

# Custom Tag Library



# 目錄

## 章節目錄

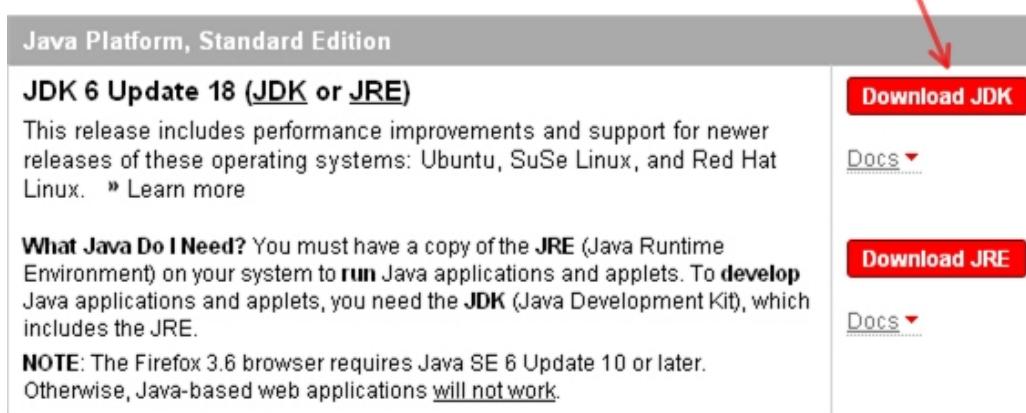
1. 開始 .....	1
開發與執行環境 .....	1
Custom Tag 介紹 .....	7
2. Tag Files .....	9
建立與使用 Tag Files .....	9
設定與使用 Attributes .....	11
包含 body 的 Tag Files .....	14
3. Simple Tag .....	17
建立與使用 Simple Tag .....	17
包含 body 的標籤 .....	21
標籤的 body 與 JspFragment .....	24
Simple Tag API .....	26
設定與使用標籤的 Attributes .....	28
設定資料給 body 使用 .....	30
巢狀標籤 .....	33
使用 SkipPageException .....	38
4. Classic Tag .....	40
Classic Tag API .....	40
Classic Tag 的生命週期 .....	44
包含 body 的 Classic Tag .....	47
巢狀標籤 .....	49



# 1. 開始

## 開發與執行環境

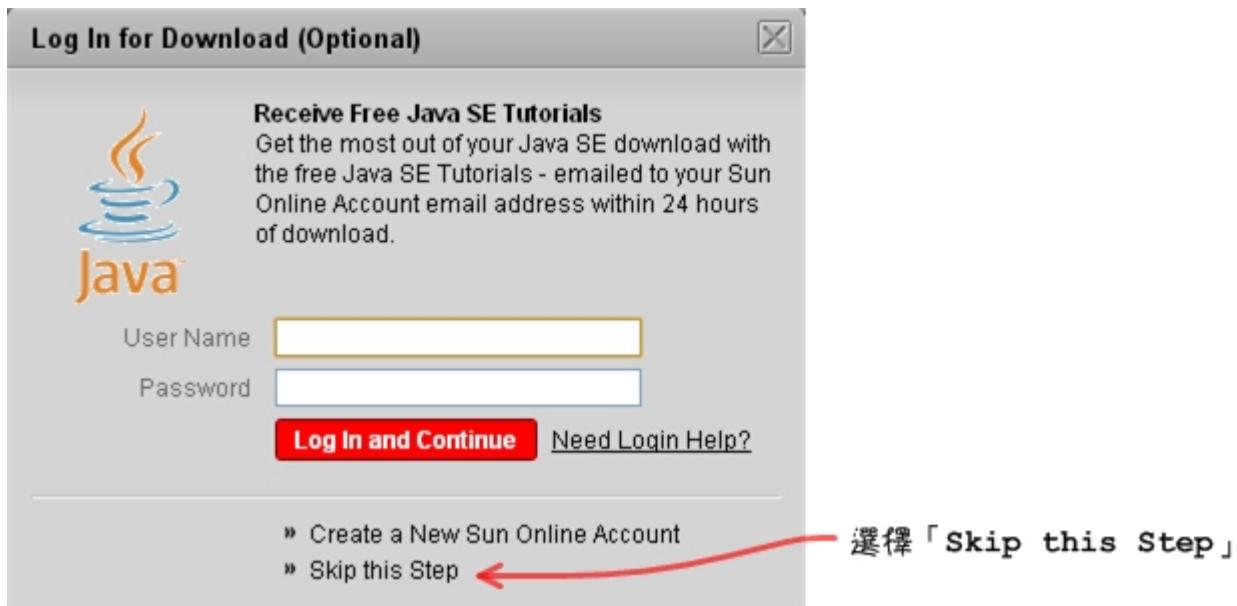
1. 到 <http://java.sun.com/javase/downloads> 下載 Java SE JDK。
2. 選擇「Download JDK」：



3. 選擇平台(Platform)後選擇「Download>>」：



4. 選擇「>>Skip this step」跳過登入畫面：



5. 儲存與執行下載的檔案，把 Java SE JDK 安裝在電腦中。  
 6. 到 <http://bits.netbeans.org/netbeans/6.5/community/latest/> 下載 NetBeans IDE。  
 7. 選擇語言和平台後，選擇 Java 的下載：

選擇「正體中文」

選擇平台

**NetBeans IDE 6.5 Build 200912090600 下載**

電子郵件位址(選擇性):

訂閱電子郵件:  每月  每周  
 NetBeans 可以透過電子郵件聯繫給你

語言: 正體中文 平台: Windows 2000/XP/Vista

備註: UML 套件存在於更新中心。

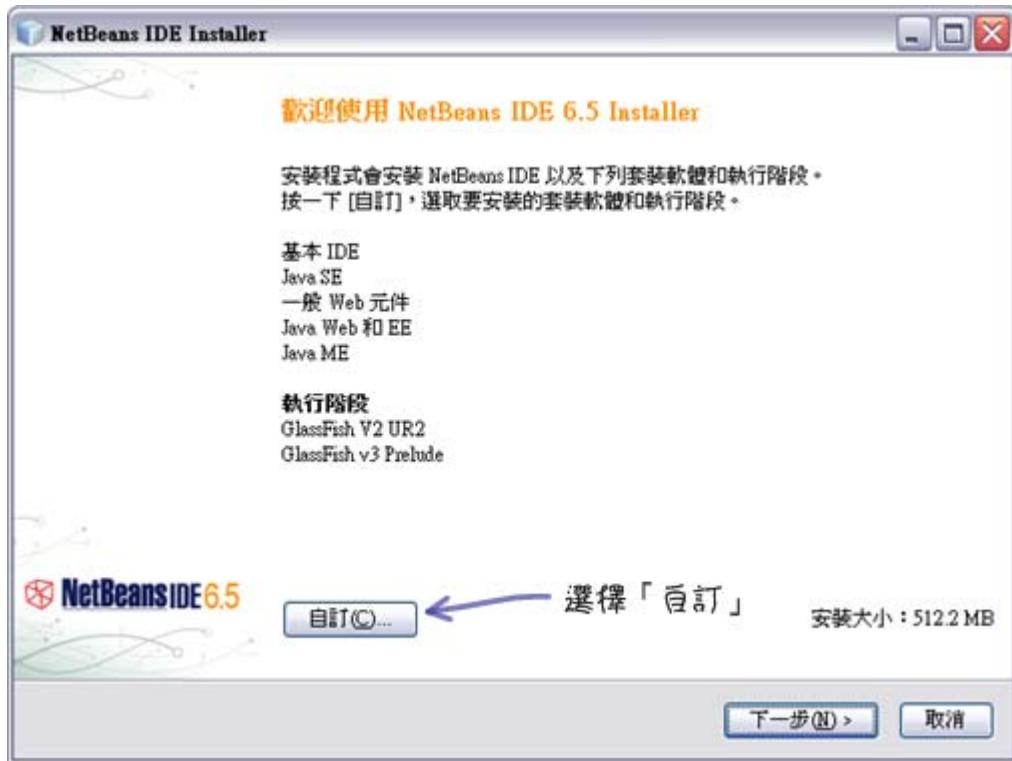
由社群貢獻的各國語言 NetBeans IDE 下載套件 <sup>1</sup>						
支援的技術 *	Java SE	Java	Ruby	C/C++	PHP	All
① Java SE	•	•				•
② Java Web 和 EE		•				•
③ Java ME		•				•
④ Ruby			•			•
⑤ C/C++				•		•
⑥ PHP					•	•
⑦ SOA						•
內建的伺服器						
⑧ GlassFish V2 UR2		•				•
⑨ GlassFish v3 Prelude		•	•			•
⑩ Apache Tomcat 6.0.18	•					•

下載 下載 下載 下載 下載 下載

免費, 35 MB 免費, 200 MB 免費, 57 MB 免費, 22 MB 免費, 24 MB 免費, 236 MB

選擇「下載」

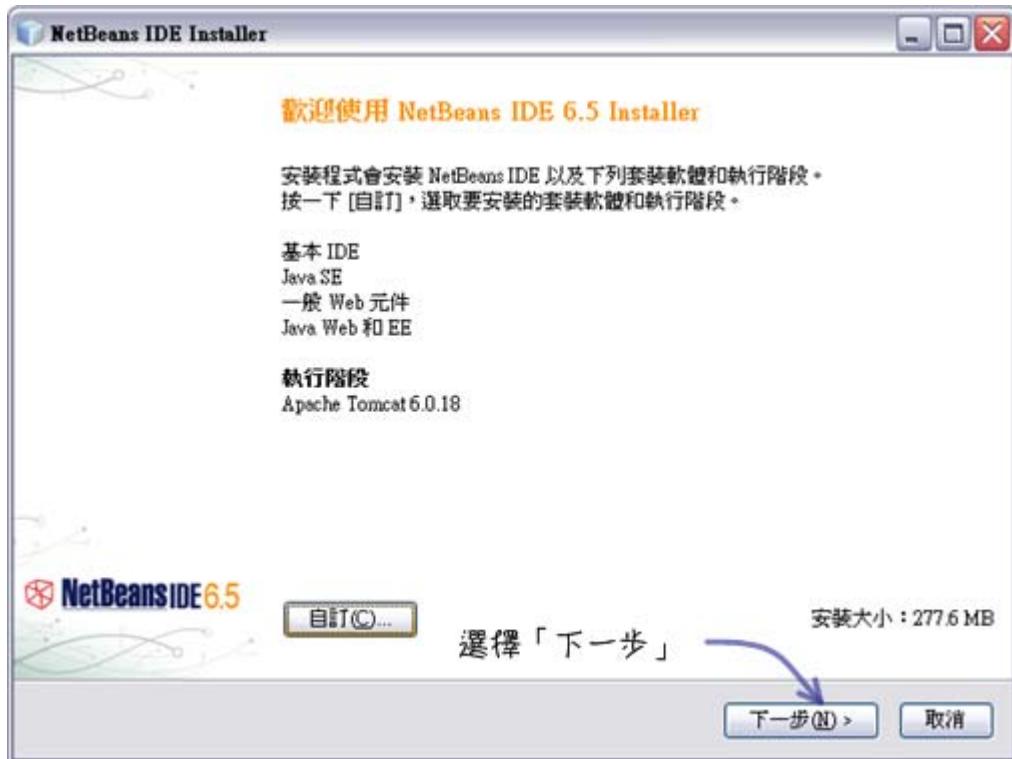
8. 儲存與執行下載後的檔案；在安裝畫面選擇「自定」：



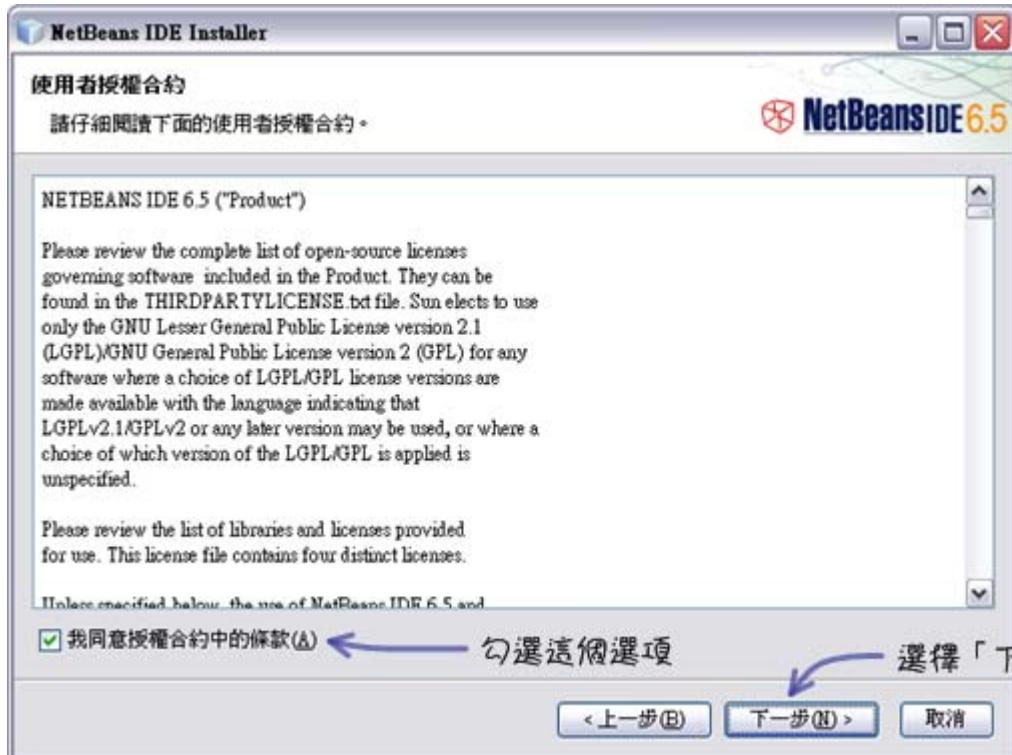
9. 選擇課程需要的選項：



10. 選擇下一步：



11. 選擇同意後選擇下一步：



12. 選擇下一步：



13. 選擇下一步：



14. 選擇安裝，開始進行安裝程序：

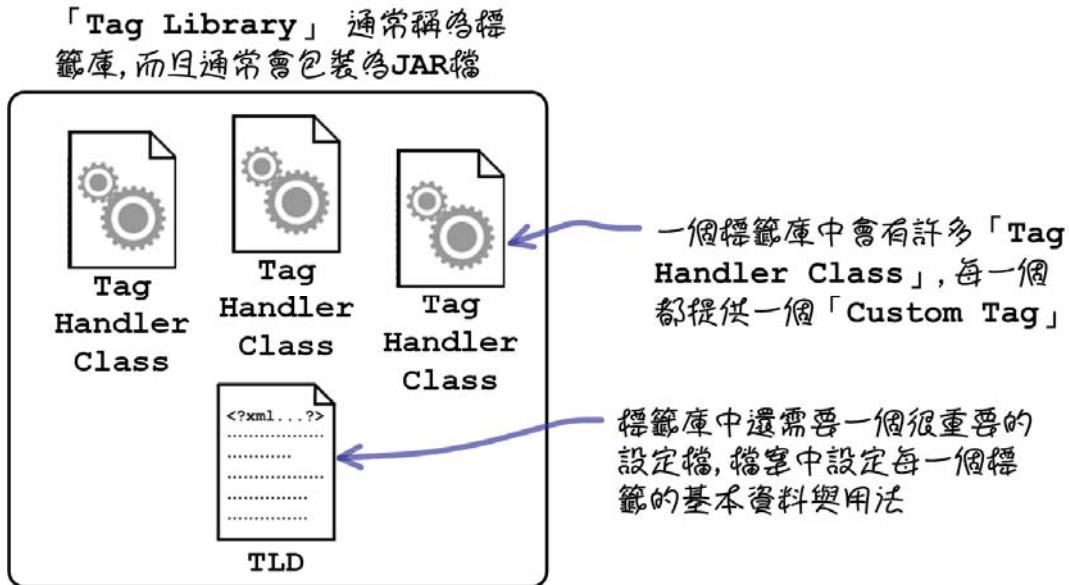


15. 取消選項後選擇完成：

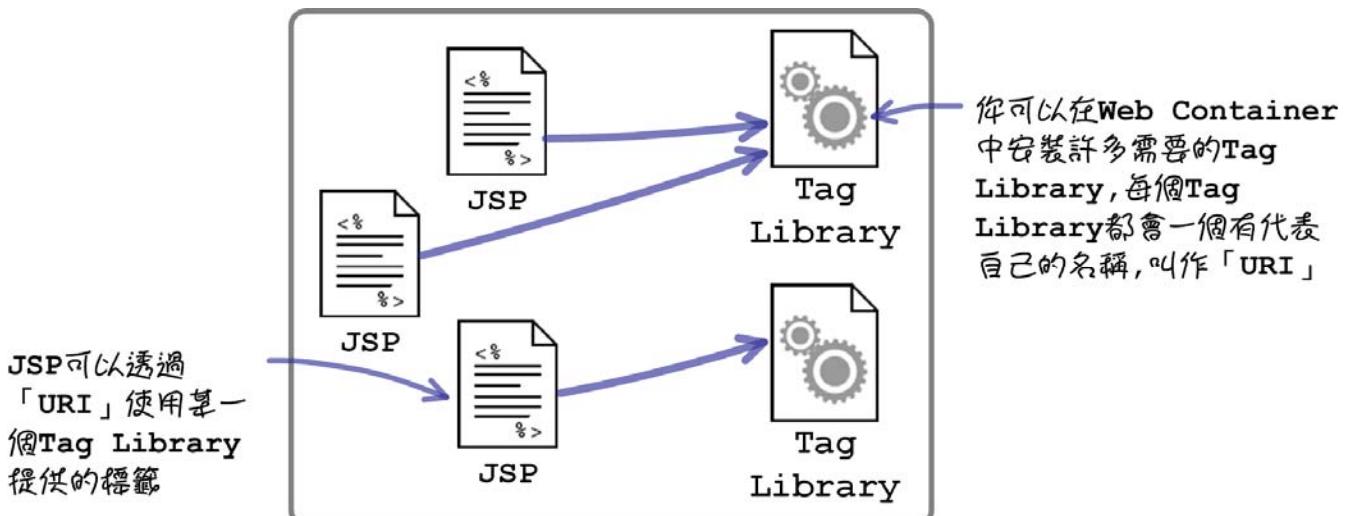


## Custom Tag 介紹

Java 動態網頁技術除了提供 Servlet、JSP 與 Filter 元件外，還可以讓你自己開發需要的標籤，來簡化 JSP 元件中的程式碼，這種技術稱為「custom tag library」：

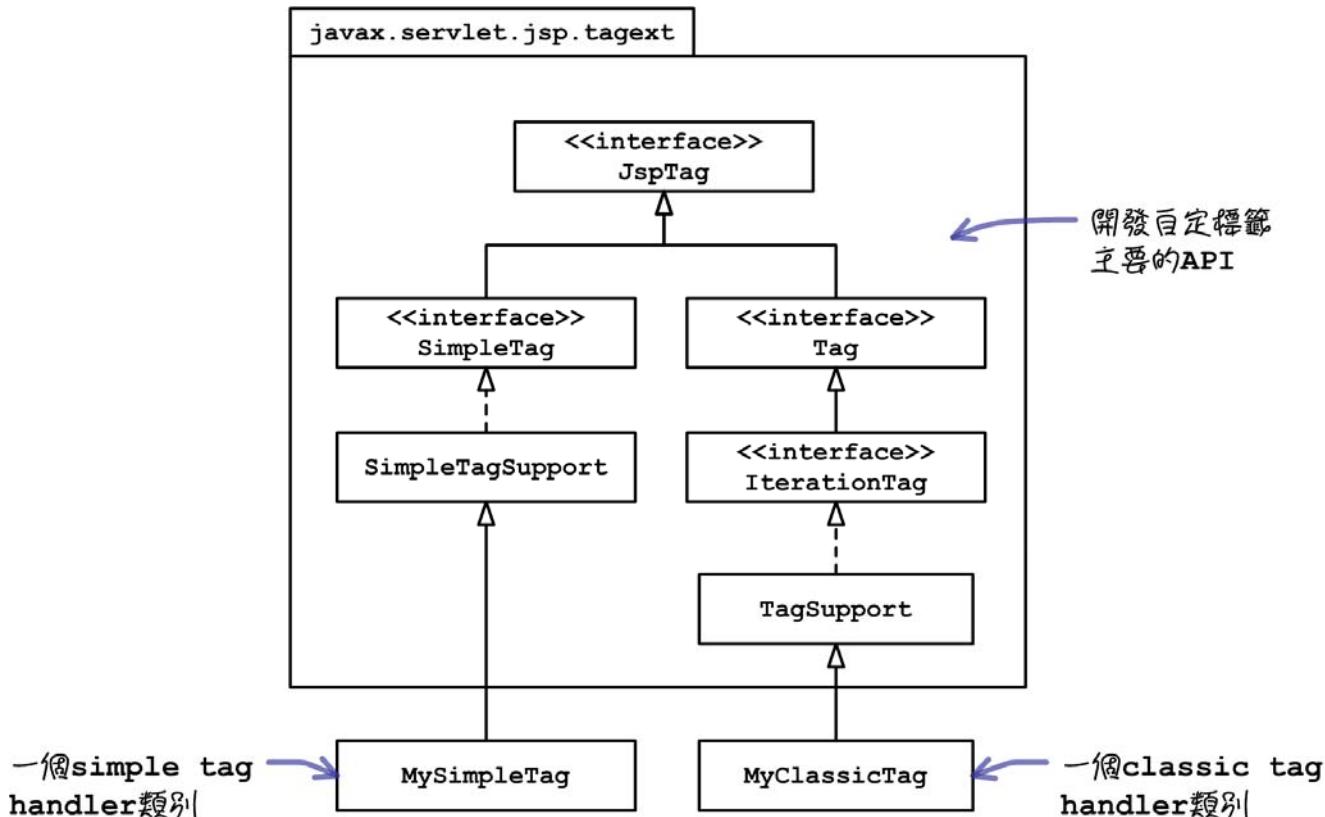


JSP 元件可以透過「URI」名稱使用 tag library 中提供的標籤：



Java 在「`javax.servlet.jsp.tagext`」套件中提供一組 API，讓你可以開發自己需要的標籤。從 JSP 2.0 開始，開發自定標籤有「classic」或「simple」兩種選擇。在 2.0 之前，只能使用「classic」的方式來開發自定標籤，這種開發的方式會比較複雜一些。在 2.0 時，Java 加入「simple」的開發方式，大幅度簡化開發標籤的工作。

下列是開發自定標籤主要的 API：



## 2. Tag Files

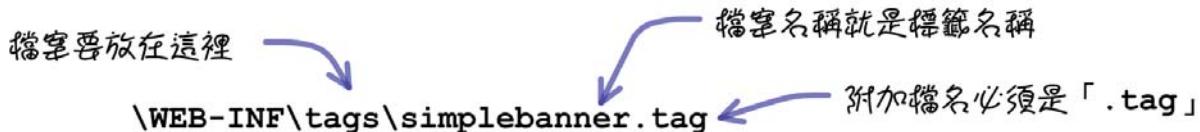
### 建立與使用 Tag Files

Java 提供一種非常簡易的方式，可以讓你自己建立一個標籤，這種作法稱為「Tag Files」。Tag Files 的用法其實和「jsp:include」與「<%@page include>」兩種標籤的應用類似，都是在 JSP 中引用一段內容。下列的步驟可以建立與使用一個簡單的 Tag File：

1. 在「WEB-INF」資料夾下建立一個存放 Tag Files 的資料夾，名稱為「tags」：



2. 在 Tag Files 資料夾下建立一個附加檔名為「tag」的檔案：



```

<table border="0" width="60%" bgcolor="#ccccff">
    <tr>
        <td align="left">My web application - Powered by Simon</td>
        <td align="right"></td>
    </tr>
</table>

```

Tag files 內容是一段用來顯示頁面標題 HTML 標籤

3. 在 JSP 中使用 Tag Files :

```

把prefix設定為「tf」          使用tagdir設定Tag
<%@ taglib prefix="tf" tagdir="/WEB-INF/tags" %>

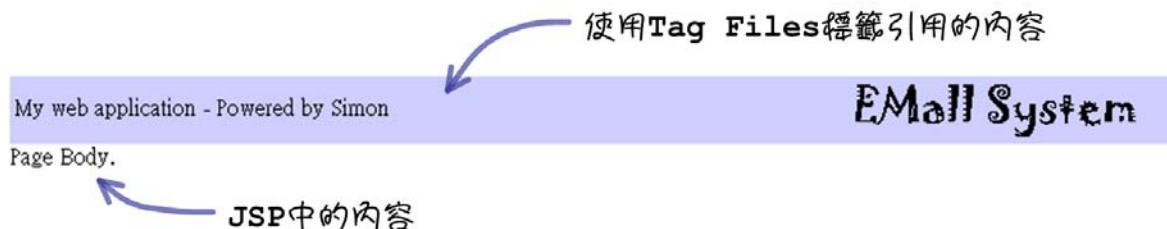
<html>
  <body>
    <tf:SimpleBanner />
    Page Body.
  </body>
</html>

```

在「taglib」中  
設定的prefix

Tag files的檔名「simplebanner」

4. 上列的 JSP 顯示的網頁會像這樣：



## 設定與使用 Attributes

在使用 Tag Files 時，也可以像搭配「jsp:include」與「jsp:param」一樣，傳遞資料給引用的檔案使用。Tag Files 的作法是使用「attribute」的設定。下列的步驟新增可以接收 attribute 資料的 tag file：

1. 你可以先參考設定 attribute 的語法：

```
<%@ attribute name="設定attribute名稱"
           required="設定true為必須, false為可有可無"
           rtxexprvalue="設定true為可以使用動態資料, false為固定的文字"
%>
```

2. 在 Tag Files 資料夾下建立一個檔名為「banner.tag」的檔案，這個 tag file 必須接收一個名稱為「webmaster」的 attribute，並且使用它來顯示：

```
設定attribute名稱 <--> 設定為必要 <--> 設定為可以使用動態資料
<%@ attribute name="webmaster" required="true" rtxexprvalue="true" %>

<table border="0" width="60%" bgcolor="#ccccff">
  <tr>
    <td align="left">My web application - Powered by ${webmaster}</td>
    <td align="right"></td>
  </tr>
</table>
```

在 Tag files 中使用 EL  
顯示 attribute 內容

3. 在 Tag Files 資料夾下建立一個檔名為「`footer.tag`」的檔案，這個 tag file 必須接收一個名稱為「`myemail`」的 attribute，並且使用它來顯示：

```

設定attribute名稱          設定為必要          設定為可以使  

                            ↓                         ↓                         ↓
<%@ attribute name="myemail" required="true" rtexprvalue="true" %>

<table border="0" width="60%" bgcolor="#ccccff">
  <tr>
    <td align="center">EMail: ${myemail}</td>
  </tr>
</table>

```

在 Tag files 中使用 EL  
顯示 attribute 內容

4. 在 JSP 中使用並傳送 attribute 給 Tag Files :

```

把prefix設定為「tf」          使用tagdir設定Tag  

                               ↓           Files資料夾的位置
                               ↓
<%@ taglib prefix="tf" tagdir="/WEB-INF/tags" %>
<html>
  <body>
    <tf:banner webmaster="Simon Johnson" />

```

Page Body.

```

    <tf:footer myemail="simon@macspeed.net" />
  </body>
</html>

```

設定webmaster  
設定myemail

5. 上列的 JSP 顯示的網頁會像這樣：



Attribute 中有一個「`rtpexprvalue`」的設定，用來規定設定 attribute 資料時，可不可以使用變數或運算式：

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

<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<c:set var="name">Simon Johnson</c:set> ← 使用JSTL設定一個名稱為「name」的資料

<html>
  <body>
    <tf:banner webmaster="${name}" /> ← 如果這個attribute的「rtpexprvalue」設定為「true」，你就可以這樣使用
    Page Body.
    <tf:footer myemail="simon@macspeed.net" />
  </body>
</html>
```

如果「`rtpexprvalue`」設定為「`false`」，設定 attribute 資料時就不可以使用變數或運算式：

「`rtpexprvalue`」設定為「`false`」...

```
<%@ attribute name="webmaster" required="true" rtpexprvalue="false" %>
...
<%@ taglib prefix="tf" tagdir="/WEB-INF/tags" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<c:set var="name">Simon Johnson</c:set>
<html>
  <body>
    <tf:banner webmaster="${name}" /> ← 這裡就不可以出現任何變數或運算式，否則會發生錯誤
    Page Body.
    <tf:footer myemail="simon@macspeed.net" />
  </body>
</html>
```

## 包含 body 的 Tag Files

使用 tag file 提供的 custom tag 時，可以依照標籤的需求，設定是否需要 body。你可以依照下列的語法在 tag file 中加入關於 body 的設定：

```
<%@ tag body-content="設定" %>
```

Tag file 關於 body 的設定有三種：

- ❑ empty：不允許使用 body
- ❑ tagdependent：把 body 中文字當成一般的文字，可以包含網頁文字，HTML 標籤
- ❑ scriptless：預設的設定；不允許 body 中使用 scriplets 與 expression；伺服器轉換 body 中的 standard tags、EL 與 JSTL

下列的表格表示 body 的設定 tagdependent 與 scriptless，body 中出現各種不同內容時，伺服器處理的規則：

設定與內容	tagdependent	scriptless
Scriptlet	忽略	不允許
Expression	忽略	不允許
Standard Tags	忽略	轉換
EL	直接顯示 EL	轉換
JSTL	忽略	轉換

以上列討論的「banner.tag」與「footer.tag」來說，它們應該加入不允許使用 body 的設定：

`<%@tag body-content="empty" %>` ← 在 tag file 中加入這個設定

```
<tf:banner webmaster="Simon Johnson">
    Banner body???
</tf:banner>
```

使用包含 body 的標籤寫法會造成錯誤

在使用 tag file 時，你可以使用 attribute 傳送資訊給它，如果要傳送的資訊內容很多的話，使用 attribute 就不太合適了。下列的步驟新增一個可以包含 body 的 tag file：

1. 在 Tag Files 資料夾下建立一個檔名為「content.tag」的檔案：

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

<%@tag body-content="tagdependent" %>
    ↑ 設定顯示內容顏色的attribute

<table border="0" width="60%" bgcolor="#ffffcc">
    <tr><td>
        <font color="\${fontColor}"><jsp:doBody/></font>
    </td></tr>
</table>
    ↑ 把body內容當成一般文字
    ↑ 輸出body內容
```

2. 在使用這個 tag file 時，可以把一大段需要顯示的內容放在 body 中：

```
<%@ taglib prefix="tf" tagdir="/WEB-INF/tags" %>
<html>
    <body>
        <tf:banner webmaster="Simon Johnson" />

        <tf:content fontColor="#0000ff">
            MACSPEED.NET at present is a non-
            ...
        </tf:content>
            ↑ 把需要顯示的內容放在body中

        <tf:footer myemail="simon@macspeed.net" />
    </body>
</html>
```

3. 上列的 JSP 顯示的網頁會像這樣：

The screenshot shows a JSP page with the following content:

My web application - Powered by Simon Johnson

**EMall System**

MACSPEED.NET at present is a non-commercial nature website, take provides the free teaching curriculum as the main service. The present teaching curriculum by the Java related technology primarily, non-periodically provides the free curriculum for all need friends, curriculas all have corresponding the discussion area, if has the question or the suggestion which the curriculum is connected. If you need other curricula, perhaps wants to join the curriculum to develop team, welcome to leave message in discussion area or email to informs me.

EMail: simon@macspeed.net

4. 如果你的 body 中會使用 standard tags、EL 或 JSTL，你應該要這樣設定：

...  
  <%@tag body-content="scriptless" %>  
  ...

↓ 設定為「scriptless」，或是  
乾脆不要加入這個設定，預設的  
設定就是「scriptless」

### 3. Simple Tag

## 建立與使用 Simple Tag

Custom tag API 分為 classic 和 simple 兩種。Simple custom tag 從 JSP 2.0 開始，它提供一種比 classic custom tag 更簡單的開發方式。下列的步驟可以建立與使用 simple custom tag：

1. 在「WEB-INF\tlds」下建立一個定義標籤用的 TLD 檔：

```
<?xml version="1.0" encoding="UTF-8"?> 表示這是一個XML文件
<taglib version="2.0" 除了設定「version」為「2.0」
    xmlns="http://java.sun.com/xml/ns/j2ee" 或「2.1」以外，其它是固定的設定
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation=
        "http://java.sun.com/xml/ns/j2ee web-jsptaglibrary_2_0.xsd">

    <tlib-version>1.0</tlib-version>
    <uri>http://macspeed.net/jsp/ctl/mysimpletag</uri> 設定使用這個TLD檔的「uri」
    ...
</taglib> 在這裡設定這個TLD檔包含的標籤，  
一個TLD檔可以定義多個標籤
```

2. 在 TLD 檔中可以使用這樣的設定來定義每一個標籤：

```
<?xml version="1.0" encoding="UTF-8"?>
<taglib version="2.0" ...>

    ...
    <tag>
        <name>標籤名稱</name>
        <tag-class>Tag handler類別的完整名稱</tag-class>
        <body-content>設定標籤的body</body-content>
    </tag>
</taglib>
```

一個標籤的設定

決定標籤在JSP中使用的名稱

如果標籤不需要body, 可以設定為「empty」

3. 使用 simple tag API 撰寫一個 tag handler 類別：

```
package net.macspeed.tags;
import java.io.IOException;

import javax.servlet.jsp.JspContext;
import javax.servlet.jsp.JspWriter; ← 與JSP相關的套件
import javax.servlet.jsp.JspException;
import javax.servlet.jsp.tagext.SimpleTagSupport; ← 與標籤相關的套件

public class MySimpleTagHandler extends SimpleTagSupport {
    ← 覆寫「doTag」方法
    public void doTag() throws JspException, IOException {
        JspContext jContext = getJspContext(); ← 取得「JspContext」物件
        JspWriter out = jContext.getOut(); ← 取得「JspWriter」物件
        out.print("Hello! Simple tag!"); ← 輸出內容
    }
}
```

## 4. 在 TLD 檔中加入這個標籤的定義：

```
<?xml version="1.0" encoding="UTF-8"?>

<taglib version="2.0" ...>

    ...
    <tag>
        <name>mySimpleTag</name>           ← 為標籤取一個名稱，就是在JSP中使用的名稱
        <tag-class>net.macspeed.tags.MySimpleTagHandler</tag-class>      ← 設定tag handler類別的完整名稱
        <body-content>empty</body-content>
    </tag>                                ← 設定這個標籤不可以使用body
</taglib>
```

## 5. 在 JSP 中使用這個標籤：

```
<%@taglib prefix="mst"
           uri="http://macspeed.net/jsp/ctl/mysimpletag" %>

<html>
    <body>
        <mst:mySimpleTag/>             ← 使用mySimpleTag標籤
    </body>
</html>
```

6. 上列的 JSP 顯示的網頁會像這樣：

Hello! Simple tag!

7. 下列是 TLD 檔與 JSP 中相關的設定：

```
...<uri>http://macspeed.net/jsp/ctl/mysimpletag</uri><tag><name>mySimpleTag</name><tag-class>net.macspeed.tags.MySimpleTagHandler</tag-class><body-content>empty</body-content></tag>...<%@taglib prefix="mst" uri="http://macspeed.net/jsp/ctl/mysimpletag" %><html><body><mst:mySimpleTag/></body></html>
```

## 包含 body 的標籤

你可以決定標籤是否可以包含 body：

```
<?xml version="1.0" encoding="UTF-8"?>
<taglib version="2.0" ...>
    ...
    <tag>
        <name>mySimpleTag</name>
        <tag-class>net.macspeed.tags.MySimpleTagHandler</tag-class>
        <body-content>設定</body-content>
    </tag>
</taglib>
```



可以設定為「empty」，「scriptless」  
或「tagdependent」

標籤 body 的設定值有三種：

- ❑ empty：不允許使用 body
- ❑ tagdependent：把 body 中文字當成一般的文字，可以包含網頁文字，HTML 標籤
- ❑ scriptless：預設的設定；不允許 body 中使用 scriplets 與 expression；伺服器轉換 body 中的 standard tags、EL 與 JSTL

下列的表格表示 body 的設定 tagdependent 與 scriptless，body 中出現各種不同內容時，伺服器處理的規則：

設定與內容	tagdependent	scriptless
Scriptlet	忽略	不允許
Expression	忽略	不允許
Standard Tags	忽略	轉換
EL	直接顯示 EL	轉換
JSTL	忽略	轉換

你可以把剛才的標籤設定修改為包含 body 的設定：

```
<?xml version="1.0" encoding="UTF-8"?>
<taglib version="2.0" ...>
...
<tag>
    <name>mySimpleTag</name>
    <tag-class>net.macspeed.tags.MySimpleTagHandler</tag-class>
    <body-content>scriptless</body-content>
</tag>
</taglib>
```

↑ 從「empty」修改為「scriptless」

一個包含 body 的標籤，可以使用下列的方式來輸出 body 的內容：

```
package net.macspeed.tags;
...
public class MySimpleTagHandler extends SimpleTagSupport {
    public void doTag() throws JspException, IOException {
        JspContext jContext = getJspContext();
        JspWriter out = jContext.getOut();
        out.print("Hello! Simple tag!");

        out.println("<p>");           ← 取得「JspFragment」物件
        JspFragment jspBody = getJspBody();

        jspBody.invoke(null);          ← 輸出body內容
        out.println("</p>");

    }
}
```

使用這個標籤的時候，就可以加入 body：

```
<%@taglib prefix="mst"
    uri="http://macspeed.net/jsp/ctl/mysimpletag" %>

<html>
    <body>
        <mst:mySimpleTag>
            This is body! ← 可以修改為使用包含body的標籤用法
        </mst:mySimpleTag>
    </body>
</html>
```

上列的 JSP 顯示的網頁會像這樣：

Hello! Simple tag!

This is body!

## 標籤的 body 與 JspFragment

標籤的 body 設定會決定它的用法,如果標籤的 body 設定為「empty」的話,在使用標籤的時候就不可以包含 body:

```
<body-content>empty</body-content>
```

標籤的body設定為「empty」  
的話,在使用標籤時就不可以包  
含body,否則會造成錯誤

```
<mst:mySimpleTag/>
```

標籤的 body 設定為「scriptless」或「tagdependent」,使用標籤的時候才可包含 body:

```
<body-content>scriptless</body-content>
```

或

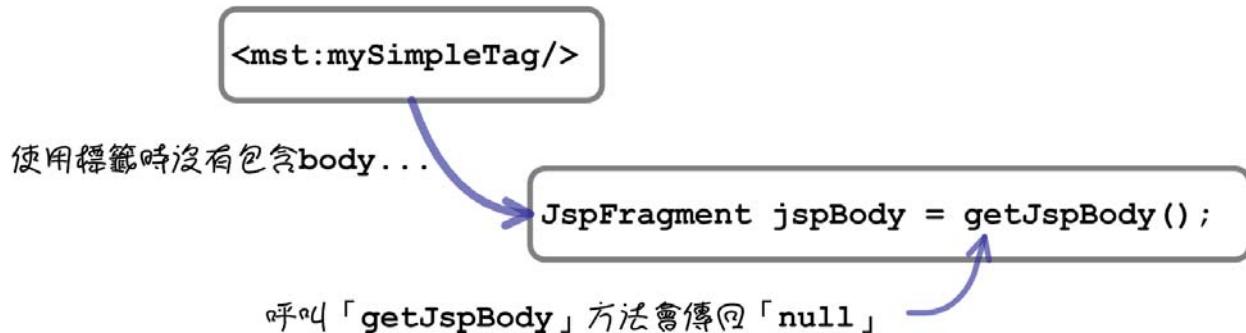
```
<body-content>tagdependent</body-content>
```

標籤的body設定為「scriptless」  
或「tagdependent」,使用標籤時有  
沒有包含body都可以

```
<mst:mySimpleTag/>
```

```
<mst:mySimpleTag>  
This is body!  
</mst:mySimpleTag>
```

因為使用標籤的時候，有沒有包含 body 都可以，所以你在 tag handler 類別中，應該要特別注意：

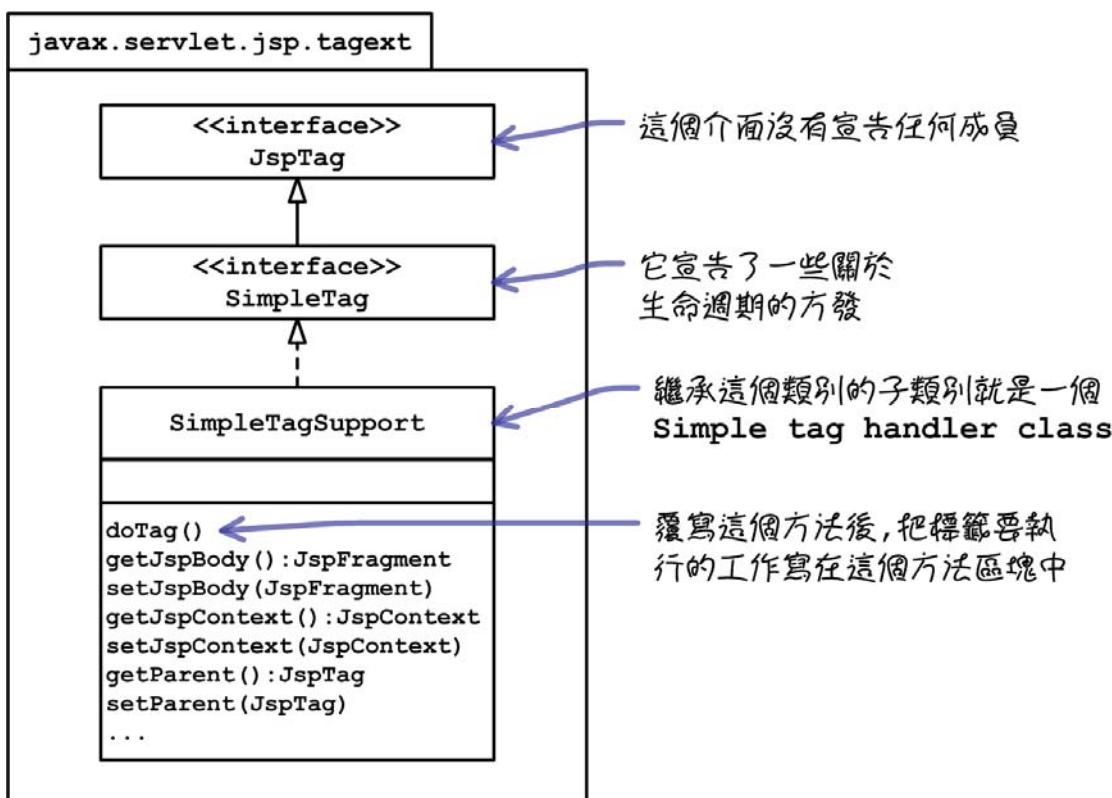


為了預防產生「NullPointerException」，通常會在 tag handler 類別中加入類似這樣的程式片段：

```
JspFragment jspBody = getJspBody();  
if (jspBody != null) { ← 如果不是「null」的話, 表示  
    out.println("<p>");   使用標籤的時候有包含body  
  
    jspBody.invoke(null); ← 再呼叫「invoke」方法, 輸出body的內容  
  
    out.println("</p>");  
}
```

## Simple Tag API

下列是 simple tag API :



當你在「doTag」方法中撰寫程式的時候，可以不用處理「JspException」與「IOException」：

「doTag」方法的宣告已經包含這些例外宣告，所以程式寫起來會比較簡單一些

```

public void doTag() throws JspException, IOException {
    ...
}

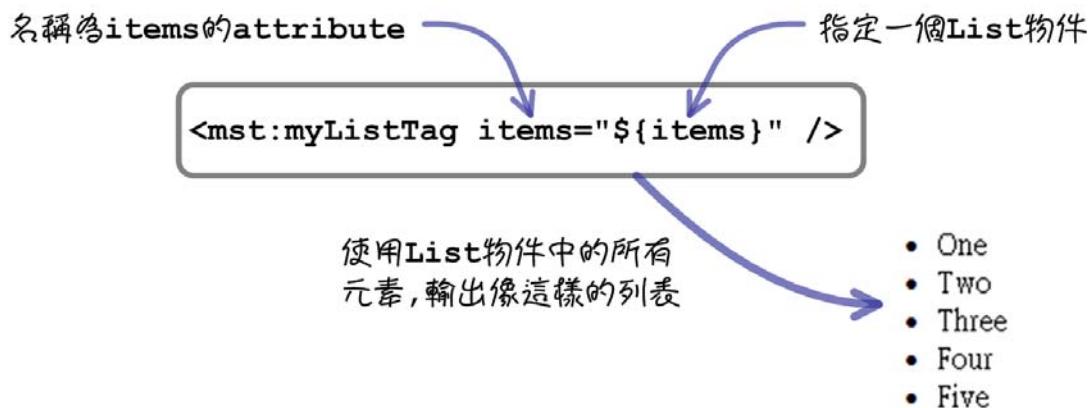
```

下列是 simple tag handler 的生命週期：



## 設定與使用標籤的 Attributes

開發 simple tag 時，可以依照需求，為標籤加入 attribute 的設定。以下列的標籤來說：



如果標籤需要 attribute 的話，你要先決定 attribute 的名稱，然後在 TLD 檔中使用「attribute」設定：

```
...
<tag>
  <name>mySimpleListTag</name>
  <tag-class>net.macspeed.tags.MySimpleListTagHandler</tag-class>
  <body-content>empty</body-content>

  <attribute>
    <name>items</name>          ← 設定 attribute 名稱
    <required>true</required>   ← 是否必要
    <rteexprvalue>true</rteexprvalue> ← 動態資料或固定文字
  </attribute>
</tag>
...
```

決定好 attribute 的名稱後，在 tag handle 類別中，也要使用 JavaBean 的規則，為每一個標籤的 attribute 宣告一個 setter 方法，伺服器會自動呼叫每一個 attribute 對應的 setter 方法設定資料：

```
<attribute>
```

```
  <name>items</name>
  <required>true</required>
  <rteprvalue>true</rteprvalue>
</attribute>
```

Attribute名稱為「items」

```
package net.macspeed.tags;
```

```
...
public class MySimpleListTagHandler extends SimpleTagSupport {
```

```
    private List items; ← 通常會宣告一個存放attribute資料的欄位
```

```
    public void setItems(List items) { ← 依照JavaBean的規則，使用attribute名稱寫一個
        this.items = items; ← setter方法
    }
```

```
    public void doTag() throws JspException, IOException {
```

```
        JspContext jContext = getJspContext();
        JspWriter out = jContext.getOut();
```

```
        if (items != null && items.size() > 0) {
            out.println("<ul>"); ← 如果List物件中有任何元素的話...
```

```
            for (Object o : items) {
                out.print("<li>");
                out.print(o);
                out.print("</li>"); ← 輸出List物件中所有元素
            }
        
```

```
        out.println("</ul>");
```

```
    } ← 輸出沒有任何資料的訊息
    else {
        out.println("<p>Empty!</p>");
```

```
}
```

## 設定資料給 body 使用

在 JSTL 的 core 中提供一個 forEach 標籤，它用來處理與輸出 Collection 或陣列資料時非常方便：

```

    ↗ 使用JSTL中的core標籤
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>

    ↗ 指定一個變數名稱
<c:forEach var="item" items="${myList}" >

        ↗ 指定一個List物件
        ↗ 使用EL輸出List物件中的元素
    </c:forEach>

```

如果想要提供一個與上列 forEach 標籤同樣功能的標籤，你會在 TLD 檔中使用這樣的設定：

```

...
<tag>
    <name>mySimpleForEachTag</name>
    <tag-class>
        net.macspeed.tags.MySimpleForEachTagHandler
    </tag-class>
        ↗ 將body設定為scriptless
    <body-content>scriptless</body-content>

    <attribute>
        <name>items</name>
        <required>true</required> ↗ 接收一個List物件的attribute
        <rteprvalue>true</rteprvalue>
    </attribute>

    <attribute>
        <name>var</name>
        <required>true</required> ↗ 指定輸出變數名稱的attribute
        <rteprvalue>false</rteprvalue>
    </attribute>
</tag>
...

```

這種標籤的設定方式會不太一樣，你必須在輸出 body 內容前，將 body 需要的資料設定好：

```
package net.macspeed.tags;  
...  
public class MySimpleForEachTagHandler extends SimpleTagSupport {  
  
    private List items;  
    private String var; ← 存放attribute資料的欄位  
  
    public void setItems(List items) { ← 「items」attribute  
        this.items = items;  
    } ← 的setter方法  
  
    public void setVar(String var) { ← 「var」attribute  
        this.var = var;  
    } ← 的setter方法  
  
    public void doTag() throws JspException, IOException {  
  
        JspContext jContext = getJspContext();  
        JspFragment jBody = getJspBody();  
  
        try {  
            if (items != null) { ← 先判斷items是否為null  
                for (Object o : items) {  
                    jContext.setAttribute(var, o); ← 使用「var」指定的名  
                    jBody.invoke(null); ← 稱，把一個元素設定為  
                } ← JspContext範圍的  
            } ← attribute  
        } ← 輸出body內容  
        catch (Exception e) {  
            throw new JspException(e);  
        }  
    }  
}
```

這個標籤使用起來的效果，就跟 `forEach` 標籤的基本用法一樣：

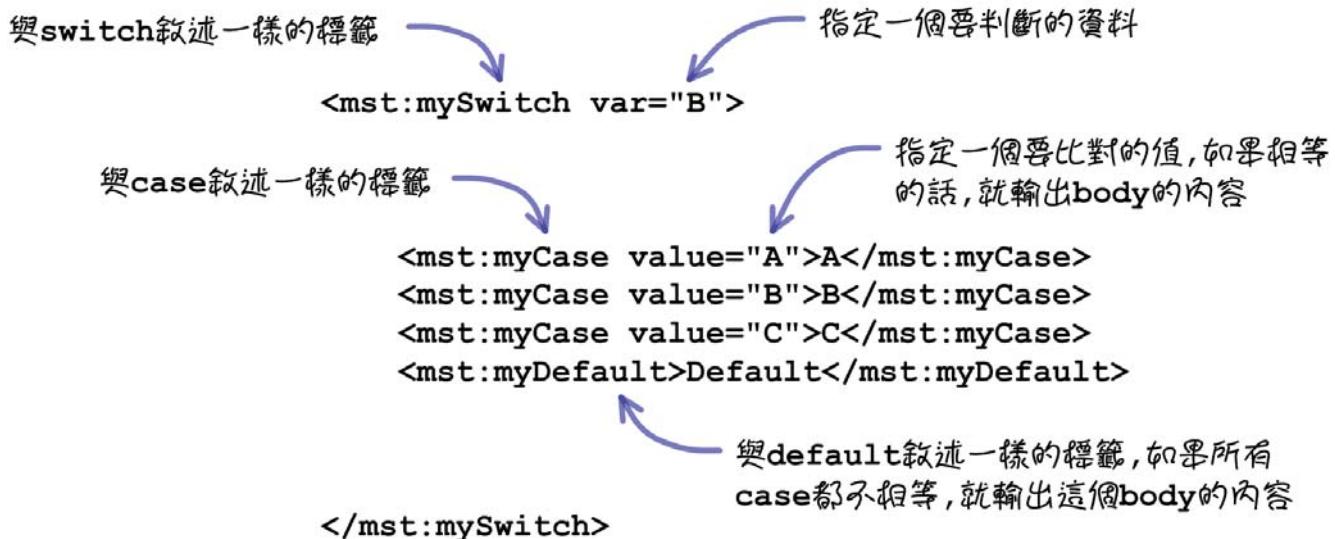
The diagram shows the XSLT code for the `<mst:mySimpleForEachTag>` tag with handwritten annotations:

- An annotation points to the `var="item"` attribute with the text "名稱為 var 的 attribute".
- An annotation points to the `items="${items}"` attribute with the text "名稱為 items 的 attribute".
- An annotation points to the `<li>${item}</li>` element with the text "在 body 中使用 var 指定的名稱輸出".
- An annotation points to the closing tag `</mst:mySimpleForEachTag>` with the text "指定一個 List 物件".

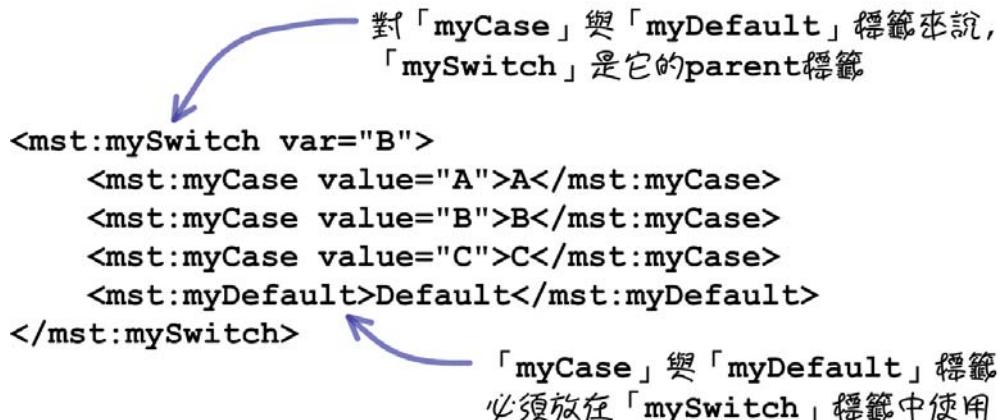
```
<mst:mySimpleForEachTag var="item" items="${items}">
  <li>${item}</li>
</mst:mySimpleForEachTag>
```

## 巢狀標籤

在 JSP 網頁中，你可能需要使用 Java 程式語言中的「switch」來執行一些判斷工作，雖然 JSTL 的 core 標籤中提供了「choose」、「when」和「otherwise」，不過它們的用法跟 switch 不太一樣。所以你也可以自己設計一組像這樣的標籤：



之前所討論過的標籤都是單獨使用的，像這樣把標籤組合起來使用的話，就會有「parent」標籤產生了：



這組標籤在 TLD 檔中的定義不會有比較特別的地方，依照標籤各自的需求來定義就可以了：

```
...
<tag>
  <name>mySwitch</name> ← 執行switch工作的標籤
  <tag-class>net.macspeed.tags.MySimpleSwitchTagHandler</tag-class>
  <body-content>scriptless</body-content>
  <attribute>
    <name>var</name>
    <required>true</required> ← 指定判斷資料的attribute
    <rteprvalue>true</rteprvalue>
  </attribute>
</tag>

<tag>
  <name>myCase</name> ← 執行case工作的標籤
  <tag-class>net.macspeed.tags.MySimpleCaseTagHandler</tag-class>
  <body-content>scriptless</body-content>
  <attribute>
    <name>value</name>
    <required>true</required> ← 和switch中比對值的attribute
    <rteprvalue>true</rteprvalue>
  </attribute>
</tag>

<tag>
  <name>myDefault</name> ← 執行default工作的標籤
  <tag-class>net.macspeed.tags.MySimpleDefaultTagHandler</tag-class>
  <body-content>scriptless</body-content>
</tag> ...
  <br/> ← 不需要attribute
```

因為要判斷的資料是儲存在 parent 標籤，而每一個「myCase」標籤都需要判斷自己指定的值是否相等，所以在設計 parent 標籤的 tag handler 類別時，要提供讓「myCase」與「myDefault」標籤執行判斷的方法：

```

package net.macspeed.tags;
...
public class MySimpleSwitchTagHandler extends SimpleTagSupport {

    private String var; ← 儲存判斷資料的欄位
    private boolean caseMatch = false; ← 紀錄是否有某個
                                        case已經相等

    public void setVar(String var) {
        this.var = var; ← 判斷資料的setter方法
    }

    public boolean isCaseMatch() { ← 傳回是否有某個
                                    case已經相等
        return caseMatch;
    }

    public boolean match(String value) {
        boolean result = var.equals(value); ← 比對資料是否相等
        if (result) {
            caseMatch = true; ← 某個case已經相等
        }
        return result;
    }

    public void doTag() throws JspException, IOException {
        JspFragment jBody = getJspBody();
        if (jBody != null) {
            jBody.invoke(null); ← 輸出body內容
        }
    }
}

```

在「myCase」標籤的 tag handler 類別中，可以取得 parent 標籤物件，並執行必要的工作：

```
package net.macspeed.tags;
...
public class MySimpleCaseTagHandler extends SimpleTagSupport {

    private String value; ← 儲存比對資料的欄位

    public void setValue(String value) {
        this.value = value; ← 比對資料的setter方法
    }

    public void doTag() throws JspException, IOException {

        JspFragment jBody = getJspBody();

        MySimpleSwitchTagHandler parent = (MySimpleSwitchTagHandler) getParent(); ← 呼叫「getParent」方法取得parent標籤物件
        ← 傳回parent標籤物件的型態為「JspTag」，所以要執行轉型

        if (parent != null && !parent.isCaseMatch()) {
            if (parent.match(value)) { ← 呼叫parent標籤的方法，判斷是否已經有case相等了
                jBody.invoke(null); ← 呼叫parent標籤的方法執行比對的工作
            }
        }
    }
}
```

比對的結果如果相等的話，就輸出body的內容

在「myDefault」標籤的 tag handler 類別中，也可以取得 parent 標籤物件，如果所有「myCase」都不相等的話，就輸出 body 內容：

```
package net.macspeed.tags;  
...  
public class MySimpleDefaultTagHandler extends SimpleTagSupport {  
  
    public void doTag() throws JspException, IOException {  
  
        JspFragment jBody = getJspBody();  
  
        MySimpleSwitchTagHandler parent =  
            (MySimpleSwitchTagHandler) getParent();  
  
        if ( jBody != null && parent != null &&  
            !parent.isCaseMatch() ) {  
            jBody.invoke(null);  
        }  
    }  
}
```

如果還沒有case相等的話...  
就輸出body的內容

## 使用 SkipPageException

在「javax.servlet.jsp」套件中，提供一個「SkipPageException」例外類別，可以在 simple tag handler 類別中使用，用來控制標籤後面的網頁是否要繼續處理：

```
...
public void doTag() throws JspException, IOException {
    JspContext jContext = getJspContext();
    JspWriter out = jContext.getOut();

    out.print("Message in doTag()<br>");

    if (somethingWrong) {
        throw new SkipPageException();
    }
}
```

如果這裡丟出這個例外...

```
...
<html>
    <body>
        Start!<br>
        <mst:skippage/>
        End!<br>
    </body>
</html>
```

「SkipPageException」並不會造成程式中斷，不過標籤後面的網頁就不會處理了

SkipPageException 的效果只有在使用標籤的 JSP，這個效過在下列的狀況中會比較清楚一些：

```
...
public void doTag() throws JspException, IOException {
    JspContext jContext = getJspContext();
    JspWriter out = jContext.getOut();

    out.print("Message in doTag()<br>");

    if (somethingWrong) {
        throw new SkipPageException();
    }
}
```

如果這裡丟出這個例外...

```
...
<html>
    <body>
        Page 1 Start!<br>
        <jsp:include page="page2.jsp" />
        Page 1 End!<br>
    </body>
</html>
```

「SkipPageException」  
的效果不會發生在這個JSP

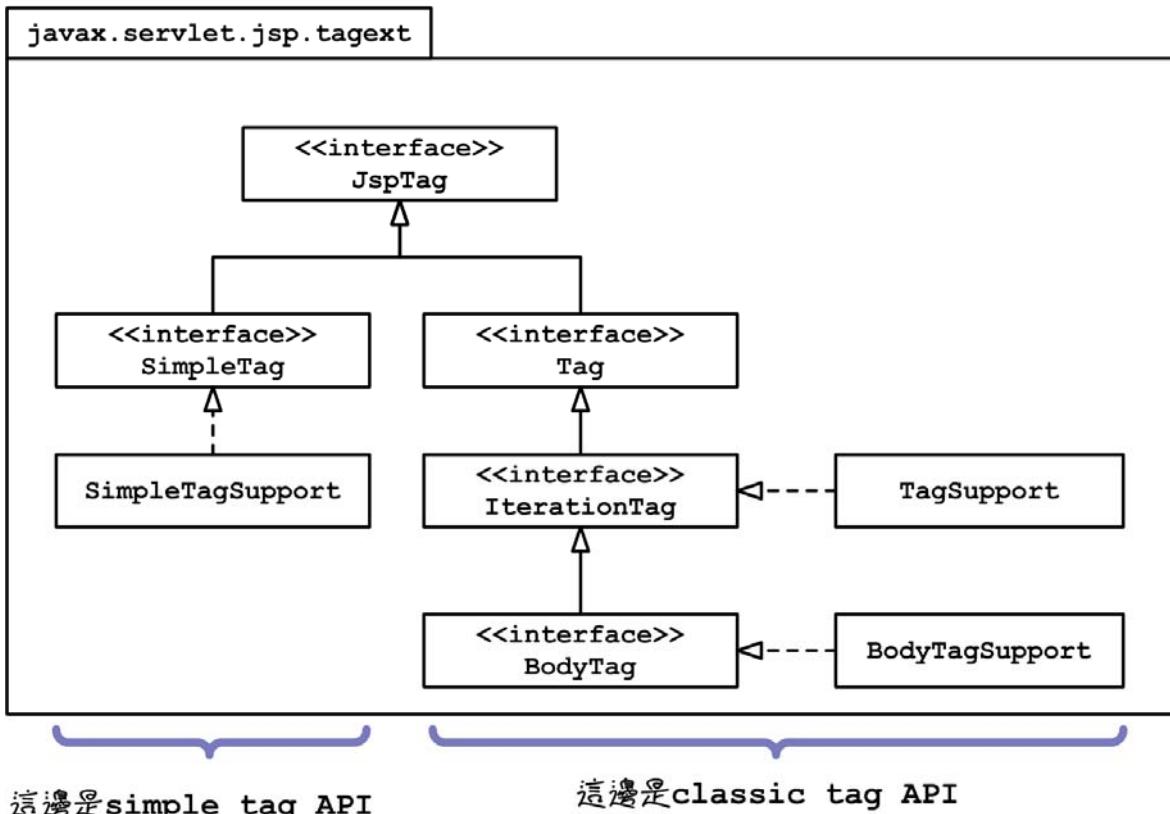
```
...
Page 2 Start!<br>
<mst:skippage/>
Page 2 End!<br>
```

標籤後面的網頁會處理，所以它不會顯示

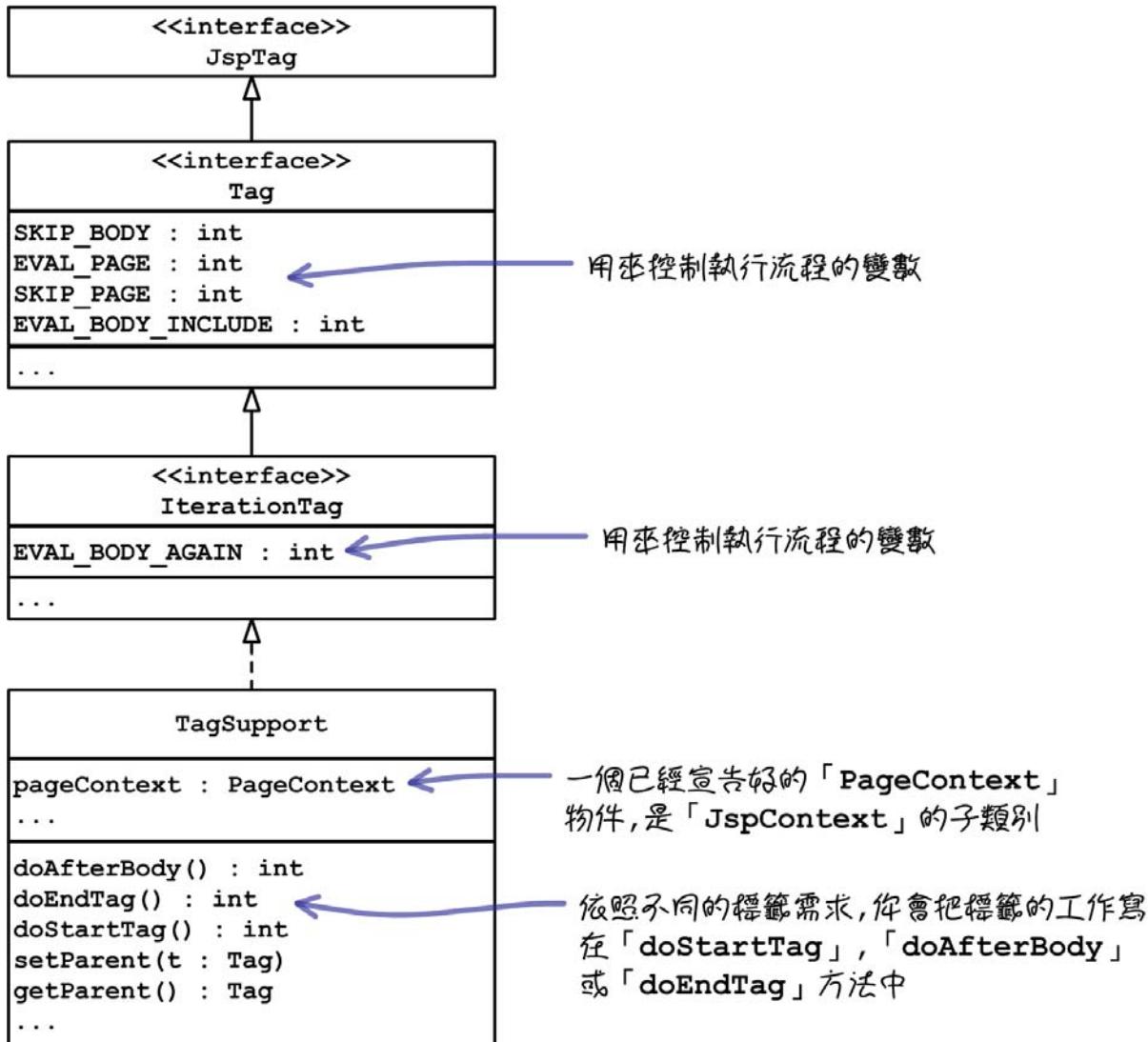
## 4. Classic Tag

### Classic Tag API

Custom tag API 在 2.0 開始才加入 simple tag API，在 2.0 之前的 custom tag API 稱為 classic。下列的圖型包含兩種 API 的主要成員：



Classic tag handler 類別要繼承自「TagSupport」：



下列的步驟可以建立與使用一個 classic custom tag :

1. 使用 classic tag API 撰寫一個 tag handler 類別：

```

package net.macspeed.tags;

import java.io.IOException;
import javax.servlet.jsp.JspException;
import javax.servlet.jsp.JspWriter;           ← 需要的套件
import javax.servlet.jsp.tagext.TagSupport;

public class MyClassicTagHandler extends TagSupport {           ← 繼承「TagSupport」

    public int doStartTag() throws JspException {           ← 覆寫「doStartTag」方法

        JspWriter out = pageContext.getOut();           ← 使用父類別宣告好的
                                                        「pageContext」取得
                                                        「JspWriter」物件

        try {           ← 輸出內容

            out.println("Hello! Classic tag!");

        }
        catch (IOException e) {           ← 要自己處理「IOException」
            throw new JspException( e.toString() );
        }

        return SKIP_BODY;           ← 沒有body要處理的話，就回傳「SKIP_BODY」
    }
}

```

2. Classic 與 simple tag 在 TLD 檔中的設定完全一樣，你以依照在 simple tag 討論過的方式，建立需要的 TLD 檔，然後加入這個標籤的定義：

```
<?xml version="1.0" encoding="UTF-8"?>

<taglib version="2.0" ...>

    ...
    <tag>
        <name>myClassicTag</name>           ← 為標籤取一個名稱，就是在JSP中使用的名稱
        <tag-class>net.macspeed.tags.MyClassicTagHandler</tag-class>   ← 設定tag handler類別的完整名稱
        <body-content>empty</body-content>
    </tag>                                ← 設定這個標籤不可以使用body

</taglib>
```

3. 在 JSP 中使用這個標籤：

```
<%@taglib prefix="mst"
           uri="http://macspeed.net/jsp/ctl/myclassictag" %>

<html>
    <body>
        <mst:myClassicTag/>             ← 使用myClassicTag標籤
    </body>
</html>
```

## Classic Tag 的生命週期

下列是 classic tag handler 的生命週期：

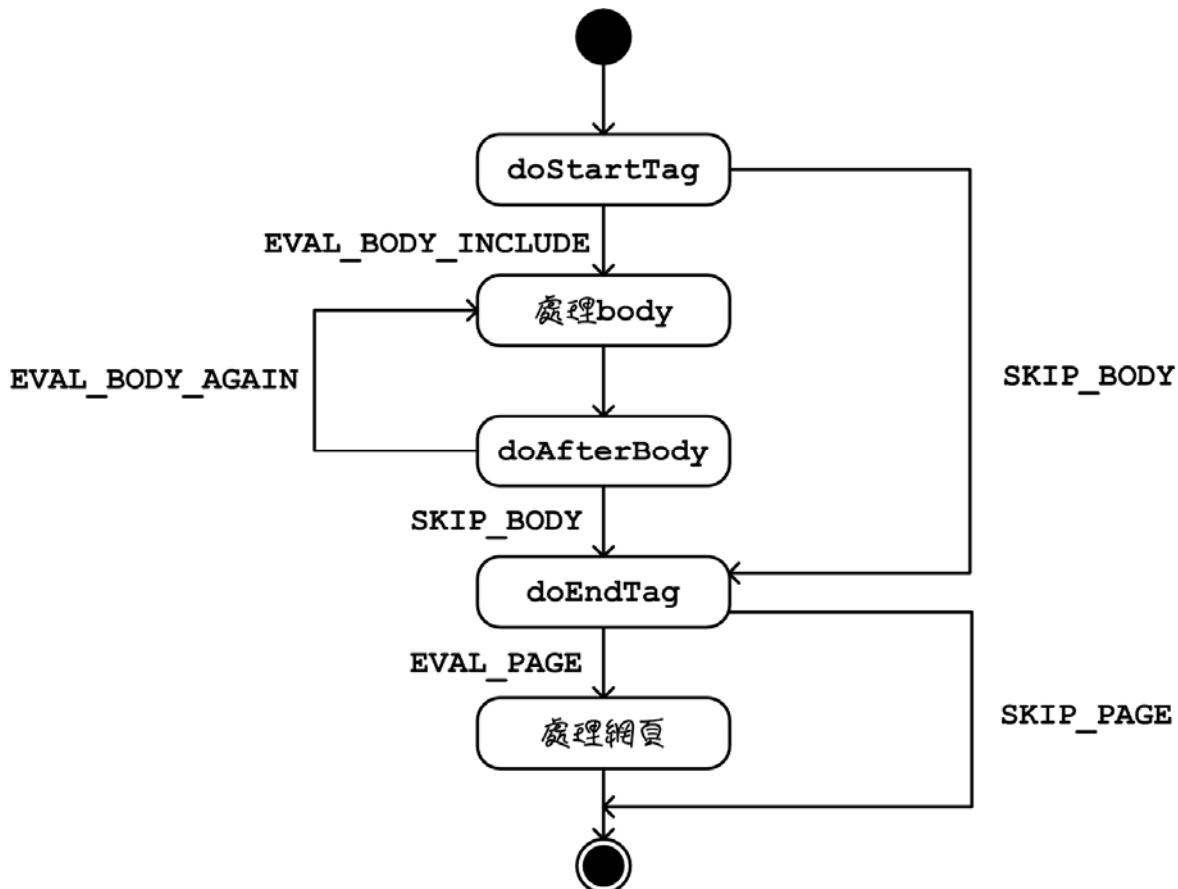


如果是一個沒有包含 body 而且比較簡單的標籤，使用 classic 與 simple 兩種 API 來開發，其實是差不多的。不過使用 classic tag API 開發包含 body 的標籤，就會比較複雜一些。

Classic tag API 的「`doStartTag`」、「`doAfterBody`」與「`doEndTag`」三個方法，都會回傳一個「`int`」整數，用來決定後續的流程與處理方式。下列的表格是方法與回傳值的效果：

方法	回傳值	後續流程
<code>doStartTag</code>	SKIP_BODY(預設)	<code>doEndTag</code>
	EVAL_BODY_INCLUDE	處理 body → <code>doAfterBody</code>
<code>doAfterBody</code>	SKIP_BODY(預設)	<code>doEndTag</code>
	EVAL_BODY_AGAIN	處理 body → <code>doAfterBody</code>
<code>doEndTag</code>	SKIP_PAGE	不處理標籤後面的網頁
	EVAL_PAGE(預設)	繼續處理標籤後面的網頁

你也可以使用下列的流程圖來瞭解它們的運作方式：



你可以先從「doEndTag」來瞭解 classic tag API 的流程控制，下列的範例在原來的 classic tag handler 類別中，加入覆寫「doEndTag」方法的程式碼：

```
package net.macspeed.tags;
...
public class MyClassicTagHandler extends TagSupport {
    public int doStartTag() throws JspException {
        JspWriter out = pageContext.getOut();
        try {
            out.println("Hello! Classic tag!");
        }
        catch (IOException e) {
            throw new JspException( e.toString() );
        }
        return SKIP_BODY; ← 向傳「SKIP_BODY」的話，會接著呼叫「doEndTag」方法
    }
}

public int doEndTag() throws JspException {
    return SKIP_PAGE; ← 覆寫「doEndTag」方法
} ← 預設的回傳值為「EVAL_PAGE」，把它改為回傳「SKIP_PAGE」
```

修改使用標籤的 JSP 後，就可以瞭解「doEndTag」方法的回傳值會有什麼效果：

```
<%@taglib prefix="mst"
    uri="http://macspeed.net/jsp/ctl/myclassicitag" %>

<html>
    <body>
        Start!<br>
        <mst:myClassicTag/><br>
        End!<br>
    </body>
</html>
```

因為這個標籤在「doEndTag」方法的回傳值為「SKIP\_PAGE」...  
所以標籤後面的網頁內容都不會處理了

## 包含 body 的 Classic Tag

如果要使用 classic tag API 開發類似 JSTL 提供的 forEach 標籤，會比使用 simple tag API 複雜一些，不過你可以透過它來瞭解 classic tag API 如何控制 body 的輸出與流程。

使用 classic tag API 開發時，在 TLD 中關於標籤還有 attributes 的設定，與 simple tag API 一樣：

```
...
<tag>
    <name>myClassicForEachTag</name>

    <tag-class>
        net.macspeed.tags.MyClassicForEachTagHandler
    </tag-class>
    <body-content>scriptless</body-content>
        ↑ 將body設定為scriptless

    <attribute>
        <name>items</name>           ← 接收一個List物件的attribute
        <required>true</required>
        <rteprvalue>true</rteprvalue>
    </attribute>

    <attribute>
        <name>var</name>             ← 指定輸出變數名稱的attribute
        <required>true</required>
        <rteprvalue>false</rteprvalue>
    </attribute>
</tag>
...

```

使用 classic tag API 開發的 forEach 標籤會像這樣：

## 巢狀標籤

Classic tag API 與 simple tag API 一樣有巢狀標籤的設計方式，你可以使用 classic tag API 開發一組與「switch」敘述一樣的標籤。下列是這組標籤在 TLD 檔中的定義：

```
<tag>
  <name>mySwitch</name> ← 執行switch工作的標籤
  <tag-class>net.macspeed.tags.MyClassicSwitchTagHandler</tag-class>
  <body-content>scriptless</body-content>
  <attribute>
    <name>var</name>
    <required>true</required> ← 指定判斷資料的attribute
    <rteprvalue>true</rteprvalue>
  </attribute>
</tag>

<tag>
  <name>myCase</name> ← 執行case工作的標籤
  <tag-class>net.macspeed.tags.MyClassicCaseTagHandler</tag-class>
  <body-content>scriptless</body-content>
  <attribute>
    <name>value</name>
    <required>true</required> ← 和switch中比對值的attribute
    <rteprvalue>true</rteprvalue>
  </attribute>
</tag>

<tag>
  <name>myDefault</name> ← 執行default工作的標籤
  <tag-class>net.macspeed.tags.MyClassicDefaultTagHandler</tag-class>
  <body-content>scriptless</body-content>
</tag>
```

← 不需要attribute

因為要判斷的資料是儲存在 parent 標籤，而每一個「myCase」標籤都需要判斷自己指定的值是否相等，所以在設計 parent 標籤的 tag handler 類別時，要提供讓「myCase」與「myDefault」標籤執行判斷的方法：

```

package net.macspeed.tags;
...
public class MyClassicSwitchTagHandler extends TagSupport {

    private String var; ← 儲存判斷資料的欄位
    private boolean caseMatch = false; ← 紀錄是否有某個
                                         case已經相等

    public void setVar(String var) {
        this.var = var; ← 判斷資料的setter方法
    }

    public boolean isCaseMatch() { ← 傳回是否有某個
                                   case已經相等
        return caseMatch;
    }

    public boolean match(String value) {
        boolean result = var.equals(value);
        if (result) {
            caseMatch = true; ← 比對資料是否相等
        }
        return result;
    }

    public int doStartTag() throws JspException {
        return EVAL_BODY_AGAIN; ← 繼續處理body內容
    }
}

```

在「myCase」標籤的 tag handler 類別中，可以取得 parent 標籤物件，並執行必要的工作：

```
package net.macspeed.tags;
...
public class MyClassicCaseTagHandler extends TagSupport {

    private String value; ← 儲存比對資料的欄位

    public void setValue(String value) {
        this.value = value; ← 比對資料的setter方法
    }

    public int doStartTag() throws JspException {
        MyClassicSwitchTagHandler parent =
            (MyClassicSwitchTagHandler) getParent(); ← 呼叫「getParent」方法取得parent標籤物件
        ← 傳回parent標籤物件的型態為
        ← 「Tag」，所以要執行轉型

        if (parent != null &&
            !parent.isCaseMatch() &&
            parent.match(value)) { ← 呼叫parent標籤的方法，判斷是否已經有case相等了
            return EVAL_BODY_INCLUDE; ← 呼叫parent標籤的方法執行比對的工作
        }
        else {
            return SKIP_BODY; ← 比對的結果如果相等的話，就處理body的內容
        }
    }
}
```

不處理body的內容

在「myDefault」標籤的 tag handler 類別中，也可以取得 parent 標籤物件，如果所有「myCase」都不相等的話，就輸出 body 內容：

```
package net.macspeed.tags;
...
public class MyClassicDefaultTagHandler extends TagSupport {

    public int doStartTag() throws JspException {

        MyClassicSwitchTagHandler parent =
            (MyClassicSwitchTagHandler) getParent();

        if (parent != null && !parent.isCaseMatch()) {
            return EVAL_BODY_INCLUDE; ← 如果還沒有case相等的話...
        }
        else {
            return SKIP_BODY; ← 不處理body的內容
        }
    }
}
```