Adv. Java means DURGA SIR..

ADV.JAVA

With

SCWCD / OCWCD

JSP Material

5. Building a Custom Tag Library

DURGA M.Tech

(Sun certified & Realtime Expert)

Ex. IBM Employee

Trained Lakhs of Students
for last 14 years across INDIA

JSP Custom Tag Library

Agenda :

- Introduction
- Components of Custom tag applications
- Tag Handler Class
- TLD(Tag Library descriptor)
- Taglib directive
- Flow of Custom Tag Application
- Tag Extension API

1) Classic Tag Model
   - Tag interface
   - IterationTag (I)
   - TagSupport (C)
   - BodyTag (I)
   - BodyContent (AC)
   - Comparision between TagSupport and BodyTagSupport classes
   - Nested Tags (or) co-operative tags
   - Getting Arbitrary Ancestor Object

2) SimpleTag Model : (jsp 2.0v)
   - SimpleTag interface
   - Jsp implicit objects and attributes in SimpleTag Model
   - Key differences between ClassictagModel and SimpleTagModel
   - DynamicAttributes

3) TagFiles (jsp2.0v)
   - Building & Using a Simple tag file
   - Declaring body-content for tag file
   - TagFiles with DynamicAttributes

Introduction :

Standard Actions, EL, JSTL are not succeeded to complete elimination of java code from Jsp.

Ex: Our requirement is to communicate with EJP or DB there is no standard action is defined for this requirement, we can defined our own tag from Jsp 1.1v onwards to meet our programming requirement such types of tags are nothing but Custom Tags.

All Custom tags can be divided into 3 parts

1. Classic Tag Model (Jsp1.1)

2. Simple Tag Model(2.0)
3. Tag files (Jsp2.0)

Components of Custom tag applications :

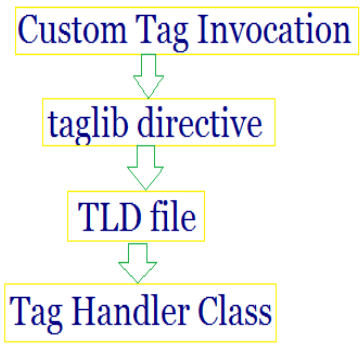
Custom tag application contains the following 3 components

1) Tag Handler Class :
   - It is a simple java class which defines entire required functionality.
   - Every Tag Handler class should compulsory implements Tag interface either directly or indirectly.
   - Web-container is responsible for creation of Tag handler class object for this always invokes public no-arg constructor , hence every tag handler class should compulsory contains public no-arg constructor.

2) TLD(Tag Library descriptor) :
   It is an xml file which provides mapping between Jsp(where custom tag functionality is required) and Tag handler class (where custom tag functional available).

3) Taglib directive :
   We can use Taglib directive to make custom tag functionality available to the Jsp, it provides location of TLD file.

Flow of Custom Tag Application :



- When ever Jsp engine encounters a custom tag it identifies prefix and checks for corresponding taglib directive with matched prefix from the taglib directive Jsp engine identifies location of TLD file.
- From the TLD file Jsp engine identifies corresponding Tag handler class Jsp engine execute the THC and provides required functionality to the Jsp.

Tag Extension API :

We can define custom tags by using only one package javax.servlet.jsp.tagext;

This package contains the following interfaces and class:

- Tag (I)
- IterationTag (I)
- BodyTag (I)
- TagSupport (C)
- BodyTagSupport (C)
- BodyContext (C)

Tag (I) :

- It act as a base interface for all THC
- Every THC should implement Tag interface either directly or indirectly.
- This interface defines 6 methods , these methods can be applicable on any THC object.
- If we want to include Tag body at most once without manipulation then we should go for Tag interface.

IterationTag (I) :

- It is the child interface of Tag , if we want to include Tag body multiple times without any manipulation then we should go for IterationTag interface.
- It defines one extra method doAfterBody().

BodyTag (I) :

It is the child interface of IterationTag, if we want to manipulate the TagBody then we should go for BodyTag interface.

It defines the following 2 methods

1. setBodyContent()
2. doInitBody()

TagSupport (C) :

1. It implements IterationTag interface and provides default implementation for all methods and hence this class act as base class to develop Tag Handler class where TagBody can be included any no.of times without manipulation.
2. This class act as a adaptor for Tag and IterationTag interfaces.

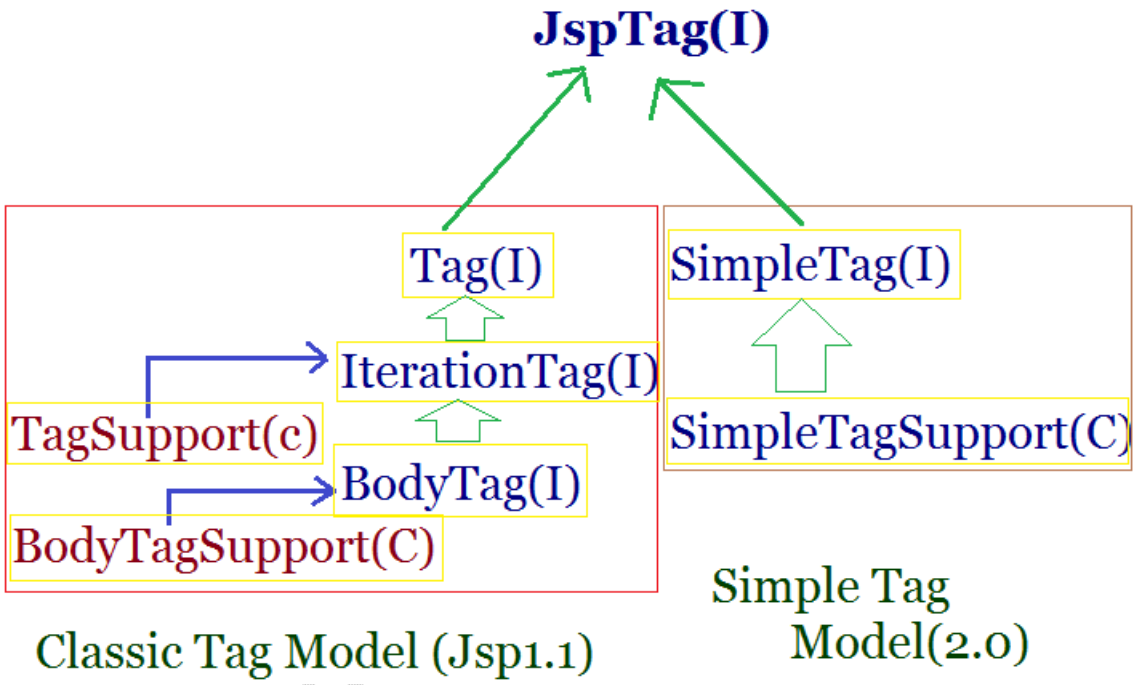
BodyTagSupport (C) :

1. This class implements BodyTag interface and provides default implementation for all its methods and it is child class of TagSupport class.

2. We can use this class as a base class for implementing custom tag that can process the tag body.
3. This class act as a adaptor for BodyTag interface.

BodyContext (C) :

1. BodyContext object act as a buffer for Tag body, it is the child class of JspWriter.
2. We can use this BodyContext class only with in BodyTag interface and BodyTagSupport class.



JspTag interface just for polymorphism purpose and doesn't contain any methods.

Classic Tag Model

Tag interface :

Tag interface defines the following 6 methods :

1. setPageContext()
2. setParent()
3. doStartTag()
4. doEndTag()
5. getParent()
6. release()

Tag interface defines the following 4 constants :

1. EVAL_BODY_INCLUDE
2. SKIP_BODY
3. EVAL_PAGE
4. SKIP_PAGE

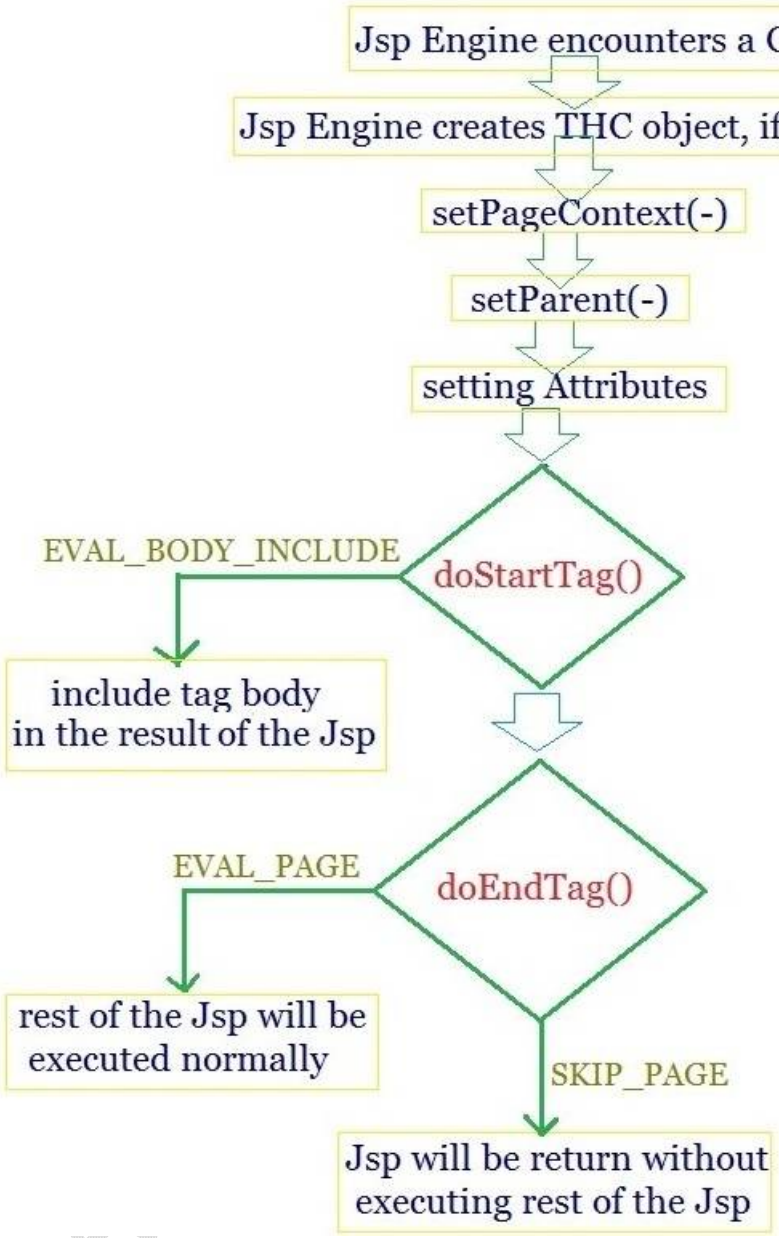
LifeCycle of Tag Handler class that implements Tag interface :

1) When ever Jsp engine encounters a Custom tag invocation it will identify corresponding tag handler class through taglib directive and TLD file.
2) Web-container will checks whether the corresponding Tag handler class object is available or not , if it is not available Jsp engine creating an object by executing public no-arg constructor , hence every Tag handler class should compulsory contains public no-arg constructor , violation leads we will get instantiation exception.
3) Jsp engine calls setPageContext(-) to make PageContext object available to the Tag handler class.
   public void setPageContext(PageContext pcontext)
4) THC can use this PageContext object to get all other Jsp implicit attributes.
5) Jsp engine calls setParent() to make parent tag object available to Tag Handler class
   public void setParent(Tag tag)
   This method is useful in nesting

Setting Attributes :

- A custom tag can be invoked with attributes also for every attribute the corresponding THC should contains one instance variable and corresponding setter methods like a bean , these are exactly same as properties of bean class for every custom tag attributes Jsp engine will execute corresponding setter method to make attribute values make its available to THC.
- Jsp engine will invoke doStartTag()
  public int doStartTag() throws JspException
- Entire custom tag functionality we have to define in this method only , this method can return either EVAL_BODY_INCLUDE or SKIP_BODY
- If this method return EVAL_BODY_INCLUDE then Tag Body will be included in the result of Jsp (output of the Jsp).
- If this method return a SKIP_BODY , then Jsp engine don't consider Tag Body.
- Jsp engine calls doEndTag() public int doEndTag() throws JspExceptiondoEndTag() can returns if this method return EVAL_PAGE the rest of the Jsp will be executed normally.
- If the returns SKIP_PAGE then Jsp engine will return without executing rest of the Jsp.
- Finally Jsp engine calls release() to perform cleanup activities when ever Tag Handler Object is no longer required.
  public void release()

For every custom tag invocation the following sequence of events will be happened (Flow Chart)



But release() method will not be called for every custom tag invocation.

custom tag example :

index.jsp

```
<%@taglib uri="customtags" prefix="mine" %>
 Hello this is Tag demo Jsp
```

```
<mine:welcome>
 This is body of the custom tag
</mine:welcome>
 This is after  tag
```

TagDemo.java

```
package com.tag;

import java.io.IOException;
import javax.servlet.jsp.JspException;
import javax.servlet.jsp.JspWriter;
import javax.servlet.jsp.PageContext;
import javax.servlet.jsp.tagext.Tag;

public class TagDemo implements Tag {

	public PageContext pageContext;

	static {
		System.out.println("class loading");
	}

	public TagDemo() {
		System.out.println("instantiation");
	}

	public void setPageContext(PageContext pageContext) {
		this.pageContext = pageContext;
		System.out.println("pagecontext object setted");
	}

	public void setParent(Tag tag) {
		 System.out.println("parent tag setted");
	}

	public int doStartTag() throws JspException {
			System.out.println("This is doStartTag() method");
			JspWriter out = null;
			try {
				out = pageContext.getOut();
				out.println("Welcome to custom tag developers");
				out.println("Welcome to custom Tag handler class");
			} catch (IOException e) {
				 e.printStackTrace();
			}
		       //return SKIP_BODY; // output 1
			return EVAL_BODY_INCLUDE; //output 2
```

```
    }

    public int doEndTag() throws JspException {
            System.out.println("This is doEndTag() method");
            return EVAL_PAGE; //output 1
          //return SKIP_PAGE; //output 2
    }
    public Tag getParent() {
           System.out.println("getParent() method");
           return null;
    }

    public void release() {
          System.out.println("release() method");
          pageContext = null;
    }
}
```

same program using annotations :

```
package com.tag;

import java.io.IOException;
import javax.servlet.jsp.JspException;
import javax.servlet.jsp.JspWriter;
import javax.servlet.jsp.PageContext;
import javax.servlet.jsp.tagext.Tag;

public class TagDemo implements Tag {

        private PageContext pageContext;

        static {
            System.out.println("class loading");
        }

         public TagDemo() {
               System.out.println("instantiation");
          }

         @Override
          public void setPageContext(PageContext pageContext) {
                   this.pageContext = pageContext;
                   System.out.println("pagecontext object setted");
          }

          @Override
          public void setParent(Tag tag) {
```

```
                System.out.println("parent tag setted");
            }

            @Override
            public int doStartTag() throws JspException {
                    System.out.println("This is doStartTag() method");

                    @SuppressWarnings("UnusedAssignment")
                    JspWriter out = null;
                    try {
                            out = getPageContext().getOut();
                            out.println("Welcome to custom tag developers<br>");
                            out.println("Welcome to custom Tag handler class<br>");
                    } catch (IOException e) {
                    }
                //return SKIP_BODY; // output 1
                    return EVAL_BODY_INCLUDE; //output 2
            }

            @Override
            public int doEndTag() throws JspException {
                    System.out.println("This is doEndTag() method");
                    return EVAL_PAGE; //output 1
                //return SKIP_PAGE; //output 2
            }

            @Override
            public Tag getParent() {
                    System.out.println("getParent() method");
                    return null;
            }

            @Override
            public void release() {
                    System.out.println("release() method");
                    setPageContext(null);
            }

            public PageContext getPageContext() {
                    return pageContext;
            }

}
```

myTld.tld

```
<taglib version="2.1">
    <tlib-version>1.2</tlib-version>
```

```
    <uri>customtags</uri>
    <tag>
        <name>welcome</name>
        <tag-class>com.tag.TagDemo</tag-class>
    </tag>
</taglib>
```

In the above program , if doStartTag() returns EVAL_BODY_INCLUDE and doEndTag returns EVAL_PAGE then the following is output :

```
Hello this is Tag demo Jsp
Welcome to custom tag developers
Welcome to custom Tag handler class
This is body of the custom tag
This is after tag

in console :
Info:  class loading
Info:  instantiation
Info:  pagecontext object setted
Info:  parent tag setted
Info:  This is doStartTag() method
Info:  This is doEndTag() method
```

Mapping of the Jsp with tld file :

approach 1 :

We can hard code the location of tld file for the uri attribute of taglib directive

Ex:

```
<%@taglib uri="/WEB-INF/myTld.tld" prefix="mine" %>
```

The problem in this approach is if there is change in name of the tld file or location of the tld file we have perform that which is complex to the programmer.

approach 2 :

Instead of hard coding the location of tld file we can define mapping through web.xml

Ex:

```
<%@taglib uri="http://jobs4times.com/scwcd/tags" prefix="mine" %>
```

web.xml

```
<web-app>
    <jsp-config>
```

```
        <taglib>
            <taglib-uri>http://jobs4times.com/scwcd/tags</taglib-uri>
            <taglib-location>/WEB-INF/myTld.tld</taglib-location>
        </taglib>
    </jsp-config>
</web-app>
```

This approach is highly recommended to use because there is any change in location of tld, just change in web.xml is enough not required to change in all Jsps

approach 3 :

We can map taglib uri attribute directly with uri attribute of tld file

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

myTld.tld

```
<taglib version="2.1">
    <tlib-version>1.2</tlib-version>
    <uri>mycustomtags</uri>
    ----------
</taglib>
```

This approach is not recommended because all web-servers may not be supported.

Structure of TLD file :

```
<taglib version="2.1">
    <tlib-version>1.2</tlib-version>
    <uri>www.jobs4times.com</uri>
    <tag>
        <name>welcome</name>
        <tag-class>com.tag.CustomTagDemo</tag-class>
        <body-content>XXXX</body-content>
        <attribute >
            <name>msg</name>
            <required>true</required>
            <rtexprvalue>true</rtexprvalue>
        </attribute>
        ----------
    </tag>
</taglib>
```

In the above tld file we can write EL functions also.

<body-content> :

It describes the type of content allow inside tag body the allowed values are

1. empty : The body of custom tag should be empty i.e., we can't take any tag body in this case we can invoke custom tag as follows <mine:mytag/>
2. tagdependent : Entire tag body will be treated plain text, Jsp engine sends tag body to the tag handler class without any processing.
3. scriptless : Tag body should not contains any scripting elements scriptlet, expression, declarations are not allowed but standard actions and EL expressions are allowed.
4. jsp : There are no restrictions on tag body whatever allowed in Jsp by default allowed in tag body also(including scripting elements).
   Note : The default value of <body-content> is "jsp".

<attribute > :

A custom tag can be invoked with attribute also we have to declare these attributes by using attribute tag , this tag contains the following child tags

1. <name> : name of the attribute
2. <required> : true means mandatory attribute, false means optional attribute, the default value is false
3. <rtexprvalue> (runtime expression value) :true means runtime expressions are allowed, false means runtime expressions are not allowed

```
<mime:mytag color="${param.color}" />
```

we have to provide only literals

```
<mime:mytag color="red" />
```

The default value is false

Write a demo program : empty custom tag with mandatory attribute :

A tag contain attribute also for each attribute we have to do the following things

- We have to declare that attribute in the TLD file by using <attribute> tag
- For each attribute THC should contain one instance variable and corresponding setter methods.
- In the case of optional attribute there may be a chance of NullPointerException will raise hence we have to handle carefull.

test.jsp

```
<%@taglib uri="http://jobs4times.com/tags" prefix="mine" %>
<%@page isELIgnored="false"%>

<mine:double number="3"/>
<mine:double number="${param.number}"/>
```

TagAttributeDemo.java

```
package com.tag;

import java.io.IOException;
import javax.servlet.jsp.JspException;
import javax.servlet.jsp.JspWriter;
import javax.servlet.jsp.PageContext;
import javax.servlet.jsp.tagext.Tag;

public class TagAttributeDemo implements Tag {

        public PageContext pageContext = null;
        private int number = 0;

        @Override
         public void setPageContext(PageContext pageContext) {
                this.pageContext = pageContext;
                System.out.println("pagecontext object setted");
         }

        @Override
         public void setParent(Tag tag) {
             System.out.println("parent tag setted");
         }

         public void setNumber(int number) {
                 this.number = number;
         }

        @Override
        @SuppressWarnings("CallToPrintStackTrace")
        public int doStartTag() throws JspException {
               System.out.println("This is doStartTag() method");

        @SuppressWarnings("UnusedAssignment")
         JspWriter out = null;
         try {
                out = pageContext.getOut();
                out.println("The double of the given no "+number+" is "+(2*number));
         } catch (IOException e) {
                e.printStackTrace();
         }
         return SKIP_BODY;
      // return EVAL_BODY_INCLUDE;
  }

    @Override
```

```
    public int doEndTag() throws JspException {
            System.out.println("This is doEndTag() method");
          //return EVAL_PAGE;
            return SKIP_PAGE;

     }

     @Override
     public Tag getParent() {
             System.out.println("getParent() method");
              return null;
     }

      @Override
       public void release() {
              System.out.println("release() method");
              pageContext = null;
       }

}
```

myTld.tld

```
<?xml version="1.0" encoding="UTF-8"?>
<taglib version="2.1" xmlns="http://java.sun.com/xml/ns/javaee">
 <tlib-version>1.2</tlib-version>
 <uri>www.jobs4times.com</uri>
 <tag>
  <description>double of number</description>
  <name>double</name>
  <tag-class>com.tag.TagAttributeDemo</tag-class>
  <body-content>empty</body-content>
  <attribute>
   <name>number</name>
   <required>true</required>
   <rtexprvalue>true</rtexprvalue>
  </attribute>
 </tag>
</taglib>
```

web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="3.1"
 xmlns="http://xmlns.jcp.org/xml/ns/javaee"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
     http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd">
```

```
<jsp-config>
 <taglib>
  <taglib-uri>http://jobs4times.com/tags</taglib-uri>
  <taglib-location>/WEB-INF/myTld.tld</taglib-location>
 </taglib>
</jsp-config>
</web-app>
```

http://localhost:8080/jstl/test.jsp?number=7

Write a demo program : empty custom tag with optional attribute :

demo.jsp

```
<%@taglib uri="http://jobs4times.com/tags" prefix="mine" %>
<%@page isELIgnored="false"%>

<mine:welcome name="Ashok"/> <br>
<mine:welcome name="Arun"/> <br>
<mine:welcome /> <br>
<mine:welcome name="Agastya"/> <br>
```

TagOptional.java

```
package com.tag;

import java.io.IOException;
import javax.servlet.jsp.JspException;
import javax.servlet.jsp.JspWriter;
import javax.servlet.jsp.PageContext;
import javax.servlet.jsp.tagext.Tag;

public class TagOptional implements Tag {

public PageContext pageContext=null;
private String name=null;

@Override
public void setPageContext(PageContext pageContext) {
this.pageContext=pageContext;
System.out.println("pagecontext object setted");
}

@Override
public void setParent(Tag tag) {
System.out.println("parent tag setted");
}

public void setName(String name){
```

```
 this.name=name;
}

@Override
@SuppressWarnings("CallToPrintStackTrace")
public int doStartTag() throws JspException {
 System.out.println("This is doStartTag() method");

 @SuppressWarnings("UnusedAssignment")
 JspWriter out=null;
 try{
  out=pageContext.getOut();
  if(name==null){
   out.println("
Hi Guest welcome to Custom Tags");
  }
  else{
   out.println("
Hi "+name+" welcome to Custom Tags");
  }
 }catch (IOException e) {
  e.printStackTrace();
}
 return SKIP_BODY;
 // return EVAL_BODY_INCLUDE;
}

@Override
public int doEndTag() throws JspException {
System.out.println("This is doEndTag() method");
 return EVAL_PAGE;
//return SKIP_PAGE;

}

@Override
public Tag getParent() {
System.out.println("getParent() method");
return null;
}

 @Override
public void release() {
System.out.println("release() method");
  pageContext=null;
}
```

```
}
```

myTld.tld

```
<?xml version="1.0" encoding="UTF-8"?>
<taglib version="2.1" xmlns="http://java.sun.com/xml/ns/javaee">
 <tlib-version>1.2</tlib-version>
 <uri>www.jobs4times.com</uri>
 <tag>
  <name>welcome</name>
  <tag-class>com.tag.TagOptional</tag-class>
  <body-content>empty</body-content>
  <attribute>
   <name>name</name>
   <required>false</required>
  </attribute>
 </tag>
</taglib>
```

web.xml

```
same as previous
```

output:

```
Hi Ashok welcome to Custom Tags
Hi Arun welcome to Custom Tags
Hi Guest welcome to Custom Tags
Hi Agastya welcome to Custom Tags
```

IterationTag(I) :

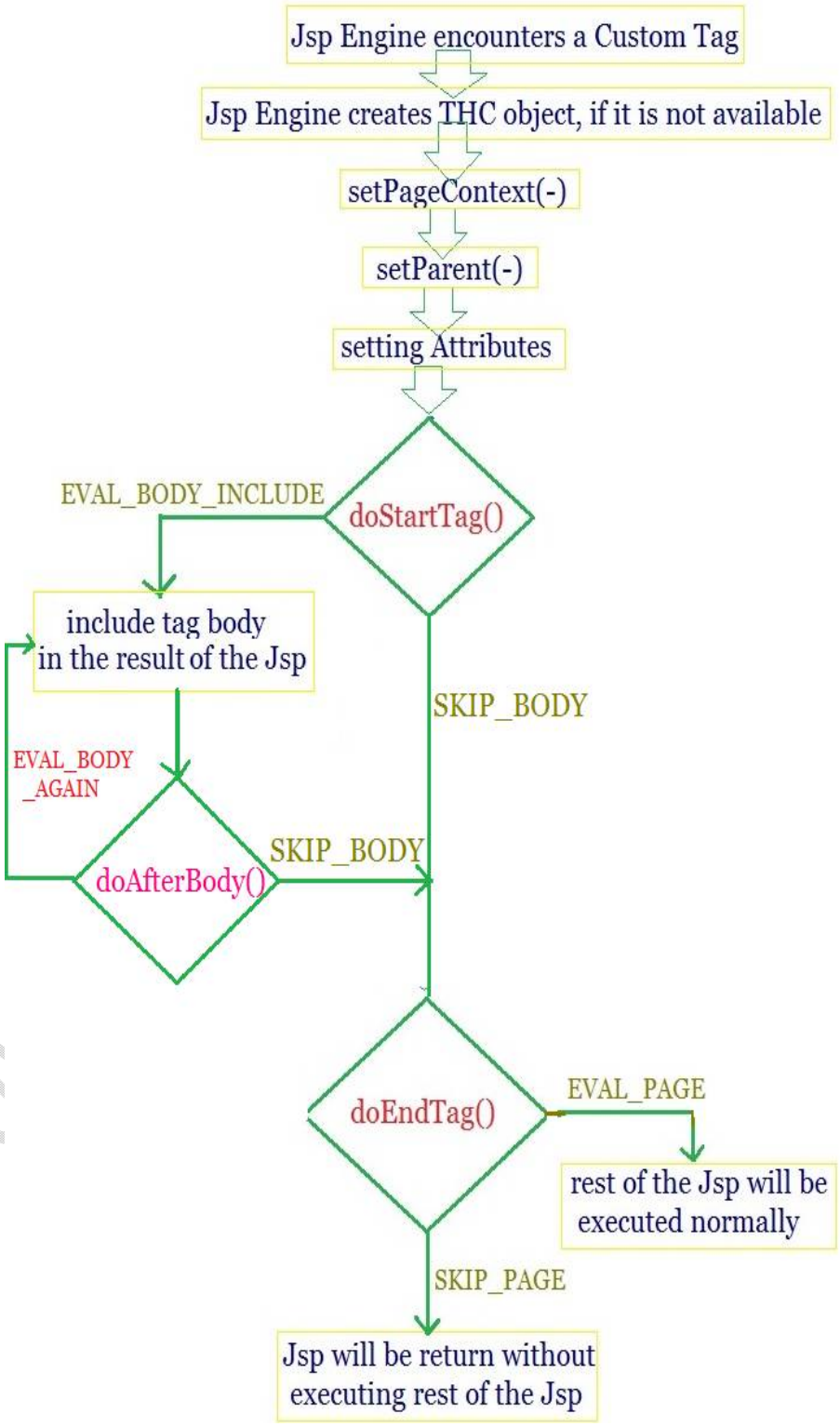
1. It is child interface of Tag
2. If we want to include tag body multiple times without manipulation then we should go for IterationTag.
3. IterationTag interface contains one extra method doAfterBody() one extra constant is EVAL_BODY_AGAIN

doAfterBody() :

public int doAfterBody() throws JspException

- This method will be executed after doStartTag() and it can return either EVAL_BODY_AGAIN or SKIP_BODY
- If it returns EVAL_BODY_AGAIN then tag body will be consider once again followed by execution of doStartBody()

Flow Chart of IterationTag :



Write a program for IterationTag :

test.jsp

```
<%@taglib uri="http://jobs4times.com/tags" prefix="mine" %>

<mine:myTag count="3">
 This is Iteration Tag Demo <br>
</mine:myTag>
Hi, This is afterBody
```

IterationTagDemo.java

```
package com.tag;

import javax.servlet.jsp.JspException;
import javax.servlet.jsp.PageContext;
import javax.servlet.jsp.tagext.IterationTag;
import javax.servlet.jsp.tagext.Tag;

public class IterationTagDemo implements IterationTag {

@SuppressWarnings("PublicField")
public PageContext pageContext=null;
private int count=0;
private Tag tag=null;

@Override
public void setPageContext(PageContext pageContext) {
this.pageContext=pageContext;
System.out.println("pagecontext object setted");
}

@Override
public void setParent(Tag tag) {
 this.tag=tag;
System.out.println("parent tag setter");
}

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

@Override
public int doStartTag() throws JspException {
 System.out.println("This is doStartTag() method");

 if(count>0){
  return EVAL_BODY_INCLUDE;
```

```
 }
 else{
  return SKIP_BODY;
 }
}

@Override
public int doAfterBody() throws JspException{
 if(--count>0)
  return EVAL_BODY_AGAIN;
 else
  return SKIP_BODY;
}

@Override
public int doEndTag() throws JspException {
System.out.println("This is doEndTag() method");
 return EVAL_PAGE;
}

@Override
public Tag getParent() {
System.out.println("getParent() method");
return tag;
}

 @Override
public void release() {
System.out.println("release() method");
  pageContext=null;
  count=0;
  tag=null;
}

}
```

myTld.tld

```
<?xml version="1.0" encoding="UTF-8"?>
<taglib version="2.1" xmlns="http://java.sun.com/xml/ns/javaee">
 <tlib-version>1.2</tlib-version>
 <uri>www.jobs4times.com</uri>
 <tag>
  <name>myTag</name>
  <tag-class>com.tag.IterationTagDemo</tag-class>
  <body-content>tagdependent</body-content>
  <attribute>
   <name>count</name>
```

```
    <required>true</required>
  </attribute>
 </tag>
</taglib>
```

web.xml

```
<jsp-config>
 <taglib>
  <taglib-uri>http://jobs4times.com/tags</taglib-uri>
  <taglib-location>/WEB-INF/myTld.tld</taglib-location>
 </taglib>
</jsp-config>
```

output :

```
This is Iteration Tag Demo
This is Iteration Tag Demo
This is Iteration Tag Demo
Hi, This is afterBody
```

TagSupport (C) :

- The main drawback of implements Tag and IterationTag interfaces directly is we have to provide implementation for all methods even though most of the times we have to consider doStartTag(), doAfterBody(), doEndTag()
- We can resolve this problem by using TagSupport class can implement IterationTag and provides default implementation for all its methods and it is very easy to extend TagSupport class and override required methods instead of implementing all methods.

Internal implementation of TagSupport class :

```
import java.io.Serializable;
import javax.servlet.jsp.JspException;
import javax.servlet.jsp.PageContext;
import javax.servlet.jsp.tagext.IterationTag;
import javax.servlet.jsp.tagext.Tag;

public class TagSupport implements IterationTag,Serializable {
 private static final long serialVersionUID = 1L;

private Tag parent;
protected PageContext pageContext;

@Override
public void setPageContext(PageContext pageContext) {
this.pageContext=pageContext;
System.out.println("pagecontext object setted");
```

```
}
@Override
public void setParent(Tag parent) {
 this.parent=parent;
System.out.println("parent tag setter");
}

@Override
public int doStartTag() throws JspException {
 System.out.println("This is doStartTag() method");
 return SKIP_BODY;
}

@Override
public int doAfterBody() throws JspException{
 System.out.println("This is doAfterBody() method");
 return SKIP_BODY;
}

@Override
public int doEndTag() throws JspException {
System.out.println("This is doEndTag() method");
 return EVAL_PAGE;
}

@Override
public Tag getParent() {
System.out.println("getParent() method");
return parent;
}

 @Override
public void release() {
System.out.println("release() method");
  pageContext=null;
  parent=null;
}

}
```

Note :

1. The default return type of doStartTag(), doAfterBody() is SKIP_BODY
2. The default return type of doEndTag() is EVAL_PAGE

pageContext variable is by default variable to the child classes hence we can use this variable directly in our tag handler classes

Write a demo program for TagSupport class :

test.jsp

```
<%@taglib uri="http://jobs4times.com/tags" prefix="mine" %>

This is before custom tag <br>
<mine:myTag>
 This is Tag body <br>
</mine:myTag>
This is after custom tag
```

TagSupportDemo.java

```
package com.tag;

import java.io.IOException;
import javax.servlet.jsp.JspException;
import javax.servlet.jsp.JspWriter;
import javax.servlet.jsp.tagext.TagSupport;

public class TagSupportDemo extends TagSupport {
 private static final long serialVersionUID = 1L;

@Override
public int doStartTag() throws JspException {
 JspWriter out=null;
 try{
  out=pageContext.getOut();
 out.println("This is THC using TagSupport class<br>");
 }
 catch(IOException e){
  e.printStackTrace();
 }

 return EVAL_BODY_INCLUDE;
}

}
```

myTld.tld

```
<?xml version="1.0" encoding="UTF-8"?>
<taglib version="2.1" xmlns="http://java.sun.com/xml/ns/javaee">
  <tlib-version>1.2</tlib-version>
  <uri>www.jobs4times.com</uri>
  <tag>
```

```
  <name>myTag</name>
  <tag-class>com.tag.TagSupportDemo</tag-class>
  <body-content>tagdependent</body-content>
 </tag>
</taglib>
```

web.xml

```
<jsp-config>
 <taglib>
  <taglib-uri>http://jobs4times.com/tags</taglib-uri>
  <taglib-location>/WEB-INF/myTld.tld</taglib-location>
 </taglib>
</jsp-config>
```

http://localhost:8080/jstl/test.jsp?

output :

```
This is before custom tag
This is THC using TagSupport class
This is Tag body
This is after custom tag
```

Example : 2

test.jsp

```
<%@taglib uri="http://jobs4times.com/tags" prefix="mine" %>

<mine:myTag count="3">
 This is Iteration body <br>
</mine:myTag>
This is after Iteration
```

TagSupportDemo.java

```
package com.tag;

import javax.servlet.jsp.JspException;
import javax.servlet.jsp.tagext.TagSupport;

public class TagSupportDemo extends TagSupport {
 private static final long serialVersionUID = 1L;

 private int count=0;

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

```
@Override
public int doStartTag() throws JspException {
 if(count>0)
  return EVAL_BODY_INCLUDE;
 else
  return SKIP_BODY;
}

 @Override
 public int doAfterBody() throws JspException{
 if(--count>0)
  return EVAL_BODY_AGAIN;
 else
  return SKIP_BODY;
}

}
```

myTld.tld

```
<?xml version="1.0" encoding="UTF-8"?>
<taglib version="2.1" xmlns="http://java.sun.com/xml/ns/javaee">
 <tlib-version>1.2</tlib-version>
 <uri>www.jobs4times.com</uri>
 <tag>
  <name>myTag</name>
  <tag-class>com.tag.TagSupportDemo</tag-class>
  <body-content>tagdependent</body-content>
  <attribute>
   <name>count</name>
   <required>true</required>
  </attribute>
 </tag>
</taglib>
```

web.xml

```
<jsp-config>
 <taglib>
  <taglib-uri>http://jobs4times.com/tags</taglib-uri>
  <taglib-location>/WEB-INF/myTld.tld</taglib-location>
 </taglib>
</jsp-config>
```

output :

```
This is Iteration body
This is Iteration body
This is Iteration body
```

This is after Iteration

BodyTag (I) :

- It is the child interface of IterationTag
- If we want to manipulate the body then we should go for BodyTag interface.
- BodyTag interface defines the following 2 extra methods
    1. public void setBodyContent(BodyContent bodyContent)
    2. public void doInitBody() throws JspException
- BodyTag interface defines the following 2 extra constants
    1. EVAL_BODY_BUFFERED
    2. EVAL_BODY_TAG (deprecated)

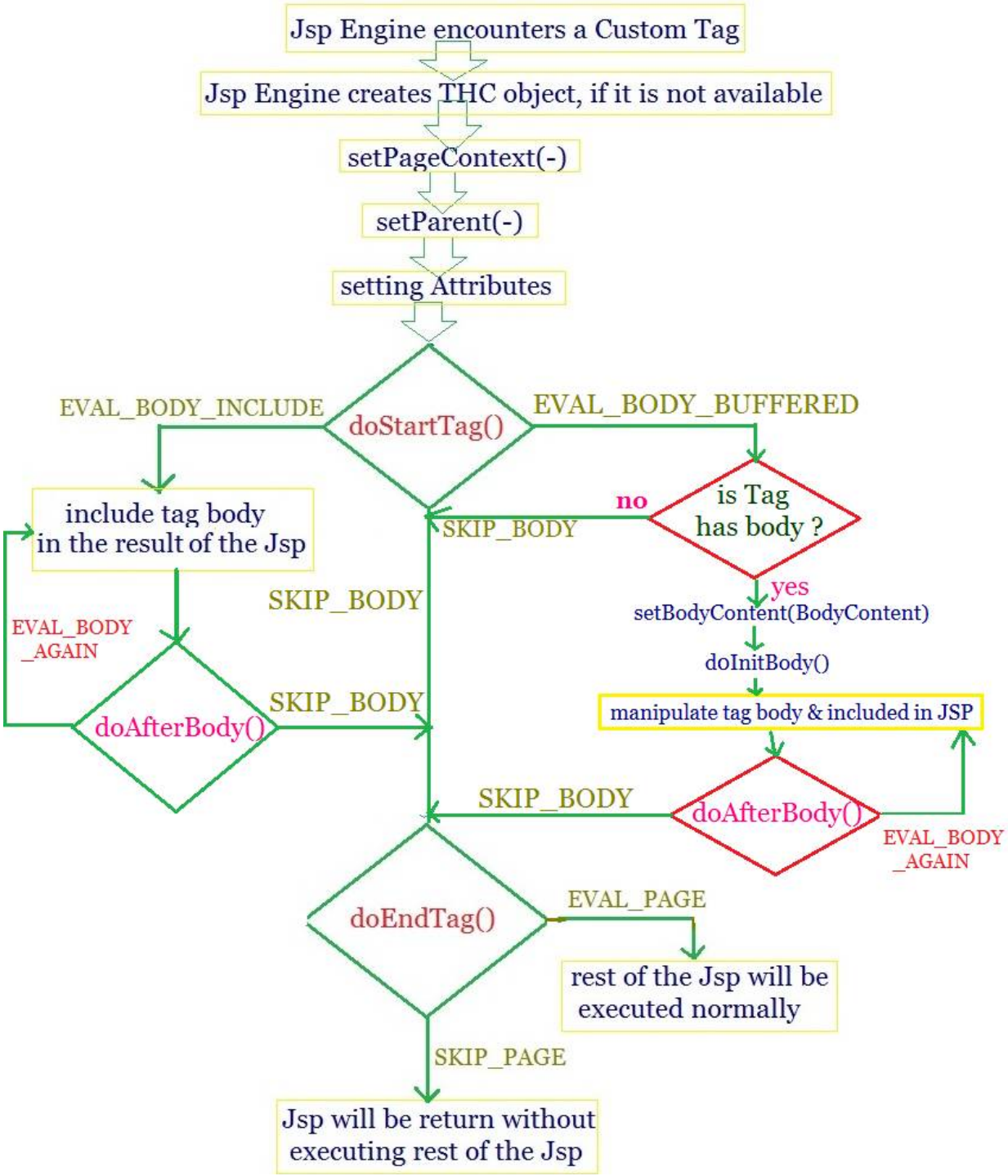
BodyContent (AC) :

- We can use this BodyContent object to hold tag body
- BodyContent class is the child class of JspWriter
- BodyContent class is an abstract class and vendor is responsible to provide implementation

Note : BodyContent class object act as a buffer for tag body , if we want to manipulate that tag body, first we have to retrieve that tag body from the BodyContent object , for that BodyContent class defines the following methods

1. public String getString() : returns tag body in the form of String.
2. public Reader getReader() : return a Reader object to extract tag body.
3. public JspWriter getEnclosingWriter() : It returns JspWriter to print data to the jsp page.
4. public void ClearBody() : To clear body present in BodyContent object i.e., entire tag body present in BodyContent will be removed.

Flow Chart of BodyTag interface :



Life Cycle of BodyTag interface :

- Life Cycle of BodyTag interface is exactly similar to IterationTag handler, and difference is doStartTag() returns either EVAL_BODY_INCLUDE or EVAL_BODY_BUFFERED.
- If the method returns EVAL_BODY_BUFFERED and tag contains body then jsp engine creates BodyContent object and invoke setBodyContent() by passing BodyContent object as a argument followed by doInitBody().

Note : we can write a logic to manipulate the tag body with in doAfterBody()

setBodyContent(-) and doInitBody() won't be executed in the following cases

case 1 : If doStartTag() returns either EVAL_BODY_INCLUDE or SKIP_BODY.

CASE 2 : If doStartTag() returns either EVAL_BODY_BUFFERED and tag doesn't contain body.

Implementation of BodyTagSupport class :

- This class extends TagSupport class and implements BodyTag interface.
- This class provides default implementation for all 9 methods available in BodyTag interface.
- It is very easy to write Tag handler class by extending BodyTagSupport class instead of implementing BodyTag interface directly.
- In this case we have to provide implementation only for required methods but not for all 9 methods.

Internal Implementation of BodyTagSupport class :

BodyTagSupport.java

```
import javax.servlet.jsp.JspException;
import javax.servlet.jsp.tagext.BodyContent;
import javax.servlet.jsp.tagext.BodyTag;
import javax.servlet.jsp.tagext.TagSupport;

public class BodyTagSupport extends TagSupport implements BodyTag{
 private static final long serialVersionUID = 1L;
 protected BodyContent bodyContent;

 @Override
 public void setBodyContent(BodyContent bodyContent){
 this.bodyContent=bodyContent;
}

 @Override
```

```
public void doInitBody(){
 System.out.println(" doInitBody method");
}

 @Override
 public int doStartTag() throws JspException{
 return EVAL_BODY_BUFFERED;
}

}
```

- pageContext, bodyContent variables are by default available to our tag handler classes we can use directly.
- The default return type doStartTag() in BodyTagSupport is EVAL_BODY_BUFFERED, doAfterBody() is SKIP_BODY and doEndTag is EVAL_PAGE.

Write a demo program for the usage of BodyTagSupport class

test.jsp

```
<%@taglib uri="http://jobs4times.com/tags" prefix="mine" %>

<mine:myTag>
 This is Body Tag Support Demo body <br>
</mine:myTag>
```

BodyTagSupportDemo.java

```
package com.tag;

import java.io.IOException;
import javax.servlet.jsp.JspException;
import javax.servlet.jsp.JspWriter;
import javax.servlet.jsp.tagext.BodyTagSupport;

public class BodyTagSupportDemo extends BodyTagSupport{
 private static final long serialVersionUID = 1L;

 @Override
 public int doAfterBody() throws JspException{
 JspWriter out=null;
 try{
  String data=bodyContent.getString();
  data=data.toUpperCase();
  out=bodyContent.getEnclosingWriter();
  out.println(data);
 }
 catch(IOException e){
```

```
 e.printStackTrace();
 }
return SKIP_BODY;
}

}
```

myTld.tld

```
<taglib version="2.1" xmlns="http://java.sun.com/xml/ns/javaee">
 <tlib-version>1.2</tlib-version>
 <uri>www.jobs4times.com</uri>
 <tag>
  <name>myTag</name>
  <tag-class>com.tag.BodyTagSupportDemo</tag-class>
 </tag>
</taglib>
```

web.xml

Same as previous

output :

```
THIS IS BODY TAG SUPPORT DEMO BODY
```

Comparision between TagSupport and BodyTagSupport classes :

Method	TagSupport (C)	BodyTagSupport (C)
doStartTag() :		
Possible return values	EVAL_BODY_INCLUDE SKIP_BODY	EVAL_BODY_BUFFERED EVAL_BODY_INCLUDE SKIP_BODY
Default return values from implemented classes	SKIP_BODY	EVAL_BODY_BUFFERED
no.of times invoke per tag	only once	only once
doAfterBody() :		
Possible return values	EVAL_BODY_AGAIN SKIP_BODY	EVAL_BODY_AGAIN SKIP_BODY
Default return values from implemented classes	SKIP_BODY	SKIP_BODY
no.of times invoke per tag	0 or more	0 or more
doEndTag() :		
Possible return values	EVAL_PAGE SKIP_PAGE	EVAL_PAGE SKIP_PAGE

Default return values from implemented classes	EVAL_PAGE	EVAL_PAGE
no.of times invoke per tag	only once	only once
setBodyContent() & doInitBody() :		
circumstances under which these methods no.of times call per tag ?	not applicable	executed only once iff doStartTag() return EVAL_BODY_BUFFERED and tag has body.

Nested Tags (or) co-operative tags :

Some times a group of tags work together will perform certain functionality such type of tags are called Nested tags

Ex : In JSTL <c:choose>, <c:when>, <c:otherwise> tags work together to implement core java if-else and switch statement such type of tags are called Co-operative & Nested Tags

Ex:

```
<c:choose>
 <c:when>
   Action
 </c:when>
 <c:otherwise>
   Default Action
 </c:otherwise>
</c:choose>
```

Ex:

```
<mine:myTag>
 <mine:myDetails />
</mine:myTag>
```

From child tag handler if we want to get parent tag object we have to use getParent() method

Write a demo program for nested custom tags :

test.jsp

```
<%@taglib uri="http://jobs4times.com/tags" prefix="mine" %>

<mine:myTag>
 <mine:myTag>
   <mine:myTag>
     <mine:myTag />
   </mine:myTag>
 </mine:myTag>
</mine:myTag>
```

NestedTagDemo.java

```
package com.tag;
```

```
import java.io.IOException;
import javax.servlet.jsp.JspException;
import javax.servlet.jsp.JspWriter;
import javax.servlet.jsp.tagext.Tag;
import javax.servlet.jsp.tagext.TagSupport;
public class NestedTagDemo extends TagSupport {
 private static final long serialVersionUID = 1L;

 @Override
 public int doStartTag() throws JspException{
  int nestedLevel=0;
  JspWriter out=null;
  Tag tag=getParent();
   while(tag!=null){
    nestedLevel++;
    tag=tag.getParent();
   }
  try{
   out=pageContext.getOut();
   out.println("<br>Nested Level : "+nestedLevel);
  }
  catch(IOException e){
   e.printStackTrace();
  }
  return EVAL_BODY_INCLUDE;
 }

}
```

myTld.tld

```
<taglib version="2.1" xmlns="http://java.sun.com/xml/ns/javaee">
  <tlib-version>1.2</tlib-version>
  <uri>www.jobs4times.com</uri>
  <tag>
    <name>myTag</name>
    <tag-class>com.tag.NestedTagDemo</tag-class>
  </tag>
</taglib>
```

web.xml

```
Same as previous
```

output :

```
Nested Level : 0
Nested Level : 1
```

```
Nested Level : 2
Nested Level : 3
```

Example :

menu.jsp

```
<%@taglib uri="http://jobs4times.com/tags" prefix="mine" %>

<mine:menu>
 <mine:menuItem item="chicken"/>
 <mine:menuItem item="mutton"/>
 <mine:menuItem item="fish"/>
 <mine:menuItem item="noodles"/>
</mine:menu>
```

Write a Tag handler class(THC) that takes the menu items fron its child and print it

MenuTag.java

```
package com.tag;

import java.io.IOException;
import java.util.ArrayList;
import javax.servlet.jsp.JspException;
import javax.servlet.jsp.JspWriter;
import javax.servlet.jsp.tagext.TagSupport;

public class MenuTag extends TagSupport {
 private static final long serialVersionUID = 1L;

ArrayList<String> menuList=null;

public void addItem(String item){
 menuList.add(item);
}

@Override
public int doStartTag() throws JspException{
 menuList=new ArrayList<String>();
 return EVAL_BODY_INCLUDE;
}

@Override
public int doEndTag() throws JspException{
```

```
JspWriter out=null;
try{
 out=pageContext.getOut();
 out.println("The menu items are : "+menuList);
}
catch(IOException e){
  e.printStackTrace();
}
return EVAL_PAGE;
}

}
```

Write a program to accept the menu items and add to menu list of its parent

MenuItemTag.java

```
package com.tag;

import javax.servlet.jsp.JspException;
import javax.servlet.jsp.tagext.TagSupport;

public class MenuItemTag extends TagSupport {
 private static final long serialVersionUID = 1L;

public String item=null;

public void setItem(String item){
 this.item=item;
}

 @Override
 public int doStartTag() throws JspException{
 MenuTag menuTag=(MenuTag)getParent();
 menuTag.addItem(item);
 return SKIP_BODY;
}

}
```

myTld.tld

```
<?xml version="1.0" encoding="UTF-8"?>
<taglib version="2.1" xmlns="http://java.sun.com/xml/ns/javaee">
 <tlib-version>1.2</tlib-version>
 <uri>www.jobs4times.com</uri>
 <tag>
   <name>menu</name>
```

```
  <tag-class>com.tag.MenuTag</tag-class>
 </tag>
 <tag>
  <name>menuItem</name>
  <tag-class>com.tag.MenuItemTag</tag-class>
   <attribute>
    <name>item</name>
    <required>true</required>
   </attribute>
 </tag>
</taglib>
```

web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="3.1"
 xmlns="http://xmlns.jcp.org/xml/ns/javaee"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
     http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd">

<jsp-config>
 <taglib>
  <taglib-uri>http://jobs4times.com/tags</taglib-uri>
  <taglib-location>/WEB-INF/myTld.tld</taglib-location>
 </taglib>
</jsp-config>
</web-app>
```

output :

```
The menu items are : [chicken, mutton, fish, noodles]
```

Getting Arbitrary Ancestor Object :

We can get immediate parent by using getParent(), TagSupport class defines the following method to get an arbitrary ancestor class object

public static Tag findAncestorWithClass(Tag t, Class c);

- **Accessing jsp implicit objects and attributes in tag handler class with in the tag handler class we can get jsp implicit objects by using PageContext object.**
- **Tag handler class can get PageContext object an argument to setPageContext().**

PageCotext class defines the following methods to get jsp implicit objects :

request	---->	getRequest()
response	---->	getResponse()

config	---->	getServletConfig()
application	---->	getServletContext()
session	---->	getSession()
out	---->	getOut()
page	---->	getPage()
exception	---->	getException()

All the methods we have to call on PageContext object

Note : exception implicit object is available only in error pages , if the enclosing jsp is not error page then getException() returns null.

Accessing attributes by using PageContext object :

PageContext class defines the following methods to perform attribute management in any scope (we can perform attribute management in tag handler classes also)

1. public void setAttribute(String name, Object value);
2. public void setAttribute(String name, Object value, int scope);
3. public Object getAttribute(String name);
4. public Object getAttribute(String name, int scope);
5. public void removeAttribute(String name);
6. public void removeAttribute(String name, int scope);
7. public Object findAttribute(String name);
8. public Enumeration getAttributeNamesInScope(int scope);

myJsp.jsp

```
<%@taglib uri="http://jobs4times.com/tags" prefix="mine" %>

JSP Implicit Objects from Custom Tags<br>
<mine:jspImplicitObjects/>
```

JspImplicitObj.java

```
package com.tag;

import javax.servlet.ServletConfig;
import javax.servlet.ServletContext;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpSession;
import javax.servlet.jsp.JspException;
import javax.servlet.jsp.JspWriter;
import javax.servlet.jsp.tagext.TagSupport;
```

```
public class JspImplicitObj extends TagSupport {
 private static final long serialVersionUID = 1L;
 private String mail="Ask";

@Override
public int doStartTag() throws JspException{
 JspWriter out=null;
 try{
  ServletRequest request=pageContext.getRequest();
  ServletResponse response=pageContext.getResponse();
  ServletConfig config=pageContext.getServletConfig();
  ServletContext application=pageContext.getServletContext();
  HttpServlet page=(HttpServlet)pageContext.getPage();
  HttpSession session=pageContext.getSession();
   out=pageContext.getOut();
   Throwable exception=pageContext.getException();

   out.println("<br>The Server details:"+request.getServerName()+" "
                          +request.getServerPort());
   out.println("<br>The content-type:"+response.getContentType());
   out.println("<br>The Session Id:"+session.getId());
   out.println("<br>The Current Servlet:"+page);
   out.println("<br>The Context Parameter value:"
                  +application.getInitParameter("uname"));

   if(exception==null){
    out.println("<br>exception is null because there is no exception code");
   }
   else{
    out.println("<br>The exception value :"+exception);
   }


   String formParamValue=config.getInitParameter("mail");

   if(formParamValue==null || mail==null){

   if(formParamValue==null){
    out.println("<br>If you want to access form parameter, "
          + "you send param name in the form of query String");
   }
   else{
    out.println("<br>The form parameter value:"+formParamValue);
   }

   if(mail==null){
```

```
   out.println("<br>If want to get the init parameter values"
         + " then you can send a request using url-pattern");
  }
  else{
   out.println("<br>The init parameter value:"+mail);
  }

  }
  else{
   out.println("<br>The form param value:"+formParamValue);
   out.println("<br>The init param value:"+mail);
  }
 }
 catch(Exception e){
  e.printStackTrace();
 }

 return EVAL_BODY_INCLUDE;
}

}
```

myTld.tld

```
<?xml version="1.0" encoding="UTF-8"?>
<taglib version="2.1" xmlns="http://java.sun.com/xml/ns/javaee">
 <tlib-version>1.2</tlib-version>
 <uri>www.jobs4times.com</uri>
 <tag>
  <name>jspImplicitObjects</name>
  <tag-class>com.tag.JspImplicitObj</tag-class>
 </tag>
</taglib>
```

web.xml

```
<web-app>
<jsp-config>
 <taglib>
  <taglib-uri>http://jobs4times.com/tags</taglib-uri>
  <taglib-location>/WEB-INF/myTld.tld</taglib-location>
 </taglib>
</jsp-config>

<context-param>
 <param-name>uname</param-name>
 <param-value>SaiCharan</param-value>
</context-param>
```

```
<servlet>
 <servlet-name>implicitObj</servlet-name>
 <jsp-file>/myJsp.jsp</jsp-file>

 <init-param>
  <param-name>mail</param-name>
  <param-value>jobs4times@gmail.com</param-value>
 </init-param>
</servlet>

<servlet-mapping>
 <servlet-name>implicitObj</servlet-name>
 <url-pattern>/test</url-pattern>
</servlet-mapping>

</web-app>
```

http://localhost:8080/jstl/test

output :

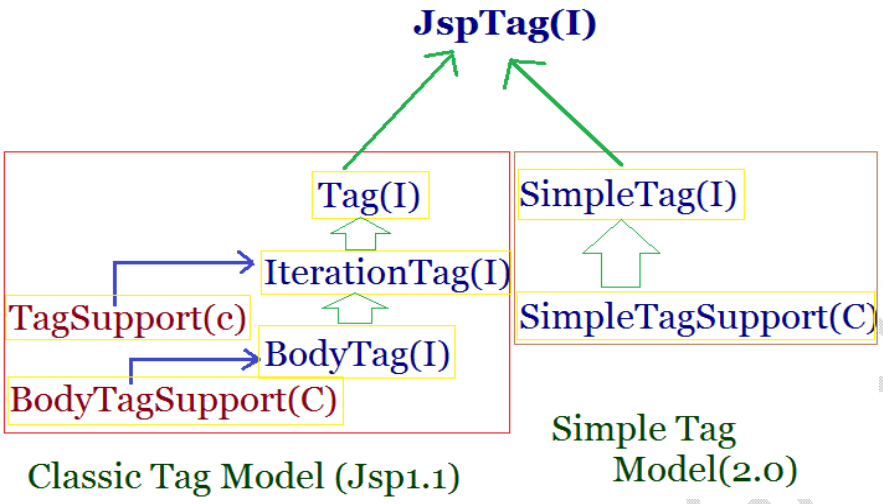
```
JSP Implicit Objects from Custom Tags

The Server details:ashok 8080
The content-type:text/html
The Session Id:089ea14fea88884de1b6f490bafe
The Current Servlet:org.apache.jsp.test_jsp@63c3024d
The Context Parameter value:SaiCharan
exception is null because there is no exception code
The form param value:jobs4times@gmail.com
The init param value:Ask
```

SimpleTag Model : (jsp 2.0v)

Implementing Custom tags by using Classic Tag Model (i.e., Tag, IterationTag, BodyTag, TagSupport, BodyTagSupport) is very complex because each tag has its own life cycle and different possible return types for every method to resolve this complexity Sun people introduced SimpleTagModel in jsp 2.0 version

In SimpleTag model we can built custom tags by using SimpleTag interface and its implementation class SimpleTagSupport.



SimpleTag interface :

It is child interface of JspTag and defines the following 5 methods

1. public void setJspContext(JspContext context)
   by using setJspContext() method we can make JspContext object available to Tag Handler class by using this object we can get all jsp implicit objects and attributes in our tag handler classes.
2. public void setParent(JspTag parent)
   This method will executed iff the tag has another tag (nested tag)
3. public void setJspBody(JspFragment jspBody)
4. public void doTag() throws JspException, IOException
5. public JspTag getParent()

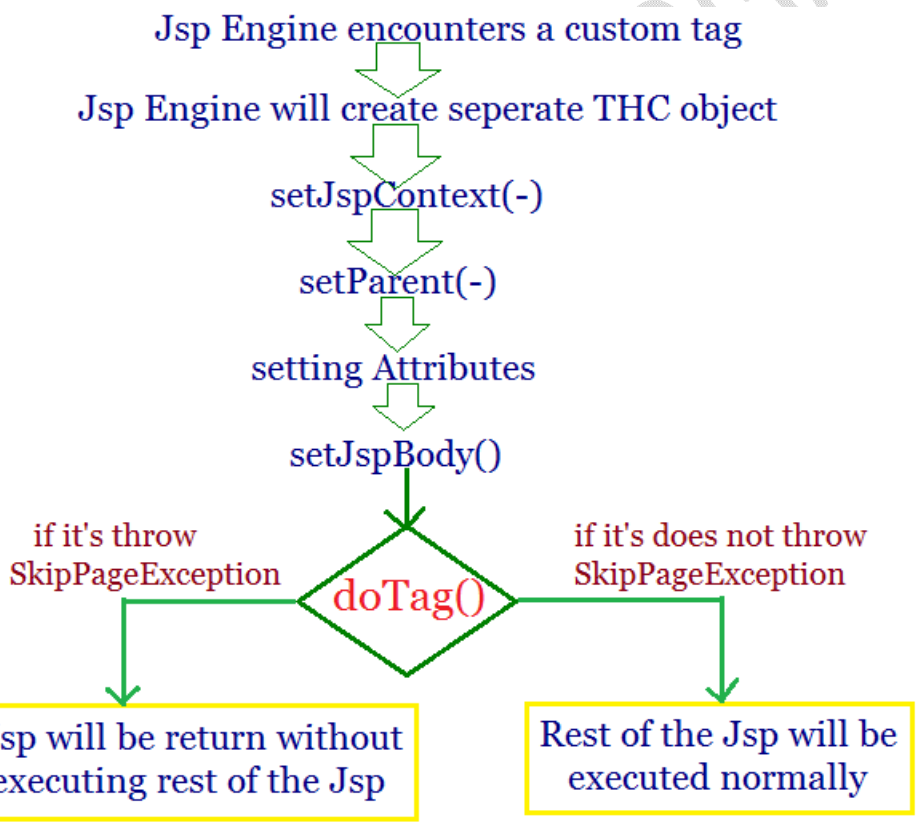
These methods are life cycle methods Jsp engine called automatically for every custom tag invocation , but not getParent() method.

Life Cycle of SimpleTag Model :

1. When ever the jsp engine encounters a custom tag in jsp it will identify corresponding tag handler class by using taglib directive and TLD file.
2. It creates a new instance of TH class by executing public no-argument constructor.
3. SimpleTag handler objects are never reused by the WC for each tag invocation in new tag handler class created.
4. Web-container executes setJspContext() to make JspContext object available to tag handler class by using this JspContext object tag handler class get all jsp implicit objects and attributes.
5. Jsp engine will call setParent() to make parent tag object available to tag handler class , this method is useful in nested tags(setParent() is only called if the element is nested in another tag invocation)
6. If a custom tag invoked with attribute then for each attribute jsp engine will call corresponding setter methods to make attribute value available to tag handler class (for every attribute one instance variable and corresponding setter method should required)

7. If a custom tag invoke with body then setJspBody() will be executed by taking JspFragment object as argument, JspFragment object represents tag body in SimpleTagModel tag body should not contains scripting elements i.e., the allowed values for the body-content tag is empty, tagdependent, scriptless from Jsp2.0v onwards body-content tag is optional, in SimpleTagModel default value is scriptless.
8. If a custom tag is empty then setJspBody() won't be called , there is no default value in body-content in SimpleTagModel.
9. Finally jsp engine will invoke doTag() to required functionality this method is equivalent to doStartTag(), doEndTag(), doAfterBody()
10. Once doTag() completes tag handler class object will be destroy by the WC.

Flow chart for SimpleTagModel :



Internal implementation of SimpleTagSupport class :

SimpleTagSupport.java

```
package com.tag;

import java.io.IOException;
import javax.servlet.jsp.JspContext;
import javax.servlet.jsp.JspException;
```

```
import javax.servlet.jsp.tagext.JspFragment;
import javax.servlet.jsp.tagext.JspTag;
import javax.servlet.jsp.tagext.SimpleTag;

public class SimpleTagSupport implements SimpleTag {
 private static final long serialVersionUID = 1L;

  @Override
 public void setJspContext(JspContext pc) {
  throw new UnsupportedOperationException("Not supported yet.");
//To change body of generated methods, choose Tools | Templates.
 }

 @Override
 public void setParent(JspTag parent) {
  throw new UnsupportedOperationException("Not supported yet.");
//To change body of generated methods, choose Tools | Templates.
 }

 @Override
 public JspTag getParent() {
  throw new UnsupportedOperationException("Not supported yet.");
//To change body of generated methods, choose Tools | Templates.
 }



 @Override
 public void setJspBody(JspFragment jspBody) {
  throw new UnsupportedOperationException("Not supported yet.");
//To change body of generated methods, choose Tools | Templates.
 }

 @Override
public void doTag() throws JspException, IOException {
  throw new UnsupportedOperationException("Not supported yet.");
//To change body of generated methods, choose Tools | Templates.
}

}
```

(OR)

Sample code :

```
import java.io.IOException;
import javax.servlet.jsp.JspContext;
import javax.servlet.jsp.JspException;
import javax.servlet.jsp.tagext.JspFragment;
import javax.servlet.jsp.tagext.JspTag;
```

```
import javax.servlet.jsp.tagext.SimpleTag;

public class SimpleTagSupport implements SimpleTag {
 private static final long serialVersionUID = 1L;

 private JspContext jspContext;
 private JspFragment jspBody;
 private JspTag parent;

  @Override
 public void setJspContext(JspContext jspContext) {
  this.jspContext=jspContext;
 }

protected JspContext getJspContext(){
 return jspContext;
}

 @Override
 public void setParent(JspTag parent) {
  this.parent=parent;
 }

 @Override
 public JspTag getParent() {
  return parent;
 }

 @Override
 public void setJspBody(JspFragment jspBody) {
  this.jspBody=jspBody;
 }

 public JspFragment getJspBody() {
  return jspBody;
 }

 @Override
public void doTag() throws JspException, IOException {
}

public static final JspTag findAncesterWithClass(JspTag tag,Class c){
 //return null;
}

}
```

Write a demo program to SimpleTagSupport class :

test.jsp

```
<%@taglib uri="http://jobs4times.com/tags" prefix="mine" %>

This is before Custom Tag <br>
<mine:myTag>
 This is the body of the tag <br>
</mine:myTag>
 This is rest of the Jsp page
```

Tag Handler Class

SimpleTagSupportDemo.java

```
package com.tag;

import java.io.IOException;
import javax.servlet.jsp.JspException;
import javax.servlet.jsp.JspWriter;
import javax.servlet.jsp.SkipPageException;
import javax.servlet.jsp.tagext.SimpleTagSupport;

public class SimpleTagSupportDemo extends SimpleTagSupport {
 private static final long serialVersionUID = 1L;


 @Override
 public void doTag() throws JspException, IOException {
 JspWriter out=getJspContext().getOut();
 out.println("This is SimpleTagSupport class<br>");
 //throw new SkipPageException();
}

}
```

myTld.tld

```
<?xml version="1.0" encoding="UTF-8"?>
<taglib version="2.1" xmlns="http://java.sun.com/xml/ns/javaee">
  <tlib-version>1.2</tlib-version>
  <uri>www.jobs4times.com</uri>
  <tag>
    <name>myTag</name>
    <tag-class>com.tag.SimpleTagSupportDemo</tag-class>
    <body-content>scriptless</body-content>
  </tag>
</taglib>
```

web.xml

```
<web-app>
<jsp-config>
 <taglib>
  <taglib-uri>http://jobs4times.com/tags</taglib-uri>
  <taglib-location>/WEB-INF/myTld.tld</taglib-location>
 </taglib>
</jsp-config>
</web-app>
```

Note : if doTag() is not throwing a SkipPageException then following is the output

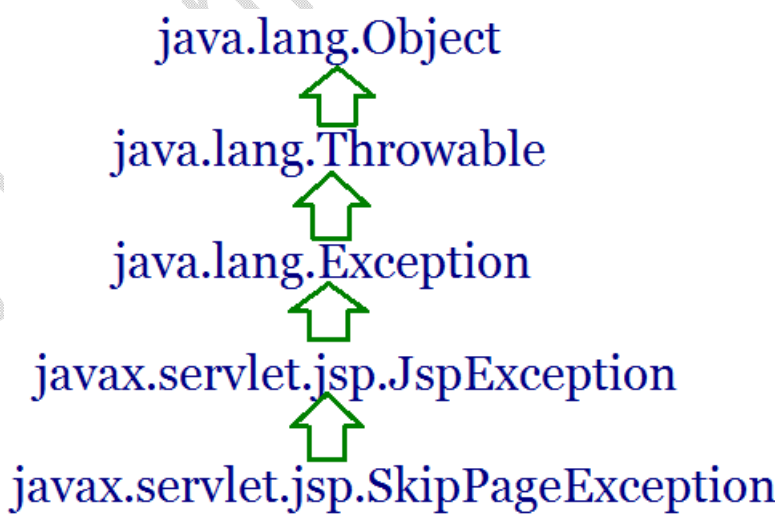
```
This is before Custom Tag
This is SimpleTagSupport class
This is rest of the Jsp page
```

If doTag() method throws SkipPageException()

```
This is before Custom Tag
This is SimpleTagSupport class
```

Bydefault the Tag Body should not be include of JSP if we want to include then we have to arrange some extra arrangement.(i.e., getterMethods, ...)

SkipPageException hierarchy :



Accessing Tag Body in SimpleTagModel

We can access tag body simple tag handler by using getJspBody() this method returns JspFragment object

public JspFragment getJspBody()

JspFragment is an abstract class at translation time, the container generate the implementation of the JspFragment abstract class capable of executing the defined fragment(tag body)

1. public JspContext getJspContext()
2. public void invoke(java.io.Writer out)

- It causes evaluation of tag body and return to the supplied writer
- If we pass null argument to invoke() then it will write directly to the jsp page

index.jsp

```
<%@taglib uri="http://jobs4times.com/tags" prefix="mine" %>

<title>Tag body in Simple tag model</title>
This is before Custom Tag <br>
<mine:myTag>
 This is the body of the tag <br>
</mine:myTag>
 This is rest of the Jsp page
```

SimpleTagSupportBodyDemo.java

```
package com.tag;

import java.io.IOException;
import javax.servlet.jsp.JspException;
import javax.servlet.jsp.JspWriter;
import javax.servlet.jsp.tagext.JspFragment;
import javax.servlet.jsp.tagext.SimpleTagSupport;

public class SimpleTagSupportBodyDemo extends SimpleTagSupport {
 private static final long serialVersionUID = 1L;

 @Override
 public void doTag() throws JspException, IOException {
 JspWriter out=getJspContext().getOut();
 out.println("This is SimpleTagSupport class<br>");
 JspFragment body=getJspBody();
 body.invoke(null);
}

}
```

myTld.tld

```
<?xml version="1.0" encoding="UTF-8"?>
<taglib version="2.1" xmlns="http://java.sun.com/xml/ns/javaee">
 <tlib-version>1.2</tlib-version>
 <uri>www.jobs4times.com</uri>
 <tag>
  <name>myTag</name>
  <tag-class>com.tag.SimpleTagSupportBodyDemo</tag-class>
  <body-content>scriptless</body-content>
 </tag>
</taglib>
```

web.xml

same as previous

http://localhost:8080/jstl/index.jsp

output :

```
This is before Custom Tag
This is SimpleTagSupport class
This is the body of the tag
This is rest of the Jsp page
```

Manipulating tag body in SimpleTagModel :

index.jsp

```
<%@taglib uri="http://jobs4times.com/tags" prefix="mine" %>

<title>Manipulating Tag body in Simple tag model</title>
This is before Custom Tag <br>
<mine:myTag>
 This is body of the tag <br>
</mine:myTag>
 This is rest of the Jsp page
```

ManipulateSimpleTagSupportBodyDemo.java

```
package com.tag;

import java.io.IOException;
import java.io.StringWriter;
import javax.servlet.jsp.JspException;
import javax.servlet.jsp.JspWriter;
import javax.servlet.jsp.tagext.JspFragment;
import javax.servlet.jsp.tagext.SimpleTagSupport;
```

```
public class ManipulateSimpleTagSupportBodyDemo extends SimpleTagSupport{
 private static final long serialVersionUID = 1L;

 @Override
 public void doTag() throws JspException, IOException {
 JspWriter out=getJspContext().getOut();
 out.println("This is SimpleTagSupport class<br>");
 JspFragment body=getJspBody();
 StringWriter stringWriter=new StringWriter();
 body.invoke(stringWriter);
 //it returns tag body into StringWriter object
 out.println(stringWriter.toString().toUpperCase());
}
}
```

myTld.tld

```
<?xml version="1.0" encoding="UTF-8"?>
<taglib version="2.1" xmlns="http://java.sun.com/xml/ns/javaee">
 <tlib-version>1.2</tlib-version>
 <uri>www.jobs4times.com</uri>
 <tag>
  <name>myTag</name>
  <tag-class>com.tag.ManipulateSimpleTagSupportBodyDemo</tag-class>
  <body-content>scriptless</body-content>
 </tag>
</taglib>
```

web.xml

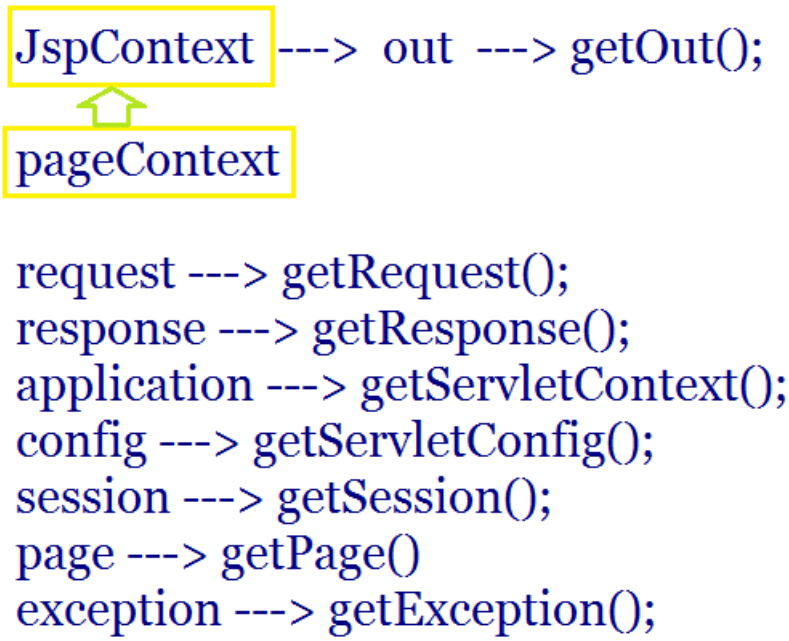
```
Same as previous
```

output :

```
This is before Custom Tag
This is SimpleTagSupport class
THIS IS BODY OF THE TAG
This is rest of the Jsp page
```

Jsp implicit objects and attributes in SimpleTag Model :

JspContext class contains getOut() but not remaining jsp implicit object methods



In Tag Handler class :

```
package com.tag;

import java.io.IOException;
import javax.servlet.ServletConfig;
import javax.servlet.ServletContext;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;
import javax.servlet.http.HttpSession;
import javax.servlet.jsp.JspException;
import javax.servlet.jsp.JspWriter;
import javax.servlet.jsp.PageContext;
import javax.servlet.jsp.tagext.SimpleTagSupport;

public class SimpleTagSupportImplicitDemo extends SimpleTagSupport {
 private static final long serialVersionUID = 1L;

 @Override
 public void doTag() throws JspException, IOException {
  PageContext pageContext=(PageContext)getJspContext();
  HttpSession session=pageContext.getSession();
  JspWriter out=pageContext.getOut();
   out.println("The session id :"+session.getId()+"<br>");
  ServletConfig config=pageContext.getServletConfig();
```

```
 ServletContext application=pageContext.getServletContext();
 ServletRequest request=pageContext.getRequest();
 ServletResponse response=pageContext.getResponse();
 Object page=pageContext.getPage();
 Throwable excetion=pageContext.getException();
}

}
```

index.jsp

```
<%@taglib uri="http://jobs4times.com/tags" prefix="mine" %>
<%@page isELIgnored="false" %>

<title>Setting attributes in Simple tag handler </title>
This is before Tag invocation<br>
<mine:myTag>
 ${movie} <br>
</mine:myTag>
 This is after tag invocation
```

AttributeSimpleTagDemo.java

```
package com.tag;

import java.io.IOException;
import javax.servlet.ServletConfig;
import javax.servlet.ServletContext;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;
import javax.servlet.http.HttpSession;
import javax.servlet.jsp.JspException;
import javax.servlet.jsp.JspWriter;
import javax.servlet.jsp.PageContext;
import javax.servlet.jsp.tagext.SimpleTagSupport;

public class AttributeSimpleTagDemo extends SimpleTagSupport {
 private static final long serialVersionUID = 1L;

 String movies[]={"movieA","MovieB","MovieC"};
 @Override
 public void doTag() throws JspException, IOException {
  System.out.println("Welcome to Attribute Mgt");
  JspWriter out=getJspContext().getOut();
  for(int i=0;i<movies.length;i++){
   getJspContext().setAttribute("movie", movies[i]);
```

```
  getJspBody().invoke(null);
 }
}

}
```

Each loop of the tag handler resets the "movie" attribute value and callsgetJspBody().invoke(-) again.

myTld.tld

```
<?xml version="1.0" encoding="UTF-8"?>
<taglib version="2.1" xmlns="http://java.sun.com/xml/ns/javaee">
 <tlib-version>1.2</tlib-version>
 <uri>www.jobs4times.com</uri>
 <tag>
  <name>myTag</name>
  <tag-class>com.tag.AttributeSimpleTagDemo</tag-class>
  <body-content>scriptless</body-content>
 </tag>
</taglib>
```

web.xml

```
Same as previous
```

output :

```
This is before Tag invocation
movieA
MovieB
MovieC
This is after tag invocation
```

What happens when the tag is invoked from on included page ?

index.jsp

```
<%@taglib uri="http://jobs4times.com/tags" prefix="mine" %>

<title>includes test Jsp </title>
This is before Jsp inclusion <br>
 <jsp:include page="test.jsp"/> <br>
This is after Jsp inclusion
```

test.jsp

```
<%@taglib uri="http://jobs4times.com/tags" prefix="mine" %>
```

```
<title>Included file</title>
This is before Custom Tag <br>
<mine:myTag>
 This is the body of the tag <br>
</mine:myTag>
 This is rest of the Jsp page
```

SimpleTagDemo.java

```
package com.tag;

import java.io.IOException;
import javax.servlet.jsp.JspException;
import javax.servlet.jsp.JspWriter;
import javax.servlet.jsp.SkipPageException;
import javax.servlet.jsp.tagext.SimpleTagSupport;

public class SimpleTagDemo extends SimpleTagSupport {
 private static final long serialVersionUID = 1L;

 @Override
 public void doTag() throws JspException, IOException {
  JspWriter out=getJspContext().getOut();
   out.println("Hello, this is from tag handler<br>");
   //throw new SkipPageException();
  }
}
```

myTld.tld

```
<?xml version="1.0" encoding="UTF-8"?>
<taglib version="2.1" xmlns="http://java.sun.com/xml/ns/javaee">
  <tlib-version>1.2</tlib-version>
  <uri>www.jobs4times.com</uri>
  <tag>
    <name>myTag</name>
    <tag-class>com.tag.SimpleTagDemo</tag-class>
    <body-content>scriptless</body-content>
  </tag>
</taglib>
```

web.xml

```
Same as previous
```

output :

```
This is before Jsp inclusion
This is before Custom Tag
Hello, this is from tag handler
```

```
This is rest of the Jsp page
This is after Jsp inclusion
```

remove comment on above THC

output :

```
This is before Jsp inclusion
This is before Custom Tag
Hello, this is from tag handler

This is after Jsp inclusion
```

What is the difference between ClassicTagModel and SimpleTagModel with respect to tag body :

- In ClassicTagModel the tag body can contains scripting elements hence the allowed values for body-content types are empty, tagdependent, scriptless, jsp and default value is jsp.
- But in SimpleTagModel the tag body should not contains scripting elements hence the allowed values for body-content types are empty, tagdependent, scriptless and default value is scriptless.

Key differences between ClassictagModel and SimpleTagModel :

Property	ClassictagModel	SimpleTagModel
key interfaces	Tag(I) IterationTag(I) BodyTag(I)	SimpleTag(I)
supporting implementation classes	TagSupport(C) BodyTagSupport(C)	SimpleTagSupport(C)
key life cycle methods that we	doStartTag() doEndTag()	doTag()

have to implement	doAfterBody()	
how to write response to jsp output stream	pageContext(). getOut().println() we should enclose this statement by using try, catch	getPageContext(). getOut().println() it is not required enclose try, catch
how to access jsp implicit objects and attributes	by using PageContext pageContext.getOut();	by using JspContext
how to include tag body in the result	in the case of Tag, IterationTag interfaces doStartTag() should return EVAL_BODY_INCLUDE but in BodyTag interface doStartTag() should return EVAL_BODY_BUFFERED	getJspBody().invoke(null);
how to stop current jsp execution	doEndTag() should return SKIP_PAGE	in side doTag() we should throw SkipPageException

DynamicAttributes :

- In general tags can contain attributes for which we have to declare in the tld file by using attribute tag for these attributes we have to maintain one instance variable and corresponding setter methods in tag handler class such type of attributes are calling static attributes.
- But we can use attributes even though tld file doesn't contain any <attribute> tag declaration such type of attributes are called dynamic attributes.
- Dynamic attributes concept applicable for both classic and simple tag models
- Dynamic attributes concept introduced jsp2.0v

To support Dynamic attribute concept we have to do the following things :

1. In the tld file we have to declare <dynamic-attributes> tag

```
<taglib version="2.1">
 <tlib-version>1.2</tlib-version>
 <uri>www.jobs4times.com</uri>
 <tag>
  .....
  <dynamic-attributes>true</dynamic-attributes>
  .....
 </tag>
</taglib>
```

2. The corresponding tag handler class should implements DynamicAttributes interface, this interface introduced jsp2.0v and contains only one method.

```
public void setDynamicAttribute(String nameSpace,  String name,Object value)
                                                        throws JspException{}
```

For every DynamicAttribute this method will be executed

test.jsp

```
<%@taglib uri="http://jobs4times.com/tags" prefix="mine" %>

This is before Custom Tag <br>
<mine:myTag name="Nandu" wife="Renu" brother="Tinku" habbits="sleep,lunch"/>
 This is rest of the Jsp page
```

DynamicAttributesDemo.java

```
package com.tag;

import java.io.IOException;
import java.util.HashMap;
import javax.servlet.jsp.JspException;
import javax.servlet.jsp.JspWriter;
import javax.servlet.jsp.tagext.DynamicAttributes;
import javax.servlet.jsp.tagext.SimpleTagSupport;

public class DynamicAttributesDemo extends SimpleTagSupport
                    implements DynamicAttributes{
 private static final long serialVersionUID = 1L;

HashMap h=new HashMap();

 @Override
 public void setDynamicAttribute(String nameSpace,
       String name,Object value) throws JspException{
 h.put(name, value);
}
 @Override
 public void doTag() throws JspException, IOException {
  JspWriter out=getJspContext().getOut();
   out.println(h +"<br>");
  }
}
```

myTld.tld

```
<?xml version="1.0" encoding="UTF-8"?>
<taglib version="2.1" xmlns="http://java.sun.com/xml/ns/javaee">
 <tlib-version>1.2</tlib-version>
 <uri>www.jobs4times.com</uri>
 <tag>
   <name>myTag</name>
   <tag-class>com.tag.DynamicAttributesDemo</tag-class>
   <body-content>scriptless</body-content>
   <dynamic-attributes>true</dynamic-attributes>
```

```
 </tag>
</taglib>
```

web.xml

```
Same as previous
```

output :

```
This is before Custom Tag
{wife=Renu, name=Nandu, brother=Tinku, habbits=sleep,lunch}
This is rest of the Jsp page
```

Combination of static & dynamic attributes :

index.jsp

```
<%@taglib uri="http://jobs4times.com/tags" prefix="mine" %>

This is before Custom Tag <br>
<mine:myTag num="2" min="5" max="10" pow="3"/>
 This is rest of the Jsp page
```

DynamicAttributesDemo.java

```
package com.tag;

import java.io.IOException;
import javax.servlet.jsp.JspException;
import javax.servlet.jsp.JspWriter;
import javax.servlet.jsp.tagext.DynamicAttributes;
import javax.servlet.jsp.tagext.SimpleTagSupport;

public class DynamicAttributesDemo extends SimpleTagSupport
                    implements DynamicAttributes{
 private static final long serialVersionUID = 1L;

String output=" ";
private int num;

public void setNum(int num){
 this.num=num;
 output=output+" The Given No is : "+num;
}

 @Override
public void setDynamicAttribute(String nameSpace,
        String name,Object value) throws JspException{
 int n=Integer.parseInt((String) value);
```

```
if(name=="min"){
 output=output+"<br>minimum value of "+num+","+n+" is : "+Math.min(num, n);
}
else if(name=="max"){
 output=output+"<br>maximum value of "+num+","+n+" is : "+Math.max(num, n);
}
else if(name=="pow"){
 output=output+"<br>power value of "+num+","+n+" is "+Math.pow(num, n);
}

}

 @Override
 public void doTag() throws JspException, IOException {
  JspWriter out=getJspContext().getOut();
   out.println(output +"<br>");
  }
}
```

myTld.tld

```
<?xml version="1.0" encoding="UTF-8"?>
<taglib version="2.1" xmlns="http://java.sun.com/xml/ns/javaee">
 <tlib-version>1.2</tlib-version>
 <uri>www.jobs4times.com</uri>
 <tag>
  <name>myTag</name>
  <tag-class>com.tag.DynamicAttributesDemo</tag-class>
  <body-content>scriptless</body-content>
  <dynamic-attributes>true</dynamic-attributes>
  <attribute>
   <name>num</name>
   <required>true</required>
   <rtexprvalue>true</rtexprvalue>
  </attribute>
 </tag>
</taglib>
```

web.xml

```
Same as previous
```

output :

```
This is before Custom Tag
The Given No is : 2
minimum value of 2,5 is : 2
maximum value of 2,10 is : 10
power value of 2,3 is 8.0
This is rest of the Jsp page
```

TagFiles (jsp2.0v) :

objectives:

1. Describes the semantics of tag file model
2. Describes application structure of tag file
3. Write a tag file and explain the constraints on jsp content in tag body
4. TagFiles concept introduced in Jsp2.0v
5. Tagfile is a jsp page or jsp document designed to be used as a custom action
6. The main advantage of tag files when compared with classic and simple tag model is we can built custom tag very easily and we are not required to write tld file and the corresponding tag handler classes
7. The main limitation of tag files is won't suggestible will doing must processing

Building & Using a Simple tag file :

1. Write a Jsp page or Jsp document and save it with .tag extension.
2. Place this tag file inside /WEB-INF/tags folder
3. Write a taglib directive in Jsp with "tagdir" attribute

index.jsp

```
<%@taglib prefix="mine" tagdir="/WEB-INF/tags"%>
<%@taglib prefix="wish" tagdir="/WEB-INF/tags/myTags"%>

<html>
<title>Tag Files Demo</title>
<body>
<h1>THis is Tag File Demo</h1>
<mine:myTag />
<wish:welcome/>
</body>
</html>
```

welcome.tag

```
<h1>This is Welcome tag file</h1>
```

myTag.tag

```
<h1>This is MyTag File</h1>
```

output:

```
THis is Tag File Demo
This is MyTag File
This is Welcome tag file
```

Note : Internally tag files will be converted into simple Tag Handlers , which are generated by JSP Engine, and it is available in work folder.

Declare tag files with attributes :

We can declare attributes for Tag files by using attribute directive.

```
<%@attribute name="title" required="true" rtexprvalue="true"%>
```

attribute directive is not part of JSP, It's part of Tag files

test.jsp

```
<%@taglib prefix="mine" tagdir="/WEB-INF/tags"%>

<title>Tag Files Demo</title>
<h1>THis is Tag File Demo</h1>
<mine:myTag title="Ashok"/>
```

myTag.tag

```
<%@attribute name="title" required="true" rtexprvalue="true"%>

<h1>This is MyTag File</h1>
title : ${title}
```

output :

```
THis is Tag File Demo
This is MyTag File
title : Ashok
```

Declaring body-content for tag file :

Tag file invocation can contain body but we are not use scripting elements i.e., inside tag body doesn't contain scriptlets, expressions, declarations but it contains EL-expressions ans Jsp standard actions.

```
<%@tag body-content="tagdependent"%>
```

- tag directive is not part of JSp, it is part of Tag files.
- body-content allowed values are empty, tagdependent, scriptless.
- The default value is scriptless.

index.jsp

```
<%@taglib prefix="wish" tagdir="/WEB-INF/tags/myTags"%>


This is Tag Files with attributes and tag body <br>
<wish:welcome fontColor="#660099">
 This is Tag Body
</wish:welcome>
```

welcome.tag

```
<%@attribute name="fontColor" required="true"%>
```

```
<%@tag body-content="tagdependent"%>

<FONT color="${fontColor}">
 <jsp:doBody/>
</FONT>
```

output :

```
This is Tag Files with attributes and tag body
This is Tag Body //colour
```

With in the tag file, we can access tag body by using <jsp:doBody/>

Classic Tag Model	EVAL_BODY_INCLUDE, EVAL_BODY_BUFFERED
Simple Tag Model	getJspBody().invoke(null)
Tag Files	<jsp:doBody/>

- Either directly or indirectly inside WEB-INF/tags folder
- Either directly or indirectly inside META-INF/tags present in jar file WEB-INF/libfolder.

Note : If we deploy Tag file in some jar file compulsory we have to write TLD file , we have to plate it inside META-INF folder.

myTld.tld (in META-INF)

```
<taglib version="2.1">
 <tlib-version>1.2</tlib-version>
 <uri>www.jobs4times.com</uri>
 <tag-file>
  <name>myTag</name>
  <path>/META-INF/tags/myTag.tag</path>
 </tag-file>
</taglib>
```

Ex : test.jsp

```
<%@taglib prefix="wish" uri="jobs4times"%>

Tag Files deployed in the form of Jar files <br>
<wish:welcome />
```

welcome.tag

```
This is Welcome tag file
 deploying in some jar file
```

myTld.tld

```
<taglib version="2.1">
 <tlib-version>1.2</tlib-version>
 <uri>jobs4times</uri>
```

```
 <tag-file>
  <name>welcome</name>
  <path>/META-INF/tags/welcome.tag</path>
 </tag-file>
</taglib>
```

TagFiles with DynamicAttributes :

- TagFiles can also include DynamicAttributes the mechanism is basically same ClassicTagModel and SimpleTagModel but with TagFiles.
- The jsp engine provide the map object for you, you can inspect and iterate over the map of attribute key-value pair using for-each JSTL .

```
<%@tag body-content="scriptless" dynamic-attributes="${mapObj}"%>
```

The value of dynamic attributes is page scoped variable that holds map or Hashmap reference.

```
<c:forEach items="mapObj" var="x">
 ${x.key} and ${x.value}
</c:forEach>
<tag>
 <name>myTag</name>
 <tag-class>com.tag.SimpleTagDemo</tag-class>
 <body-content>scriptless</body-content>
</tag>
<mine:myTag>
 ${2*3}
</mine:myTag>
```

output :

```
6
```

If it works, which of the following Simple tag Model in tag handler life cycle methods will be executed?

1. void doTag()
2. void setParent()
3. void setJspContext()
4. JspTag getParent()
5. void setJspBody()

Ans : 1, 3, 5

- The setParent() is called only when tag is involved from with in anotther tag, since this tag was not nested setParent() was not executed.
- In the case of Classic Tag Model whether the tag contains nested or not in all cases setParent() will executed i.e., setParent() is part of lifecycle method.

Disabling scripting language in web.xml

- we can disable scripting language <scripting-invalid> tag
- If we are disabling EL that time jsp engine will consider EL syntax as template text.

```
<web-app>
 <jsp-config>
  <jsp-property-group>
   <url-pattern>*.jsp</url-pattern>
   <scripting-invalid>true</scripting-invalid>
  </jsp-property-group>
 </jsp-config>
</web-app>
```

Once the <scripting-invalidate> if we are not allowed to use scripting elements in jsp violation leads translation time error we will get.

Disabling expression language globally

We can disable EL for a perticular jsp by using isELIgnored attribute of page directive.

```
<%@page isELIgnored="true"%>
```

We can disable expression language at application level as follows.

```
<web-app>
 <jsp-config>
  <jsp-property-group>
   <url-pattern>*.jsp</url-pattern>
   <el-ignored>true</el-ignored>
  </jsp-property-group>
 </jsp-config>
</web-app>
```