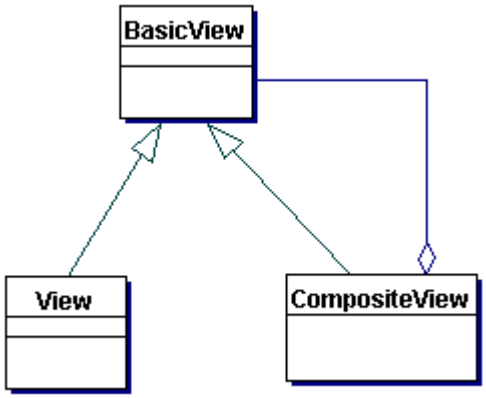

Рисунок 7.18 Классовая диаграмма паттерна Composite View

Участники и обязательства

На рисунке 7.19 представлена циклограмма паттерна Composite View.

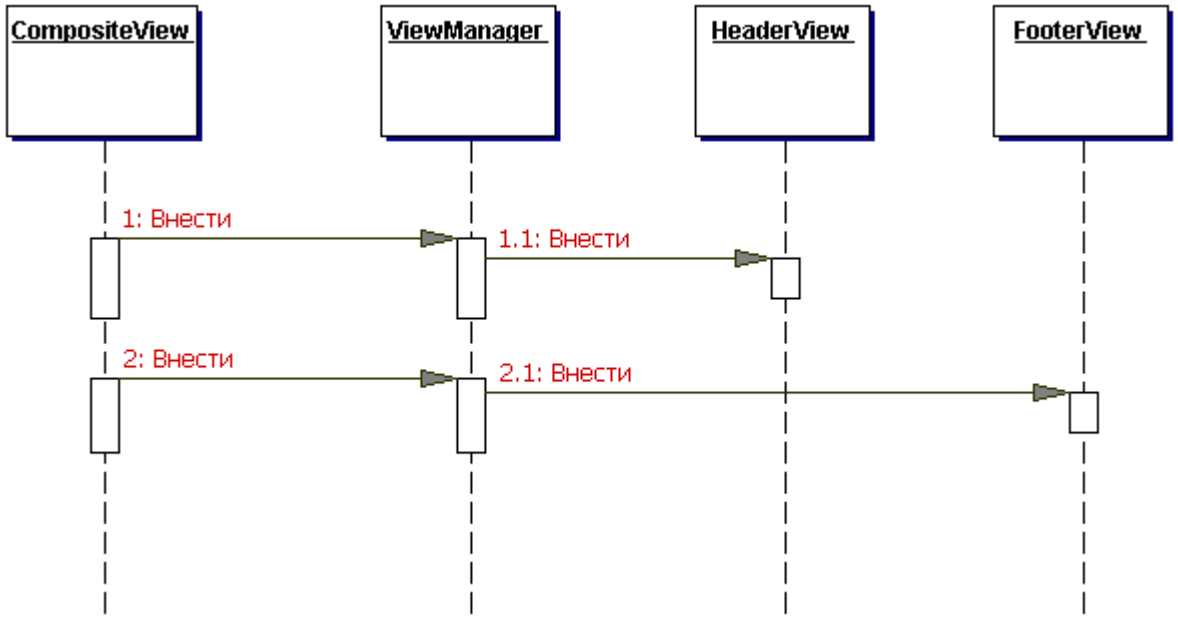

Рисунок 7.19 Циклограмма паттерна Composite View

Composite View

Составной вид состоит из множества подвидов.

View Manager

View Manager отвечает за внедрение частей шаблонных фрагментов в составной вид. View Manager может являться частью стандартного рабочего механизма JSP-страницы и иметь форму тэга include (`<jsp:include>`) JSP-страницы. Он может также быть инкапсулированным в JavaBean-хелпер (JSP page 1.0+) или хелпер заказного тэга (JSP page 1.1+). Все это придает функциональности большую прочность.

Выгода от использования другого механизма (не стандартного тэга include) заключается в простоте обычного включения. Например, определенные шаблонные фрагменты можно включать только в том случае, если пользователь выполняет определенную роль или удовлетворены конкретные условия системы. Следовательно, использование компонента helper как View Manager допускает управление структурой страницы как единым целым. Это может пригодиться при создании страницы, разметку которой можно использовать повторно.

Included View

Included view является подвидом, а именно частью общего вида. Included view также может быть составным и включать в себя множество других подвидов.

Стратегии

Стратегия JSP page View

См. главу «JSP page View Strategy».

Servlet View Strategy

См. главу «Servlet View Strategy».

Стратегия JavaBean View Management

Стратегия JavaBean View management реализована с использованием JavaBean-компонентов, как показано в примере 7.22. Вид передает полномочия JavaBean-компоненту, который реализует заказную логику для управления разметкой и

композицией вида. Разметка страницы может базироваться на информации о ролях пользователя или политике безопасности. В результате получается более мощная функциональная структура, чем стандартная JSP-страница. И хотя данная стратегия семантически эквивалентна стратегии Custom Tag View Management, она не так изящна, из-за внедрения в вид кода скриптлета.

Использование стратегии JavaBean View Management требует меньшей предварительной работы, чем в случае Custom Tag View Management, так как создавать JavaBean-компоненты и интегрировать их в среду JSP-страницы достаточно просто. Кроме того, в JavaBean-компонентах разбираются даже новички. Данная стратегия проще и с позиции управления, потому что готовые JavaBean-компоненты являются единственным артефактом, требующим управления и настройки.

Пример 7.22 Стратегия JavaBean View Management

```
<%@page
  import="corepatterns.compositeview.beanhelper.ContentHelper" %>

<% ContentHelper personalizer = new
  ContentHelper(request); %>


<table valign="top" cellpadding="30%"  width="100%">
     <% if (personalizer.hasWorldNewsInterest() ) { %>
        <tr>
            <td><jsp:getProperty name="feeder"
               property="worldNews"/></td>
        </tr>
        <%
        }
        if ( personalizer.hasCountryNewsInterest() ) {
        %>
        <tr>
            <td><jsp:getProperty name="feeder"
               property="countryNews"/></td>
        </tr>
        <%
        }

        if ( personalizer.hasCustomNewsInterest() ) {
        %>
        <tr>
            <td><jsp:getProperty name="feeder"
               property="customNews"/></td>
        </tr>
        <%
        }

        if ( personalizer.hasAstronomyInterest() ) {
        %>

        <tr>
            <td><jsp:getProperty name="feeder"
                property="astronomyNews"/></td>
            </tr>
        <%
        }
        %>
    </table>
```

Стратегия Standard Tag View Management

Стратегия View management реализована с использованием стандартных тэгов JSP-страниц, например `<jsp:include>`. Реализовать данную стратегию очень просто. Однако данная стратегия не дает той мощности и гибкости, которую обеспечивает Custom Tag View Management, так как разметка отдельных страниц внедрена в сами страницы. И хотя данная стратегия позволяет динамически изменять основной контент, любые изменения разметки в масштабах сайта потребуют отдельного изменения каждой JSP-страницы, что видно из примера 7.23.

Пример 7.23 Стратегия Standard Tag View Management

```
<html>
<body>
<jsp:include
  page="/jsp/CompositeView/javabean/banner.html"
  flush="true"/>
<table width="100%">
  <tr align="left" valign="middle">
    <td width="20%">
    <jsp:include
    page="/jsp/CompositeView/javabean/ProfilePane.jsp"
      flush="true"/>
    </td>
    <td width="70%" align="center">
    <jsp:include
      page="/jsp/CompositeView/javabean/mainpanel.jsp"
      flush="true"/>
    </td>
  </tr>
</table>
<jsp:include
  page="/jsp/CompositeView/javabean/footer.html"
    flush="true"/>
</body>
</html>
```

При создании составного отображения при помощи стандартных тэгов можно включать в него и статическое (например HTML-файл), и динамическое (например JSP-страницу) содержимое. Кроме того, содержимое можно включать во время трансляции или во время рабочего цикла. Если содержимое включается в процессе трансляции, то отображаемая страница останется без изменений до тех пор, пока JSP-страница не будет повторно скомпилирована. Иными словами, страница генерируется один раз при компиляции JSP-страницы. В примере 7.24 представлена выдержка из JSP-страницы, которая генерирует составную страницу во время трансляции при помощи стандартной директивы включения `<%@ include %>`.

Включение содержимого во время рабочего цикла подразумевает, что изменения основных подвидов будут отображаться в общей странице при следующем доступе к ней клиента. Такой подход более динамичен и может быть выполнен при помощи стандартного тэга JSP-страницы `<jsp:include>`, как показано в примере 7.24. Разумеется, при таком подходе имеют место некоторые дополнительные затраты в рабочем цикле, связанные с типом генерации вида, однако, он является лучшим из-за большей гибкости при изменении содержимого «на лету».

Пример 7.24 Composite View с включением содержимого во время трансляции

```
<table border=1 valign="top" cellpadding="2%"
    width="100%">
    <tr>
       <td><%@ file="news/worldnews.html" %> </td>
    </tr>
    <tr>
       <td><%@ file="news/countrynews.html" %> </td>
    </tr>
    <tr>
       <td><%@ file="news/customnews.html" %> </td>
    </tr>
    <tr>
       <td><%@ file="news/astronomy.html" %> </td>
    </tr>
</table>
```

Пример 7.25 Composite View с включением содержимого во время рабочего цикла

```
<table border=1 valign="top" cellpadding="2%"
  width="100%">
    <tr>
        <td><jsp:include page="news/worldnews.jsp"
            flush="true"/> </td>
    </tr>
    <tr>
        <td><jsp:include page="news/countrynews.jsp"
```

```
            flush="true"/> </td>
    </tr>
    <tr>
        <td><jsp:include page="news/customnews.jsp"
            flush="true"/> </td>
    </tr>
    <tr>
        <td><jsp:include page="news/astronomy.jsp"
            flush="true"/> </td>
    </tr>
</table>
```

Стратегия Custom Tag View Management

Стратегия Custom Tag View management реализована при помощи заказных тэгов (JSP page 1.1+). Она является наиболее предпочтительной. Логика реализуется тэгами, которые отвечают за разметку и композицию вида. Эти тэги дают гораздо более мощное и гибкое решение, чем стандартный тэг `<jsp:include>`, однако их применение требует больших усилий. Заказные действия могут базировать разметку и композицию страницы на таких понятиях, как роли пользователей или службы безопасности.

Применение данной стратегии требует большей предварительной работы, чем применение других стратегий view management, так как разработать заказные тэги сложнее, чем просто использовать JavaBean-компоненты или стандартные тэги. И дело не только в этом. Интегрировать и управлять заказными тэгами в среде также оказывается сложнее. Данная стратегия подразумевает генерацию большого количества артефактов, включая сам тэг, дескриптор библиотеки тэгов, файлы конфигураций, а также настройку среды в соответствии с ними.

В следующей выдержке из JSP-страницы примера 7.26 показана возможная реализация данной стратегии. За дополнительной информацией обращайтесь к разделу «Пример кода».

```
<region:render
    template='/jsp/CompositeView/templates/portal.jsp'>

<region:put section='banner'
    content='/jsp/CompositeView/templates/banner.jsp' />

<region:put section='controlpanel' content=
    '/jsp/CompositeView/templates/ProfilePane.jsp' />

<region:put section='mainpanel' content=
    '/jsp/CompositeView/templates/mainpanel.jsp' />

<region:put section='footer' content=
    '/jsp/CompositeView/templates/footer.jsp' />
</region:render>
```

Стратегия Transformer View Management

Стратегия Transformer View management реализована при помощи XSL Преобразователя (Transformer). Обычно данную стратегию сочетают со стратегией Custom Tag View Management, причем заказные тэги используются для реализации и передачи полномочий соответствующим компонентам. Данная стратегия помогает отделить модель от вида, так как большая часть разметки вида должна быть вынесена в отдельную таблицу стилей. В то же время, в нее входят технологии, требующие новых, более сложных навыков. Это делает данную стратегию невозможной для применения во многих окружениях, в которых эти технологии еще не были введены.

В следующей выдержке показано, как использовать заказной тэг в JSP-странице для конвертирования модели при помощи таблицы стилей и преобразователя:

```
<xsl:transform model="portfolioHelper"
   stylesheet="/transform/styles/generalPortfolio.xsl"/>
```

Стратегия Early-Binding Resource

Стратегия Early-Binding Resource является другим названием для процесса включения содержимого во время трансляции (как описано в стратегии Standard Tag View Management). Она представлена в примере 7.24. Данная стратегия подходит для поддержки и обновления относительно статического шаблона и рекомендуется в тех случаях, когда шапка и нижняя часть страницы в виде изменяются не часто.

Стратегия Late-Binding Resource

Стратегия Late-Binding Resource является другим названием для процесса включения содержимого во время рабочего цикла (как описано в стратегии Standard Tag View Management). Она представлена в примере 7.25. Данная стратегия подходит для составных страниц, которые часто изменяются. Примечание: если подвид, включенный во время рабочего цикла, является динамическим ресурсом (например JSP-страницей), тогда он тоже может быть составным видом, включающим дополнительное содержимое во время рабочего цикла. Гибкость, получаемую при помощи подобных вложенных композиционных структур, нужно сравнить с тем, как они загружают рабочий цикл, и рассматривать в свете конкретных требований проекта.

Результаты

- **Улучшение модульности и возможность повторного использования**
  Паттерн поддерживает модульный дизайн. Можно повторно использовать отдельные части шаблона в разных видах и заполнять их при необходимости требуемой информацией. Паттерн позволяет перемещать таблицу в собственный модуль и просто включать его, куда требуется. Использование данного типа динамической разметки и композиции приводит к уменьшению дублирования кода, поддержке повторного использования и упрощению обслуживания.
- **Увеличение гибкости**
  Усовершенствованная реализация может условно включать фрагменты шаблона вида, базирующиеся на решениях, принятых во время рабочего цикла (например, роль пользователя или политика безопасности).
- **Упрощение поддержки и управления**
  Гораздо эффективнее изменять части шаблона, когда он не задан жестко в самой разметке вида. Если он находится отдельно от вида, то можно изменять модульные части содержимого шаблона независимо от его разметки. Кроме того, эти изменения незамедлительно станут доступны клиенту в зависимости от выбранной стратегии реализации. Изменять разметку страницы также очень просто, потому что все изменения централизованы.
- **Уменьшение управляемости**
  Агрегирование мелких частей отображения в единый вид приводит к потенциальным ошибкам отображения, так как подвиды являются фрагментами страницы. Пока это только ограничение, которое позднее может стать проблемой управляемости. Например, если JSP-страница генерирует HTML-страницу из трех подвидов, каждый из которых включает в себя открывающий и закрывающий HTML-тэги (`<HTML>` and `</HTML>`), то полученная страница будет неправильной. Таким образом, при использовании данного паттерна важно быть уверенным, что подвид не являются завершенным видом. Для того чтобы можно было создавать правильные составные виды, использование тэгов должно быть четко учтено.
- **Воздействие на производительность**
  Генерирование отображения, включающего в себя большое число подвидов, может уменьшить производительность. Включение подвидов во время рабочего цикла приведет к задержке при каждом открытии клиентом страницы. В среде со строгим соглашением об уровне сервиса это ухудшает производительность. Такое ухудшение минимально, однако все же не является приемлемым. В качестве альтернативы предлагается включать подвид во время трансляции, хотя это и ограничит возможность изменения подвида при ретрансляции страницы.

Пример кода

Паттерн Composite View может быть реализован при помощи любого количества стратегий, но наиболее популярной из них является Custom Tag View Management. На самом деле, для реализации составных видов, которые отделяют разметку вида от его содержимого и обеспечивают модульные встраиваемые шаблоны подвидов, доступен целый ряд библиотек заказных тэгов.

В этом примере мы воспользуемся библиотекой шаблонов, автора Дэвида Геари (David Geary), которая подробно описана в его статье «Advanced JavaServer Pages».

Библиотека шаблонов описывает три основных компонента: *sections (разделы)*, *regions (зоны)* и *templates (шаблоны)*.

- Раздел является компонентом повторного использования, который отображает HTML или JSP-страницу.
- Зона описывает контент путем определения разделов.
- Шаблон управляет разметкой зон и разделов в отображаемой странице.

Зона может быть определена и отображена, как показано в примере 7.26.

Пример 7.26 Компоненты Region и Section

```
<region:render template='portal.jsp'>
  <region:put section='banner' content = 'banner.jsp' />
  <region:put section = 'controlpanel' content =
      'ProfilePane.jsp' />
  <region:put section='mainpanel' content =
      'mainpanel.jsp' />
  <region:put section='footer' content='footer.jsp' />
</region:render>
```

Зона определяет контент, устанавливая соответствие между логическими именами разделов и частью содержимого, такого как `banner.jsp`.

Разметка зоны и ее разделов определяются шаблоном, с которым ассоциирована каждая зона. В примере 7.17 шаблон называется `portal.jsp`.

Пример 7.27 Определение шаблона

```
<region:render section='banner'/>
<table width="100%">
    <tr align="left" valign="middle">
        <td width="20%">
      <!-- menu region -->
      <region:render section='controlpanel' />
        </td>
        <td width="70%" align="center">
      <!-- contents -->
      <region:render section='mainpanel' />
        </td>
    </tr>
</table>
```

На сайте с большим количеством видов и одной согласованной с ними разметкой имеется только одна JSP-страница с кодом, похожим на определение шаблона из примера 7.27, а также множество JSP-страниц (похожих на пример 7.26), в которых определены чередующиеся зоны и разделы.

Разделами являются фрагменты JSP-страниц, используемые как подвиды при создании составной страницы, как указано в шаблоне. Раздел `banner.jsp` представлен в примере 7.28.

Пример 7.28 Подвид раздела – banner.jsp

```
<table width="100%" bgcolor="#C0C0C0">
<tr align="left" valign="middle">
  <td width="100%">

  <TABLE ALIGN="left" BORDER=1 WIDTH="100%">
  <TR ALIGN="left" VALIGN="middle">
    <TD>Logo</TD>
    <TD><center>Sun Java Center</TD>
  </TR>
  </TABLE>

  </td>
```

```
</tr>
</table>
```

Составные виды являются модульным, гибким и расширяемым способом для создания видов JSP-страницы вашего J2EE-приложения.

Родственные паттерны

- **View Helper**
  Паттерн Composite View может использоваться как вид в паттерне View Helper.
- **Composite [GoF]**
  В основе Composite View лежит паттерн Composite. Он описывает частичные и целые иерархии, в которых составной объект состоит из многочисленных частей обрабатываемых эквивалентным образом.