


Customer TagLib

Allen Long
Email: allen@huihoo.com
http://www.huihoo.com
2004-04

内容安排

- TagLib基础
- Writing Tag
- Writing IterationTag
- Writing BodyTag
- Writing Collaborating Tags
- Third party TagLib

TagLib意义

业务逻辑层与表示层的分离一直是基于WEB应用开发的一大目标。因为这样可以使开发人员更加专注于自己领域的开发（程序工程师负责业务逻辑，而网页制作人员等负责表示层）。其中JavaBean就是一个很好的解决方案。自JSP1。0以后，出现了Tag Extension API(标记扩展API)。这使得WEB开发人员可以自定义标记，将其嵌在JSP页面中，来完成复杂的业务逻辑。这样，网页制作人员只需向JSP页面中添加简单、熟悉的标记就可以实现相应业务逻辑，而这些业务逻辑的封装则由程序工程师来完成。

也就是在简单的JSP标记符后面隐藏复杂的功能

Why Use Custom Tags

- Improved separation of presentation and implementation
  - Reduce/eliminate scripting tags
  - Encapsulate common or application-specific page idioms
  - Provide an HTML-friendly layer of abstraction
- Only three data-oriented JSP actions:

```
<jsp:useBean>
<jsp:setProperty>
<jsp:getProperty>
```

How Do They Work?

- Like most J2EE technologies, there are two aspects to the development of a custom tag library.
  - Individual tags are implemented as Java classes called *Tag Handlers*.
  - An XML file called the *Tag Library Descriptor* (TLD) maps a set of tag handlers into a library.
- The Java classes and TLD can be deployed individually or *via* a JAR file.

Use Custom Tag

- Like the standard actions, custom tags follow XML syntax conventions

```
<prefix:name attribute="value"
attribute="value"/>
```

or

```
<prefix:name attribute="value"
attribute="value">
    body content
</prefix:name>
```

TagLib构成

- TagLib处理程序
- TagLib指令
- 标记库描述符

Tag Handlers

- A tag handler class must implement one of the following interfaces:
  - javax.servlet.jsp.tagext.Tag
  - javax.servlet.jsp.tagext.IterationTag
  - javax.servlet.jsp.tagext.BodyTag
- A tag handler can optionally implement javax.servlet.jsp.tagext.TryCatchFinally
- Tag attributes are managed as JavaBeans properties (*i.e.*, *via* getters and setters)

Tag Library Descriptor

- TLD defines tag syntax
- TLD maps tag names to handler classes
- TLD constrains tag body content
- TLD specifies tag attributes
  - Attribute names and optional types
  - Required *vs*. optional
  - Compile-time *vs*. run-time values
- TLD specifies tag variables (name, scope, *etc*.)
- TLD declares tag library validator and lifecycle event handlers, if any

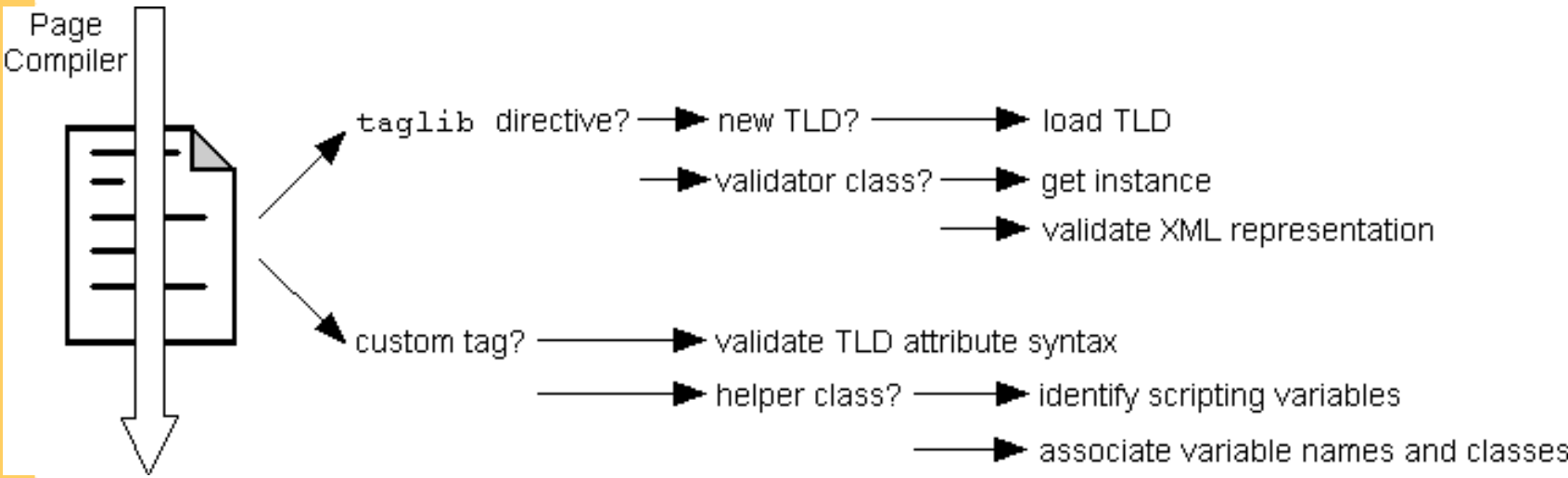
Using Custom Tags

- Tags are made available within a JSP page *via* the **`taglib`** directive:

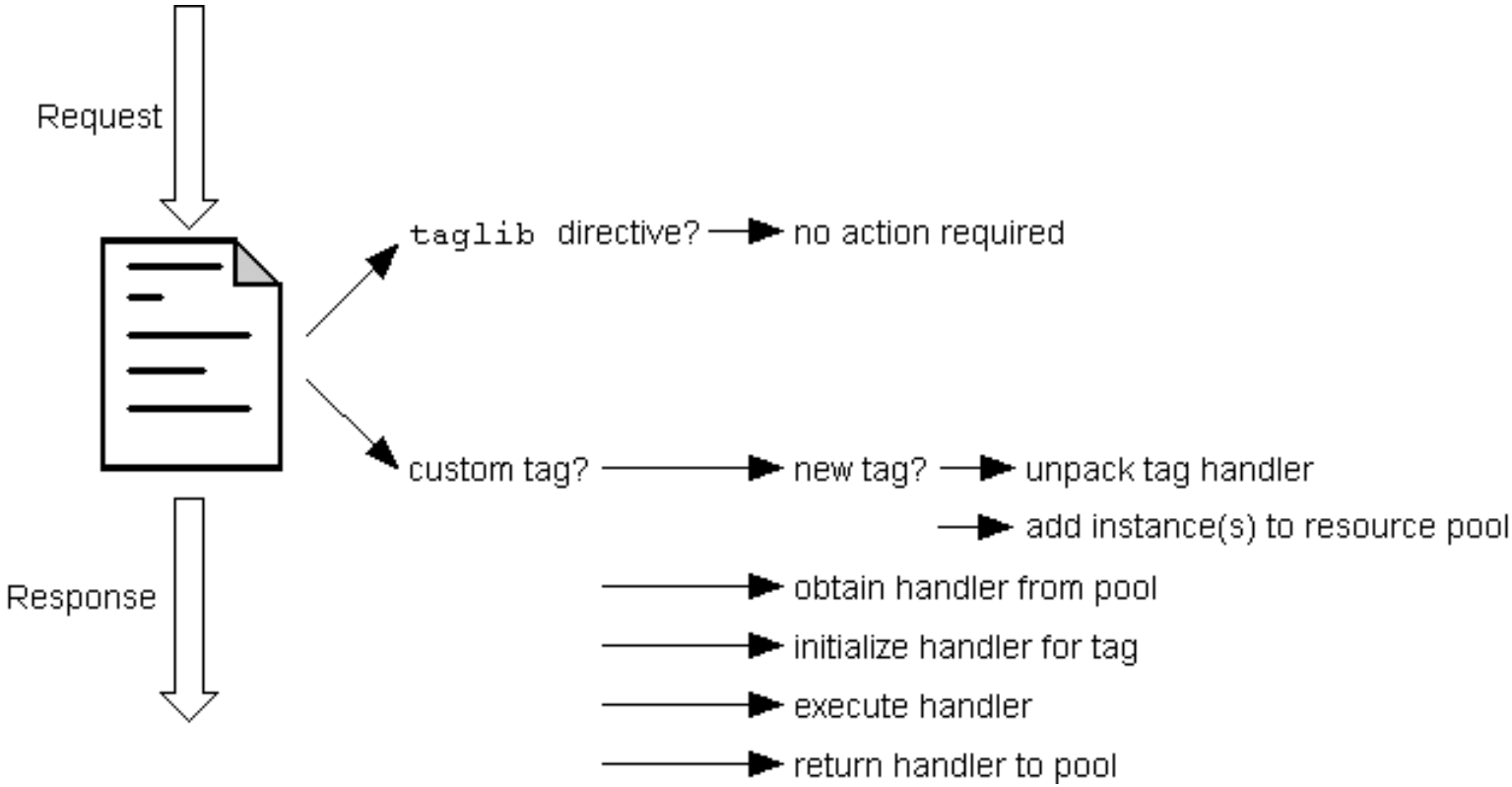
```
<%@ taglib uri="uri" prefix="prefix" %>
```

- Directive's **`uri`** attribute references the TLD (established via WAR file's **`web.xml`**)
- Directive's **`prefix`** attribute provides a local namespace for the TLD's tags

Page Compilation



Page Execution

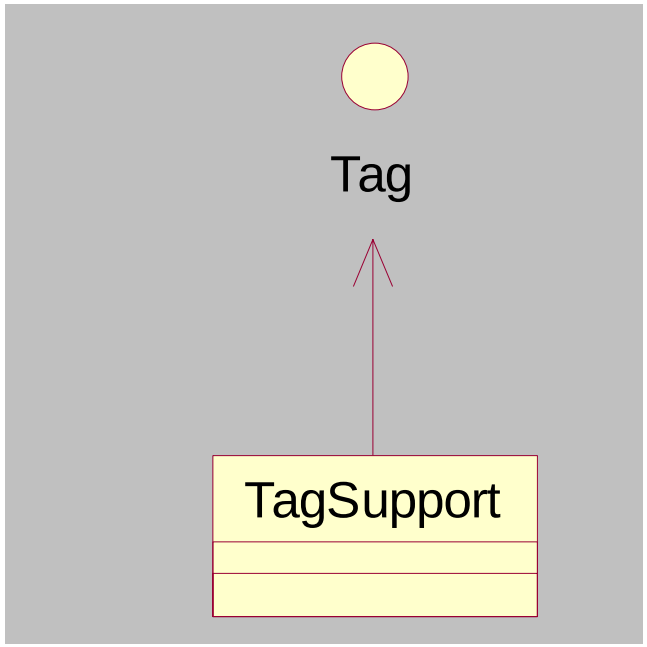




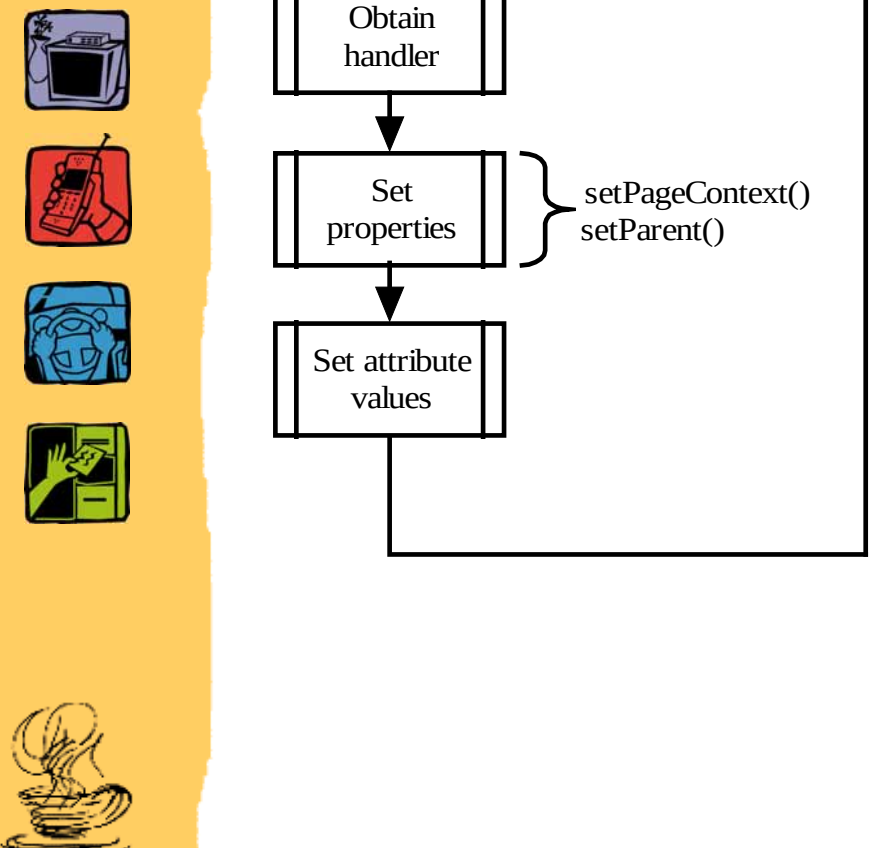

Simple Tag

Tag TagSupport

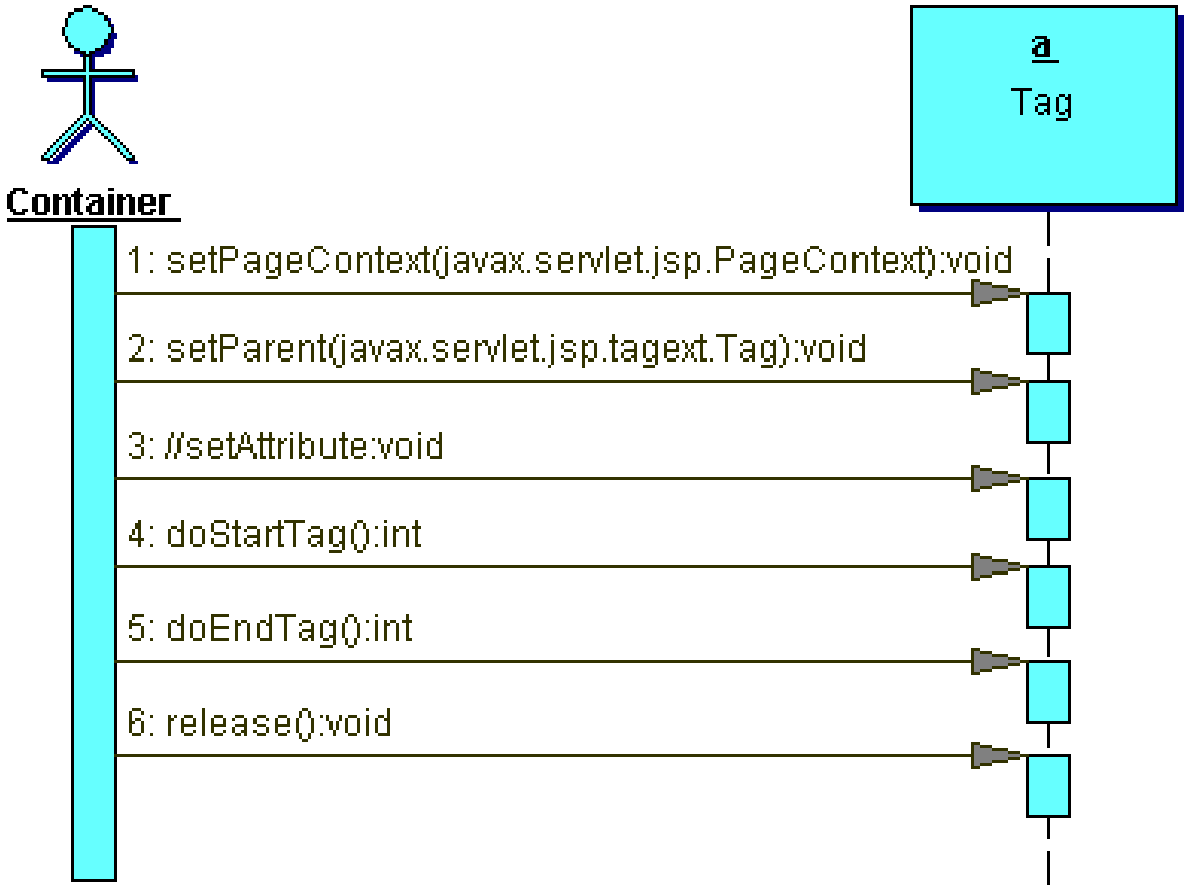


Tag Invocation



Container & Tag



Container & Tag

- 1. Use the setPageContext() method of the Tag to set the current PageContext for it to use.
- 2. Use the setParent() method to set any parent of the encountered Tag (or null if none).
- 3. Set any attributes defined to be given to the Tag.
- 4. Call the doStartTag() method. This method can either return EVAL_BODY_INCLUDE or SKIP_BODY. If EVAL_BODY_INCLUDE is returned, the Tags body will be evaluated. If SKIP_BODY is returned, the Container will not evaluate the body of the Tag.
- 5. Call the doEndTag() method. This method can either return EVAL_PAGE or SKIP_PAGE. IF EVAL_PAGE is returned, the Container will continue to evaluate the JSP page when done with this Tag. If SKIP_PAGE is returned, the Container will stop evaluating the page when it done with this Tag.
- 6. Call the release() method.

Implementing Custom Tags

- Basic tags implement the Tag interface
  - Generate page content
  - Conditionally execute body content
  - Conditionally halt page execution
- **`TagSupport`** class provides default implementation

Sample


```
public class demo1 implements Tag {
    private PageContext pagecontext=null;
    public void setPageContext(PageContext pc) {
        this.pagecontext=pc;
    }
    public void setParent(Tag t) {
    }
    public Tag getParent() {
        return null;
    }
```

Sample

```
public int doStartTag() throws JspException {
        try {
            pagecontext.getOut().print("This is a test");
        }
        catch (IOException ex) {
        }
        return this.EVAL_BODY_INCLUDE;
    }
    public int doEndTag() throws JspException {
        return this.EVAL_PAGE;
    }
    public void release() {
        System.out.println("release");
    }
```

TLD and web.xml


```
<tag>
    <name>display</name>
    <tagclass>simapleTag.demo1</tagclass>
    <bodycontent>JSP</bodycontent>
  </tag>
```

web.xml

```
 <taglib>
    <taglib-uri>testtaglib</taglib-uri>
    <taglib-location>/WEB-INF/tlds/counter.tld</taglib-location>
  </taglib>
```

TLD描述

- Taglib 父类
- Tlibversion
- Jspversion
- Shortname 标记库的句柄
- Tag
  - Name 标记的名称
  - Tagclass 类名
  - Bodycontent 内容类型(JSP empty)
  - Info
  - Attribute
    - Name
    - Required 是否必须(true,false)
    - Rtexprvalue 值由JSP解析(true false)

JSP

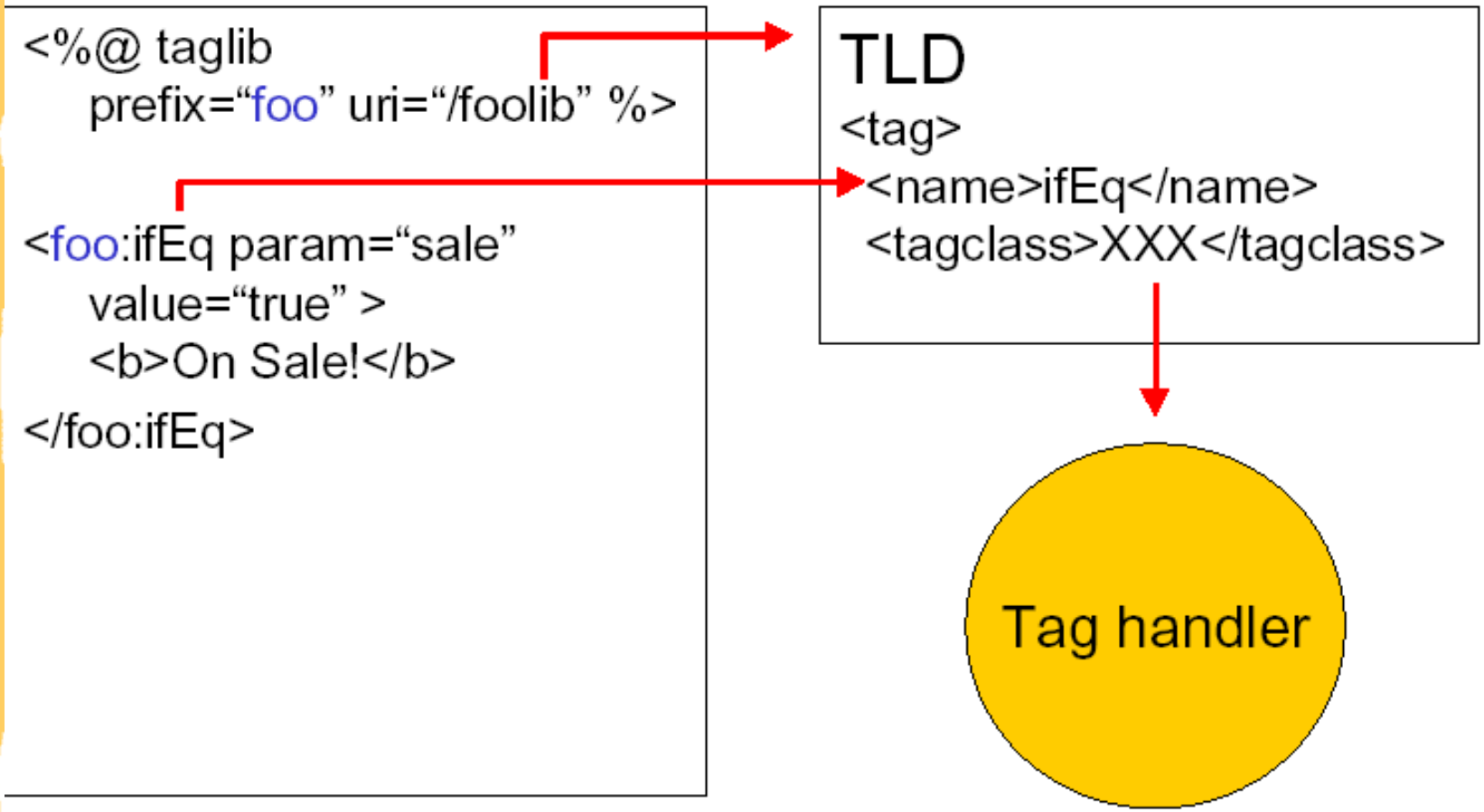



```
<%@ page contentType="text/html; charset=GBK" %>
<%@ taglib uri="testtaglib" prefix="counter" %>
<html>

<counter:display>Body Context</counter:display>
<BR>


</html>
```

How It All Fits Together



Arribute

- 语法 （name,value)
- 属性的类型
  - 转化时的值
  - 请求时的值
- 是否必须
- 类型转化

Implementing Custom Tags

- Attributes are modeled as JavaBeans properties
  - Values are set *via*
    ```
    public void setAttribute (type value)
    ```
  - Values are retrieved *via*
    ```
    public type getAttribute ()
    ```

Implementing Custom Tags

- Tags access JSP environment *via* the **`PageContext`** object
  - **`pageContext`** instance variable
- Accessing implicit objects
  - **`pageContext.getRequest()`**
  - **`pageContext.getResponse()`**
  - **`pageContext.getOut()`**

Implementing Custom Tags

- Accessing a page's JavaBeans
  - **`pageContext.getAttribute(`*`name`*`, `*`scope`*`)`**
  - **`pageContext.findAttribute(`*`name`*`)`**
- Introducing a bean/variable
  - **`pageContext.setAttribute(`*`name`*`,`**
    `                    `*`object`*`, `*`scope`*`)`

Sample- HelloTag

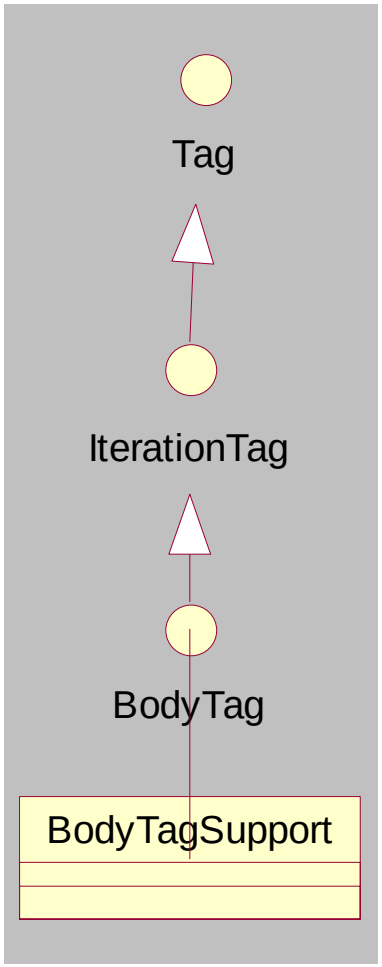
```
public class HelloTag extends TagSupport
{
    private String name="";
    public HelloTag(){
          super();
    }

    public void setName(String name){
          this.name=name;
    }
    public int doEndTag() throws javax.servlet.jsp.JspTagException {
          try{
                pageContext.getOut().write("Hello "+name+"!");
          }catch(java.io.IOException e){
                throw new JspTagException("IO Error: " + e.getMessage());
          }
          return EVAL_PAGE;
    }
}
```




Writing a BodyTag

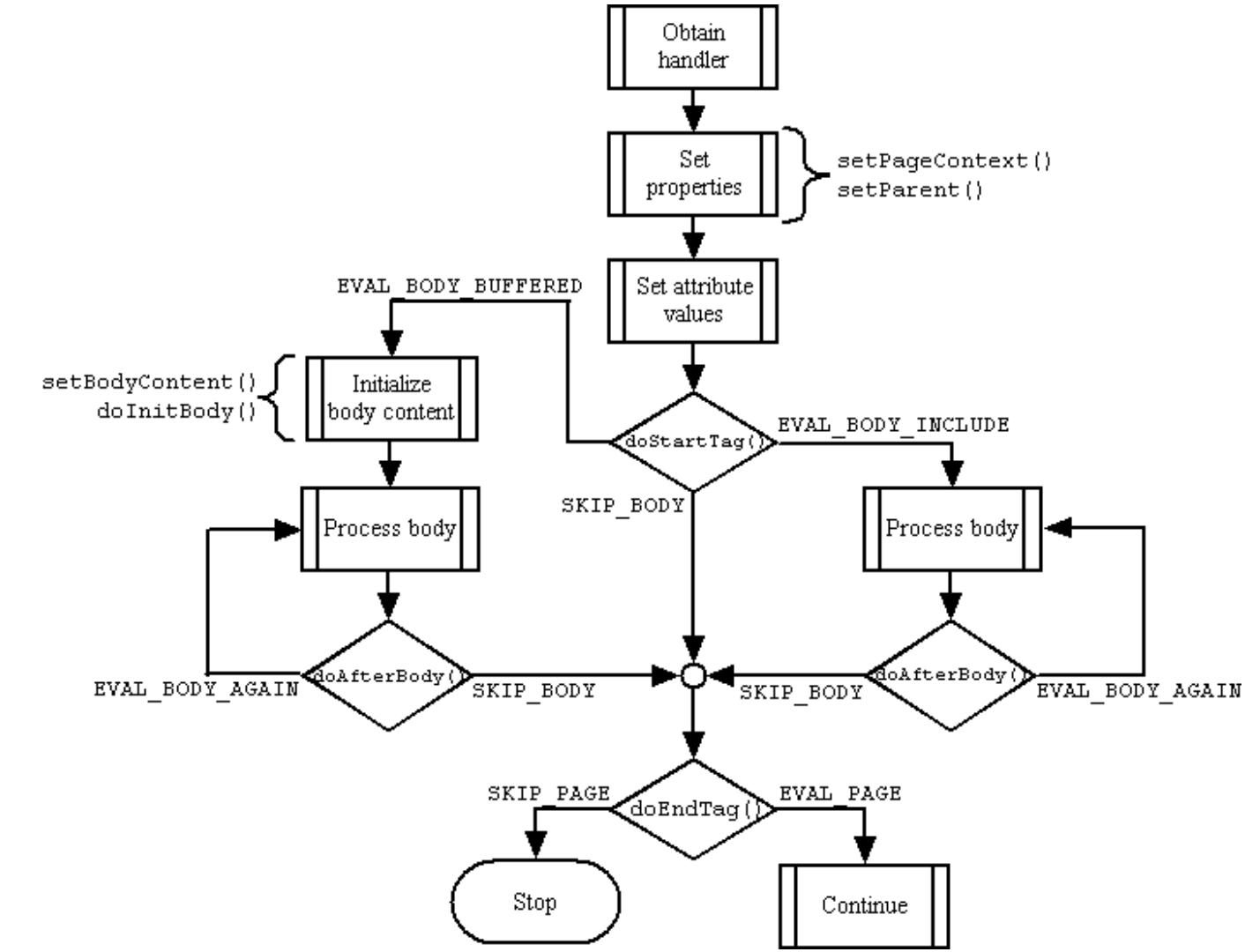
Body Tag



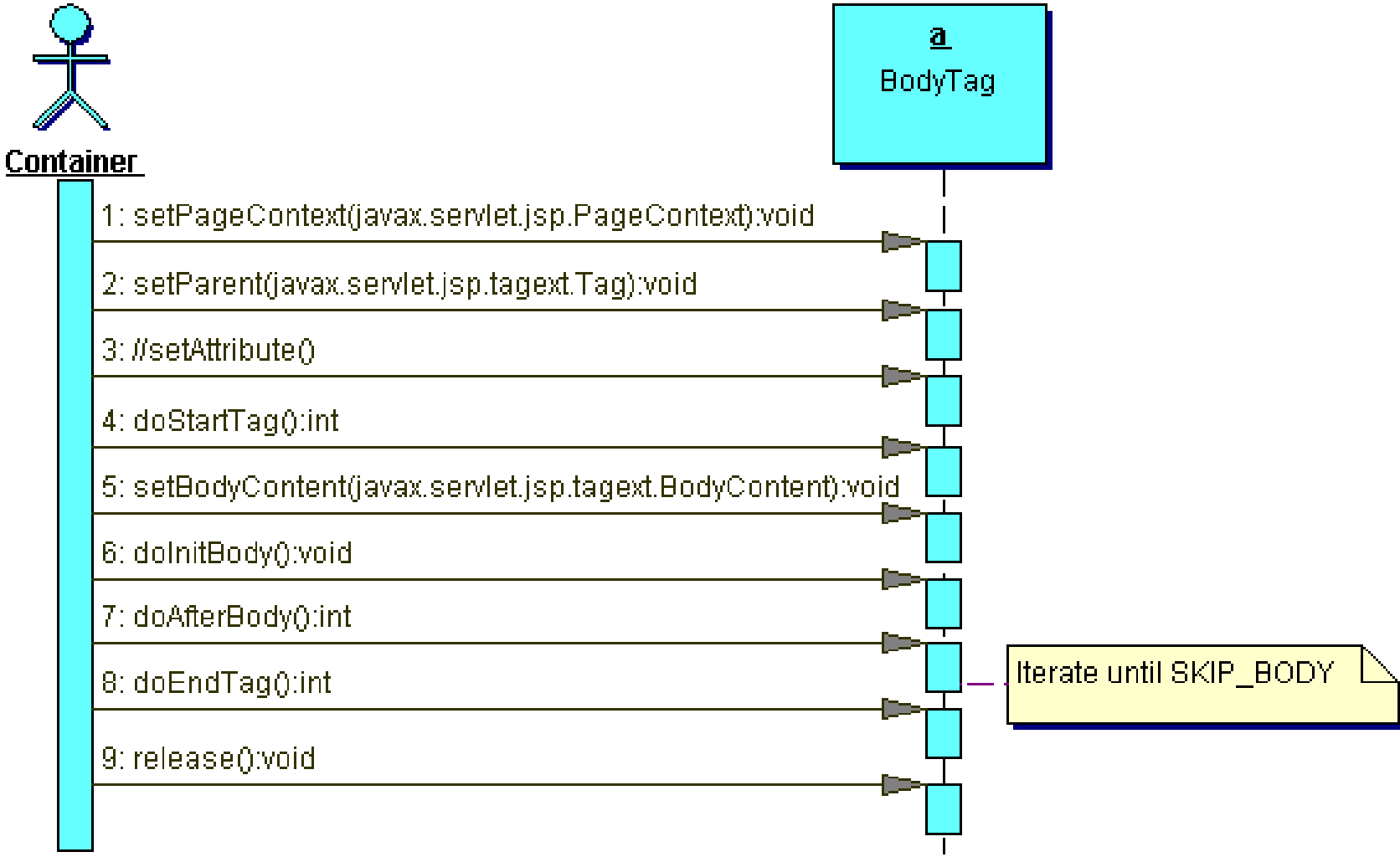
Implementing Custom Tags

- Complex tags implement the BodyTag interface
  - Generate page content
  - Conditionally execute body content
  - Process content generated by body
  - Repeat execution of body content
  - Conditionally halt page execution
- **`BodyTagSupport`** class provides default implementation

BodyTag Life-Cycle

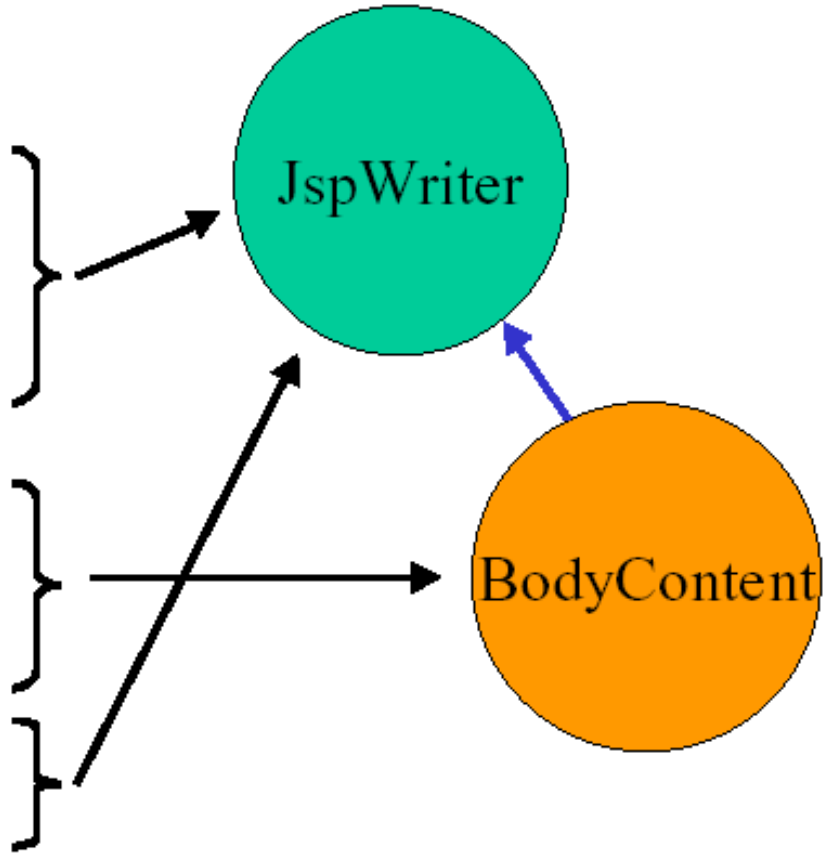


Container interact with a BodyTag



Buffering Hierarchy

```
<%@ taglib … %>
<html>
  Some text
  <jsp:getProperty …/>
  <foo:mail …>
    Dear
      <jsp:getProperty />
  </foo:mail>
</html>
```



BodyContent

BodyContent

(from tagext)

- BodyContent()
- clearBody()
- flush()
- getEnclosingWriter()
- getReader()
- getString()
- writeOut()

BodyContent

The BodyContent class extends the JspWriter class and can be used to process body evaluations so they can be retrieved later on. The content of a BodyContent class can be read with the Reader returned by the getReader() method. The content can also be read as a String by calling the getString() method. The BodyContent's content can be cleared by calling the clearBody() method. In order to write content to the BodyContent, the writeOut() method can be called with a writer (such as the JspWriter returned by the getEnclosingWriter() method).

FilterTag


```
public class FilterTag extends BodyTagSupport {
  public int doEndTag() throws JspException {
    if (bodyContent != null) {
      try {
        String content = bodyContent.getString();
        content = filter(content);
        // now clear the original body content and write back
        // the filtered content
        bodyContent.clearBody();
        bodyContent.print(content);
        // finally, write the contents of the bodyContent object back to the
        // original JspWriter (out) instance
        bodyContent.writeOut(getPreviousOut());
      } catch (IOException ioe) {
        throw new JspTagException(ioe.getMessage());
      }    }
    return EVAL_PAGE;
  }
```

TLD

```
<tag>
    <name>FilterTag</name>
    <tagclass>BodyTag.FilterTag</tagclass>
    <bodycontent>JSP</bodycontent>
  </tag>
```

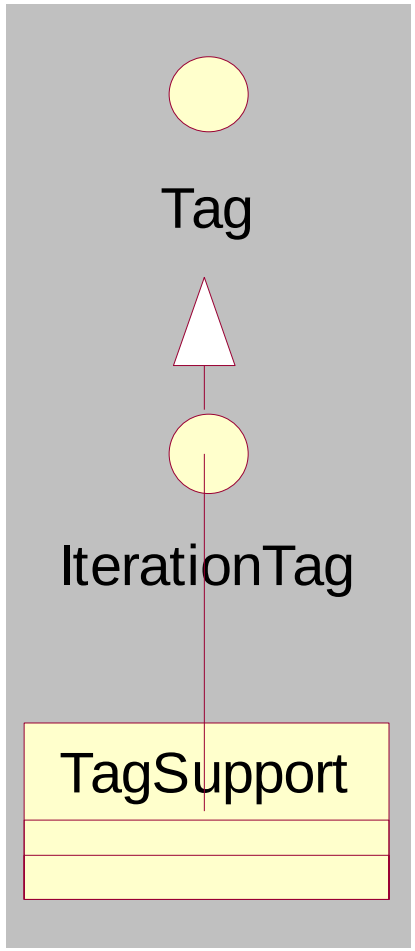
JSP

```
<body:FilterTag >
  This is a test
  fTMDd Hehe!!!
</body:FilterTag>
```




Write Iteration Tag

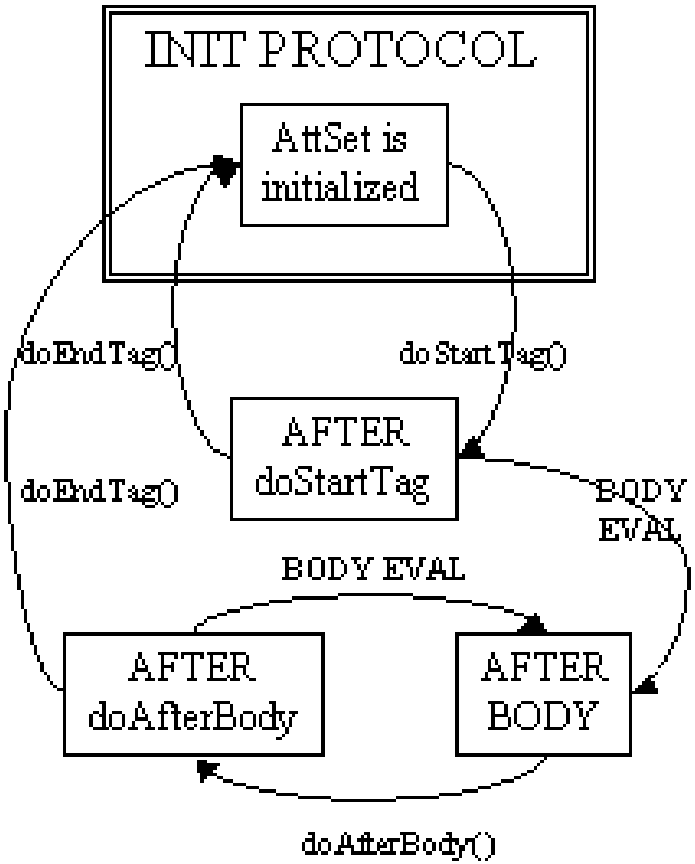
Iteration Tag



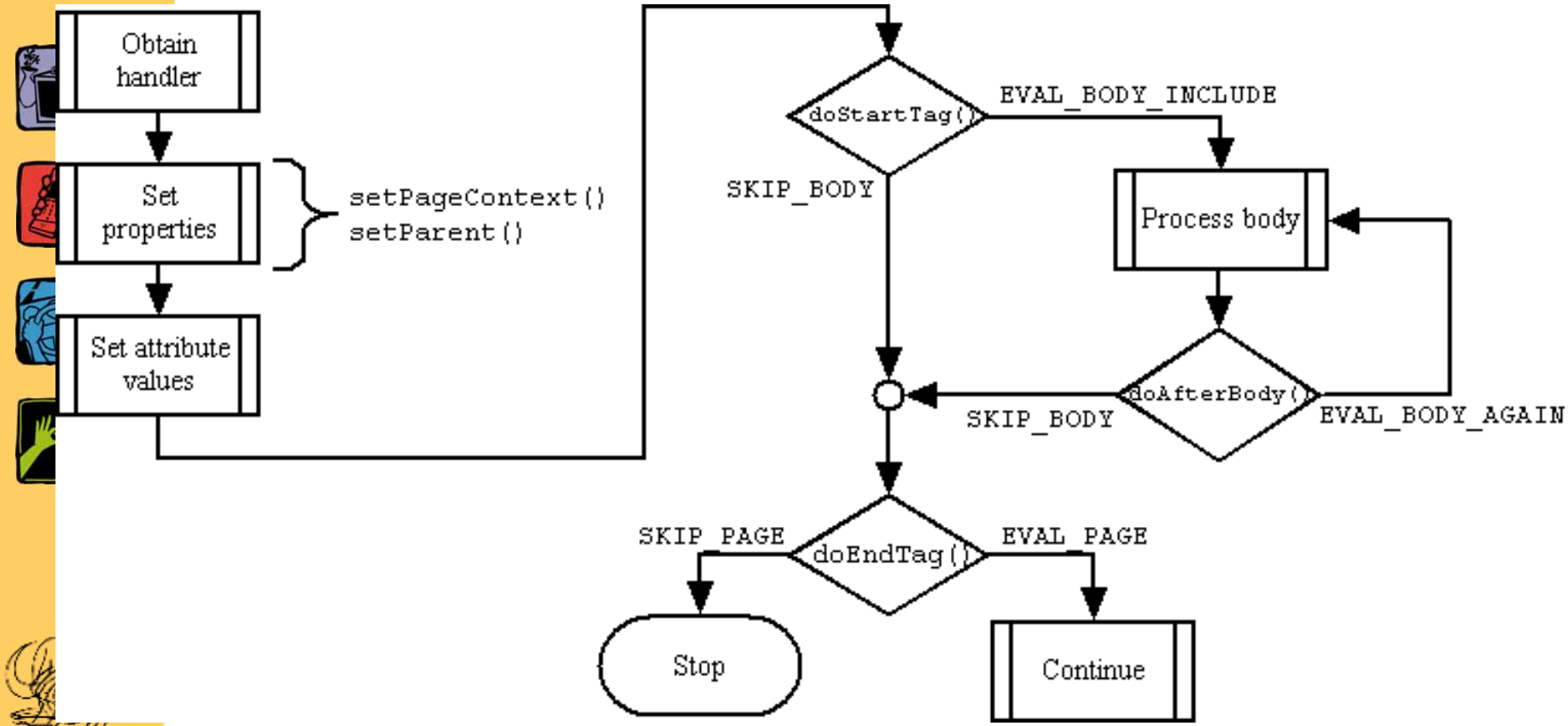
Implementing Custom Tags

- Iterative tags implement the IterationTag interface
  - Generate page content
  - Conditionally execute body content
  - Repeat execution of body content
  - Conditionally halt page execution
- `TagSupport` class provides default implementation
- Introduced in JSP 1.2

IterationTag Lifecycle



IterationTag Interface



Sample

```
public int doStartTag() throws JspTagException {
        Collection coll = (Vector)pageContext.getAttribute(var);

        iterator = coll.iterator();
else {if (iterator.hasNext()) {
            pageContext.setAttribute("Test", iterator.next());
            return EVAL_BODY_INCLUDE;
        }

            return SKIP_BODY;
        }
    }
```

Sample

```
public int doAfterBody() {
        if (iterator.hasNext()) {
            pageContext.setAttribute("Test",
  iterator.next());
            return EVAL_BODY_AGAIN;
        }
        else {
            return SKIP_BODY;
        }
    }
```




Writing Collaborating Tags

Collaborating Tags

嵌套的标记可能也会引用包含它的标记（诸如父标记和祖父标记），允许被链接的类相互调用对方的方法和特性。这样，子标记和父标记就可以共享数据。

使用如下两种方法之一，嵌套标记就可以引用祖先标记：

- TagSupport.getParent()：返回父标记；也就是包围该标记的最里层标记。
- TagSupport.findAncestorWithClass(from,class)：在特定的标记层次结构未知或必须预先设置时所用。

Sample- SwitchTag


```
public class SwitchTag extends TagSupport{
        String value;
        public SwitchTag(){
                super();
        }
        public void setValue(String value){
                this.value=value;
        }
        public String getValue(){
                return value;
        }
        public int doStartTag(){
                return EVAL_BODY_INCLUDE;
        }
}
```

Sample- CaseTag


```
public class CaseTag extends TagSupport{
    public int doStartTag() throws JspTagException{
        //SwitchTag
    parent=(SwitchTag)findAncestorWithClass(this,SwitchTag.class);
        SwitchTag parent=(SwitchTag)this.getParent();
            try{
                    if(parent.getValue().equals(getValue())){
                            return EVAL_BODY_INCLUDE;
                    }else{
                            return SKIP_BODY;
                    }
            }catch(NullPointerException e){
                    return SKIP_BODY;
            }
    }
}
```

JSP


```
<%@ taglib uri="mytags" prefix="mt" %>
<HTML>
<BODY BGCOLOR="#FFFFFF">
    <P>This example will show how to use collaborating Tags.</P>
    <mt:switch value="dark">
        <mt:            <P>This is Dark.</P>case value="light">
            <P>This is light.</P>
        </mt:case>
        <mt:case value="Dark">

        </mt:case>
        <mt:case value="dark">
            <P>This is dark.</P>
        </mt:case>
    </mt:switch>
</BODY>
</HTML>
```

TryCatchFinally

TryCatchFinally

- `doCatch(java.lang.Throwable t)`
- `public void doFinally()`

Sample


```
public class Catchandfinally extends TagSupport implements
    TryCatchFinally{
   public int doStartTag() throws JspException {
          throw new JspException("This is a  test");
   }
   public void doCatch(Throwable t) throws Throwable {
      System.out.println(t.getMessage());
   }
   public void doFinally() {
      System.out.println("doFinally");
   }

}
```




TagLib Project

Struts Taglibs

- Jakarta Struts 项目是 MVC的实现
- http://jakarta.apache.org/struts/index.html

Jakart Taglib

- Jakarta Taglibs 项目是 JSTL 1.0 参考实现的起源。
- http://jakarta.apache.org/taglibs/index.html

JSTL

- JSP Standard Tag Library (JSTL)
- http://java.sun.com/products/jsp/jstl/index.html

内容回顾

- TagLib基础
- Writing Tag
- Writing IterationTag
- Writing BodyTag
- Writing Collaborating Tags
- Third party TagLib

总结

- JSP 标准标记库（JSP Standard Tag Library，JSTL）是一个实现 Web 应用程序中常见的通用功能的定制标记库集，这些功能包括迭代和条件判断、数据管理格式化、XML 操作以及数据库访问。
- JSP 标准标记库（JSTL）是 JSP 1.2 定制标记库集，这些标记库实现大量服务器端 Java 应用程序常用的基本功能。通过为典型表示层任务（如数据格式化和迭代或条件内容）提供标准实现，JSTL 使 JSP 作者可以专注于特定于应用程序的开发需求，而不是为这些通用操作“另起炉灶”。
- JSTL 1.0 发布于 2002 年 6 月，由四个定制标记库（core、format、xml 和 sql）和一对通用标记库验证器（ScriptFreeTLV 和 PermittedTaglibsTLV）组成。

参考资料

- http://java.sun.com/products/jsp/jstl/index.html

sun公司的JSTL站点

- http://jakarta.apache.org/taglibs/index.html

Apache的Taglibs项目

- http://www.huihoo.com

国内一个关于中间件的专业站点

结束

谢谢大家！

Allen@huihoo.com
http://www.huihoo.com