

行前準備

本書與 Java Server Page

Java Server Page 版本

現在要寫 JSP - Java Server Page 的文件，實在是非常困難的事情，因為目前的 JSP 版本 0.92 → 0.91 → 1.0 改變很大，實作最新版本 1.0 的產品還不多，而且由於已經利用 0.9x 版製作出來的系統不能輕易地放棄，所以本篇會將這三個主流的版本加以介紹。

整篇我會以 JSP 1.0 的規範為核心並以其它舊版本為輔作介紹，舊版本的介紹便是以 Websphere 1.0（0.91 改）的語法為主，方便使用這類產品的讀者。

JSP 版本規格書

以下是各版 JSP 規格

1.1 public draft

這是 JSP 1.1 的草案規格，目前仍未有公開的測試版本，僅有草案規格書開放供人參考，並附有一些參考的原始碼，**不建議初學者閱讀它**。其與 JSP 1.0 差異請參考“JSP 個版本間的差異”。網址：

<http://java.sun.com/products/jsp/techinfo.html>

1.0

目前各家廠商加緊腳步要追趕上的版本，目前尚未有穩定的版本、JSP 第一版的正式規格，同時也是本書撰寫 Java Server Page 篇的主要介紹核心。網址：

<http://java.sun.com/products/jsp>

0.92

整體 JSP 架構最完整的版本，為後來的 JSP 1.0 訂立了許多新觀念，同時也是目前最受歡迎的 JSP 版本，雖然在 JSP 1.0 中，語法有了大量的改變，不過主要的因素是要讓 JSP 1.0 正規化，而不是 0.92 功能不夠完整，

甚至有些 0.92 的功能，由於 1.0 來不及實作，變成選擇性的實作規格，而放到 1.1 才是建議規格。網址：

<http://java.sun.com/products/jsp/jsp092.html>

0.91

0.91 版是最多實作支援的版本，目前多數的 JSP 規格制定前的版本大多都至少支援 0.91。網址：

http://www.burridge.net/jsp/Spec91/jsp_spec.html

關於本書 JSP 範例

JSWDK1.0 EA

由於本書的寫到此時，支援 JSP 1.0 的產品還不是很多，大多是只支援 0.92、0.91 至於 1.0 的語法。由於本書以 1.0 規格為主，所以我會在接下來的文章中教導各位如何使用 JSWDK 1.0EA，不過 JSWDK 只支援 ASCII 編碼，中文會有問題，所以各位試驗時看到的都會是英文的 JSP，但我想這不構成學習上的問題。

WebSphere 1.0

本書還會介紹 WebSphere 1.0，而 WebSphere 使用的是 JSP 0.91 加上一些特別的語法，這些範例你可以拿到 WebSphere 測試，除了 WebSphere 特別的語法外，理論上拿到其它支援 0.91 語法的 JSP 引擎上，應該也是可以執行。

範例檔的問題

接下來是範例檔的問題，本書所有的例子都可以在下面的網址下載

<ftp://ftp.ncnu.edu.tw/JavaDownload/usr/brenden/books/ServletGuide>

在篇會用到的例，它的位址會在下面這個目錄

```
/src/jsp_sample
```

而當我說

```
/src/jsp_sample/someone.zip
```

時，表示我現在要討論的是，someone.zip 裡面的範例，而若我說

```
/src/jsp_sample/someone.zip:src/somenoe.jsp
```

或 someone.zip:src/someone.jsp 時，表示我要討論或所指的是 someone.zip 這個檔案所解開來裡面的 src/目錄下的 someone.jsp 這個檔案，其餘類推。

範例的 JSP 版本

裡面所有的 JSP 範例檔案，在說明前，我都會以 **JSP0.91** / **JSP0.92** / **JSP1.0** 的記號來說明相容於何種版本的 JSP，至於界於 0.91 與 1.0 之間的 1.0 Public Draft 則是 1.0 的草稿規格，雖然有部份的實作以此為準，但是它們都不是正式的版本，所筆者並不想浪費時間討論它們，不過兩者的版本的差異可以參考“Java Server Page 版本差異”該章。

Java Server Page 實作與本書的假設立場

有許多可能的 JSP 實作方式，包含它的可重用元件、Scripting 語言以及 JSP 網頁編譯成 Java 物件型態。由於 JavaSoft 的 JSP 引擎，JSWDK，目前只打算支援 JavaBean 做為其可重用的元件而以 Java 做為它的 Scripting 語言，目前其它受歡迎的 Scripting 語言，如 JavaScript。不過以 JSP 未來的規格來看，就算支援其它的 Scripting 語言，同樣也是會以 Java 平台為準，因為 JavaSoft 並不打算讓 JSP 支援 JavaBean 以外的元件模型。

由於這些可能的實作仍在制定中，所以 Java Server Page 目前會被編譯成爲 Java 的物件，通常是成爲一個 Servlet 的子類別，不過目前已經有些實作是以其它的編譯型式存在了。

因此本書爲了專注於討論 Java Server Page 技術實際層面（這本書本來就是實作為主☺），我們必需選擇一種實作，爲了簡單化本書的撰寫，筆者決定以最常見的 Java Server Page 實作型式

- 可重用元件－JavaBeans
- Scripting 語言－Java
- Java Server Page 會編譯成的 Java 物件－Java Servlet

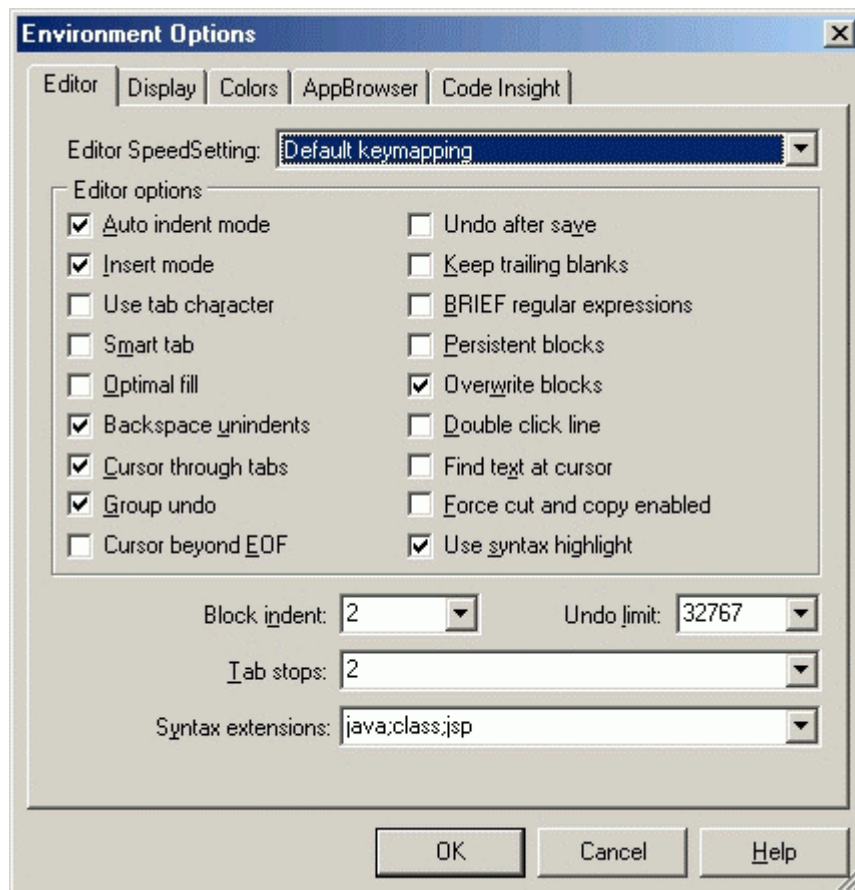
來討論接下來的課題。

開發工具

與 JBuilder 配合

其實 JBuilder 是不支援 Java Server Page 的，不過呢，對於習慣使用 JBuilder 做整合開發的人，JBuilder 裡還是有點小小的支援，這需要點技巧，打開選單的

Tools → Enviroment Option，你就可以看到如下圖的畫面，更改它的 **Syntax Extensions**，加入 **jsp**，你就可以得到部份的 JSP 中 Java 語法的語法高彩顯示，有沒有用就看個人的喜好了☺



圖片 1、JBuilder 支援 JSP 撰寫

不過其它的 JBuilder 功能，如 Code Insight 等，都不會有作用㊸。

支援語法高彩的編修工具

NetObjects ScriptBuilder 3.0

NetObjects ScriptBuilder 3.0 有相當好的 JSP 0.91 規格的語法支援，它與 WebSphere Application Server 支援的 JSP 版本相同，因為它就是 WebSphere 家族的配套工具，所以如果你要採用 WebSphere 做開發，請務必使用它來撰寫 JSP 檔

<http://www.netobjects.com/products/html/nsb3.html>

摘要：

WebSphere Application Server 的 0.91 語法含有不正式的語法，這些是 WebSphere 獨有的延伸機制，如 “<INSERT BEAN>”，不過 ScriptBuilder 完全支援它，這算是家族成員所必要做到的吧㊸，所以筆者建議用它，算是相當不錯的工具。

HomeSite

<http://www.allaire.com>

HomeSite 4.0.1 直接支援 JSP 0.91 與 0.92 檔案編修時的關鍵字高彩，不過如果你是舊的 4.0 版，它並不直接支援，但也可以取得下面網址所附的 Plug-in

<http://www.burridge.net/jsp/homesite.html>

JavaServer Page 簡介

Java Server Page 相關觀念與名詞解釋

Scripting Language

一般我們指程式語言（Programming Language）通常想到如 C/C++、Java、Basic 等等，這些語言通常要經過一段的學習時間，克服完學習曲線才能掌握它的使用。它們的功用主要是透過其語法掌控系統資源。

而 Scripting Language 的發展主要目的是為了提供掌控某些技術而設計的界面，例如 JavaScript 的原本設計就是為了掌控瀏覽器與其頁面資料，還有像是 Unix Shell，也通常有其 Scripting 語言來操控 Shell，更有複雜的是 Microsoft 的 VBA，Microsoft 的系列產品幾乎都有它的影子。

JSP 不一定是
包裝 Servlet
與 Java 語
言，也有可
能是其它的

所以通常我們可以把 Scripting 語言定義成某種子系統專用的語言，所以 Java Server Page 會被定義成 Scripting 語言的一種，主要是因為它是包裝 Java Servlet 這個子系統的界面，藉以簡化 Servlet 與 Java 語言的難度，並無法超脫它們而存在，所以不被認為是一種 Programming Language。

另一種 Script 語言的定義是指比程式語言更簡單些，複雜度少一些，不用懂程式語言的理論，可以輕易學會的語言，但也意味著會比普通程式語言受限語多；例如 Perl 早期許多人都是把它當 Script 語言看待，即使是今日 Perl 已經超出原本 Script 語言的功能，還是有許多人認為他是 Script 語言，不過這是題外話了。

Dynamic HTML

基本上 Server-Side Script 的目的即在於簡化類 CGI 程式寫作動態 HTML 的麻煩，因為如果我們以程式語言的角度來支援動態 HTML—Dynamic HTML，簡稱 DHTML，的產生，不管如何，對一個 HTML 寫作者來說，無疑是一種障礙。

延伸 HTML 語法

因此，

Java Server Page 與 DHTML

JSP 提供網頁的動態執行能力，使用的語法是有由一些 HTML（或 XML）延伸的標籤（Tag）或是 Java 語言，就像是寫一般 Servlet 一樣；通常它是以.jsp 為副檔名。

Client Side

Client 端上的網頁動態能力，以 Java 來說，最重要的莫過於 Applet 了。

Server-Side

所謂的 Server Side Script 的出現與起源，筆者目前並不是很清楚，若有較詳盡的資料，會立即補上。

以 HTML 寫作者的角度，延伸 HTML 的語法定義，增加動態的語法部份，以混合 HTML 的靜態部份，便是 Server-Side Script 的中心思想。

另外被稱之為 Server-Side 的意思，指的便是這類語法的處理會是在 Web Server 上面，而非 Client 上，例如 Java-Script 便是個 Client 的 Script，與它們正好相反。因此 Server-Side Script 的 Client 而言是透明的，與瀏覽器的版本無關，而且可以在其中做一些安全性的動作，而不用擔心 Client 端的問題

Java Server Page 與 Servlet、HTML 的關係

Java Server Page 技術主要是將 HTML（或 XML）的標籤延伸以放為 Java 程式碼，當 Web Server（或 Servlet 引擎、Application Server，怎麼稱呼都可以啦）支援 Java Server Page 時，它會依它的語法，將之轉換成 Servlet 程式。因此 JSP 檔就會變成 servlet，接著它就會被編譯成 Java 的執行碼（bytecode），以一般 Servlet 的運作方式，載入、執行等...。JSP 與 Servlet 的差異只是在於 Java Server Page 提供不了解 Servlet 的人，也可以簡單地利用延伸的標籤做一些動態網頁。

Java Server Page 語法

一個 JavaServer Page 檔是一種 HTML 檔再加上一些額外的程式碼在其中，利用<%%>與一般的 HTML 標籤語法做區別

```
<% JSP 程式碼從這邊開始 %>
```

簡單的 JSP 例子

下面是一個簡單的 JSP 檔例子

JSP0.91 JSP0.92 JSP1.0

原始碼 1、最基本的 JSP 網頁

```
<HTML>
<HEAD>
<TITLE>Simple Example</TITLE>
</HEAD>
<BODY>
<% out.println("Hello, world!"); %>
</BODY>
</HTML>
```

除了“<%...%>”內，就是一般的 Java 程式語法外，JSP 檔內還有什麼特別之處呢？這與 Servlet 又有什麼差異呢？請耐心的聽完我們接下來的文章，在更後文章中解釋。

技術的取舍

這點不全然正確，這是指 JSP 以 Servlet 引擎實作的情況才會成立

JSP 實際上會被轉譯成 Servlet，接著 Servlet 引擎就以，這轉譯規則這點，雖然有規格書來規範，但是產生的結果（轉譯成的程式碼），則是依各家的 JSP 引擎而有所差別，以目前 JSP 的規格來說，並沒有強制編譯後的 Servlet 形式為何，所以 JSP 一直到目前為止，都是止於 JSP 語法上的可攜性，至於編譯後的物件並不具可攜性，當然這裡指的可攜性是指跨越不同的 JSP 引擎，而不是指作業平台，例如 A 公司 JSP 引擎與公司 JSP 引擎兩者所產生的 JSP 實作物件並不能互換。

所以，JSP 技術的重點是什麼？為什麼不直接用 Servlet 就好了？對大多數的人來說，它有以下兩個優點

方便與 HTML 混合

以 HTML 來說 Java 以 JSP 的語法包裝延伸，使得 HTML 更容易加入動態部份，相反的 Servlet 卻要對 HTML 的輸出做額外的心力處理（像是“\”這類的跳脫字元），尤其在複雜的 HTML 網頁中，加入動態的部份，以 Servlet 處理簡直是惡夢

比 Servlet 開發方便

修改 JSP 後，馬上就可以看到執行結果，不需要編譯（不用你自己動手）；而 Servlet 卻要編譯，放入執行碼等複雜的動作。不過以程式語言的角度來說，HTML 與程式語法混合，會讓程式混亂，除錯與維護都不易，傳統上的程式開發中的語法與執行時的除錯已經發展得不錯，但是 JSP 則不然，因此，“比 Servlet 開發方便”是專指對 Java 語言不熟的人而

更強大的功能與架構的定位

Java Server Page 設計的目標

Java Server Page 技術主要是要幫助讓動態網頁更容易與更簡單的實作，而且要發揮最大的威力與彈性。Java Server Page 設計的目標與理想主要有以下幾點：

寫一次、到處可執行

Java Server Page 技術是完全的與平台無關的設計，包含它的動態網頁與底層的 Server 元件設計。你可以以任何作業平台撰寫動態網頁並且在任何支援 Java Server Page 的 Server 系統上執行，當然，它是與瀏覽器端是獨立而沒有任何關係的（有關係的只會是你輸出的 HTML 檔©）。

你也可以建立 Server 的元件，並在 Java Server Page 中使用它，目前主要是 JavaBean 與 Java Servlet，而它們也都是 Java 保證下的跨平台規格。

加強元件能力

目前 Server-Side 的動態網頁主要所欠缺的就是完整而強大的元件模型，Java Server Page 技術以 Java 的元件模型 JavaBean 來加強 Java Server Page 的元件使用能力。

這省去你開發的時間，因為 JavaBean 與 Servlet 是早已存在的技術。而因為元件處理掉多數的工作，這可以幫助你將網頁設計與 Business logic 程式的撰寫分開，有助於你的產能與維護的便利性，尤其是 Server 們都是異質作業平台時。

作業 Java Enterprise 平台的門戶

Java Server Page 是 Java Enterprise 平台（Java 專注於企業應用方面的平與技術，如 JDBC、JNDI、JINI 等等）的高度整合部份。你可以利用 Java 的 Enterprise API 開發企業的各種需求系統，而 Java Server Page 正適合用來做為這些技術的前端。當你想要升級你的程式時，你可以升級你的元件與動態網頁部份，這些都存在於你的 Server，所以你只需要修改 Server 的資料，所有使用者端的展現層就會跟著改變。

更容易建立動態網頁

Java Server Page 是 HTML 檔，它混合標準 HTML 語法與 Java Server Page 的自己的語法標籤，並以 Java 作為其 Scripting 語言。一個 JSP 檔通常是以 .jsp 為副檔名，並使用 Server 上的可重用元件，如此簡單。

目前 Java Server Page 的元件基礎是 JavaBean 或是 Servlet，至於 Enterprise JavaBean 則在未來的版本會加入支援。

JSP 架構與大型商務

因此對於更大型的網頁系統而言 JSP 有以下的優點：

更明確的開發角色定位

Java Server Page 設計時，考量到大型商務網頁開發時的團隊合作時的分工方式，而將其定立為以下的角色，以下是一般的情況：

網站或網頁設計者

- HTML 3.2 以上（含）的知識背景
- Java 語言的知識背景
- 元件設計的知識背景
- 不假設動態網頁內容產生的知識背景
- 不假設 Scripting 語言的知識背景

元件設計師

- Java 語言的知識背景
- JavaBean 的寫作經驗
- 不假設動態網頁內容產生的知識背景
- HTML 3.2 以上（含）的知識背景
- 不假設 Scripting 語言的知識背景

動態網頁或瀏覽器 Scripting 語言寫作者

- 不假設 Java 語言的知識背景
- Microsoft 元件或物件的經驗
- Active Server Pages 或 NetFusion 的經驗
- HTML 3.2 以上（含）的知識背景
- Scripting 語言的知識背景

Java Servlet 設計師

- Java 語言的知識背景
- Servlet 的寫作經驗
- Page Compilation 的知識經驗
- HTML 3.2 以上（含）的知識背景
- Scripting 語言的知識背景

編譯式的 Server-Side Script ?

雖然說，實際的運作法式取決於你的 JSP 引擎的運作法式，不過一般來說，JSP 為編譯式而非 Microsoft ASP-Active Server Page 所採用的直譯方式。

JSP 實作與運作方式

新的 JSP 規格中，並不強制 JSP 裡的 Scriptlet 也就是程式碼的部份語法必須要是 Java 語法，以筆者目前所知，有幾個 Servlet 利用 EMAC-Script 或是 WebL 這些架構在 Java 上面的另類直譯語言來做為 Scriptlet 的語法，不過它們仍是編譯成 Servlet 的模式來執行，這主要是方便與原有的 Servlet 引擎配合。換句話說，你可以做出以其它語言，其它平台，完全與 Java 或 Servlet 無關的 JSP 實作？

筆者認為這是肯定的，但是實情是十分困難，因為 JSP 的複雜架構必需由 JavaBean 來達成，這種架構在其它語言想要整合是十分困難或是缺乏經濟效益的，因此筆者不太認為會有超出 Java 形式存在的 JSP，最多是以非 Servlet 的形式實作，但仍然是架構在 Java 上面。

筆者可以很明確的告訴你，JSP 的優勢在於 Java 背後所存在的強力 API 與企業支援而非 JSP 語法本身，JSP 語法本身与其它 Server-Side Script 比較，並無特出之處，也並非最優秀。

編譯成 Servlet 的好處

為何當初在設計 JSP 時會是編譯式，而非直譯式的呢？其實只是純粹容易實作，而且 JSP 本身變成 Servlet，也有助於原本的 Servlet 引擎可以直接管理 JSP 的執行，而 JSP 引擎的實作就可以放心的把這方面的任務交給 Servlet 引擎。

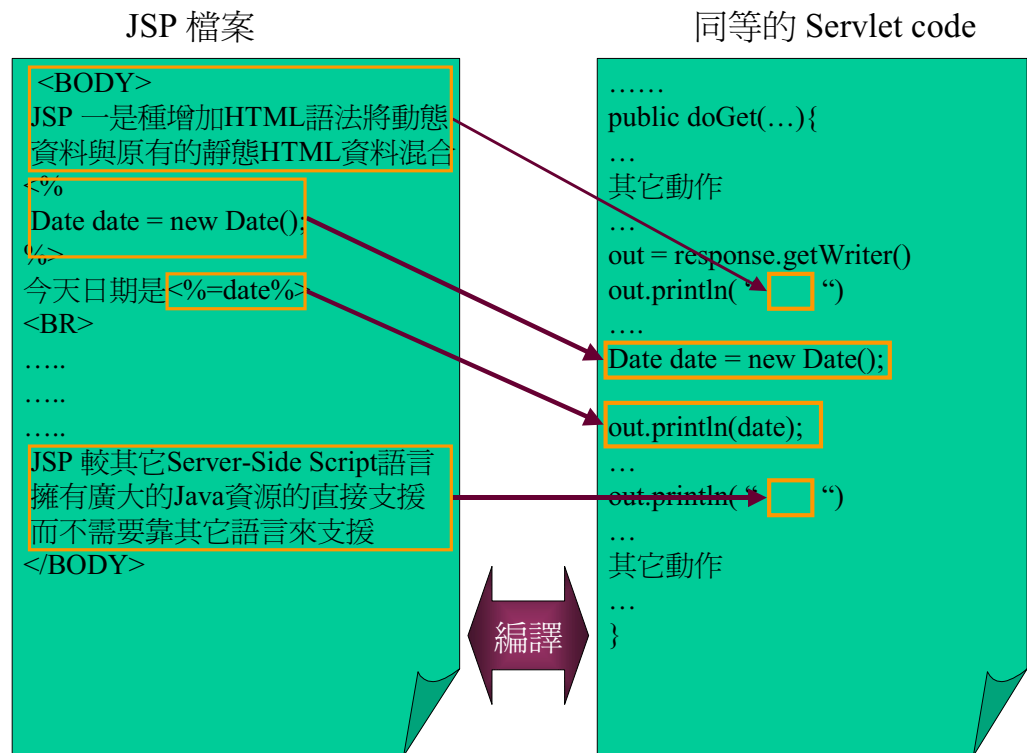
運作概念

JSP 的運作模式如下並請參考下面的圖片

JSP 檔案 → Servlet 碼(.java) → Java 執行碼 → 以 Servlet 形式存在

其實除去轉譯與編譯的部份，JSP 的運作其實與 Servlet 的差異便不太大了。你會覺得 JSP 只是幫助你方便地寫 Servlet。

其實在最早先的 JSP 原型提出，也就是 Java Compile Page 的時代，JSP 原型看起來就像是方便「與 HTML 混寫」、並具「自動編譯」的變型 Servlet，然而後期的 JSP 規格改變了這個關點。我們在後面會再慢慢談到這點。

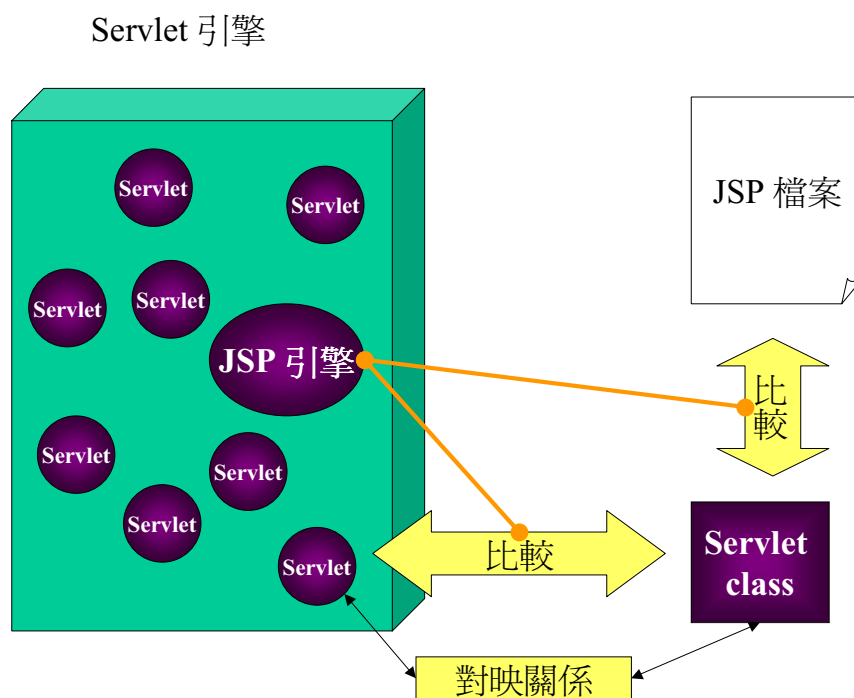


圖片 2、編譯 JSP 的概念

再論運作概念

基本上 JSP 引擎，通常都是架構在 Servlet 引擎上面，常見是以 Servlet 的形式存在，利用 Servlet 引擎的副檔名 (.jsp) 或是 Content Type 對映，把 JSP 檔案轉譯成 Servlet 的原始碼，再呼叫 Java 編譯器，編譯成 Servlet，這也就是為什麼第一次的 JSP 執行較花時間。因此除了在第一次編譯，之後的 JSP 速度與 Servlet 相同，這也是 JSP 的優點之一。

至於 JSP 修改的立即更新架構繼承於 Servlet 的動態更新，首先 JSP 引擎會檢查編譯好的 JSP（已經變 Servlet 了）是否比原始的 JSP 檔還新，如果不是，表示 JSP 檔有更新，就會重新執行轉譯與編譯的動作。剩下的部份，就與 Servlet 的動態更新相同，請往前面相關章節參考。



圖片 3、JSP 轉譯成 Servlet 概念圖

運作系統

運作平台

JSP 引擎實作

名稱	JSP 版本	網址	註解
JSP Reference	1.0 PD*	http://java.sun.com/products/jsp	JavaSoft 官方 JSP 實作參考
JavaServer Web Development Kit	1.0	http://java.sun.com/products/jsp	JavaSoft 官方 JSP 實作參考與 JSDK 合併後的總稱
GNU JSP	0.91	http://www.klomp.org/gnujsp	GPL
Java Web Server 2.0 beta	1.0PD	http://java.sun.com/products/jws	JavaSoft 的 Web Server 系統

PolyJSP	0.92	http://www.plenix.org/polyjsp	以 ML/XSL 為基礎， 可以延伸成支援任何 語言，目前支援有 EcmaScript、WebL *
Resin	0.92 1.0PD	http://www.caucho.com	支援 JavaScript
SJSP	0.92	http://web.telecom.cz/sator/jsp/index.html	
zJSP	0.91	http://www.zachary.com/creemer/zjsp.html	
JRUN	0.92 1.0	http://www.livesoftware.com	LiveSoftware 是 Servlet 主要的技術領導
Orion	0.92, 1.0PD	http://orion.evermind.net	Web server 內含 Servlet 與 JSP 支援
ServletExec	0.91	http://www.newatlanta.com	支援許多 Web Server

摘要：

* PD-Public Draft，公開測試版

** WebL- <http://www.research.digital.com/SRC/WebL>

Application Server

Application Server 是指佈署 Web Application 的中介軟體的系統，用以連接 Client 與 Server 間，下面這個網址是專門討論 Java 的 Server-side 應用技術的網站，其中有簡單的各家 Application Server 的資料

<http://www.interpasnet.com/JSS/textes/jsp2.htm>

下面是這些資料是各家 Application Server 支援 Java Server Page 的統整結果，以英文字順序排：

產品名稱	支援 JSP 版本
BEA Weblogic 4.0	無
BEA Weblogic 4.5 (ex 4.1) http://www.beasys.com/	1.0
Sapphire/Web 6.0	無
Sapphire/Web 6.1	1.0
Gemstone/J 3.0 http://www.gemstone.com/	0.91 (ServletExec)
HahtSite 4.0 http://www.haht.com/	無
IBM WebSphere 1.0	0.91
IBM WebSphere 2.02 http://www.software.ibm.com/webservers	0.91
IBM WebSphere 3.0 (未推出) http://www.ibm.com/	1.0、0.91

Inprise Application Server 1.0	無
Inprise Application Server 4.0 (未推出) http://www.inprise.com/	1.0
Netscape Application Server 2.1	0.92
Novera jBusiness 4.0 http://www.novera.com/	???
Oracle Application Server 4.08 http://www.oracle.com/	無
Persistence PowerTier for EJB http://www.persistence.com/	???
Progress Apptivity 3.0 http://www.progress.com/	無
SilverStream 2.5 http://www.silverstream.com/	???
Sun NetDynamics 5.0 http://www.netdynamics.com/	無
Sybase Enterprise App Server 3.0 http://www.sybase.com/	???

Web Server

Microsoft IIS

IIS 不支援 Servlet 功能，不過可以配合的有 IBM 的 WebSphere, LiveSoftware 的 JRun 與 New Atlanta 的 ServletExec 這些產品主要都是提供 IIS 4.0 嵌入 Servlet 引擎的技術。

Apache

IBM 的 WebSphere, LiveSoftware 的 JRun 都可以支援，而 M 的 WebSphere 其產品本身就含有 Apache 改來的 Web Server 配合執行。

JSP 與其它 Scripting 語言

JSP 是“Java” Server Page，而 0.92 版本後，就定義了“支援 Sscripting 語言”的指令語法，所以實際上 JSP 的程式語法是可以為其它語言的，下面是兩個其它語言的實作

名稱	JSP	支援語言	網址
JRun 1.3.2	1.0	Java、WebL*	http://www.livesoftware.com
PolyJsp	0.92	Java EcmaScript	http://www.plenix.org/polyjsp
Resin	0.92	Java、 compiled JavaScript	http://www.caucho.com

摘要：

* WebL 以 Java VM 為基礎的直譯式語言

<http://www.research.digital.com/SRC/WebL/>

其它動態網頁的技術

JSP vs CGI

不論ASP或JSP（還有很多其它的產品與它的功能類似）都是直接與Web server做銜接（透過動態連結檔DDL或是其它相同的動態連接的技術），因此不需要透過花時間又花資源的標準輸出輸入介面來執行程式再將資料傳回，因此傳統的CGI比這類技術的執行效能差很多。

例如，每當某個 Client 要執行你的 CGI 程式時，Server 就要建立一個新的 Process 來執行這個程式，而且如果你用像 Perl 之類的直譯式語言，Server 還必需要花時間去執行另一個 Perl 直譯器，在作業系統中大量的 Process 產生與清除是十分花時間與耗資源的，這個情況在每天有數仟人次登入的 Web 系統尤然。

另外CGI的Web程式也難以維護它們的架構，因為CGI沒有自己專有的技術可以將一群CGI程式組成一個Web程式，並且分享之間的資料。由於條JSP與ASP這類的技術，它們的系統與Web Server同時一起跑，而且是一直執行，直到Web Server關閉為止，因此他們很容易地撰寫行程管理這種維護連線資訊的系統，而CGI則不然，因此CGI不適合發展複雜Web程式。

Active Server Page 與其它內嵌 Scripting 的動態網頁

Server-side 的程式設計過去來說都十分困難，透過 CGI 來撰寫需要一些古怪深奧的程設知識加上 Perl 或 C 之類語言的熟悉。但是 HTML 檔內嵌 Scripting 語言與其它的延伸標籤使得這些情況改變，使得沒有堅深程式語言基礎的人，也可以輕易地撰寫 Server-side 的動態網頁，甚至是完全沒有程式語言學習經驗的人，也可以透過這些延伸標籤做出動態網頁。一個 Active Server Page 檔案是 HTML 語法加上一些額外的程式碼，JSP 與 ASP 的技術概念相似，正確地說，JSP 有許多都是向 ASP “借來的” ☺。

特色	Active Server Page	Java Server Page
可以在各種 Server 上使用	否，主要是 IIS（市場佔有率 23%）	可以（Apache、Netscape、IIS，市場佔有率 85%以上）
可應用在多種作業平台	否，只能用在 Microsoft Windows 系統	可以（Windows 系列平台、Sun Solaris 平台、Mac OS、Linux

自行定義延伸標籤	否	等) 可以 (1.1 才是正式規格)
跨平台的可重用元件	曾	可以，使用 Java Bean 元件架構
開放規格與各家合作	否	是 (包含 IBM&WebLogic、Oracle、Netscape)
可以將資料與程式邏輯分開	可；使用 COM 元件	否；使用 JavaBean 技術
Scripting 語言支援	VBScript、JScript	Java、JavaScript 等，還會有新的加入
可以建立大型 Web 程式	可	可
與傳統資料庫連結	可，ODBC 或 COM 元件	可，利用 JDBC API 或 JavaBean
副檔名	.asp	.jsp
行程管理	有	有
執行模式	編成 p-code	以 Java byte code 存在
自定函式	可	可
定位	MicroSoft 產品	Java 規格書

我們稱之為 Scripting 語言，Server 會它與 HTML 的標籤分開處理。但是光是內嵌 Scripting 語言與其它的延伸標籤這樣是不夠的，Server-side 動態網頁可以內嵌 Scripting 語言不光只是為了方便 Web 程式的撰寫，更重要的是要將 Web 程式的邏輯 (Business logic) 與 Web 程式的展現層，而 Server-side 動態網頁正是負責展現層的前端部份，而負責處理 Web 程式的邏輯部份，會以一種元件模型呈現，例如 JSP 以 JavaBean 而 ASP 以 ActiveX，不具有使用元件支援的 Server-side 動態網頁技術 (如 PHP) 未來將難以有競爭力。

PENDDING..more.

JSP 的未來與展望

JSP 技術尚未完整地規畫出來，到本書所寫為止，我們主要的平台已經開始轉移到 JSP 1.0 版本，這個版本加入了一些新的特色 (特別是加入 XML 語法與輸出緩衝以解決重導的錯誤等)，而有些 0.92 的功能被取消 (為了將它納入“延伸標籤庫”的規格中，而 1.0 又為了時效上的目題，而先將它移出規格中；這也使得有些人仍然不願用 1.0)

走向 XML 相容語法

XML 愈來愈熱門，不久將來 XML 將會成為主流，因此 JSP 1.0 開始制定支援 XML 的語法，以期與 XML 相容，然而 1.0 中 XML 語法支援不是強制實作的，但是未來的 JSP 1.1 則是強制實作（還有“延伸標籤庫”規格也是）。

JSP 1.1 Public Draft

總覽

JSP 1.1 與 JSP 1.0 是同時開發，但是 JSP 1.1 需要新的 Servlet API 支援，所以未來的 JSP 1.1 會等到新的 Servlet API 公佈後再推出。

新執 Tag Extension 機制

除了 JSP 本身的標準語法標籤外，JSP 1.1 打算提供具有可攜性的自定語法標籤機制，使用者可以自行定義自己的語法標籤，而且可以在任何支援 JSP 1.1 規格的引擎上執行。

新增的特色

JSP 1.1 的規格會新增

- 對 Web 程式的支援加強，如 Web 程式的全域狀態控制
- Web 程式的全域事件處理器
- 另外會與 J2EE 1.0 一起配合起來整合的功能，如 Web 程式包裝的資訊與程式佈署的資訊

J2EE 1.0 的支援

JSP 1.1 要與 J2EE 1.0 的下面技術整合

1. Security.
2. Transactions.
3. Session state.

與 1.0 版的差異

PENDING

Java 2 Enterprise Edition

JSP 1.1 將會成為 JavaSoft 的重要 Java 平台規格—Java 2 Enterprise Edition 1.0 版中的必要支援規格。雖然目前 Enterprise Java 的許多產品仍未為 JSP 設計，但是不久的未來 Enterprise Java 架構將會十分仰賴 JSP 的技術。

參考 JavaOne 99 中，JavaSoft 對 JSP 規格與 J2EE 的解說

Java™ 2 Platform, Enterprise Edition

☐ JavaServer Pages™ Technology

<http://industry.java.sun.com/javaone/99/event/0,1768,678,00.html>

Java Server Page 第一步

取得執行系統

JSWDK 1.0EA

在本書主文中，主要是介紹 JavaServer Web Development Kit - JSWDK，因為目前較完整實作 Java Server Page 1.0 規格的就只有它了。

注意：

JSWDK 1.0EA 仍有些許的臭蟲與未實作規格書的部份

目前的版本為 1.0 EarlyAccess 1.0（1999/7/8），可以在下面的網址中取得

<http://developer.java.sun.com/developer/earlyAccess/jsp/index.html>

或是

ftp://ftp.ncnu.edu.tw/JavaDownload/JavaX/Servlet/JSWDK/1.0ea/jswdk1_0ea-win.zip

WebSphere

本章範例

本會採用的範例是下面的壓縮包

`/src/jsp_samples/FirstJSP.zip`

JWDK 1.0EA

使用 JSWDK 請把它解開放到下面的目錄

```
你的JSWDK 安裝路徑\webpages\
```

WebSphere

而使用 JSWDK 請把它解開放到下面的目錄

```
C:\WebSphere\AppServer\samples\
```

使用 JSWDK 執行 JSP

以 JSWDK 執行 JSP

前面的 Servlet 部份我們已經提到 JSWDK 可以執行 Servlet，而它其實也包含了 JSP 1.0 Public Draft 的規格支援。

現在我以 Windows 系統為主說明，其它作業平台應該也是差不多的（JSWDK 是純 Java 寫成）。

執行 JSWDK Server

以 startserver.bat 執行 JSWDK Server，你應該會看到以下的畫面

```
JavaServer(tm) Web Dev Kit Version 1.0 EA  
Loaded configuration from file:C:\Java\app\jswdk-1.0-  
ea\default.cfg  
endpoint created: :8080
```

接下來你便可以在瀏覽器上打入 <http://localhost:8080>，進入的網頁就是

```
你的JSWDK 安裝路徑\webpages\index.html
```

設定 Server

JSWDK Server 的設定檔在

```
你的JSWDK 安裝路徑\default.cfg
```

中，將它打開，你可以看到一些參數的設定，其中下面幾個參數是較為重要的

server.port

Web Server 的通訊埠（port）預設是 8080，所以你的瀏覽器需要在 hostname 後打:8080 以指定它的通訊埠，改變這個值，當然你的瀏覽器打的 URL 也要跟著改變。

由於一個通訊埠只能系統佔有，所以如有必要，你可以修改避免衝突；同時，這也方便你同時執行數個 JSWDK，只要它們是不同的通訊埠，同時數個是不會有問題的。

STARTSERVER.BAT 這個啟動 Server 的檔案也可以在執行時，加上“-p 通訊埠”的參數來指定執行的通訊埠，如下：

```
STARTSERVER -P 80
```

Server.docbas

便是你的 JSWDK Server 網頁的根目錄，預設值便是相對映於

```
你的JSWDK 安裝路徑\webpage
```

server.workdir

Server.workdir 則是 JSP 引擎產生 Servlet 程式碼與編譯完成的執行碼放置的根目錄，預設值便是相對映於

```
你的JSWDK 安裝路徑\work
```

server.webapp.examples.mapping

JSWDK 也提供了新增“自定對映 URL”的功能，設定一樣是在 JSWDK 的設定檔 default.cfg 中，你可以在預設的設定檔中發現這兩段

```
server.webapp.examples.mapping=/examples  
server.webapp.examples.docbase=examples
```

server.webapp.examples.mapping=/examples 是指你要將

```
http://localhost:8080/examples/
```

對映到

```
你的JSWDK 安裝路徑\新的路徑\examples
```

而這樣同時也會造成新的 Servlet 對映，也就是

```
http://localhost:8080/examples/servlet/myservlet
```

會對映到


```
你的JSDK 安裝路徑\examples\WEB-INF\servlets\myservlet.class
```

myservlet.class 是一個編譯好的 servlet 程式；以上例來說，Servlet 引擎就會將 **myservlet.class** 這個類別碼檔載入 Java VM 中執行，同樣的，這是 Servlet 引擎的責任，你不需要去了解它是怎麼做到的，只需要知道該怎麼放置它便可。而 JSP 檔也是一樣，例如

```
Http://localhost:8080/examples/jsp/cal/call.jsp
```

會對映到

```
你的JSDK 安裝路徑\examples\jsp\cal\call.jsp
```

JSDK 執行 JSP 出錯？

當你裝好 JSDK 並執行 Server 後，測試其中的 JSP Page，你可能會遇到下面的錯誤，而 JSP 完全不能執行：

```
java.lang.NoClassDefFoundError: sun/tools/javac/Main
    at com.sun.jsp.compiler.Main.compile(Main.java:194)
    at
com.sun.jsp.runtime.JspLoader.loadJSP(JspLoader.java:114)
    at
com.sun.jsp.runtime.JspServlet$JspServletWrapper.loadIfNecessary
(JspServlet.
java:69)
    at
com.sun.jsp.runtime.JspServlet$JspServletWrapper.service(JspServ
let.java:77)
    at
com.sun.jsp.runtime.JspServlet.serviceJspFile(JspServlet.java:12
5)
    at
com.sun.jsp.runtime.JspServlet.service(JspServlet.java:152)
    at
javax.servlet.http.HttpServlet.service(HttpServlet.java:840)
```

這個問題是出在 JSP 執行需要編譯 Java 程式，而使用到 JDK 裡的編譯器，但是 JDK 1.2 裡的編譯器部份的程式被抽離 Java VM 的主要類別庫包裹裡，所以會有以上找不到 sun.tools.javac.Main，解決的方法是修改 STARTSERVER.BAT 裡的 CLASSPATH

```
set JAVA_HOME=c:\jdk1.2
.....
set CLASSPATH=%JAVA_HOME%\lib\tools.jar.....
```

使用 WebSphere 執行 JSP

保留 JSP 轉譯成 Servlet 的原始碼

本書討論的 JSP 實作，都是以 Java + Servlet 的基礎，如果你已經有 Servlet 的基礎，想要進一步了解 Java Server Page 在搞些什麼鬼，那麼最好的方法，就是觀察 JSP 轉譯成 Servlet 後的程式碼，大部份的 JSP 引擎都會將之保留放在特定的目錄中：

JSWDK 1.0 EA

WebSphere

由於 IBM 的 WebSphere 把

```
http://yourhost:port/IBMWebAS/samples
```

對映到下面這個目錄

```
C:\WebSphere\AppServer\samples\
```

所以，當你在瀏覽器上打入下面的 URL

```
http://yourhost:port/IBMWebAS/samples/JSP_Date_Demo.jsp
```

而它對映到的 JSP 網頁檔案就是

```
C:\WebSphere\AppServer\samples\JSP_Date_Demo.jsp
```

而你產生的 JSP 程式碼會在（假設你把 WebSphere 裝在 C:\WebSphere）

```
C:\WebSphere\AppServer\servlets\pagecompile\_IBMWebAS\samples\_JSP__Date__Demo_xjsp.java
```

再舉個例子，例如

```
C:\WebSphere\AppServer\samples\tets.jsp
```

產生的 JSP 程式碼會是在

```
C:\WebSphere\AppServer\servlets\pagecompile\_IBMWebAS\samples\_test_xjsp.java
```

第一個 JSP 網頁

顯示時間

Java Server Page 的語法乃是部份抄襲 ASP 得來的，像是 `<% 程式碼... %>` 及 `<% 參數宣告="參數值" %>` 就跟 ASP 一樣，在 JSP1.0 規格書裡面，我們稱之為“scriptlet”，Scriptlet 是規格中放入 Scripting 語法的地方，在這裡面，你可以做任何一般 Java 程式可以做的動作，當然，這是 JSP 的 Scripting 語言是以 Java 為其語法的。在這個 Scriptlet 中 `out` 變數也就是我們在 Servlet 中 `HttpServletResponse.getWriter()` 所得到的 HTTP 輸出資料流物件，它是 `javax.servlet.PrintWriter` 的物件。因此此時的 Scriptlet 內必須是 Java 程式語言的語法：

更明確地說是
`javax.servlet.jsp`
`.JspWriter` 這點
我們後面再談

原始碼 2、JSP_Date_Demo.jsp—簡單的 JSP 例子 [JSP0.91](#) [JSP0.92](#) [JSP1.0](#)

```
<HTML>
<HEAD>
  <TITLE>JSP Date Demo Page</TITLE>
</HEAD>
<BODY>

<H1>JSP Date Demo Page</H1>

The current date is <%
java.util.Date date = new java.util.Date();
out.println(date);
%>.
<br>expression syle <%=date%>
</BODY>
</HTML>
```

而 `<%=.....%>` 也是從 ASP 學來的，會將 `=` 後面接著的是 Scripting 語言的變數或敘述，會變成字串產生出來。

那麼在 JSP 引擎上所產生的 Java 原始碼，大致上就會變成這個樣子

同樣的，這只是可能的實作情況，實際轉換結果並不一定如些

```
import java.util.Date;
import javax.servlet.*;
import javax.servlet.http.*;
...省略....
...省略....
...省略....
public class date extends HttpServlet{
  ...省略....
  Response.setContentType("text/html");
  PrintWriter out = Response.getWriter();
  Throwable exception =
(Throwable)request.getAttribute("ls.jsp.error");
  out.println("<HTML>\n<HEAD>\n<TITLE>JSP date page</TITLE>");
  out.println("<H1>JSP date page</H1>");
  out.print("The current date is ");
  java.util.Date date = new Date();
  out.print( date );
```

```

    out.println("<br>expression style" + date);
    out.println("</BODY>");
    out.println("</HTML>");
    ...省略....
}
...省略....

```

執行它的方法：

JSWDK 1.0EA

`http://localhost:8080/FirstJSP/JSP_Date_Demo.jsp`

WebSphere

`http://yourhost:port/IBMWebAS/samples/FirstJSP/JSP_Date_Demo.jsp`

而執行結果，在瀏覽器上的畫面就是

JSP date page

The current date is Mon May 10 02:12:28 PDT 1999.
expression syle Mon May 10 02:12:28 PDT 1999.

概念上，很簡單，不是嗎？☺

for 迴圈

現在我們來看另一個例子

原始碼 3、JSP_For_Loop_Demo.jsp 簡單的 JSP 例子 [JSP0.91](#) [JSP0.92](#) [JSP1.0](#)

```

<HTML>
<HEAD>
  <TITLE>JSP For Loop Demo Page</TITLE>
</HEAD>
<BODY>
<H1>JSP For Loop Demo Page</H1>
<%for(int i = 0; i < 10; i++){%>

for loop display :<%=i%><br>

<%}%>
</BODY>
</HTML>

```

執行它的方法：

JSWDK 1.0EA

`http://localhost:8080/FirstJSP/JSP_Date_Demo.jsp`

WebSphere

```
http://yourhost:port/IBMWebAS/samples/FirstJSP/JSP_For_Loop_Demo.jsp
```

PENDDING

Java Server Page 基礎概念

Java Server Page 基本概念

名詞定義

API

程式開發介面的縮寫；參考 Application Programming Interface

Application Programming Interface (API)

程式開發介面；定義程式設計者如何寫一個程式與類別、物件的行為和狀態溝通

例如 Java 的核心類別庫

application scope

程式活動領域；也稱之為程式的生命周期

Client

向 Server 取得資源的一方，在本篇未說明都是指瀏覽器的一端。

content type

內文格式；參考 MIME type

GET

HTTP 協定的一種方法，允許使用者取得 Server 上的某些資源，請參考“Servlet 與 HTTP 協定篇”。

header

HTTP 協定訊息格式中的一部份，以 key-value 格式存在，其意義與 HTTP 規格書及 Web Server 本身有關，請參考“Servlet 與 HTTP 協定篇”。

JAR 檔(.jar)

Java Archive 是一種 Java 平台內建的壓縮檔格式，可以用來將程式資源整合在一起，更與 Java 平台功能整合，提供抽象資源（abstract resource）與安全性機制（數位簽證－digital signature）。

JavaBeans

Java 平台的一種跨平台的可重用元件。

Java Database Connectivity (JDBC)

JDBC 是 Java 平台中與業界合作的企業規格，提供與平台無關的資料庫存取，提供 Java 平台的資料庫連結能力。JDBC 提供一組 API 以 SQL 語法存取資料庫。

JavaScript

由 Netscape 開發的一種開放、跨平台的物件（Object-base）程式語言，原本是設計用來控制 Netscape 瀏覽器用，但目前已有許多其它應用，常常與 Java 搞混。

JSP element

JSP 元素；Java Server Page 語法構成元素，共分為三大部分：指令語法、Scripting 語法與動作語法。

MIME

參考 Multi-Purpose Internet Mail Extensions.

Multi-Purpose Internet Mail Extensions (MIME)

一種用來傳遞多媒體格式 e-mail 與訊息的格式

PENDING

POST

HTTP 協定的一種方法，允許使用者取得 Server 上的某些資源，請參考“Servlet 與 HTTP 協定篇”。

PUT

HTTP 協定的一種方法，允許使用者放置某些資源到 Server 上，請參考“Servlet 與 HTTP 協定篇”。

Request

請求；Client 送到 Server 的訊息，通常為請求 Server 上的某項資源（resource），其規格由 HTTP 協定定義，請參考“Servlet 與 HTTP 協定篇”。

request headers

參考 headers.。

request scope

也稱為 page scope，只存在於 Client 與 Server 間的一次連線（request/response）之間。

response

回應；Server 回應 Client 的一部份，其規格由 HTTP 協定定義，請參考“Servlet 與 HTTP 協定篇”。

resource

資源；Client 可以透過 URL 向 Server 取得的資料，可能是某個文件、目錄或是某個時式，Web 程式便是由其構成。

resource abstraction

資源抽象。Java 平台的一種概念，提供與檔案系統無關的資料取得方法，例如 Class 物件的 `getResource()` 方法以及 JSP 的 `application` 變數的 `getResource()` 方法，這在 JSP 的應用上實分重要，我們會在後面的文章詳細介紹。

scope

活動領域；也稱之為生命週期。

server

為 Client 提供各種服務，在本篇中，未說明的話，都是指 Web Server。

session scope

一個 Client 對 Server 取得資源的連續構成的連線間，便是一個 session 的活動領域。

type

型別；指 Java 的類別或是界面

Uniform Resource Identifier

用來定位某種網路協定上的資源文法格式。

PENDDING

“Uniform Resource Locators” (URLs)是 URI 的子集，例如 HTTP 的 URL

<http://www.ncnu.edu.tw/index.html>

PENDDING

URI

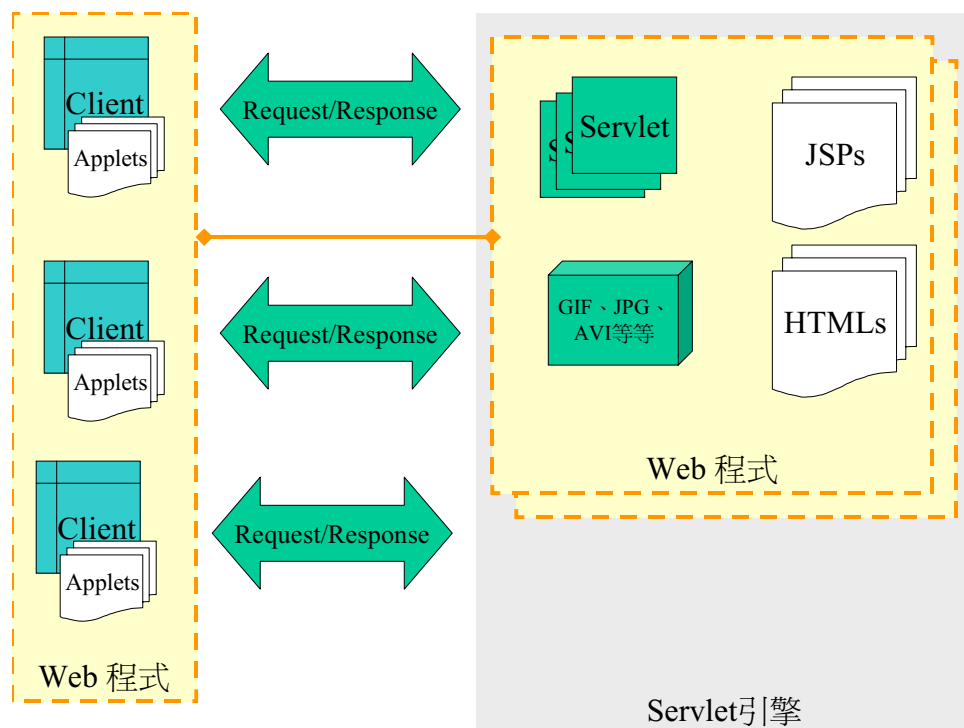
參考 Uniform Resource Identifier

Web 程式概念觀

定義

Web application 的原型可由下面來構成可能很簡單，也可以很複雜，端看你怎麼設計，而我們稱構成 Web 程式的所有部份為“資源”而資源又有分動態的資料（如 HTML 檔）與動態部分（如 Servlet、Applet、ASP）：

- Java 的執行環境於 Server-side 執行，這是所有於 Server 的動態資源部分所存在的平台；
- JSP 網頁與 Servlet 負責處理 Client 端的請求與產生動態的內容；
- Server-side JavaBean 元件－提供行為與狀態；
- JSP/Servlet 引擎負責管理所有 Client 的行程資料；
- 靜態資料如 HTML、DHTML、XHTML、XML 等資源；
- Client-side 上的 Java Applet、JavaBean 元件與其它的 Java 類別檔的組合，雖然是在 Client 上執行，不過仍是由 Server 提供；
- Java 的執行環境於 Clie-side 執行；



PENDDING

Web 程式由這些靜態資料與程式構成，由上圖你可以了解，傳統的 Web 程式開發的問題在於構成元素複雜而且零亂，每個資源間如何分享彼此的資料，如 Servlet 怎麼跟另一個 Servlet 或是 JSP 溝通，JavaBean 如何應用，Web 程式共用的物件如何管理，每個 Client 之間的行程資料如何維護與如何分享等。

PENDDING

為了解決這些問題，各家 Server-side 技術提供商有著各種不同的解決方法，目前正在運行的也不少，但是彼此之間並不相同，替 JSP/Servlet 的規格能更嚴謹，更能達到跨越不同平台的規格，JSP 規格書企圖定義一些觀念與實作機制來幫助 Web 程式開發上的問題，而且提供可攜性的解進方案。

Web 程式與 JSP

Web 程式，如同我們前面定義的，是由一群資源構成，而這些資料全部都可以透過 URL 來取得。這些資料包含了 JSP 網頁、Java Servlet、Java Applet 與靜態資料還有其它 Java 技術構成的資源。

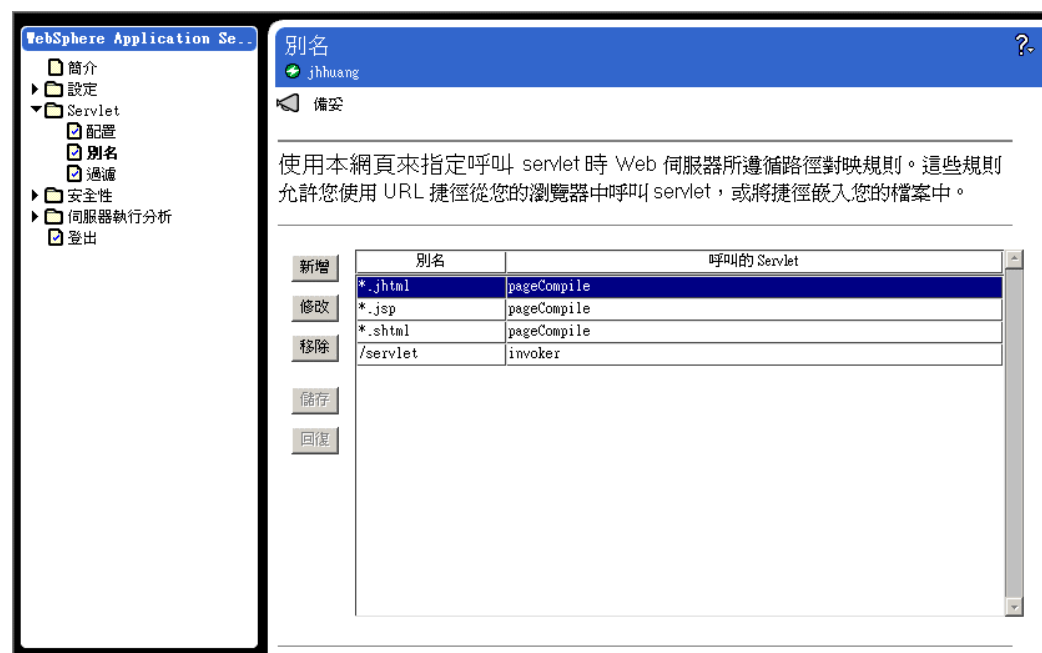
對 Web 而言，所謂的程式，包含了資料存取還有資源間的分享資料。當 JSP 網頁編譯成 Servlet 後，Java Server Page 1.0 要求所有的資源可以透過 **application** 這個隱含的變數取得，這方面的支援會依照 JSDK 2.1 的規格

來實作。`application` 變數實其是一個 `ServletContext` 物件，我會在後面的章節中討論它們之間的關係

URL 對映

所謂的 URL 的對映，就是 Server 上提供的機制，允許你將某個 URL 位置對映到某個 Server 上的資源，舉例來說，以 WebSphere 而言，它提供了所謂的別名（alias）來做 Servlet 程式資源的 URL 的映

圖片 4、別名與 URL 對映



以上圖來說，`/servlet` 對映到 `invoker` 這個 servlet 程式，也就是所當你以 URL 指定 WebSphere 上的資料，如下時

```
http://yourhost:port/servlet/HelloWorldServlet
```

WebSphere 系統便會將這個 Client 對 URL 資源的請求交由 `invoker` 這個 Servlet 來請求，而 `invoker` 便會負責載入 WebSphere 的 Servlet 目錄下的 `HelloWorldServlet` 這個 Servlet 程式，並將之執行。

JSP 元素與 relative URL 規格

JSP 元素（包含指令語法、Scripting 語法與動作語法）的語法可能會用到 relative URL 的規格，在 JSDK 2.1 中稱之為“URI 路徑”，它們的格式由 RFC 2369 所制定。雖然 JSDK 2.1 已經支援利用 URI 路徑來取得資源的功能（這是一種資源抽象的概念）。這些對映的機制在 JSP 1.0 規格中並沒有制定，是與你的 JSP 引擎實作有關，在未來的 JSP 規格中，應該會加入各 JSP 引擎都可攜的設定方式，做為佈署 Web 程式之用。現在我們以 JSDK 1.0 EA 中的 URL 對映機制如何與 Web 程式開發配合來做說明

URL 對映機制與 Web 程式－JSWDK 1.0EA

我面在“Java Server Page 第一步”說明了“自定對映 URL”的功能，其設定檔中的

```
server.webapp.examples.mapping=/examples  
server.webapp.examples.docbase=examples
```

告訴我們會將 Servlet 的 URL

```
http://localhost:8080/examples/
```

也就是 URI 爲“/example”對映到

```
你的JSWDK 安裝路徑\新的路徑\examples
```

而這樣同時也會造成新的 Servlet 對映，也就是

```
http://localhost:8080/examples/servlet/myservlet
```

會對映到

```
你的JSWDK 安裝路徑\examples\WEB-INF\servlets\myservlet.class
```

而 JSP 的 bean 所放置的位置則會對映到

```
你的JSWDK 安裝路徑\examples\WEB-INF\jsp\beans
```

這時，你在 JSP 中利用 **jsp:useBean** 語法建立 bean 時，JSP 尋找 bean 類別的載入 CLASSPATH 便是這裡。

以 Web 程式的角度來看，便是把整個

```
http://localhost:8080/examples/
```

當成一個程式。例如 JSWDK 的範例 Calendar 例子，便算是它的一個子程式，你可以由以下進入

```
http://localhost:8080/examples/jsp/cal/login.html
```

我們把它當成一個 Web 程式的話，那麼它的 Web 程式的佈署就是

靜態資源 URI 路徑（絕對路徑）：

```
/jsp/cal/login.html
```

對映到

```
你的JSWDK 安裝路徑\新的路徑\examples\jsp\cal\login.html
```

動態資源 URI 路徑（絕對路徑）：

```
/jsp/cal/cal1.jsp  
/jsp/cal/cal2.jsp
```

對映到

```
你的JSDK 安裝路徑\examples\jsp\cal\cal1.jsp  
你的JSDK 安裝路徑\examples\jsp\cal\cal1.jsp
```

在 `jsp:include` 與 `jsp:forward` 標準動作語法裡，我們便會用到這些概念，例如 JSDK 的 `include` 範例

```
http://localhost:8080/examples/jsp/include/include.jsp
```

要引入另一個 JSP 資源時，利用下面的語法

```
<%@ include file="/jsp/include/foo.jsp" %>
```

`file` 屬性利用的就是 URI 路徑的資源定位方法，使它對映到整個 Web 程式

```
http://localhost:8080/examples/
```

的其中一個 JSP 資源

```
http://localhost:8080/examples/jsp/include/foo.jsp
```

PENDING: a better descript

需要用到 URL 對映的 JSP 元素有

- `page` 指令語法的 `errorPage` 屬性
- `include` 指令語法的 `file` 屬性
- `jsp:include` 標準動作語法的 `page` 屬性
- `jsp:forward` 標準動作語法的 `page` 屬性

裡用到

bean 元件全名：

```
cal.Entries  
cal.Entry  
cal.JspCalendar  
cal.TableBean
```

對映到

```
你的JSDK 安裝路徑\examples\WEB-INF\jsp\beans\cal\Entries.class  
你的JSDK 安裝路徑\examples\WEB-INF\jsp\beans\cal\Entry.class
```



```
你的JSDK 安裝路徑\examples\WEB-INF\jsp\beans\cal\
JspCalendar.class
你的JSDK 安裝路徑\examples\WEB-INF\jsp\beans\cal\TableBean.class
```

需要用到 bean 元件資源的語法為 **jsp:useBean** 標準動作語法

資源抽象與 Java Server Page

在 JSP 1.0 與 Servlet 2.1 中，HTTP 協定程式是由 URL 對映到各個資源將之整合，這群 URL 是由 JSP 引擎（或是 Servlet 引擎）串連在一起的。每個 URL 的動態資源（包含 JSP 與 Servlet）都有一個唯一的實體，利用 **javax.servlet.ServletContext** 包裝它。Servlet 與 JSP 網頁在同一個可以透過 **ServletContext** 的 **setAttribute()** 與 **getAttribute()** 方法與 **removeAttribute()** 來分享物件，有關這些物件與方法的使用討論請參考“Servlet 與 Servlet 間的溝通篇”。

JSP 透過隱含變數 **application** 可以取得任何的資源，例如

```
<HTML>
<HEAD>
<TITLE>Hello World</TITLE>
</HEAD>
<BODY>
<h1>Hello World</h1>
***include file start here***<br>
<%
    InputStream in = application.getResourceAsStream("/test.txt");

    if(in != null){
        byte[] ab = new byte[1024];
        int len = in.read(ab);
        do{
            out.print(new String(ab, 0, len));
            len = in.read(ab);
        }while(len >0);
    } else {
        out.println("Resource doesn't exist!!");
    }
}>
***include file end here***<br>
</BODY>
</HTML>
```

PENDING: more description

或是透過 **jsp:include** 與 **jsp:forward** 標準動作語法將這些用法包裝起來，自動幫你處理。

JSP 1.0 的規格中，並沒有提供可攜性的機制做這些資源的對映的管理，這將會是未來版本的目標

PENDING: more description

在 Web 程式中，JSP 的資源反映在 **application** 這個隱含的物件中，而它影響了 JSP 幾個元素的語義。

- **page** 指令語法的 **errorPage** 屬性
- **include** 指令語法的 **file** 屬性
- **jsp:include** 標準動作語法的 **page** 屬性
- **jsp:forward** 標準動作語法的 **page** 屬性

PENDDING: more description

行程管理—Session Management

這使得 HTTP 是一個“無狀態”（Stateless）的協定，也就是任意的連線間，對 HTTP Server 而言，都是一樣的，也就是各交易通訊間，不會保留連線時的資訊，這點對 HTTP 來說，是優點、也是缺點，HTTP 協定原本的設計上在於分享資源，因此“無狀態”的設計是較有效率也容易實作的方式，因為你不需要去追蹤 Client 的存在，對 Server 而言，可以減少很多系統資源。

與但是對於電子商務而言，常常需要與 Server 間有著複雜的通訊方式，“無狀態”的協定在此時就不適用了。

JSP 透過 **jsp:useBean**、**jsp:useBean** 與 **jsp:getProperty** 標準動作語法，自動幫你的 bean 物件管理行程，供 JSP 與 Servlet 間的分享。

PENDDING:more session concept with JSP

JSP 語法基礎

Java Server Page 與 XML

PENDDING

語法規則

下面幾點是與所有的 JSP 語法有關的觀念。

Java Server Page 指令部份與 HTML 部份

JSP 原始碼中包含了 JSP 元素與 Template data 兩類，Template data 指的是 JSP 引擎不處理的部份，也就是除了 JSP 語法外，JSP 引擎讀解不了的資料的東西，例如 HTML 的內容等，會直接丟到 Client；而 JSP 元素則是 JSP 所有語法的總稱。JSP 元素的共由三個部份構成，指令語法

（directive）元素、Scripting 語法元素與動作（action）元素語法；相關的語法我們會在後面提到。

JSP 元素的語法

大多 JSP 語法中的 Tag 是根據 XML 規格所制定的，JSP 元素基於 XML 語法，他們不是包含，一個啓始 Tag（可能帶有些屬性），還有選擇性的內文部份，而終結於另一個結束的 Tag；而它們也有可能會是 Empty tag（沒有內文的一種 Tag）而帶有屬性（選擇性或非選擇性）。

所以 JSP 的 Tag 會是像下面的 Tag 例子。

```
<mytag attr1="attribute value" ...>
...內文部份
</mytag>
```

或是這樣子的 Empty Tag

```
<mytag attr1="attribute value" .../>
```

JSP Tag 是大小寫有差別的，這是依照 XML 規格的習慣而如此制定的。Scripting 元素與 Directive 元素使用還使用了一種傳統的文法（其實是抄 ASP 的），它們較為方便撰寫（其實是 XML 式的 JSP 語法是後來才有的），它們是以 `<%...%>` 這樣的 Tag 為基本模式變化

所有的 JSP 網頁都對映的 XML 文件的規格，未來 JSP 的規格書會要求 JSP 引擎必需要接受 JSP 網頁與同等意義的 XML 規格文件。

只有 Directive 元素與 Scriptlet 元素都有傳統與 XML 樣式的語法，這些語法並不是要給 JSP page 用的，而是要給同等的 XML 文件使用的

標準動作指令
Standard Action
沒有傳統格式

我們會在介紹這些語法時，同時介紹傳統與 XML 樣式的語法

標籤的開始與結束

JSP 元素有不同的開始與結束的標籤（中間包含的就是內文部份）

```
<mytag>
...內文部份...
</mytag>
```

這些規則也用於傳統的 JSP 語法上，例如

```
<% ...內文部份... %>
```

以`<%`開頭，而以`%>`結尾，中間是它的內文。

所有的開始與結束都是在同一個實際檔案上（如 `hello.jsp` 檔）

空白元素

或是 Empty Tag，乃依循著 XML 的規格，一個元素使用 Empty tag（沒有內文的一種標籤）會與有開始 Tag、空白的內文與結束 Tag 不同。例如 `<mytag/>` 與 `<mytag></mytag>` 兩者並不相同，而且與 XML 定義一樣，JSP 的元素，並沒有像 HTML 的 `
` 這樣的單對標籤存在。

屬性值

也是依循著 XML 的規格，屬性值的表示需要被 quoted

PENDING:

空白

在 HTML 與 XML 中，空白通常是沒有意義，除了一些情況下；例如，XML 檔案中，要以 `<?xml` 開頭，而且不能有空白在前面

摘要：

所謂的空白，包含了

Space – ASCII 20

Tab – ASCII 09

換行 – ASCII 0D 0A

JSP 規格依循 XML 中的定義，也就是說所有在內文中的空白，都是沒有意義的，例如 `<mytag>` 與 `< mytag >` 意義一樣，但是空白在 JSP 引擎的處理與產生輸出後之間，都會被保留下來，並不會刻意清楚掉它。

```
<?xml version="1.0" ?>
<%@ page buffer="8kb" %>
文件的其它部份
```

會被轉化成

```
<?xml version="1.0" ?>
文件的其它部份
```

中間會有一行空白，是因為 `<%.....%>` 雖然被 JSP 引擎處理完後，沒有任何的輸出，但是 `%>` 後面還有一個換行碼（以 Windows 平台而言，就是 0D0A），所以仍然被保留下來，所有變成一行空白在那裡

另一個例子

```
<% response.setContentType("...");
whatever... %><?xml version="1.0" ?>
```

```
<%@ page buffer="8kb" %>
文件的其它部份
```

產生的結果

```
<?xml version="1.0" ?>
<%@ page buffer="8kb" %>
文件的其它部份
```

疑？沒多一行空白空白了？，是因為 `<%.....%>` 被 JSP 引擎處理完後，沒有任何輸出，而且 `%>` 與 `<?xml>` 間沒有任何東西，當然也就沒啥好保留的了。

Quoting、Escape 規則

Quote 是指爲了避免與語法衝突而使用的一種轉換規則。例如，HTML 都是以 `<tag_name>` 爲其標籤語法的基礎，因此讀到”`<`”就會期望與接下來的”`>`”之間的內文，會是一種標籤語法，所以當你要在一般的 HTML 內容中輸出”`<`”怎麼辦呢？HTML 便是以 `<` 來表示一般內容的”`<`”。下面就是 JSP 的 Quoting 原則：

Scripting 元素內文的 Quoting

與 HTML 的標籤語法類似，JSP 以 `<% %>` 爲其 Scripting 元素的語法基礎，所以也需要類似的方式來避免誤判。

因爲 Scripting 元素以”`<%`”爲開始，而以”`%>`”爲結尾，所以當你要在 Scripting 元素的內文表示”`%>`”時怎麼辦呢？例如

```
...前面部份
<%
int i;
out.println("JSP 中的%>是 Scriptlet 的結尾");

%>
...後面部份
```

這個例子被 JSP 處理時，因爲處理到”`JSP 中的%>`”的尾端時，會誤以爲到了 Scriptlet 的尾端，因此這個 Java 碼會變成錯誤的寫法，因爲他會被這樣子轉成 Servlet 碼的一部份

```
...其它的 Servlet 碼
int i;
out.println("JSP 中的
```

爲了避免這樣的誤判，你應以”`%\>`”取代”`%>`”

```
...前面部份
<%
int i;
```

```
out.println("JSP 中的%\>是 Scriptlet 的結尾");  
  
%>  
...後面部份
```

這樣它才會正確地被 JSP 引擎轉換成這樣的 Servlet 程式碼：

```
...其它的 Servlet 碼  
int i;  
out.println("JSP 中的%>是 Scriptlet 的結尾");  
...其它的 Servlet 碼
```

爲了前面 A literal %> is quoted by %\>

Template Data 的 Quoting

同樣的因爲 Scripting 元素以 “<%” 爲開始，所以當你要在 Template Data 中顯示 “%>” 時，你應該要用 “<%” 取代 “<%”，它才不會被誤判成 Scripting 元素的開始；例如

```
<html>  
<body>  
JSP 中的<%是 Scriptlet 的開頭  
<%  
你的 Scriptlet 內容  
%>  
...
```

在執行後輸出時，就會變成

```
<html>  
<body>  
JSP 中的<%是 Scriptlet 的開頭  
...你的 Scriptlet 輸出  
...
```

屬性值內文的 Quoting

屬性值是以單引號 “'” 開始，並也是以單引號 “'” 結束所包含的內文部份，所以屬性值也一樣要避免與 “'” 衝突，而要以 “\’” 取代之。

同時，屬性值要避免衝突的還有：

- 單引號 “'” 取代爲 “\’”
- “%>” 取代爲 “%\>”
- “<%” 取代爲 “<%”

舉以下爲例

```
<%@ page info="測試屬性值四個要避免衝突的字串 - \ " \ ' <% %\>" %>
```

...JSP 檔剩下的部份

此時，這個 JSP 的說明資訊就是

測試屬性值四個要避免衝突的字串－" ' <% %>

JSP 語法的分類

我們在前面提到過 JavaServer Page 技術將資料的展現層（HTML 語法）與管理其動態內容的工作簡化。

因此 JSP 提供延伸標籤（非 HTML 語法標籤）的語法讓撰寫 JSP 者可以在舊有的 HTML 網頁上，嵌入存取動態資料的能力，而這些語法也包含程式的邏輯判斷，但是 JSP 的建議是將這些邏輯判斷部份教給可以重用的元件，與 JSP 檔案分開。

JSP 的語法分為下面兩類

與 Java Server Page 網頁有關的標籤

與 JSP 網頁本身有關的一些設定都是與 JSP 指令語法有關，包含

- page 指令
- include 指令

與 Scripting 語法有關的標籤

0.91、0.92 與 1.0 版間大多相同

Scriptlet

0.91、0.92 與 1.0 均是以下格式

```
<%...Scripting 語言內容...%>
```

Declaration

1.0 以

```
<%!...宣告部份...%>
```

而 0.91 與 0.92 以

```
<SCRIPT runat=server>  
...宣告部份...
```

```
</SCRIPT>
```

Expression

0.91、0.92 與 1.0 均是以下格式

```
<%=...Scripting 語言敘述句...%>
```

與 Java Server Page 的 bean 元件有關的標籤

1.0

- `jsp:useBean`
- `jsp:setProperty`
- `jsp:getProperty`

0.92

下面的幾個語法是用來與 `JavaBean` 溝通，以及取得 `JavaBean` 的 `Property` 用的：

- `USEBEAN` 標籤
- `SETFROMREQUEST` 標籤
- `SETONCREATE` 標籤
- `DISPLAY` 標籤

0.91

- `BEAN` 標籤
- `PARAM` 標籤

延伸標籤語法機制

目前延伸標籤語法機制在 `JSP 1.0` 中為選擇性實作規格，而 `JSP 0.92` 有部份的語法被實作出來。

與 Bean 有關

0.92 中與 `Bean` 有關的延伸標籤包含：

- LOOP 標籤用來顯示含有多項值的 bean 屬性值（如 String[]）

流程控制

0.92 中的流程控制延伸標籤包含

0.92 中的流程控制延伸標籤通常與 bean 配合，用來判斷 bean 的屬性，來決定程式的流程；共有以下幾種：

- INCLUDEIF 標籤用來判斷 bean 屬性為真時，執行它的內文部份；
- EXCLUDEIF 標籤則正好與 INCLUDEIF 相反

JSP 指令類語法—Directive Syntax

指令類語法與 XML

{ }* 是正規語
言(regular
expression)的
表示法

Directives—指令語法主要是用來與 JSP 引擎間溝通用，在 JSP 1.0 中它是以下的格式

```
<%@ 指令名 {屬性="屬性值" }* %>
```

因為 JSP 希望走向 XML 相容的語法，因此 JSP 的指令格式就分為“相容 XML”與“傳統、不相容 XML”的指令格式，以 JSP 指令語法而言，相容 XML 式的語法為

```
<jsp:directive.指令名 { 屬性="屬性值" }* />
```

JSP1.0 的傳統與 XML 樣式指令語法範例

下面是一個用傳統 JSP Page 指令語法的例子

```
<%@ page info="my latest JSP Example V1.0" %>
```

這則是它以 XML 樣式的表示方法，兩者對 JSP 引擎的處理結果完全一樣：

```
<jsp:directive.page info="my latest JSP Example V1.0" />
```

這則是另一個例整

```
<%@ page buffer="none" isThreadSafe="yes"
      errorpage="/oops.jsp" %>
```

對映的 XML 樣式語法

```
<jsp:directive.page buffer="none" isThreadSafe="yes"
      errorpage="/oops.jsp" />
```

XML 型式的 JSP Page 指令

```
<jsp:directive.page info="my latest JSP Example V1.0" />
```

page 指令語法

JSP 1.0 page 指令列表

語法：

```
<%@ page {屬性="屬性值" }* %>
```

XML：

```
<jsp:directive.page { 屬性="屬性值" }* />
```

`{}`* 是正規語言(regular expression)的表示法，上面表示可以在一個 `<%@page ... %>` 指令間有好幾對 `屬性="屬性值"` 的屬性設定。

page 指令列表：

```
page 指令列表 ::= { language=" script 語言" }
                  { extends="承繼類別名" }
                  { import=" Java 引入包裹列表" }
                  { session="true|false" }
                  { buffer="none| 緩衝大小(kilo-byte)" }
                  { autoflush="true| false" }
                  { isthreadsafe="true|false" }
                  { info="網頁資訊" }
                  { errorpage="錯誤處理所指向的 URL" }
                  { iserrorpage="true|false" }
```

language

定義 Scriptlet、expression scriptlet 還有 declarsion 內文中用的 Scripting 語言，還有其包含進遼的

在 JSP 1.0 中只有定義而且要求支援 Java 語言，所以它的值只會是“java”，本書只討論當 language 的屬性值是“java”時的語義

當這個屬性值被設定為“java”時，這些原始碼片段必須符合 Java Programminng Language 的規格。

所有的 Scripting 語言必需提供一些隱含的物件，提供 JSP 開發者可以使用 declariion、scriptlet 還有 expression，這些特別的物件可在....找到

所有的 scripting 語言必需支援 JRE，所有的 scripting 語言必需可以在它的語法中銜接 Java 的物件模型，特別是隱含的變數，JavaBean 的 properties 還有公開的方法。

未來的 JSP 規格可能會定義更多的屬性值，

當 scripting 元性出現後，還利用指令語法下 language 參數，這是嚴重的錯誤

extends

這個參數值是完整的 Java 類別名稱（包含類別與包裹名）而且必須是實作 `javax.servlet.jsp.HttpJspPage` 的類別。

這個屬性值必需被很小心地使用，因為 JSP 引擎提供的 JSP 實作，實際上會預設它繼承某個 `HttpJspPage` 的實作類別，這個子類別是由 JSP 引擎的實作者來撰寫的，他可能會有一些特別的處理，例如 JSWDK 就是實作一個叫“`com.sun.jsp.runtime.HttpJspBase`”的類別，它是實作 `HttpJspPage` 界面的一個類別。

你可以利用它繼承你自己的 `HttpJspPage` 實作類別，做一些特殊應用。但不建議初學者輕易用它。

import

這個屬性值用來描述那些型別允許在 Scripting 語法元素中使用，這個值如同我們在 Java 語言中的 `import` 宣告，可以是類別的全名或是包裹（package）名接著“`.*`”字串，表示定義在這個包裹中所有的公開型別。在 Java 語言中我們是使用分號“`;`”來做為清單的分隔字元，例如

```
import java.io.*;
import java.awt.*;
```

而 JSP 則是使用逗號 “,” 做為清單的分隔字元，因此上面的 Java 語法會相當在 JSP 語法中的

```
<%@page import="java.io.*,java.awt.*"%>
```

import 清單應要被 JSP 網頁實作的物件引入，如些以來，我們便可以在 scripting 環境中使用，這種使用習慣與 Java 程式設計一樣，但是**一定要用逗號 “,” 做為分隔字元，不要誤植為分號 “;”**，這個值目前只有定義當 language 屬性值是 “java 時” 才有意義。另外，以下的語法與上面的意義相同：

```
<%@page import="java.awt.*"%>
<%@page import="java.io.*"%>
```

注意：

如果你真的使用 JSWDK 1.0EA 來測試 **import** 的，那麼你會發覺到根本不管用，這是一個 1.0EA 的臭蟲，語法是 **import** 沒錯，但是 JSWDK 的研發小組誤植為 **imports**，但因為本書是寫給 JSWDK 1.0EA 用的，所以範例會還是會用 **imports** 方便執行。

session

預設為 “true”

指定這個 page 需不需要加入一個 **session**（http 協定的）物件管理 Client 連線的行程資訊。如果為真，那麼所有的隱含變數 “**session**”（它的型別是 **javax.servlet.http.HttpSession**）便可以參考到本頁目前的 session 或建立新的 **session** 物件管理行程資訊。

如果 **false**，那麼這個網頁不加入行程管理，“session” 隱含變數就是 **null**，而任何參考到它的指令都會產生 JSP 引擎的編譯時期的錯誤；而同時這也會造成 JSP Bean 只能的 **scope** 屬性值只能是 “**page**”，若是指定 **page** 以外的值，也會產生 JSP 引擎的編譯時期的錯誤。請參考後面的 “JSP Bean 活動領域與生命週期”。

buffer

其值可能為 “**none**” 或一個數值。

指定處理 JSP 實作中的 **JspWriter**（也就是隱含變數 **out**）處理網頁輸出內容的模式。如果其值為 “**none**”，那麼就不會有緩衝，而所有的輸出都

是直透過 `ServletResponse` 的 `PrintWriter`（經由 `getWriter()` 得到的）。若是一個數值，表示緩衝的大小是指定的這個數字大小，單位為 kilo-byte，那麼輸出的緩衝實際大小至少會大於這個值。

根據 `autoFlush` 屬性值而定，輸出內容超出緩衝的大小，要不是自動被清楚（`autoFlush="ture"`）要不就是會在緩衝滿溢時產生例外（`autoFlush="false"`），實作上，緩衝的預設大小為 8kb 以上。

autoFlush

其值可能為 “false” 或 “true”，預設值是 “true”。

回應上一個 page 指令的屬性值，它是用來指定緩衝填滿時要不要自動清出（`autoFlush="ture"`），要不就是在緩衝滿溢時產生例外（`autoFlush="false"`）

注意：

同時設定 `autoFlush="false"` 和 `buffer="none"` 是不合法的，你會得到一個編譯時期的錯誤。

因為沒有緩衝時，JSP 網頁的 `JspWriter` 本身的行為就是會自動出清。

我們以下面的例子來說明

`/src/jspsample/ErrorMessage_JSP10_Demo.zip`

原始碼 4、BufferOverflowDemo

```
<%@ page autoFlush="false" buffer="1" errorPage="errorpge.jsp"
%>
<HTML>
<HEAD>
  <TITLE>JSP 1.0 Buffer Overflow Demo</TITLE>
</HEAD>

<BODY>
<H1>JSP 1.0 Buffer Overflow Demo</H1>
<a href="BufferOverflowDemo.jsp?loop=5">
  print loop = 5 (wont overflow)</a><br>
<a href="BufferOverflowDemo.jsp?loop=15">
  print loop = 15 (wont overflow)</a><br>
<a href="BufferOverflowDemo.jsp?loop=25">
  print loop = 25 (will overflow)</a><br>
<%
String loop = request.getParameter("loop");
int loop_times = 5;
if(loop != null)
  loop_times = Integer.parseInt(loop);
```

```

        for(int i=0; i < loop_times; i++){
            out.print( i + " =====<br>");
        }
    %>
</BODY>
</HTML>

```

你可以透過 GET 的 Query String 傳遞 loop 參數來決定 “====” 被輸出幾次，這個範例把緩衝設定成 1kb 因此很容易溢滿，當你的迴圈達到一定次數時，輸出就會超出緩衝所能容納的了，因此會丟出例外，本例以一個 `errorPage` 來顯示這個結果

摘要：

關於 `errorPage` 請參考下面的 `errorPage` 屬性值設定的說明，本範例用的錯誤處理網頁與該段文章用的錯誤處理網頁一樣。

isThreadSafe

預設值為 “true” 。

`isThreadSafe` 的意思是告訴 JSP 引擎，你的 JSP 網頁在處理物件間的存取是 thread-safe 與否，我們前面已經探討過 Servlet 的 thread-safe 處理模式，如有不清楚的地方，請參考 “Servlet 進階篇－多執行緒問題” 。

因此我們若是設 “false” 等於是告訴 JSP 引擎你的 JSP 網頁（其實就是 Servlet）應該要實作 `javax.servlet.SingleThreadModel` 這個介面，請自行以 `helloworld.jsp` 前面加下以下這段

```
<%@ page isThreadSafe="false"%>
```

並找出編譯成 Servlet 的原始碼來觀察。

以下是不設定，或 `isThreadSafe="ture"` 時 JSP 網頁轉譯結果（部份）

```

public class helloworld_jsp_1 extends HttpJspBase {

    // begin [file=C:\Java\app\jswdk-1.0-
    ea\examples\helloworld.jsp;from=(7,0);to=(7,62)]
    // end
    ...

```

以下是 `isThreadSafe="false"` 時 JSP 網頁轉譯結果（部份）

```

public class helloworld_jsp_2 extends HttpJspBase
    implements
    SingleThreadModel
{

    // begin [file=C:\Java\app\jswdk-1.0-
    ea\examples\helloworld.jsp;from=(7,0);to=(7,62)]
    // end

```

...

info

其值為任意的字串。

相同於重載 `Servlet` 中 `Servlet.getServletInfo()` 這個方法，作用也與 `Servlet` 同。

isErrorPage

其值為 “true” 或 “false”，預設為 “false”。

用來指定目前 JSP 網頁是否是另一個 JSP 網頁的錯誤顯示頁 “errorPage”。如果為值，那麼 Scripting 語言中隱含的 “exception” 變數就會被定義，而這個變數也就是，丟出這個例外的 JSP 網頁的錯誤所產生的 `Throwable` 物件。

若是沒有設定為 true 的話，在 Scripting 語法元素中使用 exception 變數是不被允許的，會造成 JSP 引擎的編譯時期的錯誤。

errorPage

其值必須是一個以 “URL 路徑” 描述的 JSP 網頁，而這個網頁 “通常” 會是 `isErrorPage="true"`。

當 `errorPage` 被定義，JSP 網頁發生的錯誤仍然會產生，只是此時 `catch` 例外的責任就不再是該 JSP 網頁本身，而是產生一個 `Throwable` 物件，並把他重導給 `errorPage` 所指定的網頁。

JSP 引擎的實作利用 JSDK API 2.1 的重導物件 `RequestDispatcher` 的 `include()` 方法，並且利用這方法的 `ServletRequest` 參數的 `setAttribute()` 方法，把產生的 `Throwable` 物件傳入，JSDK 的實作將它命名為 “`javax.servlet.jsp.jspException`”

參考 `RequestDispatcher` 的 `include()` 方法

```
public void include(ServletRequest request,
```

```
    ServletResponse response)
```

```
    throws ServletException,
```

```
    java.io.IOException
```

範例

/src/jspsample/ErrorPage_JSP10_Demo.zip

原始碼 5、ErrorPageDemo.jsp – 丟出例外的 JSP 網頁

```
<%@ page errorPage="errorpge.jsp" %>
<HTML>
<HEAD>
  <TITLE>JSP 1.0 Error Page Demo</TITLE>
</HEAD>

<BODY>
<H1>JSP 1.0 Error Page Demo</H1>

<%
  String s = null; //字串是空的
  s.getBytes(); //這會丟出 NullPointerException 例外
%>
</BODY>
</HTML>
```

原始碼 6、errorpage.jsp – 接受例外的 JSP 網頁

```
<html>
<body bgcolor="black" text="#FFFFFF">
<!--@ page isErrorPage="true" -->
<h1> Attention the fellowing error occurs</h1><br>
<pre><%= exception.getMessage() %></pre>
</body></html>
```

你可以試試執行 ErrorPageDemo.jsp 這個例子，因為在 **String s** 還是 **null** 前對它做任何物件方法的操作，因此會丟出 **NullPointerException** 的例外，而如果有 **errorPage** 的設定，就會重導到該網頁，以本例子來說，也就是會重導到 **errorpage.jsp** 這個 JSP 網頁，而同時 **errorpage.jsp** 這個 JSP 網頁的 **exception** 變數也會被定義成 ErrorPageDemo.jsp 丟出的 **Throwable** 物件。

摘要：

接受例外的網頁不一定要是 **isErrorPage="true"**，但是若不設定的話，**exception** 變數就不會被定義，也就是不會有這個隱含變數的存在。

值得注意的是，如果 **autoFlush= "true"** 那麼將錯誤訊息重導到另一個網頁便會可能失敗，失敗的原因是因為第一個 **JspWriter** 已經將內容出清到 **ServletResponse** 資料流了，因為重導是受限於 HTTP 協定的重導規則，一但有輸出內文後，就不能下重導 **Header** 了。關於這點 “Servlet 與 HTTP 協定有詳細討論”。

關於這點，我們以以下的例子來做測試

原始碼 7、FailErrorPageDemo.jsp – 丟出例外的 JSP 網頁

```
<%@ page buffer="none" errorPage="errorpge.jsp" %>
<HTML>
<HEAD>
  <TITLE>JSP 1.0 Error Page Fail Demo</TITLE>
</HEAD>

<BODY>
<H1>JSP 1.0 Error Page Fail Demo</H1>

<%
  String s = null; //字串是空的
  s.getBytes(); //這會丟出 NullPointerException 例外
%>
</BODY>
</HTML>
```

上面的例子與 ErrorPageDemo.jsp 相近，但是我們在 FailErrorPageDemo.jsp 這個例子中，把它的 **buffer** 設到無（**buffer="none"**），也就是說它完全不會用到緩衝，寫入什麼就立即輸出什麼到 **ServletResponse** 的資料流，因此錯誤重導的 Header 一定無法使用。

若你的 JSP 網頁的 **errorPage** 總是不能重導成功，解決的方法之一，就是加大 **buffer** 的值，使得允許的緩衝區超過錯誤發生前，可能的輸出資料大小。

contentType

定義 JSP 網頁檔案的字元編碼與其對 **response** 輸出的資料流編碼，以及 **response** 資料的 MIMIE type，這方面的規範與 Servlet 相同，請參考“JSP 輸出輸入篇－Servlet 輸出”。

設定值與 Servlet 一樣以“TYPE; charset=CHARSET”為格式，TYPE 的預設值為“text/html”，而“CHARSET”預設值為“ISO-8859-1”。

JSP 0.92 page 指令列表

注意，在 JSP 0.92 與 0.91 間只有 **Page** 指令語法，而 JSP 1.0 的指令語法除了 **page** 指令外還有 **include** 指令，而在 0.92 與 0.91 則沒有 **include** 指令，但是有同樣功能的語法，但是有其它的稱呼。

語法：

```
<%@ (屬性="屬性值")+ %>
```

“()+”也是 regular expression 的表示法，用來表示一個指令語法可以包含一個以上的 **屬性="屬性值"**。

language

範例

```
<%@ language="java" %>
```

與 JSP 1.0 的

```
<%@ page language="java" %>
```

意義完全一樣

errorpage

與 JSP 1.0 的 **errorPage** 屬性相近，但不同的是，其值可以是靜態的 HTML 檔案資源，如下

```
<%@ errorpage="contactwebmaster.html" %>
```

也因此 0.92 沒有 **isErrorPage** 的指令，但是可以在接受錯誤的 JSP 檔中利用，下面這種方式印出錯誤訊息

Scriptlet 語法

```
<% printStackTrace(out); %>
```

Expression 語法

```
<%= exception.getMessage() %>
```

0.92 的 Bean DISPLAY 標籤語法

```
<DISPLAY PROPERTY="exception:message" >
```

PENDDING: more

import

範例

```
<%@ import="java.io.*,java.util.Hashtable" %>
```

其值的意義、用法與 JSP page 指令的 **import** 屬性一樣，

JSP 0.91 Page 指令列表

語法與 0.92 同

language

同 0.92

method

這是用來指定 JSP 實作所產生的程式碼是重載什麼方法

```
<%@ method="doPost" %>
```

這個指令語義在 JSP 0.92 認為是錯誤的設計，因為 Servlet 本身在處理 HTTP 的 GET 或 POST 的 API，除了特殊的情況（multi-part post data）之外，幾乎是一模一樣的，而 JSP 定位絕對不會遇到這種特別的情況，也不需要處理它，因此在 JSP 1.0，這個語義完全被取消

import

同 0.92

content_type

範例

```
<%@ content_type="text/html; charset=UTF-8" %>
```

其值的意義、用法與 JSP page 指令的 **contentType** 屬性一樣，

implements

範例

```
<%@ extends="javax.servlet.http.HttpServlet" %>
```

其值的用法與 **import** 屬性一樣，用來指定 JSP 實作程式所 **implements** 的界面，0.92 版後就被取消掉。

extends

範例

```
<%@ extends="javax.servlet.http.HttpServlet" %>
```

其值的意義、用法與 JSP page 指令的 **extends** 屬性一樣，

JSP 1.0 的 Page 指令與中文問題

指定 Content Type

細心一點的人，或許會注意到，之前在 Servlet 中文處理的介紹中，不是一在的強調應該用 `setContentType("text/html; charset=Big5");`；而不是 `setContentType("text/html");`；嗎？是的，你想的沒錯，而錯是錯在於 JRun 用了特殊的方式，來決定輸出的編碼，而不是在 `setContentType()` 方法的參數上。這樣的做法很不好，

因此，在此不得不提一點就是，不是所有的 JSP 編譯引擎都能正確處理中文，目前筆者使用過的 JRun 與 GNUJSP 都可以，而 Sun 的 JSSDK 一向不可以。

使用 GNUJSP 的使用者，請得記一定要使用

```
<%@ content_type="text/html; charset=Big5" %>
```

來宣告你使用的 Big5 編碼，GNUJSP 引擎很有名，以至於 JSP 0.92 規格書有關編碼的部份，採用 GNUJSP 的作法。但是怪異的是 JSP 1.0 草案，這部份的資料，卻消失掉了。筆者在 JSP Mailing List 上幾番的詢問，至今仍未獲得解答，只有讓讀者們依自己的 JSP 編譯引擎試驗來決定囉！

PENNDING

至於 Sun JSP 1.0 Reference Implementation 的話，則是向來完完全全不支援中文，各位不必費心去想怎麼解決☺。

中文編碼宣告與 JSP 規格書

JSP 規格書中並沒有詳加定義檔案編碼與 JSP 語法的關係，似乎有意將這類工作交給 JSP 的轉譯器來處理，以至於筆者沒有辦法給讀者們任何有效的建議。

JSP 網頁可以以 page 指令的 `contentType` 屬性來指定 response 的內文格式，

PENDDING

contentType 的格式

<ftp://venera.isi.edu/in-notes/iana/assignments/media-types/media-types>

include 指令語法

JSP 網頁插入其它的檔案有兩種語法可以達到，兩者的語義、用途有很大的不同，一種被規類為指令語法，也就是本節要談的 **include** 指令，另一個則是 **Java Server Page** 標準動作語法之一的 **include** 動作，我們先來了解 **include** 指令的意義

JSP 1.0 include 指令

現在我們來看個例子

```
/src/jsp_sample/Directive/Include_Directive_Demo_JSP10.zip
```

原始碼 8、JSP 插入範例—Include_Directive_Demo_JSP10.jsp **JSP1.0**

```
<%@ page language="java" %>
<HTML>
<HEAD>
<TITLE>Include Directive Demo - JSP 1.0</TITLE>
</HEAD>
<BODY>
<h1>Include Directive Demo - JSP 1.0</h1>
<%@ include file="inc.txt" %>
</BODY>
</HTML>
```

下面這個，則是 **hello.jsp** 所要 **include** 指令所要插入的檔案

原始碼 9、JSP 插入範例—inc.txt **JSP1.0** **JSP0.92** **JSP0.91**

```
inc.txt start here<br>
<%
for(int i = 0; i < 10; i++)
    out.println("number " + i + "<br>");
    out.flush();
%>
inc.txt end here<br>
```

下面則是 JSP 引擎的輸出結果

```
<HTML>
<HEAD>
<TITLE>Include Directive Demo - JSP 1.0</TITLE>
</HEAD>
<BODY>
<h1>Include Directive Demo - JSP 1.0</h1>
inc.txt start here<br>
number 0<br>
```

```
number 1<br>
number 2<br>
number 3<br>
number 4<br>
number 5<br>
number 6<br>
number 7<br>
number 8<br>
number 9<br>

inc.txt end here<br>
</BODY>
</HTML>
```

發覺到了嗎？沒錯，就算插入的檔案，如上例的 `test.txt` 不是 `jsp` 結尾，但是內部的 JSP 語法一樣有用，這是為什麼呢？這是因為 `include` 指令是的編譯時期的指令，也就是 JSP 的轉譯會先把資料插入 JSP 檔案中，再行轉譯成 Servlet，所有 `include` 指令所插入的檔案，其內容就需要符合 JSP 語法！

值得注意的是，如果你改變了插入的檔案，則 JSP 不會跟著改變，這是因為 `include` 指令是的編譯時期的處理，如果你在 JSP 編譯完後才改變插入的檔案，那麼對 JSP 本身沒有影響，解決的方法是把 JSP 檔也改一下，讓 JSP 重新編譯一次。

PENDDING: more

語法

傳統格式語法是

```
<%@ include file=... %>
```

語法－XML

它的 XML 樣式語法

```
jsp:directive.include
```

JSP 0.92 include 指令

在 0.92 並沒有 `include` 指令的稱呼，它是以 `Server-side includes (SSI)` 稱呼它。它是以 `Center for Supercomputing Applications (NCSA)` 所制定的語法格式來制定，共有兩種型式

```
<!--# include file="copyright.jsp" -->
```

這種型式的 `file` 屬性的值與 `include` 指令的 `file` 屬性效果相同，另一種型式則多加上 `virtual` 屬性來指定對映的目錄

```
<!--#include virtual="/path/" file="copyright.jsp" -->
```

例如你的 Web Server 的檔案目錄在 “c:\htdocs\” 若你要插入下面的檔案

```
c:\htdocs\estroe\copyright.jsp
```

則在你的 JSP 檔中加入

```
<!--#include virtual="/estore/" file="copyright.jsp" -->
```

關於 NCSA Server-Side Include 你可以參考以下的網址

<http://hoohoo.ncsa.uiuc.edu/docs/tutorials/includes.html>

PENDING: sample

JSP 0.91 include 指令

沒有類似的語法。

JSP 動作語法一 Action Syntax

JSP 1.0 動作指令的形式都是以 XML 為準的，也就是說，它們的**大小寫會有差別**，因此以 JSP 1.0 而言，`<jsp:useBean>`不等於`<jsp:usebean>`，前者是標準動作之一，而後者什麼都不是，這點要十分注意。

除了「標準動作指令」外，JSP 1.0 還有所謂的「延伸動作指令」機制，這些指令在 JSP 1.0 中不強制實作，但是 JSP 的制定小組希望部份的延伸動作指令在 1.1 版之後，成為標準動作指令，這些我們後面會談到。

include 動作語法

JSP 1.0 定義了一些基本的動作指令（Standard Action），這些動作指令是任何支援 1.0 以後版本的 JSP 引擎都必須實作的。不論你用那家的 JSP 引擎與 Servlet 引擎都必須要支援這些動作指令。

JSP 與插入檔案

在有兩種插入檔案的語法，一個就是我們前面是到的 `include` 指令，另一個則是我們現在要談的：

語法：

```
<jsp:include page="urlSpec" />
```

下面是一個例子

```
<jsp:include page="/templates/copyright.html"/>
```

`include` 動作與 `include` 指令最大的不同在於，我們前面提供 `include` 動作它是執行時期的語法，也就是說你只能引入靜態的資源，它不會隨著你插入的資源改變而改而，而 `include` 指令則正好相反，它是屬於動態的指令，會隨著插入的資源改變而改變

我以 JWSDK 1.0 EA 的例子來做說明

```
http://localhost:8080/examples/jsp/include/include.jsp
```

PENDDING

JSP 0.92 include 動作語法

0.92 沒有類似的語法

解進的方法

JSDK2.1

```
out.flush();  
getServletContext().getRequestDispatcher("/Include_Action_JSP10/  
test.txt").include(request, response);
```

JSP 0.91 include 動作語法

同 0.92

forward 動作語法

forward 是重導的意思，所謂的重導，就是在進入 A 網頁後，資料卻是顯示 B 網頁的資料。

它的型式共有兩種，一種是 Client Side 的重導，另一種則是 Server Side 的重導，兩者的重導效果略有不同

Client-Side 的重導

當你在你的 JSP 中產生下面的資料時

```
<SCRIPT LANGUAGE="JavaScript">
```



```
location.href='error.html';  
</SCRIPT>
```

如果你的網頁是

```
http://localhost:8080/client_forward.jsp
```

Client 在載入完這個 JSP 後，會立刻重導到

```
http://localhost:8080/error.html
```

而 Client 瀏覽器上網址看到的，就會是上面這個值

另一種則是 Server-Side 的重導，例如 JSWDK 1.0EA 的這個例子

```
http://localhost:8080/examples/jsp/forward/forward.jsp
```

這個例子會檢查你的 Server 的 JVM 上的記憶體資源剩下多少，剩下多於 50% 重導到 one.jsp 若是少於 50% 則重導到 one.jsp，但是你在瀏覽器上看到的卻永遠是上面的 URL，這是因為這些動作都發生在 Server-side。

由於 JSP 的 forward 動作是利用緩衝機制，把 JSP 中的資料先放在緩衝中，因此在還沒確定到底要不要 forward 之前，資料都不會寫入，這才重導才會正確，也因此如果你的 JSP 並沒有設定緩衝（**page** 指令的 **buffer="none"**），或是緩衝區太小而你的 **page** 指令的 **autoFlush="true"**，那麼也會失敗，而系統會丟出 **IllegalStateException** 的例外。

範例

PENDING

語法

```
<jsp:forward page="relativeURLspec" />
```

下面這種語法，forward 動作支援，

例如你要重導的網頁可能是會變化的，例如

```
<% String whereTo = "/templates/"+someValue; %>  
<jsp:forward page='<%= whereTo %>' />
```

當 **someValue="one.jsp"** 時，就會被重導到 **"/templates/one.jsp"** 而若 **someValue="two.jsp"** 時，就會被重導到 **"/templates/two.jsp"**。

`page='<%= whereTo %>'`這種語法，JSP 規格書稱之為 Request-time 屬性值，也就是屬性值可以在一次連線（Request/Response）期間，才決定它的值。

摘要：

JSWDK 1.0EA 還未支援 Request-time 屬性值的設定，所以你不能拿來使用。

JSP Scripting 語法—Scripting Syntax

所有的宣告，它的都是 `page` 變數（可見域），也就是整個 JSP Page 中任何部份都可以存取到它，包含對所有的方法（`class` 方法）

宣告語法是用來宣告 JSP 網頁中的變數與方法（函式），宣告裡用的程式語法，與`<%@page language="xxxx"%>`的 `xxxx` 有關，例如定義成 `java`，自然它的語法就是使用 Java 的正規語法。

物件與變數

一個物件可以被 scripting 元素的某段程式中被存取到，透過 scripting 語言的變數，因此在 Scripting 元素裡的變數，其實指的就是 Java 的物件。一個元素可以在兩個地方定義 scripting 變數，

PENDING

宣告語法—Declarations Syntax

語法用途與用法

宣告語法 JSP 1.0

JSP1.0

```
<%! int i = 0; %>
<%! public void foo() { ...Java 程式碼 } %>
```

摘要：

宣告語法 JSP 1.0—XML

PENDING

```
<jsp:declaration> <![CDATA[ int i = 0; ]]> </jsp:declaration>
<jsp:declaration> <![CDATA[
```

```
public void f(int i) { ...Java 程式碼 }
]]> </jsp:declaration>
```

摘要：

JSP 中的 DTD 片段 `<!ELEMENT jsp:declaration (#PCDATA) >`

宣告語法 JSP 0.92

可以用 JavaServer Page 0.92 中的 `<SCRIPT ></SCRIPT>` 標籤來義。它的型式是以

JSP0.91 **JSP0.92**

```
<SCRIPT RUNAT="location">
...Java 程式碼
</SCRIPT>
```

RUNAT 指定這個 `<SCRIPT ></SCRIPT>` 包含的宣告內容應該是在那裡執行

摘要：

如果 **RUNAT** 參數沒有指定的話，它的預設值是“client”，而這個 `<SCRIPT ></SCRIPT>` 包含的宣告內容會被直接送到 Client 端去，不被處理。

一般我們的用法，應該是把它例如下面的程式碼

JSP0.91 **JSP0.92**

```
<SCRIPT RUNAT=server>
int i = 0;
public void foo() {
...Java 程式碼
}
</SCRIPT>
```

宣告語法 JSP 0.91

宣告語法 JSP 0.92 與 JSP 0.91 兩者並沒有改變。

Expression 與 Java 語法

JSP 變數的可見域問題

任何定義在`<%...%>`中的變數都是區域變數的，其它的函式都看不到，即使你定義在一個 JSP 中；例如：

JSP1.0

```
<%  
int i = 3;  
%>  
...  
<%!  
public void foo() {  
    //這裡看得到 i 變數  
}  
%>
```

為什麼？因為 JSP 轉譯成 Servlet 程式後，`i` 最後會變成 `service()` 方法中的一個區域變數，而 `foo()` 則會變成這個 Servlet 的一個公開方法

另一個例子

JSP1.0

```
<%!  
int i = 3;  
%>  
...  
<%!  
public void foo() {  
    //這裡看得到 i 變數  
}  
%>
```

為什麼？因為 JSP 轉譯成 Servlet 程式後，`i` 最後會變成 Servlet 的私有成員變數，當然，所有的物件方法，如上面的 `foo()` 都可以見到它。

摘要：

上面的使用方法都是以 1.0 版的宣告語法為準，0.92 與 0.91 則是使用 `SERVER` 標籤，請參考前面的文章。

重點是，要了解 Servlet 環境是多執行緒的，一個 Servlet 可能會有同時多個物件在不同執行緒中執行。因此，所有的處理 Request 的 JSP Page（現在是 Servlet）都可以取得 `fooVar`，也同時會引發執行緒的多步化問題。所以我們應該小心的使用它。

Objects 與具活動領域

在 JSP 中，你可以建立 Java 的物件，而你建立在 Scriptlet 中的物件，它的活動領域只有在該次的 request 中而已。

而 JSP 規格書指出有些物件是隱含的，也就是不用你建立它就是存在的，這些物件其實就是我們在 Servlet 中，如下

```
...
public void service(HttpServletRequest request,
    HttpServletResponse response)
    throws ServletException, IOException
{
    PrintWriter out = request.getWriter();

}
...
```

其中的 **request** (HttpServletRequest 的物件) 與 **response** (HttpServletResponse 的物件) 這些 JSP 會自動幫我們包裝好

page

request

session

application

預先存在的變數（物件）

當你在寫 JSP 時，已經有一些物件是早已經宣告好存在在那兒的了，例如最常出現的 **out** 物件現在我們來看看還有什麼物件存在呢？

隱含的物件	型別	用途	活動領域
request	javax.servlet.HttpServletRequest	包含了請求端方面的資訊	request

<code>response</code>	<code>javax.servlet.HttpSer vletResponse</code>	用以設定 JSP 回 應資訊的資料內 容設定	<code>page</code>
<code>pageContext</code>	<code>javax.servlet.jsp.Pag eContext</code>		<code>page</code>
<code>session</code>	<code>javax.servlet.http.Ht tpSession</code>	在同一連線作業 中，所產生的 Session 資料， 目前只對 HTTP 協定有意義	<code>session</code>
<code>application</code>	<code>javax.servlet.Servlet Context</code>		<code>applica- tion</code>
<code>out</code>	<code>javax.servlet.jsp.Jsp Writer</code>	回應資料流的標 準輸出	<code>page</code>
<code>config</code>	<code>javax.servlet.Servlet Config</code>		<code>page</code>
<code>page</code>	<code>java.lang.Object</code>	意同於 Java 中 的 this	<code>page</code>

request

response

pageContext

session

application

out

config

預先存在的變數－JSP 0.92

Scriptlet 語法

語法用途與用法

Scriptlet 中可以包含程式的片段，只要它符合 Scripting 語言的語法即可，例如 Java 語言。Scriptlet 的片段是在 Server 在處理一次 Client 連線（Request-processing）的時間時執行，也就是類似放在我們在 Servlet 中的 `service()`、`doGet()`、`doPost()` 等處理 request 的方法中，我們在 Scriptlet 除了能使用隱含的變數（如 `out`）之外，還可以使用利用 `jsp:useBean` 動作所產生的 bean 物件。例如我們建立

```
<jsp:useBean name="date" typ="java.util.Date"
```

那麼我們也就可以在 Scriptlet 存取它，例如

```
...  
<%  
out.println(date);  
%>  
...
```

還要注意的一點就是，如果你要在 Scriptlet 表示 “<%” 的話，就必須用 “<\%” 取代它，如下

```
out.println("print <\%");
```

Scriptlet 語法 JSP 1.0

如下

```
...  
<%  
out.println(date);  
%>  
...
```

Scriptlet 語法 JSP 1.0－XML

In the XML document corresponding to JSP pages, directives are represented using the syntax:

摘要：

JSP 中的 DTD 片段 `<!ELEMENT jsp:scriptlet (#PCDATA) >`

Scriptlet 語法 JSP 0.92

與 1.0 版同

Scriptlet 語法 JSP 0.91

與 1.0 版同

Expression 與 Java 語法

Expression 語法

語法用途

Expression 標籤以 `<%=` 為開始，其中間包含一段 Java 語言的表示式，並以 `%>` 結束；注意，不須要像 Java 語法一樣以分號 “;” 做結尾。

JSP0.91 **JSP0.92** **JSP1.0**

注意，
JSWDK
1.0EA 並不
支援中文，
所以你想測
試的話，請
不要打中文

```
<%  
String str = "這是一個字串";  
int i = 5;  
java.util.Date date = new Date();  
%>  
str 會被取代成 "<%= str + i + " " + date%>"
```

其輸出結果

```
str 會被取代成 "這是一個字串 5 Fri Jul 23 03:18:11 PDT 1999"
```

很簡單吧？其實 JSP 引擎只是會將它轉換成

這只是可能
的轉換結
果，實際上
還是要看 JSP
引擎，各家
的方法不一

```
String str = "這是一個字串";  
int i = 5;  
out.println("str 會被取代成\"\" + str + i + " " + date + "\"")
```


就跟我們平常在 Java 中常用 `System.out.println()` 的用法一樣，所以 `<%=....%>` 其內文包含的表示式中的變數是 String、Date 這類的物件，或是 int、float 之類的轉換，就如同 `System.out.println()` 裡的作法一樣，（物件都是透過它們的 `toString()` 方法）我想想應該是不難理解才是☺。

語法規則

Expression 語法 JSP 1.0

如

```
...
<%
out.println(date);
%>
...
```

可以改爲

```
...
<%=date%>
...
```

相當方便不是嗎？

Expression 語法 JSP 1.0 – XML

改以 `<jsp:expression>` 爲開始，而以 `</jsp:expression>` 結束，把上面的例子改過成 XML 樣式的版本，就是如同下面的結果

JSP1.0

```
<jsp:expression>
String str ="這是一個字串";
int i = 5;
java.util.Date date = new Date();
</jsp:expression>
str 會被取代成 "<%= str + i + " " + date%>"
```

摘要：

JSP 中的對映的 DTD 片段

```
<!ELEMENT jsp:expression (#PCDATA) >
```

Expression 語法 JSP 0.92

與 1.0 版同

Expression 語法 JSP 0.91

與 1.0 版同

Java Server Page 註解

在 JSP 中有兩種註解，

會輸出到 Client 的註解

在這邊的 Client 指的就是瀏覽器上，輸出到 Client 自然是指會輸出到回傳給瀏覽器的 HTML 上囉。第一種其實就是 HTML 本來就有的註解語法

HTML 註解

JSP0.91 **JSP0.92** **JSP1.0**

```
<!-- comments ... -->
```

另一種則是加強上面用法的新語法

動態註解

0.91 **0.92** **1.0**

```
<!-- ...你的註解... <%= expression %> 更多的註解... -->
```

它利用了 JSP 的 Expression 語法，藉以輸出動態的註解，這就跟輸出使用者可以看到的動態資料一樣，只不過對 HTML 語法而言，它不會顯示在瀏覽器上面，也就是說你得看這個輸出頁面的 HTML 原始碼；當然，這個語法不是用來處理 HTML 頁面的展現層的，它是開發專用的。

也就是說它的用途就是用來觀察你的 JSP 運作得怎麼樣，你可以在這裡輸出一些除錯資訊，是 JSP 網頁開發的好工具。

Expression 語法在後面的會提到。

摘要：

雖然動態註解上面指名是 **0.91** **0.92** **1.0** 版都通用，但是其實 0.92、0.91 並沒有規範他們，不過筆者的經驗是多數都可以用，因為這種功能實作很直覺也很重要，所以 1.0 才會把它納入規範中☺。

不會輸出到 Client 的註解

JSP 1.0

JSP 用的註解，但是這種註解不會輸出到 Client 端

1.0

```
<%-- ....你的註解，連%>也不會被輸出...--%> ...但這段會被顯示出來...--%>
```

在以<%--到以--%>結尾的內容就是註解的內容。

如何，與 HTML 的註解很像吧，但是這個註解完全不會輸出到 Client 端上。這有什麼用呢？當然有，當你要把 JSP 某段的資料取消掉，例如

0.91

0.92

1.0

```
<html>
....
<!-- 開始：本段是用來輸出使用者相關的資料 -->
<%
Connection connDB =
DriverManager.getConnection( "jdbc:odbc:MyDSN",
"username", "password" );

...輸出資料庫的內容
%>
<!-- 結束：本段是用來輸出使用者相關的資料 -->

...
</html>
```

好了，現在你要把這“輸出使用者相關的資料”這個功能拿掉怎麼辦呢？把它改成

0.91

0.92

1.0

```
<!--
Connection connDB =
DriverManager.getConnection( "jdbc:odbc:MyDSN",
"username", "password" );

...輸出資料庫的內容
-->
```

這樣子嗎？的確，這是可以取消掉它的功能的，但是卻會變成把這段程式碼給輸出到 Client 端上，如果這是正在執行的系統，那就糟了，全部的程式碼都會被輸出！

簡單地改成

1.0

```
<%--  
Connection connDB =  
DriverManager.getConnection( "jdbc:odbc:MyDSN",  
"username", "password" );  
  
...輸出資料庫的內容  
--%>
```

你只需要幾個字元就可以解決了☺！

當然，下面這種語法的註解方法也可行，不過這種用法是指在<%%>中的 Script 語言的專用註解，

0.91 0.92 1.0

```
<% /** this is a comment ... **/ %>
```

以上例下來，如果你用的是 Java 語言，上面這種用法就可行，也就是說這個用法端看你用的是那一種類的程式語言，作為 JSP 的 Scriptlet 的程式語言而定。

JSP 0.92

JSP 0.92 用就是前面 JSP 1.0 所用的非正規，與 JSP 使用的 Scriptlet 語言有關的語法，例如以 Java 為 Scriptlet 的語言

0.91 0.92 1.0

```
<% /** this is a comment ... **/ %>
```

當然，就這是為什麼 JSP 1.0 要再制定一個與 Scriptlet 無關的註解

1.0

```
<%-- ....你的註解....--%>
```

用來做為簡單，而且適用各種作為 Scriptlet 的語言的註解，這種註解法是所有語言都通用的。

JSP 0.91

0.91 與 0.92 同

JSWDK 例子

XML 樣式語法補充

XML 樣式語法有傳統語法沒有

JSP, HTML, and XML

JSP 規格設計用來產生動態資料，通常是用來嵌入 HTML 與 XML 中。一般來說，JSP 依據 GET 或 POST 方法傳入的參數而變化的產生資料...

PENDDING: more XML

JSP 1.0 XML 語法 DTD

```
<!ENTITY % jsp.body "  
(#PCDATA  
|jsp:directive.page  
|jsp:directive.include  
|jsp:scriptlet  
|jsp:declaration  
|jsp:expression  
|jsp:include  
|jsp:forward  
|jsp:useBean  
|jsp:setProperty  
|jsp:getProperty  
|jsp:plugin)*  
">  
  
<!ELEMENT jsp:useBean %jsp.body;>  
<!ATTLIST jsp:useBean  
  id ID #REQUIRED  
  class CDATA #REQUIRED  
  scope (page|session|request|application) "page">  
  
<!ELEMENT jsp:setProperty EMPTY>  
<!ATTLIST jsp:setProperty  
  name IDREF #REQUIRED  
  property CDATA #REQUIRED  
  value CDATA #IMPLIED  
  param CDATA #IMPLIED>  
  
<!ELEMENT jsp:getProperty EMPTY>  
<!ATTLIST jsp:getProperty  
  name IREF #REQUIRED  
  property CDATA #REQUIRED>  
  
<!ELEMENT jsp:includeEMPTY>  
<!ATTLIST jsp:include  
  page CDATA #REQUIRED>
```

```
<!ELEMENT jsp:forward EMPTY>
<!ATTLIST jsp:forward
  page CDATA #REQUIRED>

<!ELEMENT jsp:scriptlet (#PCDATA)>

<!ELEMENT jsp:declaration (#PCDATA)>

<!ELEMENT jsp:expression (#PCDATA)>

<!ELEMENT jsp:directive.page EMPTY>
<!ATTLIST jsp:directive.page
  language CDATA "java"
  extends CDATA #IMPLIED
  contentTypeCDATA "text/html; ISO-8859-1"
  import CDATA #IMPLIED
  session (true|false) "true"
  buffer CDATA "8kb"
  autoFlush (true|false) "true"
  isThreadSafe(true|false) "true"
  info CDATA #IMPLIED
  errorPage CDATA #IMPLIED
  isErrorPage(true|false) "false">

<!ELEMENT jsp:directive.include EMPTY>
<!ATTLIST jsp:directive.include
  file CDATA #REQUIRED>

<!ELEMENT jsp:root %jsp.body;>
<!ATTLIST jsp:root
  xmlns:jsp CDATA #FIXED "http://java.sun.com/products/jsp/dtd/
  jsp_1_0.dtd">
```

JavaBean 與其動作指令語法

JavaBeans 概念篇

由於本書不是 Java 語言方面的教學書，所以對 JavaBean 的觀念只是點到為止。

物件與元件模型

JavaBeans 是 Java 的元件模型

元件是物件更高層次的觀點，

，所謂的元件，就是擁有自行管理自己內部的運作的一個或多個類別所組成的群體，對這一個群體而言，除了它自己提供給外部的操作界面之外，其內部的資料以及運行方式，使用它的對象是不需要知道的。

這種獨立、自主 PEDDING: more bean's benefit and feature must be decript here

這就是 Bean 嗎？幾乎與一般類別沒什麼不同嘛。

JavaBean 與屬性

目前開發工具主要支援的元件模式就是 JavaBean，一個 Bean 很簡單地就是由數個屬性構成，你可以透過它提供方法藉以改變它的狀態，例如在 JBuilder 中提供了元件檢視器

何謂屬性

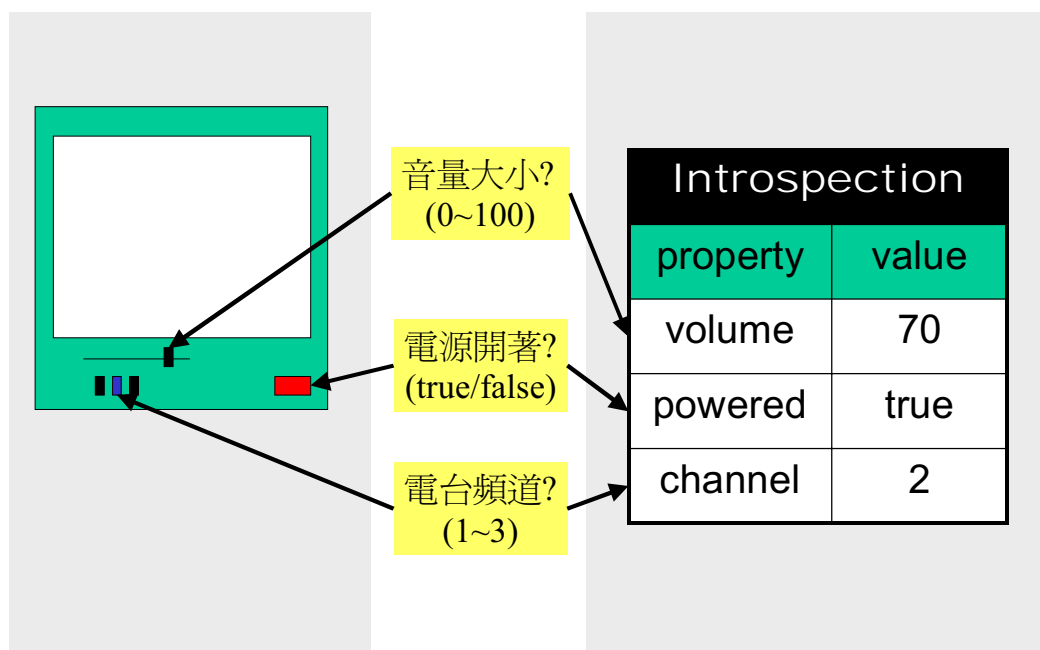
設定屬性值

name	this
background	<input type="checkbox"/> Control
disposeOnClose	True
enabled	False
exitOnClose	True
font	"Dialog", 0, 12
foreground	<input checked="" type="checkbox"/> Black
iconImageName	
iconImageURL	
layout	BorderLayout
menuBar	menuBar1
resizable	True
title	Welcome to JBuilder
visible	True

Properties Events

圖片 5、JBuilderd 的元件檢視器

屬性的內視



圖片 6、元件內視與屬性

Introspection	
property	value
volume	70
powered	true
channel	2

Bean 與一般 Java 物件的差異

Beans 不是一種類別，對 Java 類別而言，作為一個 Bean 是不需要去繼承 Beans 物件，相反的，你只要遵行某些特定的撰寫規範，或是實作某些特定的介面，PENDING list what's kind of bean interface

一般來說，能夠稱之為 Bean 的 Java 物件，必需具備下面幾個功能：

物件內視

供設定與改變的介面

事件

永續性

建立 MailBean

```
packagemail;

import java.io.*;
import java.util.*;
import sun.net.smtp.SmtpClient;

/**
 * <CODE>MailBean</CODE>是一個簡單的 Bean，可以將 HTML Form 裡
 * 的值轉為電子郵件，特色是簡單，而且支援中文
 * ；缺點是它使用 JDK 裡 sun.*包裹，這是非公開 API，所以你必需確定你的 JVM
 * 的版本有這個包裹，JSP 1.0 的使用方法請參考伴隨的 sendmail.jsp 與
 * mailform.html
 *
 * @author Jen Hsian Huang (黃任賢)
```

```

* @version 1.0, 1999/07/21
*/
public class MailBean implements Serializable {
    private String server, from, to, subject, body;

    public MailBean() {
        server = "mail"; //預設的 mail server 位置
        from = "unknown-sender@somewhere.com";
        to = "unknown-recipient@somewhere.com";
        subject = "<No subject specified>";
        body = "";
    }

    /**
     *
     * @return SMTP server 的 Host 名稱
     */
    public String getServer() {
        return server;
    }

    /**
     *
     * @param server SMTP server 的 Host 名稱
     */
    public void setServer(String server) {
        this.server = server;
    }

    /**
     *
     * @return 寄件者的 Mail Address
     */
    public String getFrom() {
        return from;
    }

    /**
     *
     * @param from 電子郵件的寄件者的 Mail Address
     */
    public void setFrom(String from) {
        this.from = from;
    }

    /**
     *
     * @return 收件者的 Mail Address
     */
    public String getTo() {
        return to;
    }

    /**
     *
     * @param to 收件者的 Mail Address
     */
    public void setTo(String to) {
        this.to = to;
    }
}

```

```

/**
 *
 * @return 電子郵件的主題
 */
public String getSubject() {
    return subject;
}

/**
 *
 * @param subject 電子郵件的主題
 */

public void setSubject(String subject) {
    this.subject = subject;
}


/**
 *
 * @return 電子郵件的內文
 */
public String getBody() {
    return body;
}

/**
 *
 * @param body 電子郵件的內文
 */
public void setBody(String body) {
    this.body = body;
}

/**
 *
 * @exception IOException I/O or SMTP 的錯誤
 */
public void send() throws IOException {
    SmtplibClient sendmail = new SmtplibClient(server);
    sendmail.from(from);
    sendmail.to(to);
    PrintStream mailOut = sendmail.startMessage();
    mailOut.println("From: " + from );
    mailOut.println("To: " + to );
    mailOut.println("Subject: " + subject );

    mailOut.println(body);

    mailOut.print("\r\n");
    mailOut.flush();
    mailOut.close();
    sendmail.closeServer();
}
}

```

JavaBeans – Java 的元件模型，參考文件：

<http://java.sun.com/docs/books/tutorial/javabeans/index.html>

JavaBeans Tutorial <http://java.sun.com/beans/docs/javaBeansTutorial-Nov97/javabeans/index.html>

JavaBeans Development Kit (BDK)

http://java.sun.com/beans/software/bdk_download.html.

JavaBean 常見問題集 <http://java.sun.com/beans/>

使用 Bean

通常，使用 bean 的原因是要計算一些運算，例如對一些資料庫做查詢，而這些運算的可重用性很大（通常我們稱之為 Business logic），你可以利用設定這些 bean 的屬性來 customize 它的行為。

JSP 現在訂立了語法規格，使得它具有存取 bean 屬性的能力，這些 bean 必須首先在某個 scope 下註冊自己的名稱，接著 JSP 便可以利用 **displayProperty** 標準動作語法來顯示這些 bean 的屬性。

動作指令：使用 Bean `<jsp:useBean>`

`<jsp:useBean>` 這個指令是 JSP 的重大精髓之一，這個動作指令，使得 JSP 加上了 JavaBean 的概念而成為了具有強大擴充性與易維護性的架構。

宣告它時，你必須與它代號 – **id** 與它的生存期限 – **scope**，還有它的類別 – **class**，然後你就可以在 JSP 使用它。

動作指令：取得 Bean 屬性 `<jsp:getProperty>`

動作指令：設定 Bean 屬性 `<jsp:setProperty>`

JSP Bean 生命週期與活動領域

行程追蹤－Session

Servlet 在這個問題上，提供了 Session 方面的支援，而 JSP 繼承了 Servlet 的 API 的功能，直接在 JSP Bean 的使用語法上支援行程追蹤的功能，JSP 引擎會自動幫我們把 session 的資料儲存還有取回來，因為 Java Server Page 有對多執行緒競爭的問題做處理，你可以確定同時只有一個執行緒對你的 **session** 物件做動作－同樣的 **application** 還有 **request** 物件也都是 thread-safe。

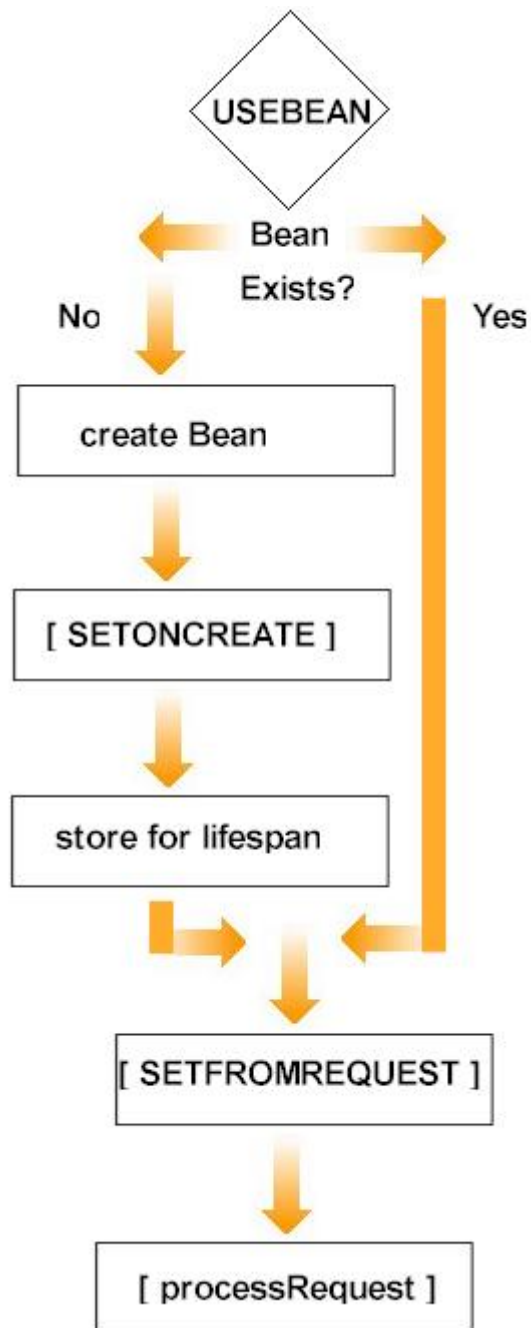
每個 Client（連線）都可以指定一個 session（`javax.servlet.http.HttpSession`）唯一的分別它。Servlet 與 JSP 在同樣的“**application**”中可以分享全域的 session 有關的狀態，透過 `HttpSession` 的 `putValue()` 與 `getValue()` 與 `removeValue()` 方法來達成，關於這點，請參考“Servlet 進階篇－行程管理”

JSP 與 Servlet 若是處理同一個 `javax.servlet.ServletRequest` 可以利用它的 `setAttribute()`、`getAttribute()` 與 `removeAttribute()` 方法傳遞資訊。

摘要：

行程追蹤的行為：多少物件可以放入 session 中、session 可以維持多久，還有其它的 session 特色等－都是由 JSP／Servlet 引擎上可以設定的。

JSP Bean 生命週期—建立的規則



JSP Bean 生命週期—何時被刪除？

當 Bean 超出它的活動領域外時，它的生命週期就告終了，而與其相關的物件也跟著清除，bean 的 **scope** 有三種 **page**、**session** 與 **application**，所以 bean 的生命週期也有三種，如果生命週期的長短來看的話，**application**>**session**>**page**，而每個 bean 的生存範圍，以目前的 JSP 規格來說，僅定義存在於同一個 JVM 機器下的物件，至於其它有些 JSP 引擎實作的多重 JVM 機器的支援，目前的 JSP 並沒有規範，我們僅就同一個 JVM 機器下的情況來討論。

Bean 與活動領域（scope）的關係

下圖便是各種活動領域（**application, session, page**）下，每個 JSP Bean 的存活區域，最大的黃色區域便是整個 JVM 的執行空間，一但你在這個空間下加入了一個 bean，不論是那個 session，那個 request/resonse 都會看到它。

而綠色部份則是一個 Client 與各個 JSP 之間的數個連線，我們統稱這些連線間為一個行程。舉例來說，當使用者打開他的瀏覽器，並連上這些 JSP 與其它資源所構成的 Web 程式時，這些 JSP 之間的物件就可以透過 session 來分享資料，這部份的動作是自動的，例如

在 foo1.jsp 中

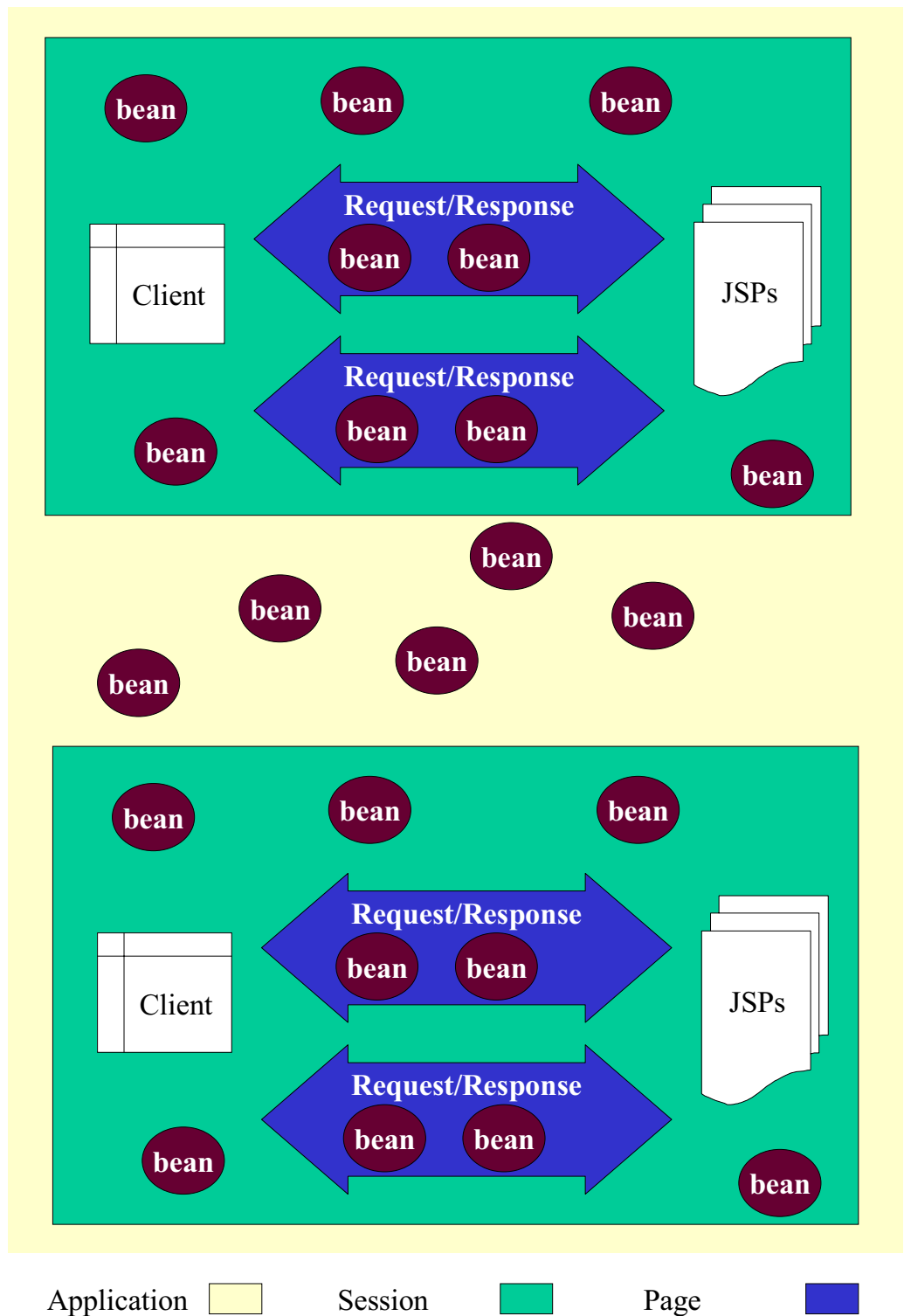
```
<jsp:useBean id="counter" scope="session"
class="session.CounterBean" />
```

與在 foo2.jsp 中

```
<jsp:useBean id="counter" scope="session"
class="session.CounterBean" />
```

你同時都建立了 session 活動領域的 bean，那一個會先被執行？不知道，但可以確定的是，若其中一者是在 session 首先被呼叫的，那麼它就是建立這個 bean 的 JSP 了，而另一圈 JSP 見到 session 裡已經有了 counter 這個 bean，就不會重建一個，而直接拿它來用，因此

而存在在這些行程的 bean，就是存在 JSP 的 session 變數裡，所有



我們先來討論每種 **scope** 值對 JSP 裡的 bean 的生命週期的影響

scope = “session”

有關 session，請參考前面的“Servlet 間的溝通篇”

這種活動領域的 bean 其生命週期是 HTTP session 之間，當 Server 終結掉一個 session 時，伴隨它的 JavaBean 物件也會跟著一起被清除掉

接下來我們以下面的範例檔

src/jsp_samples/CounterBean_Session_Demo.zip

來說明 **scope= “session|page|application”** 的 bean 之間的不同，整個程式就僅只是利用 session 的 scope 來指定 ConterBean 的生命週期，ConterBean 就能

首先看到的是 CounterBean.java 這個 bean 的原始檔

原始碼 10、BeanCuonter.java **JSP1.0**

```
package session;

/**
 * <CODE>CounterBean</CODE>是一個簡單的 Bean
 * 純粹只是記錄一個數值並累加它
 *
 * @author Jen Hsian Huang (黃任賢)
 * @version 1.0, 1999/07/21
 */
public class CounterBean{
    int val;
    public CounterBean(){
        this.val = 1;
    }

    /**
     * 屬性<CODE>counter</CODE>的取得方法
     * 取得後，還會累加一次
     *
     * @return 目前計數器的值
     */
    public int getCount(){
        return val++;
    }

    public String toString(){
        return (new Integer(val)).toString();
    }
}
```

這並不是很符合 JavaBean 規格的 bean，過爲了簡化，我只寫需要的部份，

scope = “page”

scope = “page”的 bean 它的生命週期只有在每次的 Client-request 連線而已，當 request 終了時，也就是回應結束時，這類的 bean 的生命就會被終結掉，所以不同的連線（Request/Response 之間），它們的 bean 是不可分享的

scope = “application”

Application 的生命週期是全域的，從 Server 的啟動到結束，都是它的活動領域，所以一但你建立它以後，直到 Server 關閉前，它都會存在

以 session scope 而言，可以試著兩個瀏覽器開打它，你會發覺兩個瀏覽器的計數是獨立的，

接著你再改成 application scope 再試著用兩個瀏覽器開打它，你會發覺兩個瀏覽器的計數是計數器是共享的

PENDDIGN: more

JSP Bean 生命週期—0.92 版

JSP0.92

```
<html>
<body>

<USEBEAN NAME="cart" TYPE=sessions.DummyCart LIFESPAN=session>
<SETFROMREQUEST BEANPROPERTY="*">
</USEBEAN>

You have the following items in your cart: <ol>
<LOOP PROPERTY=cart:items PROPTYELEMENT=x>
<li> <DISPLAY PROPERTY=x> </li>
</LOOP>
</ol>

</body>
</html>
```

JSP 系統架構與 JavaBean

如果你詳閱過 JSP 0.92 的文件，你應該會讀到有關 JSP 與 Servlet 定位的關係，其中最引人興趣的就是“Model I”與“Model II”這兩個設計的樣版（或許該稱之為運作模式？），這兩種模式的採用，決定於你個人的喜好（如果你掌控整個設計的話）、團隊合作策略或是你的物件導向設的信仰

較不正規的定義，Model I 是把你的程式的商務邏輯與你的 HTML 碼混合在一起的技術，而 Model II 則把商務邏輯獨立出來，如此一來它的功能便可以方便地被擴充，與 HTML 的展現層分離。

JavaBean 對 JSP 的潛力

在 JSP 的遠景中，我們知道，想要利用 JSP 開發真正可重用的 Web 程式，我們需要的是更加

因此我們可以利用 Bean 的元件模式

將扮演極重要的角色，在下面的“JSP 動作指令篇”我們將會看到 JSP 如何作為 Web 程式的展現部份並將 Web 程式的邏輯部份放入 Bean 中，使得以 JSP 開發的 Web 程式，可以真正的高度的重用性。

MailBean 與 JSP

mailform.html

```
<HTML>
<HEAD>
<TITLE>JSP 1.0  send mail form</TITLE>
</HEAD>
<BODY>

<H1>JSP 1.0  send mail form</H1>

<FORM action="sendmail.jsp" method="POST">
  <TABLE>
    <TR>
      <TH align=right>From:</TH>
      <TD>
        <INPUT type=text name=from size=60>
      </TD>
    </TR>
    <TR>
      <TH align=right>To:</TH>
      <TD>
        <INPUT type=text name=to size=60>
      </TD>
    </TR>
    <TR>
      <TH align=right>Subject:</TH>
      <TD>
        <INPUT type=text name=subject size=60>
      </TD>
    </TR>
    <TR>
      <TH align=right>Your message:</TH>
    </TR>
    <TR>
      <TD colspan=2>
        <textarea name=body cols=60 rows=20
wrap=virtual></textarea>
      </TD>
    </TR>
    <TR>

```

```

        <TD></TD>
        <TD align=left>
            <INPUT type=submit value="  Send message  ">
        </TD>
    </TR>
</TABLE>
<INPUT type=hidden name=server value="mail.ncnu.edu.tw">
<!-- $Q¥Î¥|“Ó³]@w Mail Server -->
</FORM>

```

sendmail.jsp

```

<%@ page errorPage="errorpge.jsp" %>
<HTML>
<HEAD>
    <TITLE>JSP 1.0 send mail page</TITLE>
</HEAD>

<BODY>
<H1>JSP 1.0 send mail page</H1>

<!-- 建立 Mail.bean-->
<jsp:useBean id="mailer" scope="page" class="mail.MailBean" />

<!-- 把 HTTP Form 的值送到 "mailer" 這個bean 對映的 Properties-->
<jsp:setProperty name="mailer" property="*" />

<%
mailer.send();
%>

Your email message was sent succesfully.

</BODY>
</HTML>

```

errorpage.jsp

```

<html>
<body bgcolor="black" text="#FFFFFF">
<%@ page isErrorPage="true" %>
<h1> Attention the fellowing error occurs</h1>
<table width="75%" border="1" bgcolor="red">
    <tr>
        <td><pre><%= exception.getMessage() %></pre></td>
    </tr>
</table>
<font size="1"> </font>
<h1>please check your input</h1>
</body></html>

```

JSP 進階

JSP 延伸動作指令

目前的規範

所謂的延伸動作指令，就是指各家不同的 JSP 引擎與版本不一定會支援的動作指令，各家的延伸動作指令將會成為各家 JSP 引擎的賣點。因此使用延伸動作指令時，要格外注意它的可攜性。

這裡所謂的可攜性，指的是跨 JSP 引擎的可攜性，而不是跨作業系統平台的可攜性。

在 JSP 1.0 的規格中，有定義了延伸動作指令的宣告語法，希望在未來，延伸動作指令的語法，將會是可攜性的；換句話說，你可以在 A 公司的 JSP 引擎上，開發自己的 JSP 延伸動作指令，而可以不經修改地，搬到 B 公司下的 JSP 引擎上使用。但是要注意的是 JSP 1.0 並沒有強制“延伸動作指令”需要具有可攜性。JSP 1.1 才會將延伸動作指令的可攜性列為強制實作的功能。

在可預期的未來，JSP 除了會有各式各樣的 JSPBean 被開發出來，更會有各種多樣的延伸動作指令程式庫被開發出來，那麼 JSP 的優勢就不再只是具有 Java 的強力功能而已，具延展性與易用性也將會是大賣點。各位可以參考 JRun 的<CF Anywhere>這項產品，延伸動作指令的發展遠景應該就是如此，意思就是，未來的 JSP 想要將其它的 Server-Side Script 包含在其中應該是可能的，JRun 的<CF Anywhere>便是利用這種概念，將 Alliare ColdFusion 這個有名的 Server-Side Script 包含在 JRun 裡面。

延伸動作指令的概念，以筆者的觀點來看，應該是來自於 JRun 的 Taglet 的功能，也因此，未來延伸動作指令的開發方式，應該與 Taglet 相似，而昇陽公司也應該會發表延伸動作指令的標準程式庫介面才是。

JSP 與 HTML、HTTP

JSP 與 Servlet API

與 Servlet 溝通

1.0

FooServlet.java

```

import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import modell.Foo;

public class FooServlet extends HttpServlet
{
    public void service(HttpServletRequest req,
        HttpServletResponse res)
        throws ServletException, IOException
    {
        String s[] = new String[] {
            "blue", "green", "red" };
        Foo f = new Foo(s);
        req.setAttribute("foo", f);

        getServletContext().getRequestDispatcher(
            "/samples/modell/foo.jsp").forward(req, res);
    }
}

```

foo.jsp

```

<html>
<usebean name=foo type=modell.Foo lifespan=page>
</usebean>
<%
String[] pros = foo.getList();
%>
<ul>
<%
    for(int i = 0; i < pros.length; i++
%><li> <%=pros[i]%>
<%}%>
</loop>
</ul>
</html>

```

Foo.java

```

package modell;

public class Foo {
    String s[];
    public String[] getList() { return s; }
    public Foo(String s[]) { this.s = s; }
}

```

0.92

foo.jsp

```

<html>
<usebean name=foo type=modell.Foo lifespan=page>
</usebean>
<ul>

```

```
<loop property=foo:list propertyelement=x>
<li> <display property=x>
</loop>
</ul>
</html>
```

JSP 與其它系統

與 Servlet 溝通

如何在 JSP 叫用 Servlet

0.92

1) FooServlet.java

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import modell.Foo;

public class FooServlet extends HttpServlet
{
    public void service(HttpServletRequest req,
        HttpServletResponse res)
        throws ServletException, IOException
    {
        String s[] = new String[] {
            "blue", "green", "red"
        };
        Foo f = new Foo(s);
        req.setAttribute("foo", f);

        getServletContext().getRequestDispatcher("/samples/modell/foo.
        jsp").forward
            (req, res);
    }
}
```

2) foo.jsp

```
<html>
<usebean name=foo type=modell.Foo lifespan=page>
</usebean>
<ul>
<loop property=foo:list propertyelement=x>
<li> <display property=x>
</loop>
</ul>
</html>
```

3) Foo.java

```
package modell;  
  
public class Foo {  
    String s[];  
    public String[] getList() { return s; }  
    public Foo(String s[]) { this.s = s; }  
}
```

如何傳遞資料給一個 Servlet

版本差異

1.0 與 1.0 Public Draft

1.0 到 1.0PD 之間只是修正一些語法使用習慣上的誤植，採用與 Java 語言相同的使用習慣，與一些小錯誤，基本上除了你用的軟體是以 1.0PD 為準外，不建議參考本文。

- Request-time 的屬性 expression 語法（"**<%=expr%>**"）編入強制實作的規格，這功能實作上不是問題，而且對於某些情況像很有用，例如與 **jsp:include** 指令配合，可以動態地插入資料
- **jsp:include** 動作語法與 **jsp:request include** 動作語法混合，這樣的簡化使得功能上少了一些；**jsp:request forward** 動作語法改為 **jsp:forward**。
- **page** 指令屬性改為大小寫混合，方便閱讀以及與 Java 語言的習慣配合，如下表格

舊名稱	新名稱
autoflush	autoFlush
isthreadsafe	isThreadSafe
iserrorpagq	isErrorPage
errorpage	errorPage

新增功能

- 增加 JSP 專用註解 **<%--...--%>**
- 加入 **page** 指令語法的 **contentType** 屬性
- 加入 **jsp:useBean** 標準動作語法的選擇性屬性：**type** 與 **beanName**

0.92 到 1.0

1.0 與 0.92 間有很大的規格變動，幾乎所有的語法都被變改了。

改變

- SSI 以 `<%@ include` 指令取代
- 標籤是大小寫有別的，如同 XML 規格一般
- 標準的標籤依照 Java 平台的大寫小混合的習慣來改寫（例如 `isErrorPage`，這跟 Java 語言定義 `identifier` 的習慣是一樣的
- 定義 `jsp:setProperty`、`jsp:getProperty`
- `<SCRIPT>...</SCRIPT>` 被取代成 `<%!...%>`

移除

NCSA-style SSI 不再存在於規格中

關於 NCSA Server-Side Include 你可以參考以下的網址

<http://hoohoo.ncsa.uiuc.edu/docs/tutorials/includes.html>

保留為下一版的功能

0.92 部份的功能在 1.0 中消失或成為選擇性支援的規格，預備放在 1.1 版中才會成為正式規格的一部份

目前有支援全域事件處理的只有 Livesoftware 的 JRun，因這本來就是他們產品提出的規格

- 全域事件處理支援；
- 流程控制的動作指令，如（`LOOP`、`ITERATE`、`INCLUDEIF`）已經被保留，這是為了等待 JSP 的 Tag 延伸機制的完成，以訂立更一致的功能；
- USEBEAN 的 `processRequest` 的機制

新增

- 加入 `jsp:request` 動作語法，提供執行時期的 `forward` 與 `include` 動態資源的能力
- 加入輸出的資料的緩衝機制
- `page` 指令

0.91 到 0.92

JSP 相關資源

軟體系統

JSP 引擎

WebSphere 0.91*

JavaServer Web Development Kit 1.0EA 1.0

文件資料

問答集

JavaSoft 的官方問答集

<http://java.sun.com/products/jsp/faq.html>

新聞群組或 Mailing List

LiveSoftware <news://news.livesoftware.com/livesoftware.jsp>

Mailing List 的收集區

JSP 官方 Mailing List 收集 <http://archives.java.sun.com/archives/jsp-interest.html>

Java/JSP/Servlet 世界觀

IBM 的 JSP 教學

<http://www.software.ibm.com/developer/education/java/online-courses.html>

IBM Red Book <http://www.redbooks.ibm.com/SG245423/sg245423.pdf>

IBM 其它與 JSP 有關的資訊

<http://www.software.ibm.com/webserver/appserv/doc/v20dcadv/doc/index.html>

Servlets Taverne

<http://www.interpasnet.com/JSS/>

Oi Servlet World
<http://i.am/servletforme>