Шаблоны JSP

Используйте JSP шаблоны для инкапсуляции компоновки Web-страниц и поддержки модульного проектирования.

Layout Management

Дэвид Гери

Графические пакеты обычно обеспечивают механизм компоновки, который позиционирует компоненты (widgets) в контейнере. Например, в AWT и Swing есть менеджеры компоновки (layout managers), а в VisualWorks Smalltalk есть "обертки" (wrappers). Эта статья представляет механизм шаблонов для JSP, который позволяет инкапсулировать и повторно использовать компоновку. Шаблоны JSP минимизируют влияние изменений компоновки и стимулируют модульное проектирование. *(1600 слов)*

Хотя инструменты Web-разработки быстро развиваются, они всё ещё отстают от большинства графических пакетов, таких как Swing или VisualWorks Smalltalk. Например, традиционные графические пакеты обеспечивают в той или иной форме менеджеры компоновки, которые позволяют инкапсулировать и повторно использовать алгоритмы компоновки. Эта статья исследует механизм шаблонов для JavaServer Pages (JSP), который, подобно менеджерам компоновки, инкапсулирует компоновку. Так что эту компоновку можно повторно использовать, а не дублировать.

Из-за того, что компоновка подвергается многим изменениям в процессе разработки, важно инкапсулировать эту функциональность, чтобы её можно былю изменять с минимальным влиянием на остальное приложение. В действительности, менеджеры компоновки демонстрируют пример одного из принципов объектно-ориентированного проектирования: *инкапсулировать понятия, которые могут измениться,* который также является фундаментальной темой многих шаблонов проектирования (design patterns).

Copyright © 1999 - 2007, IT-archiv. All rights reserved.

JSP не обеспечивает прямой поддержки для инкапсуляции компоновки, поэтому Web-страницы с одинаковыми форматами обычно повторяют компоновочный код. Например, Рис.1 показывает Web-страницу, содержащую заголовок, "подвал", боковую полосу и главный раздел.

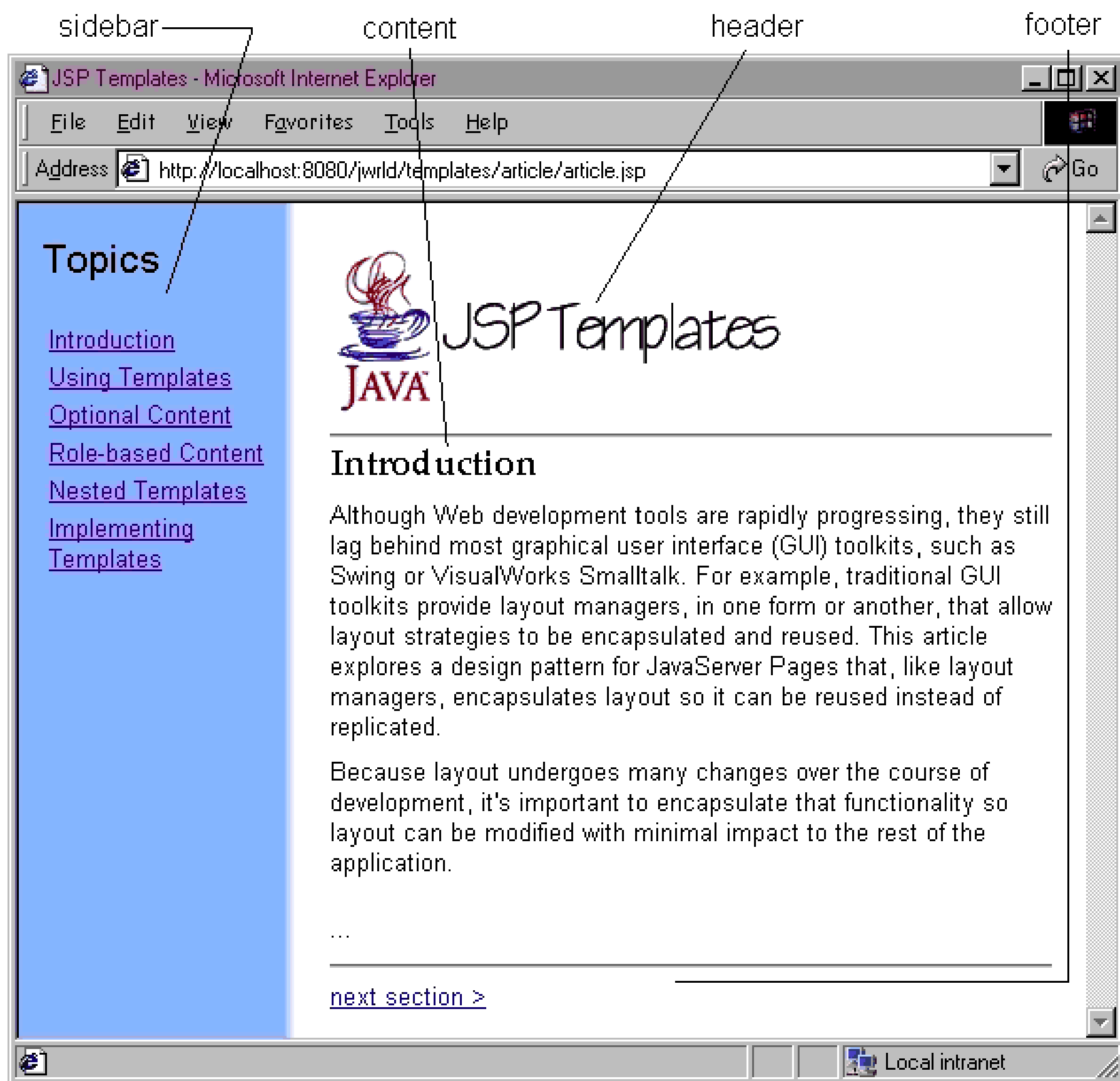

Рисунок 1. Компоновка Web-страницы.

Компоновка страницы на Рис.1 реализована табличными тегами HTML:

Copyright © 1999 - 2007, IT-archiv. All rights reserved.

Пример 1. Включение содержания

```
<html><head><title>JSP Templates</title></head>
 <body background='graphics/background.jpg'>

 <table>
    <tr valign='top'><td><%@include file='sidebar.html'%></td>
       <td><table>
          <tr><td><%@include file='header.html'%></td></tr>
          <tr><td><%@include file='introduction.html'%></td></tr>
          <tr><td><%@include file='footer.html'%></td></tr>
          </table>
       </td>
    </tr>
 </table>
 </body></html>
```

В примере выше содержимое вставляется JSP-директивой `include`, что позволяет изменять содержание страницы изменяя только включаемые файлы без модификации самой страницы. Однако, так как компоновка жестко задана в коде, изменение компоновки требует модификации страницы. Если на Web-сервере много страниц с одинаковым форматом, то даже простые изменения компоновки требуют модификации всех таких страниц.

Для уменьшения влияния изменений компоновки нам нужен механизм для включения компоновки в добавок к содержанию страницы. Таким образом, и компоновку и содержание можно менять без модификации файлов, которые их используют. Этот механизм — JSP шаблоны.

1. Использование шаблонов

Шаблоны — это JSP файлы, которые включают параметриризированное содержание. Шаблоны, обсуждаемые в этой статье, реализованы с помощью набора заказных тегов (custom tags): `template:get`, `template:put`, и `template:insert`. Тэг `template:get` осуществляет доступ к параметризированному содержанию, как демонстрируется в Примере 2.a, который производит Web-страницы в формате, показанном на Рис.1.

Copyright © 1999 - 2007, IT-archiv. All rights reserved.

Пример 2.a. Шаблон

```
<%@ taglib uri='/WEB-INF/tlds/template.tld' prefix='template' %>

 <html><head><title><template:get name='title'/></title></head>
 <body background='graphics/background.jpg'>

 <table>
    <tr valign='top'><td><template:get name='sidebar'/></td>
       <td><table>
             <tr><td><template:get name='header'/></td></tr>
             <tr><td><template:get name='content'/></td></tr>
             <tr><td><template:get name='footer'/></td></tr>
          </table>
       </td>
    </tr>
 </table>
 </body></html>
```

Пример 2.a почти идентичен Примеру 1, за исключением того, что мы используем `template:get` вместо директивы `include`. Давайте исследуем, как работает `template:get`.

Тег `template:get` возвращает Java bean с указанным именем из области действия запроса (request scope). Bean содержит URI (Uniform Resource Identifier — унифицированный идентификатор ресурса) Web-компонента, который включается тегом `template:get`. Например, в листинге Пример 2.a, `template:get` получает URI — `header.html` — из bean-а с именем `header` в области действия запроса. В последствии `template:get` включает `header.html`.

Тег `template:put` вставляет bean-ы в область действия запроса, который впоследствии возвращается тегом `template:get`. Шаблон вставляется тегом `template:insert`. Пример 2.b иллюстрирует применение тегов `put` и `insert`:

Пример 2.b. Применение шаблона из Примера 2.a

```
<%@ taglib uri='/WEB-INF/tlds/template.tld' prefix='template' %>

 <template:insert template='/articleTemplate.jsp'>
   <template:put name='title' content='Templates' direct='true'/>
   <template:put name='header' content='/header.html' />
```


Copyright © 1999 - 2007, IT-archiv. All rights reserved.

```
   <template:put name='sidebar' content='/sidebar.jsp' />
   <template:put name='content' content='/introduction.html'/>
   <template:put name='footer' content='/footer.html' />
 </template:insert>
```

Начальный тег `insert` определяет шаблон для включения, в данном случае шаблон из Примера 2.a. Каждый тег `put` сохраняет bean в области действия запроса и конечный тег `insert` включает шаблон. Шаблон впоследствии получает доступ к bean-ам как описано выше.

Для тега `template:put` может быть определён атрибут `direct`. Если `direct` установлен в `true`, то содержимое, связанное с тегом, не включается тегом `template:get`, но печатается прямо в неявную переменную `out`. Например, в Примере 2.b содержание заголовка — JSP Templates — используется для заголовка окна.

Вебсайты, содержащие много страниц одинакового формата, могут иметь один шаблон, такой как показан в Примере 2.a, и много JSP страниц, таких как в Примере 2.b, которые используют этот шаблон. *Если формат изменится, потребуется изменить только шаблон.*

Другое преимущество разделения шаблонов и содержания вообще — модульное проектирование. Например, JSP файл из Примера 2.b в конце концов включает `header.html`, показанный в Примере 2.c.

Пример 2.c. `header.html`

```
<table>
    <tr>
       <td><img src='graphics/java.jpg'/></td>
       <td><img src='graphics/templates.jpg'/></td>
    </tr>
 </table><hr>
```

Так как `header.html` включает содержание, его не нужно дублировать среди страниц, отображающих заголовок. Так же, хотя `header.html` — HTML файл, он не содержит обычного заголовка с HTML тегами, такими как <html> или <body>, потому что эти теги уже определены в шаблоне. То есть, так как шаблон включает `header.html`, эти теги не нужно повторять в `header.html`.

Copyright © 1999 - 2007, IT-archiv. All rights reserved.

Замечание:

JSP обеспечивает два пути для включения содержания: статически директивой `include` и динамически, операцией `include`. Директива `include` включает исходный текст целевой страницы во время компиляции и эквивалентна директиве `#include` в C или `import` в Java. Операция `include` включает в целевую страницу результат, генерируемый во время выполнения. Подобно JSP операции `include`, шаблоны включают содержание динамически. Таким образом, хотя JSP страницы в Примере 1 и Примере 2.b функционально идентичны, первый из них включает содержание статически, а второй — динамически.

2. Дополнительное содержание

Всё содержание шаблона — дополнительное, что делает один шаблон полезным для большего числа Web-страниц. Например, Рис. 2 и Рис. 3 показывает две страницы — страницу регистрации и опись товаров, которые используют один и тот же шаблон. Обе страницы имеют заголовок, "подвал" и главное содержание. Страница описи содержит панель редактирования (которая отсутствует на странице регистрации) для внесения изменений в опись.

Copyright © 1999 - 2007, IT-archiv. All rights reserved.

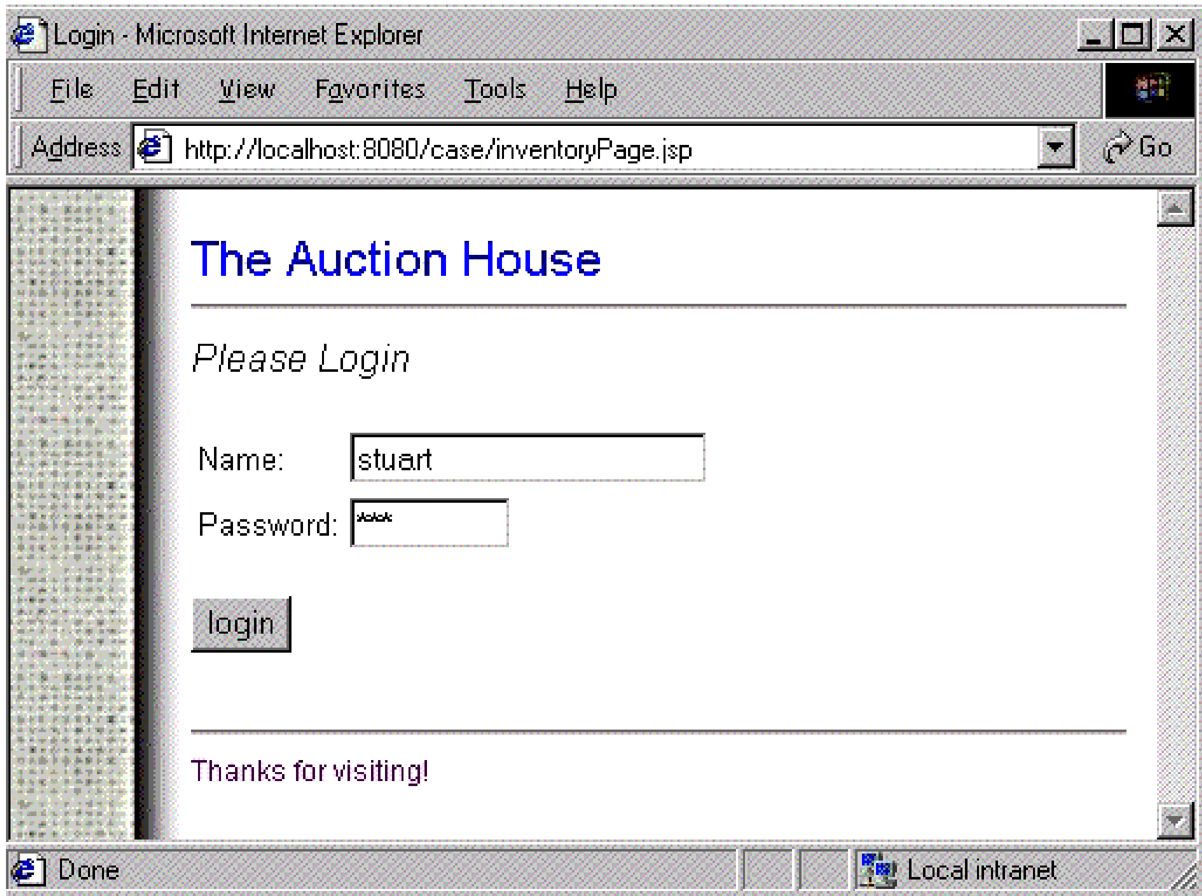

Рисунок 2. Страница регистрации.

Copyright © 1999 - 2007, IT-archiv. All rights reserved.

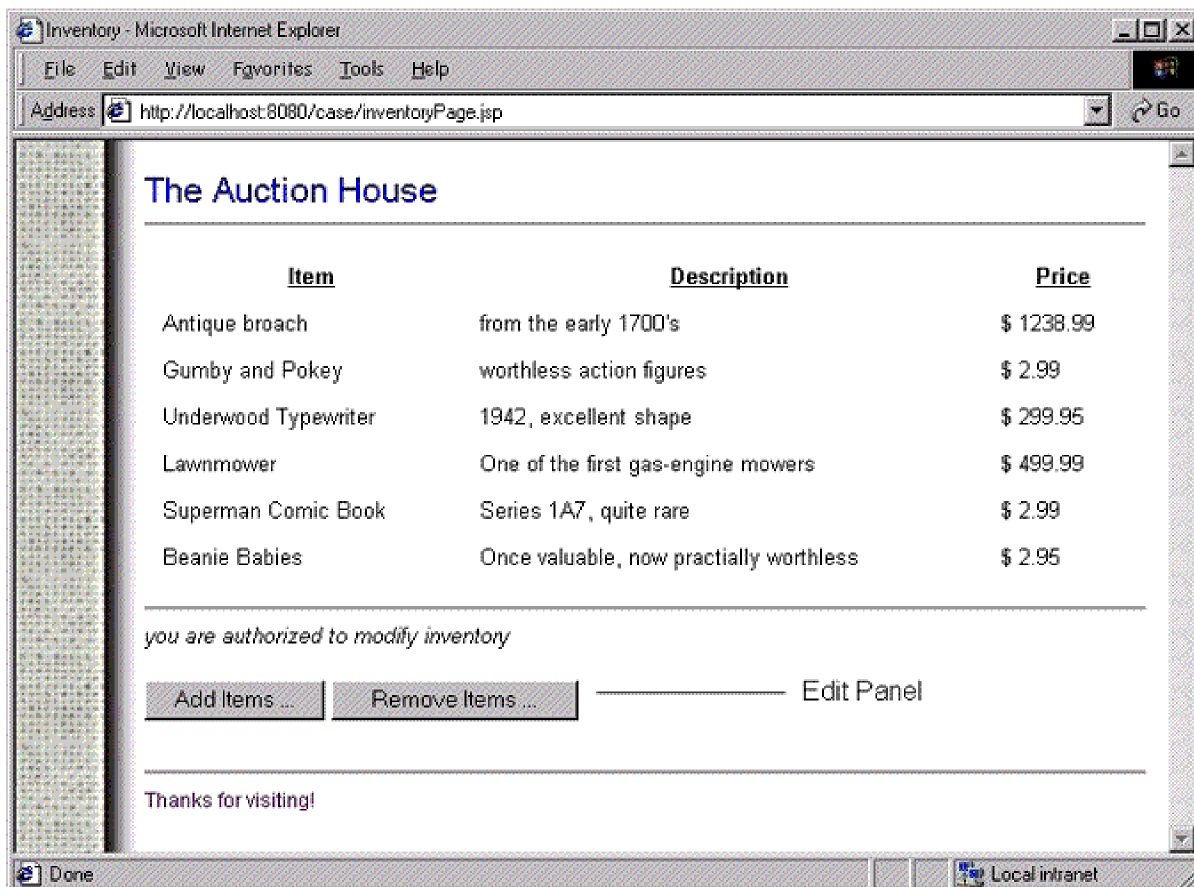

Рисунок 3. Страница описи.

Ниже вы видите шаблон, разделяемый страницами регистрации и описи:

```
<%@ taglib uri='template.tld' prefix='template' %>
 ...
 <table width='670'>
    <tr><td width='60'></td>
       <td><template:get name='header'/></td></tr>

    <tr><td width='60'></td>
       <td><template:get name='main-content'/></td></tr>

    <tr><td width='60'></td>
        <td><template:get name='editPanel'/></td></tr>
```


Copyright © 1999 - 2007, IT-archiv. All rights reserved.

```
   <tr><td width='60'></td>
   <td><template:get name='footer'/></td></tr>
</table>
...
```

Страница описи товаров использует шаблон, показанный выше и определяет содержимое панели редактирования:

```
<%@ taglib uri='template.tld' prefix='template' %>
<%@ taglib uri='security.tld' prefix='security' %>

<template:insert template='/template.jsp'>
   ...
   <template:put name='editPanel'
                 content='/editPanelContent.jsp'/>
   ...
</template:insert>
```

Страница регистрации, напротив, не определяет содержание панели редактирования:

```
<%@ taglib uri='template.tld' prefix='template' %>

<template:insert template='/template.jsp'>
   <template:put name='title' content='Login' direct='true'/>
   <template:put name='header' content='/header.jsp'/>

   <template:put name='main-content'
                 content='/login.jsp'/>

   <template:put name='footer' content='/footer.jsp'/>
</template:insert>
```

Так как страница регистрации не определяет содержание панели редактирования, она не включается.

3. Ролевое содержание

Web приложения часто различают содержание, основываясь на роли пользователя. Например, один и тот же JSP шаблон, который включает панель редактирования только когда роль пользователя — администратор, порождает две страницы, показанные на Рис. 4 и 5.

Copyright © 1999 - 2007, IT-archiv. All rights reserved.

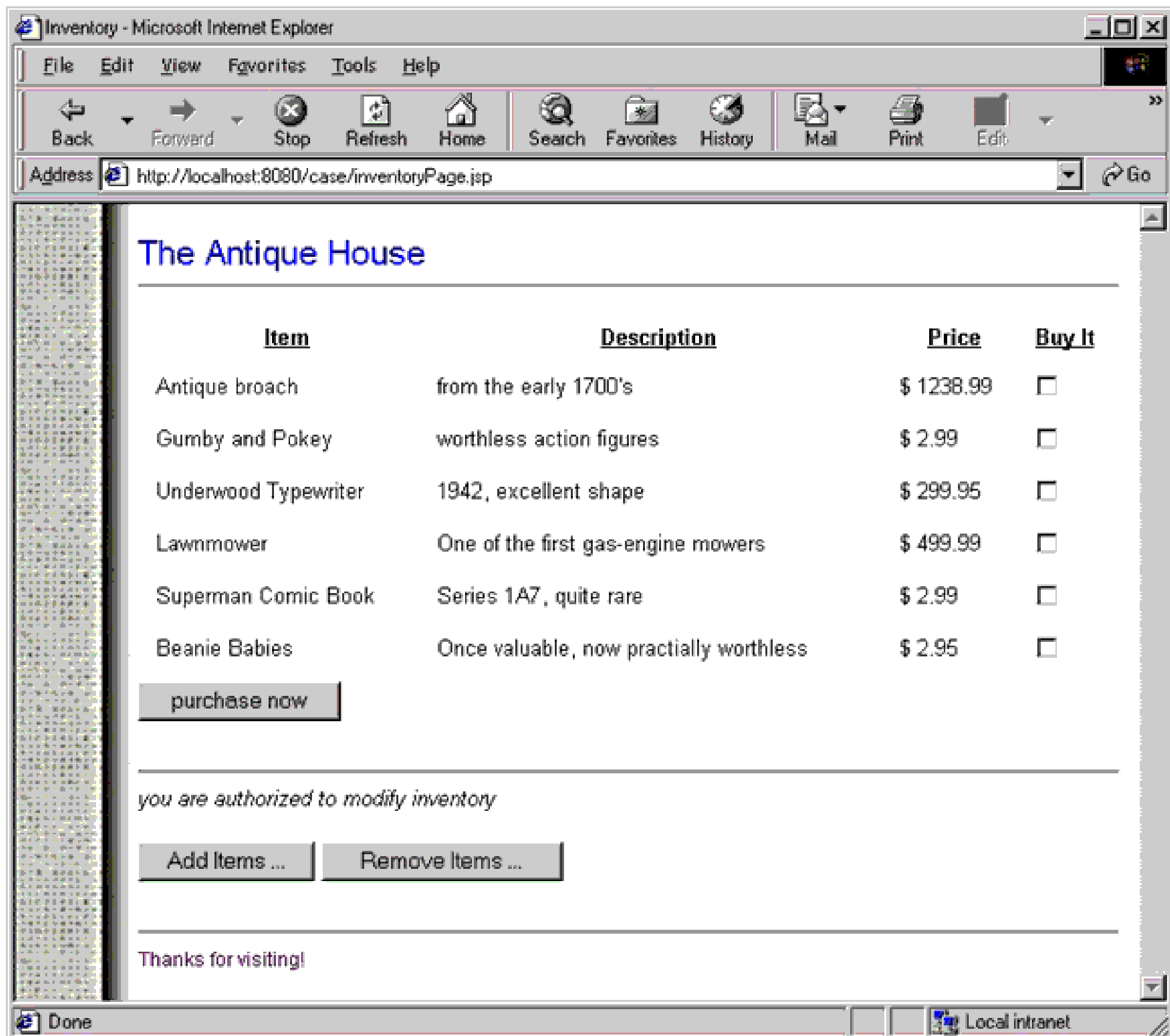

Рисунок 4. Страница описи для администраторов.

Copyright © 1999 - 2007, IT-archiv. All rights reserved.

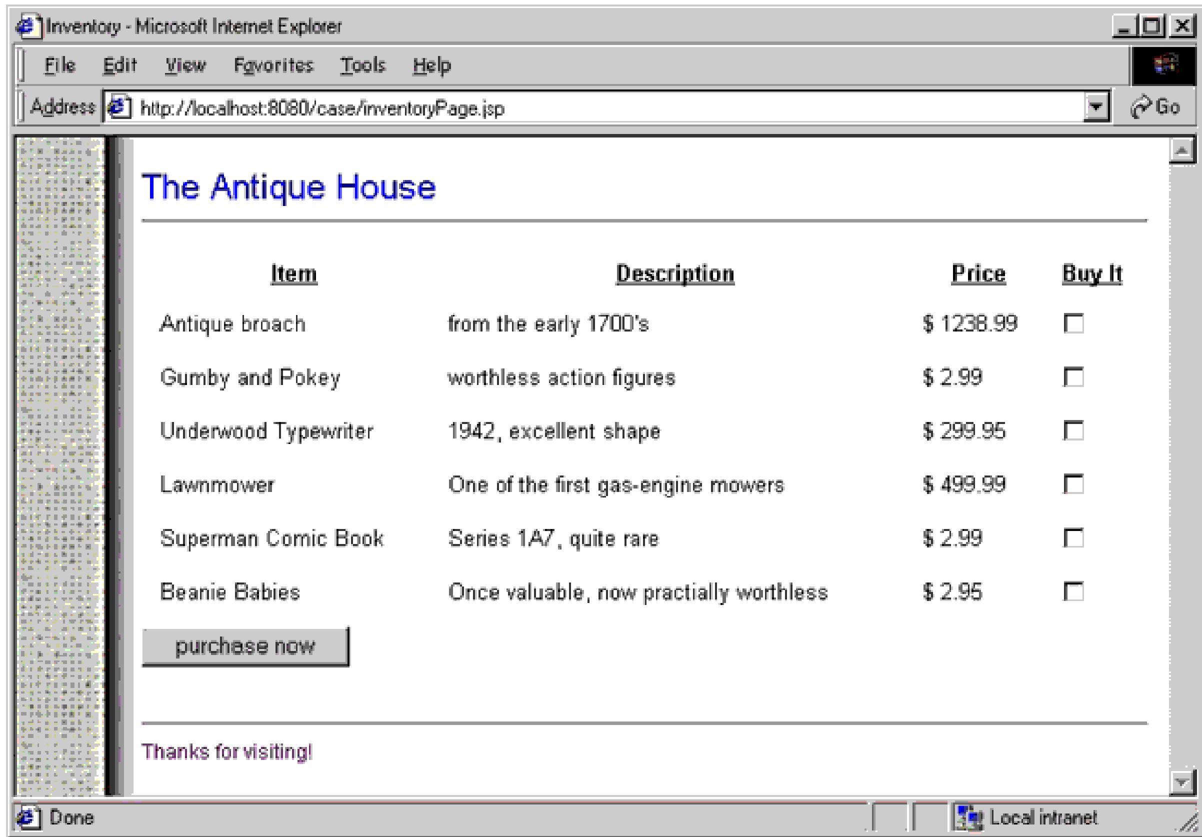

Рисунок 5. Страница описи для других пользователей.

Шаблон, используемый в Рис. 4 и 5, использует атрибут `role` тега `template:get`:

```
<%@ taglib uri='template.tld' prefix='template' %>
 ...
 <table>
    ...
    <td><template:get name='editPanel' role='curator'/></td></tr>
    ...
 </table>
 ...
```

Тег `get` включает содержание, только если роль пользователя совпадает со значением атрибута `role`. Давайте посмотрим, как обработчик тега для `template:get` использует атрибут `role`:

```
public class GetTag extends TagSupport {
    private String name = null, role = null;
```


Copyright © 1999 - 2007, IT-archiv. All rights reserved.

```
    ...
    public void setRole(String role) { this.role = role; }
    ...
    public int doStartTag() throws JspException {
       ...
       if(param != null) {
          if(roleIsValid()) {
             // включает или печатает содержимое ...
          }
       }
    ...
    }
    private boolean roleIsValid() {
       return role == null || // правильно, если роль не установлена
          ((javax.servlet.http.HttpServletRequest)
           pageContext.getRequest()).isUserInRole(role);
    }
}
```

4. Реализация шаблонов

Шаблоны, обсуждаемые в этой статье, реализованы с тремя заказными тегами (custom tags):

- `template:insert`
- `template:put`
- `template:get`

Тег `insert` включает шаблон, но перед этим тег `put` сохраняет информацию (имя, URI, и логическое значение, определяющее должно ли содержимое быть включено или напечатано напрямую) о содержимом шаблона. Тег `template:get`, который включает (или печатает) заданное содержание, впоследствии осуществляет доступ к информации.

Тег `template:put` сохраняет beans в области действия запроса (request scope) но не *напрямую*, потому что если два шаблона используют одно и то же имя содержимиго, вложенный шаблон будет перекрывать содержимое объемлющего шаблона.

Чтобы гарантировать, что каждый шаблон имеет доступ только к его собственной информации, `template:insert` поддерживает стек хеш-таблиц. Каждый начальный

Copyright © 1999 - 2007, IT-archiv. All rights reserved.

тег `insert` создает хеш-таблицу и кладет её в стек. Вложенный тег `put` создает beans и сохраняет их в вновь созданной хеш-таблице. Впоследствии тег `get` во включенном шаблоне осуществляет доступ к beans в хеш-таблице. Рис. 6 показывает как поддерживается стек для вложенных шаблонов.

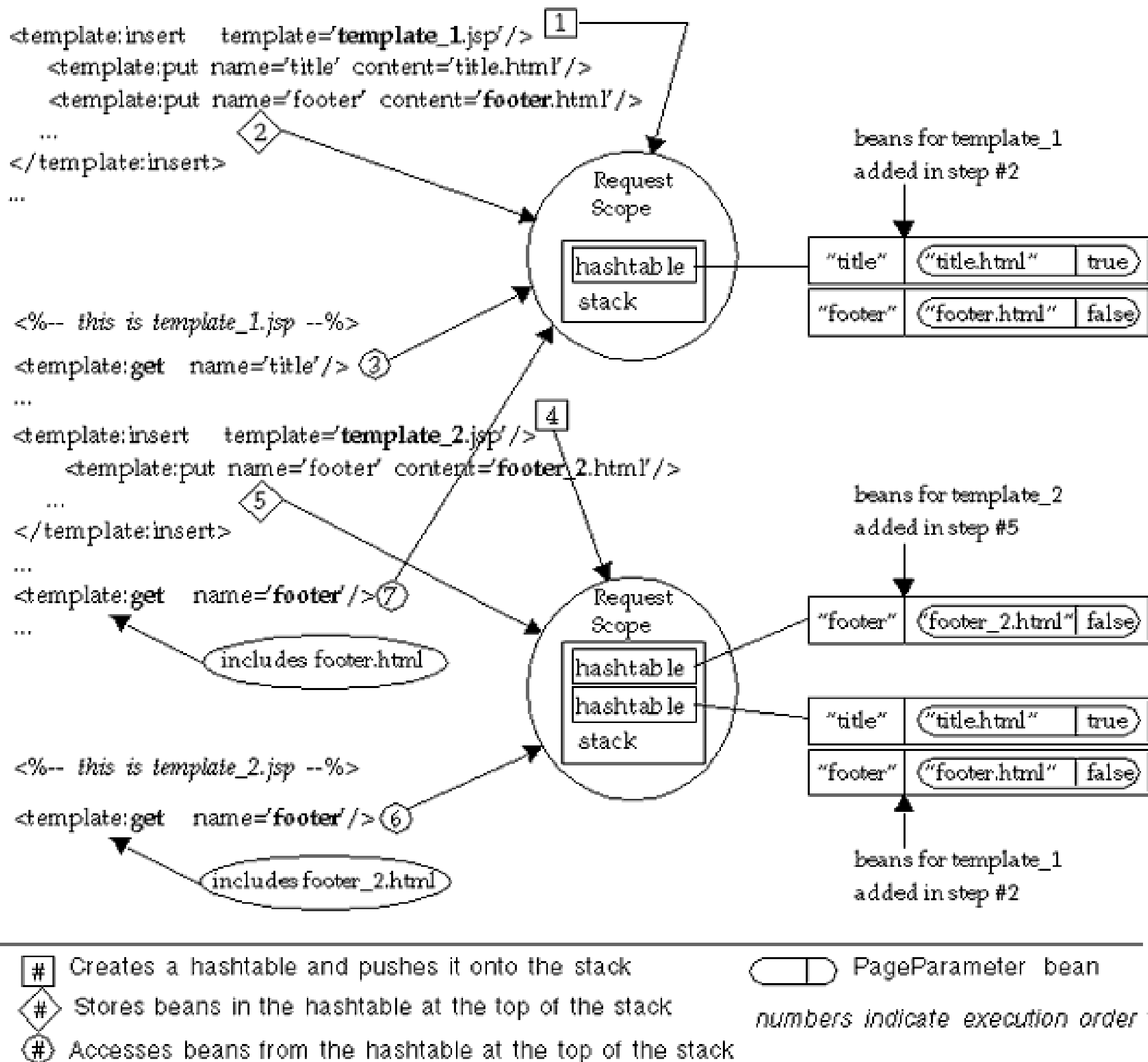

Рисунок 6. Сохранение параметров шаблонов в области действия запроса.

Каждый шаблон на Рис. 6 осуществляет доступ к правильному "подвалу" — footer.html для `template_1.jsp` и footer_2.html для `template_2.jsp`. Если beans были сохранены прямо в области действия запроса, шаг 5 в Рис. 6 будет заменять

Copyright © 1999 - 2007, IT-archiv. All rights reserved.

"подвальный" bean определенный на шаге 2.

Реализация тегов шаблона Остаток этой статьи исследует реализацию трех тегов шаблона (template tags): `insert`, `put` и `get`. Мы начнём с диаграм последовательности событий, начиная с Рис. 7. Она показывает последовательность событий для тегов `insert` и `put`, когда используется шаблон.

Copyright © 1999 - 2007, IT-archiv. All rights reserved.

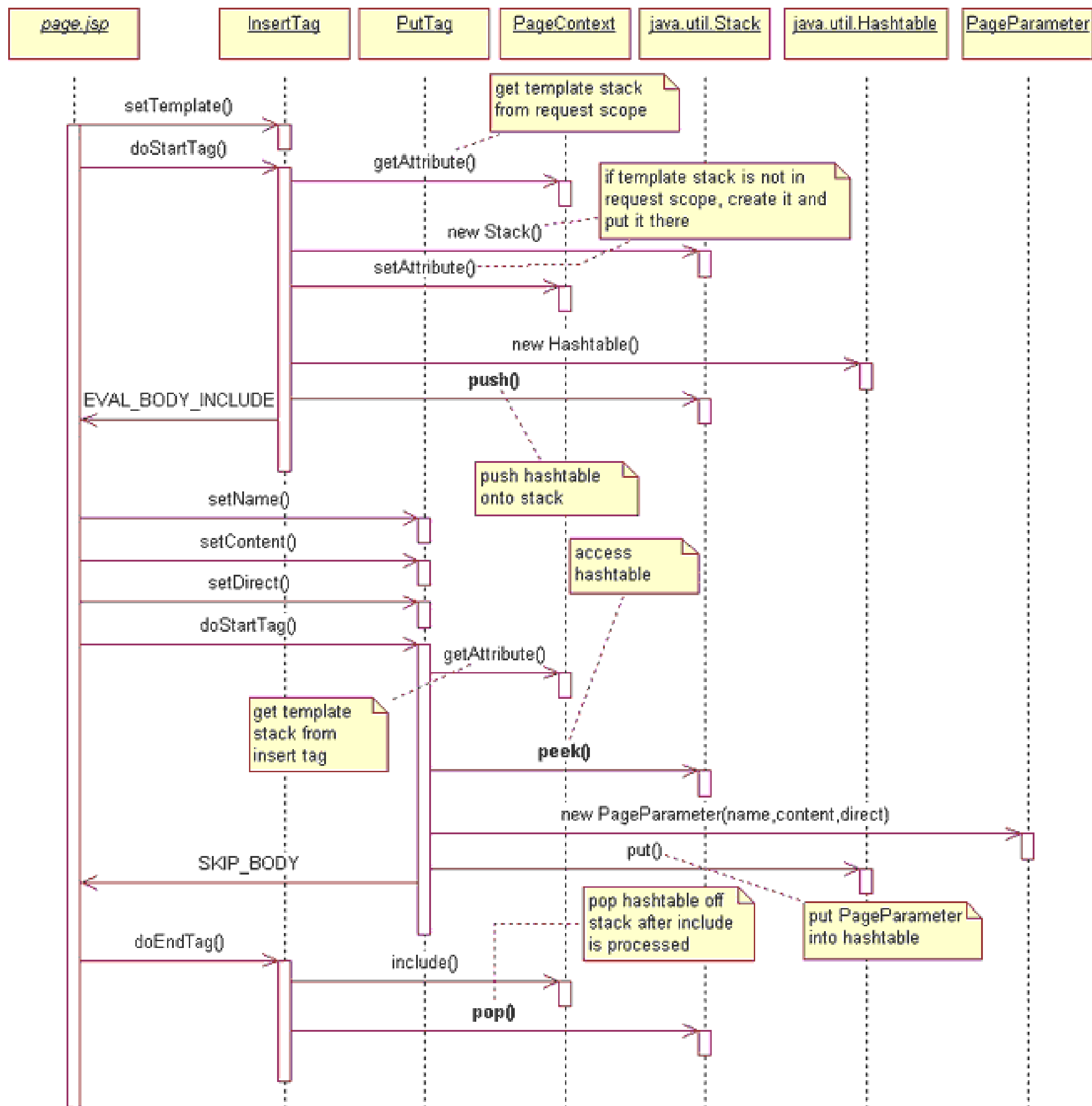

Рисунок 7. Диаграмма последовательности событий для тегов put и insert.

Если стек шаблона ещё не существует, первый тег `insert` создает его и помещает его в область действия запроса. После этого создается хеш-таблица и помещается в стек.

Каждый стартовый тег `put` создает bean `PageParameter`, сохраняемый в

Copyright © 1999 - 2007, IT-archiv. All rights reserved.

хеш-таблице, созданной объемлющим тегом `insert`.

Вставка тега `end` включает шаблон. Шаблон использует теги `get` для доступа к бинам, созданным тегами `put`. После обработки шаблона хеш-таблица, созданная стартовым тегом `insert`, выталкивается из стека.

Рис. 8 показывает диаграмму последовательности событий для `template:get`.

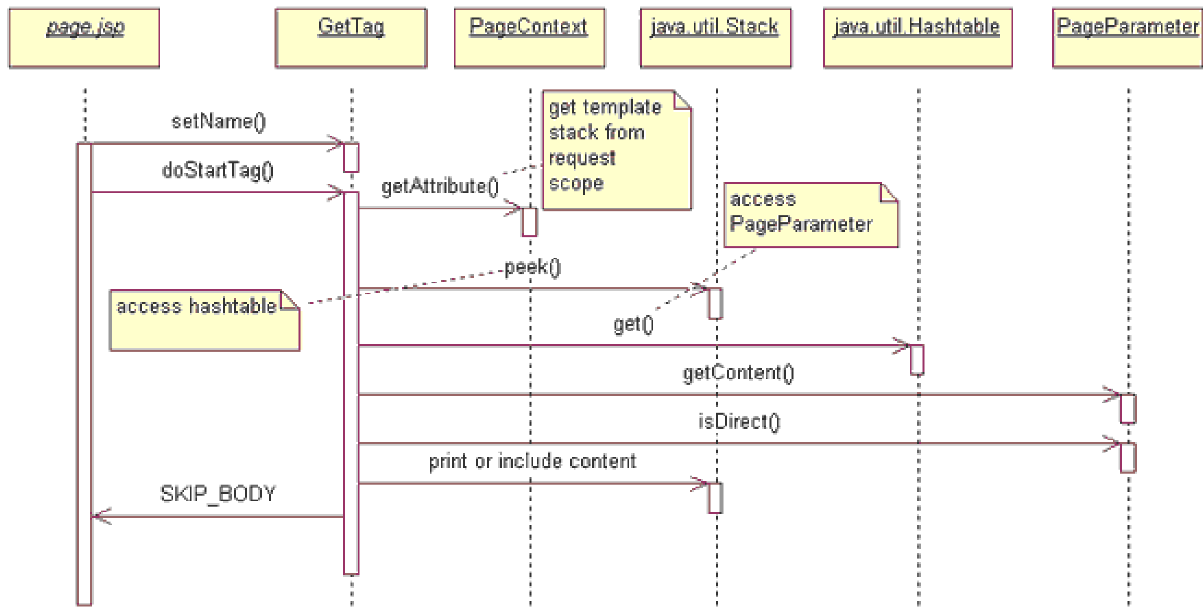

Рисунок 8. Диаграмма последовательности событий для тега get.

Листинги тегов шаблона (Template tag listings) Реализация обработчиков для тегов шаблона оказывается прямолинейной. Пример 3.a показывает класс `InsertTag` — обработчик тега `template:insert`.

Пример 3.a. `InsertTag.java`

```
package tags.templates;

 import java.util.Hashtable;
 import java.util.Stack;
 import javax.servlet.jsp.JspException;
 import javax.servlet.jsp.PageContext;
 import javax.servlet.jsp.tagext.TagSupport;

 public class InsertTag extends TagSupport {
```


Copyright © 1999 - 2007, IT-archiv. All rights reserved.

```
private String template;
private Stack stack;

// метод для настройки атрибута template
public void setTemplate(String template) {
   this.template = template;
}
public int doStartTag() throws JspException {
   stack = getStack(); // получаем ссылку на стек шаблона
   stack.push(new Hashtable()); // помещаем новую хеш-таблицу в стек
   return EVAL_BODY_INCLUDE; // передаем тело тега без изменений
}
public int doEndTag() throws JspException {
   try {
      pageContext.include(template); // включаем шаблон
   }
   catch(Exception ex) { // IOException or ServletException
      throw new JspException(ex.getMessage()); // вызываем исключение другого типа
   }
   stack.pop(); // выталкиваем хеш-таблицу из стека
   return EVAL_PAGE; // вычислить остаток страницы после тега
}
// обработчики тегов всегда реализуют release(), потому что
// обработчики могут повторно использоваться JSP контейнерами
public void release() {
   template = null;
   stack = null;
}
public Stack getStack() {
   // попытаемся получить стек из области запроса
   Stack s = (Stack)pageContext.getAttribute(
                    "template-stack",
                    PageContext.REQUEST_SCOPE);

   // если нет стека — создать новый
   // и положить его в область действия запроса
   if(s == null) {
      s = new Stack();
      pageContext.setAttribute("template-stack", s,
                    PageContext.REQUEST_SCOPE);
```


Copyright © 1999 - 2007, IT-archiv. All rights reserved.

```
      }
      return s;
   }
}
```

Пример 3.b — листинг класса `PutTag`, обработчик тега `template:put`:

Пример 3.b. `PutTag.java`

```
package tags.templates;

import java.util.Hashtable;
import java.util.Stack;
import javax.servlet.jsp.JspException;
import javax.servlet.jsp.tagext.TagSupport;
import beans.templates.PageParameter;

public class PutTag extends TagSupport {
   private String name, content, direct="false";

   // методы настройки атрибутов тега Put
   public void setName(String s) { name = s; }
   public void setContent(String s) {content = s; }
   public void setDirect(String s) { direct = s; }

   public int doStartTag() throws JspException {
      // получаем ссылку на объемлющий тег insert
      InsertTag parent = (InsertTag)getAncestor(
                              "tags.templates.InsertTag");

      // теги put должны включаться в тег insert
      if(parent == null)
         throw new JspException("PutTag.doStartTag(): " +
                                "No InsertTag ancestor");

      // получить стек шаблонов из тега insert
      Stack template_stack = parent.getStack();

      // стек шаблона не должен быть пустым
      if(template_stack == null)
         throw new JspException("PutTag: no template stack");
```


Copyright © 1999 - 2007, IT-archiv. All rights reserved.

```
       // прочитать хеш-таблицу на вершине стека
       Hashtable params = (Hashtable)template_stack.peek();

       // хеш-таблица тоже не должна быть пустой
       if(params == null)
          throw new JspException("PutTag: no hashtable");

       // положить новый PageParameter в хэш-таблицу
      params.put(name, new PageParameter(content, direct));

      return SKIP_BODY; // не нужно тело тега, [даже] если есть
   }
   // обработчики тегов всегда должны реализовать release(), потому что
    // обработчики могут повторно использоваться JSP контейнерами
   public void release() {
      name = content = direct = null;
   }
   // удобный метод для поиска имен предшественников (ancestor)
    // для заданного имени класса
   private TagSupport getAncestor(String className)
                                 throws JspException {
      Class klass = null; // нет переменных с именем "class"
      try {
         klass = Class.forName(className);
      }
      catch(ClassNotFoundException ex) {
         throw new JspException(ex.getMessage());
      }
      return (TagSupport)findAncestorWithClass(this, klass);
   }
}
```

`PutTag.doStartTag` создает bean `PageParameter` — распечатанный в Примере 3.c. — впоследствии сохранённый в области действия запроса.

Пример 3.c. `PageParameter.java`

```
package beans.templates;

 public class PageParameter {
    private String content, direct;
```


Copyright © 1999 - 2007, IT-archiv. All rights reserved.

```
   public void setContent(String s) {content = s; }
   public void setDirect(String s) { direct = s; }

   public String getContent() { return content;}
   public boolean isDirect() { return Boolean.valueOf(direct).booleanValue(); }

   public PageParameter(String content, String direct) {
      this.content = content;
      this.direct = direct;
   }
}
```

`PageParameter` служит простой меткой-заполнителем (placeholder) для атрибутов `content` и `direct`, устанавливаемых в теге `template:put`. Мы видим класс `GetTag`, обработчик тега `template:get`, в Примере 3.d:

Пример 3.d. `GetTag.java`

```
package tags.templates;

import java.util.Hashtable;
import java.util.Stack;
import javax.servlet.jsp.JspException;
import javax.servlet.jsp.PageContext;
import javax.servlet.jsp.tagext.TagSupport;

import beans.templates.PageParameter;

public class GetTag extends TagSupport {
   private String name;

   // метод для установки значения атибута name
   public void setName(String name) {
      this.name = name;
   }
   public int doStartTag() throws JspException {
      // получаем ссылку на стек шаблона
      Stack stack = (Stack)pageContext.getAttribute(
               "template-stack", PageContext.REQUEST_SCOPE);
```


Copyright © 1999 - 2007, IT-archiv. All rights reserved.

```
      // стек не должен быть null
      if(stack == null)
         throw new JspException("GetTag.doStartTag(): " +
                                "NO STACK");

      // считать(peek) в хеш-таблицу
      Hashtable params = (Hashtable)stack.peek();

      // хеш-таблица не должна быть null
      if(params == null)
         throw new JspException("GetTag.doStartTag(): " +
                                "NO HASHTABLE");

      // берем параметр страницы из хеш-таблицы
      PageParameter param = (PageParameter)params.get(name);

      if(param != null) {
         String content = param.getContent();

         if(param.isDirect()) {
            // печатаем содержимое (content), если атрибут direct == true
            try {
               pageContext.getOut().print(content);
            }
            catch(java.io.IOException ex) {
               throw new JspException(ex.getMessage());
            }
         }
         else {
            // включить содержимое (include content), если атрибут direct == false
            try {
               pageContext.getOut().flush();
               pageContext.include(content);
            }
            catch(Exception ex) {
               throw new JspException(ex.getMessage());
            }
         }
      }
      return SKIP_BODY; // пропустить тело тега
```


Copyright © 1999 - 2007, IT-archiv. All rights reserved.

```
    }
    // обработчики тегов всегда должны реализовать release(), потому что
    // обработчики могут повторно использоваться JSP контейнерами
    public void release() {
       name = null;
    }
 }
```

`GetTag.doStartTag` восстанавливает bean `PageParameter` из области действия запроса и получает свойства `content` и `direct` из бина. Впоследствии содержание либо включается, либо печатается, в зависимости от значения свойства `direct`.

5. Заключение

Шаблоны — простая, но полезная концепция, которая должна быть в наборе каждого JSP разработчика. Шаблоны инкапсулируют компоновку и поэтому минимизируют усилия по её изменению. Кроме того, шаблоны могут различать содержание, основанное на ролях пользователей и могут быть вложенными в другие шаблоны и страницы JSP.

Шаблоны — нестандартное свойство JSP, но вы можете легко реализовать их с заказными тегами (custom tags), как обсуждалось в этой статье. Эти теги могут использоваться как есть, или они могут использоваться как базовые для механизма шаблонов с бОльшими возможностями.

6. Об авторе

Дэвид Гери (David Geary), автор серии *Graphic Java* от Sun Microsystems Press, сейчас работает над книгой *JSP: Advanced Topics*, планируемой к январю 2001.

С 1984, Дэвид занимался разработкой объектно-ориентированного программного обеспечения для широкого диапазона приложений от видеоигр до многоуровневых бизнес-приложений. Он работал с многочисленными ОО языками, включая C++, Objective-C, Smalltalk, Eiffel, и конечно, Java. Дэвид также изучал C, C++ и ООП более пяти лет.

7. Ресурсы

Copyright © 1999 - 2007, IT-archiv. All rights reserved.

- Исходный код из этой статьи в виде JAR-файла: http://www.javaworld.com/javaworld/ jw-09-2000/jspweb/jw-0915-jspweb.zip
- *Core Servlets and JSP* Марти Нолла (Marty Hall) (Prentice-Hall May 2000) дает превосходное введение в сервлеты и JSP: http://www.amazon.com/exec/ obidos/ASIN/0130893404/ qid%3D965923926/104-4930578-8363958
- *Designing Enterprise Applications with the Java 2 Platform, Enterprise Edition* Николаса Кассема (Nicolas Kassem) (Addison-Wesley 2000) обсуждает JSP шаблоны: http://www.amazon.com/exec/obidos/ ASIN/0201702770/ref=sim_books/ 104-4930578-8363958

Reprinted with permission from the September 2000 edition of JavaWorld magazine. Copyright © ITworld.com, Inc., an IDG Communications company.
View the original article at: http://www.javaworld.com/ javaworld/jw-09-2000/jw-0915-jspweb.html

Перевод на русский © Владимир Дузбаев, 2000

Copyright © 1999 - 2007, IT-archiv. All rights reserved.