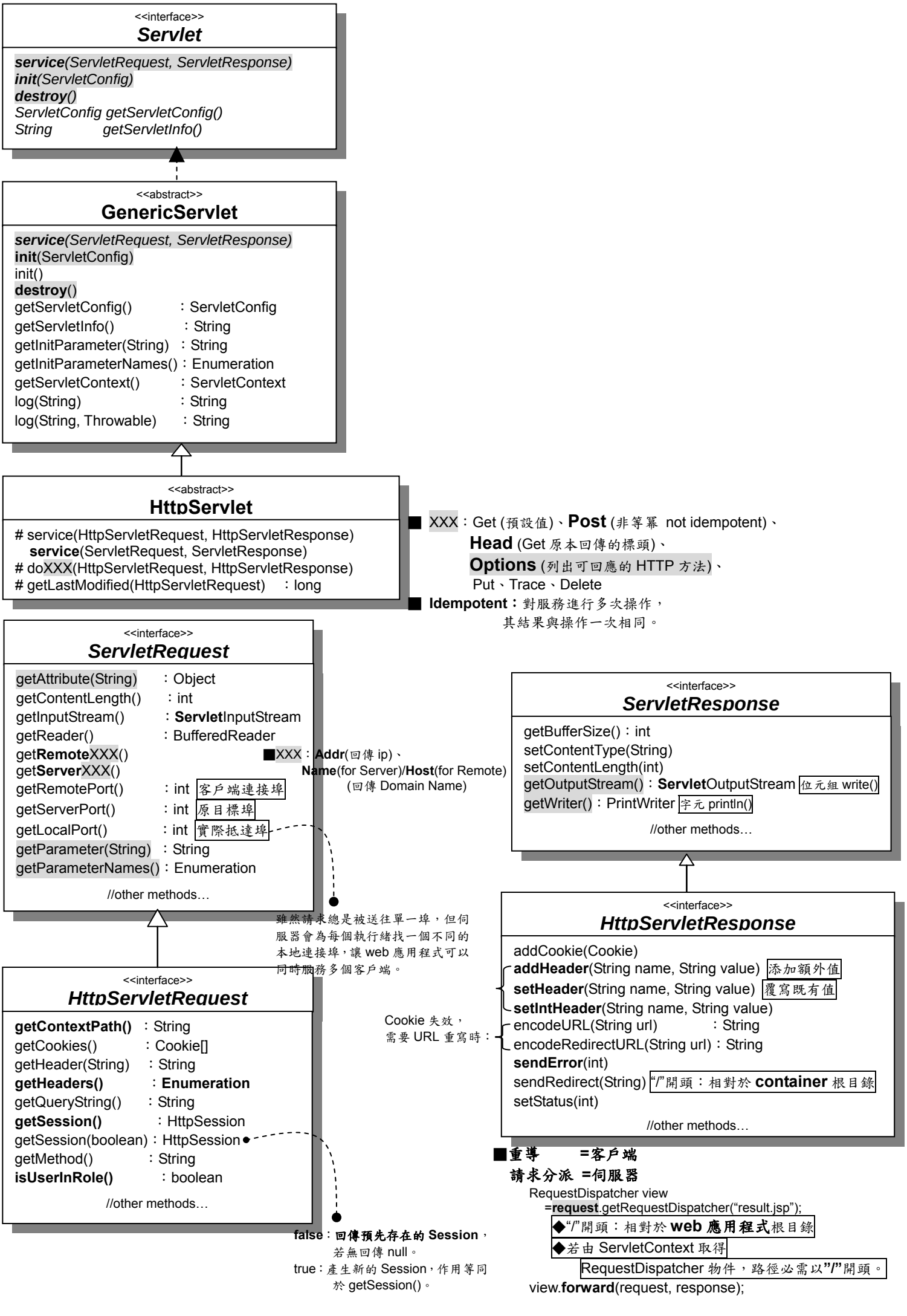
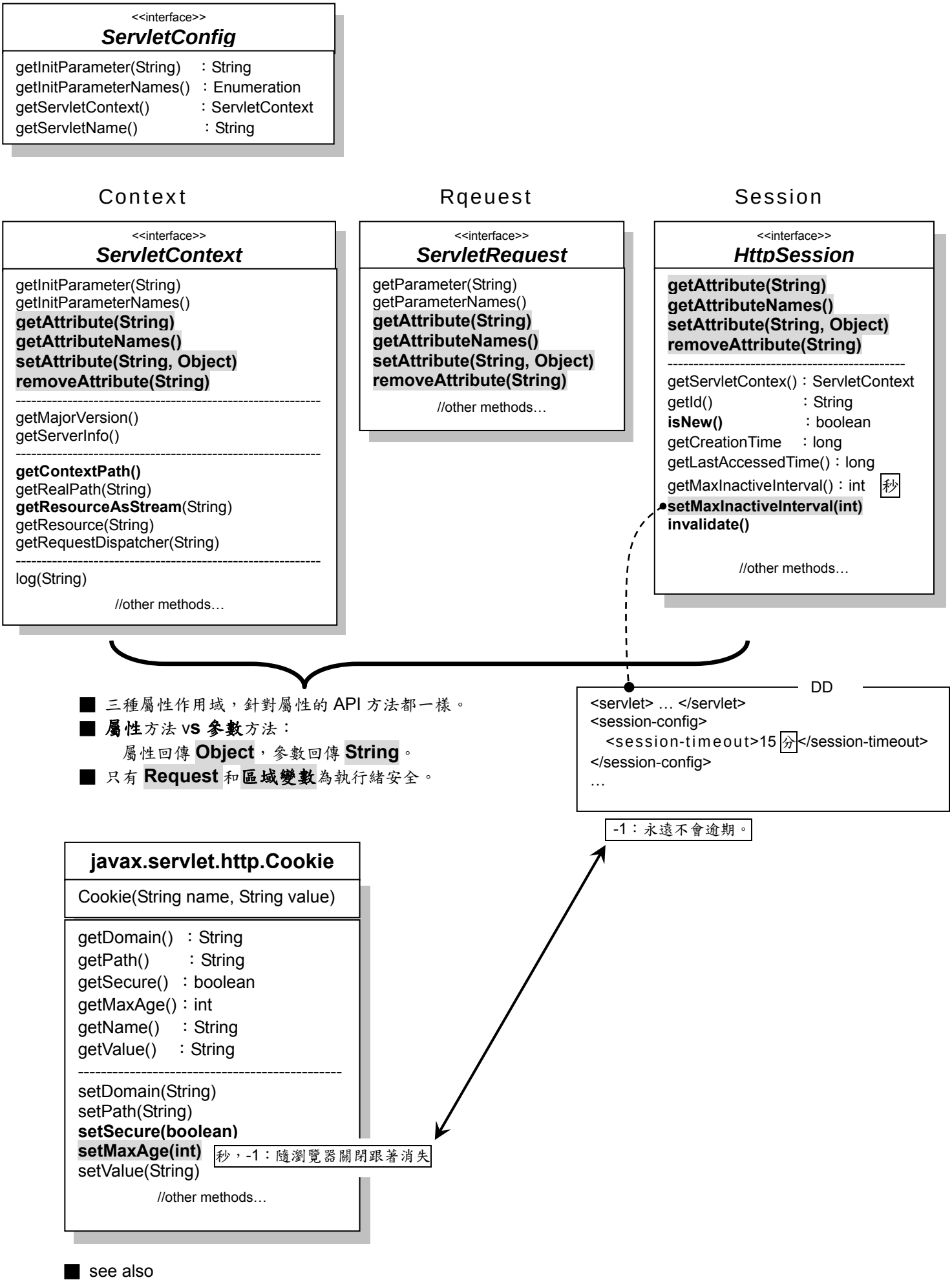

<<interface>>
Servlet
service(ServletRequest, ServletResponse)
init(ServletConfig)
destroy()
ServletConfig getServletConfig()
String getServletInfo()
<<abstract>>
GenericServlet
service(ServletRequest, ServletResponse)
init(ServletConfig)
init()
destroy()
getServletConfig() ：ServletConfig
getServletInfo() ：String
getInitParameter(String) ：String
getInitParameterNames() ：Enumeration
getServletContext() ：ServletContext
log(String) ：String
log(String, Throwable) ：String
<<abstract>>
HttpServlet
# service(HttpServletRequest, HttpServletResponse)
service(ServletRequest, ServletResponse)
# doXXX(HttpServletRequest, HttpServletResponse)
# getLastModified(HttpServletRequest) ：long
■ XXX：Get (預設值)、Post (非等冪 not idempotent)、
Head (Get 原本回傳的標頭)、
Options (列出可回應的 HTTP 方法)、
Put、Trace、Delete
■ Idempotent：對服務進行多次操作，
其結果與操作一次相同。
<<interface>>
ServletRequest
getAttribute(String) ：Object
getContentLength() ：int
getInputStream() ：ServletInputStream
getReader() ：BufferedReader
getRemoteXXX()
getServerXXX()
■XXX：Addr(回傳 ip)、
Name(for Server)/Host(for Remote)
(回傳 Domain Name)
getRemotePort() ：int 客戶端連接埠
getServerPort() ：int 原目標埠
getLocalPort() ：int 實際抵達埠
getParameter(String) ：String
getParameterNames() ：Enumeration
//other methods…
雖然請求總是被送往單一埠，但伺服器會為每個執行緒找一個不同的本地連接埠，讓 web 應用程式可以同時服務多個客戶端。
<<interface>>
HttpServletRequest
getContextPath() ：String
getCookies() ：Cookie[]
getHeader(String) ：String
getHeaders() ：Enumeration
getQueryString() ：String
getSession() ：HttpSession
getSession(boolean) ：HttpSession
getMethod() ：String
isUserInRole() ：boolean
//other methods…
false：回傳預先存在的 Session，
若無回傳 null。
true：產生新的 Session，作用等同
於 getSession()。
<<interface>>
ServletResponse
getBufferSize()：int
setContentType(String)
setContentLength(int)
getOutputStream()：ServletOutputStream 位元組 write()
getWriter()：PrintWriter 字元 println()
//other methods…
<<interface>>
HttpServletResponse
addCookie(Cookie)
addHeader(String name, String value) 添加額外值
setHeader(String name, String value) 覆寫既有值
setIntHeader(String name, String value)
Cookie 失效，
需要 URL 重寫時：
encodeURL(String url) ：String
encodeRedirectURL(String url)：String
sendError(int)
sendRedirect(String) "/"開頭：相對於 container 根目錄
setStatus(int)
//other methods…
■重導 =客戶端
請求分派 =伺服器
RequestDispatcher view
=request.getRequestDispatcher("result.jsp");
◆"/"開頭：相對於 web 應用程式根目錄
◆若由 ServletContext 取得
RequestDispatcher 物件，路徑必需以"/"開頭。
view.forward(request, response);


<<interface>>
ServletConfig
getInitParameter(String) ：String
getInitParameterNames() ：Enumeration
getServletContext() ：ServletContext
getServletName() ：String
Context
Rqeuest
Session
<<interface>>
ServletContext
getInitParameter(String)
getInitParameterNames()
getAttribute(String)
getAttributeNames()
setAttribute(String, Object)
removeAttribute(String)
getMajorVersion()
getServerInfo()
getContextPath()
getRealPath(String)
getResourceAsStream(String)
getResource(String)
getRequestDispatcher(String)
log(String)
//other methods…
<<interface>>
ServletRequest
getParameter(String)
getParameterNames()
getAttribute(String)
getAttributeNames()
setAttribute(String, Object)
removeAttribute(String)
//other methods…
<<interface>>
HttpSession
getAttribute(String)
getAttributeNames()
setAttribute(String, Object)
removeAttribute(String)
getServletContex()：ServletContext
getId() ：String
isNew() ：boolean
getCreationTime ：long
getLastAccessedTime()：long
getMaxInactiveInterval()：int 秒
setMaxInactiveInterval(int)
invalidate()
//other methods…
■ 三種屬性作用域，針對屬性的 API 方法都一樣。
■ 屬性方法 vs 參數方法：
屬性回傳 Object，參數回傳 String。
■ 只有 Request 和區域變數為執行緒安全。
DD
<servlet> … </servlet>
<session-config>
<session-timeout>15 分</session-timeout>
</session-config>
…
-1：永遠不會逾期。
javax.servlet.http.Cookie
Cookie(String name, String value)
getDomain() ：String
getPath() ：String
getSecure() ：boolean
getMaxAge()：int
getName() ：String
getValue() ：String
setDomain(String)
setPath(String)
setSecure(boolean)
setMaxAge(int)
秒，-1：隨瀏覽器關閉跟著消失
setValue(String)
//other methods…
■ see also
HttpServletRequest、HttpServletResponse.

Listener

使用情節	監聽器介面	事件類型
希望知道 Context 是否被建立或銷毀。	**ServletContextListener** context**Initialized** contextDestroyed	ServletContextEvent
希望知道 Context 中的屬性何時被新增、移除或取代。	**ServletContextAttributeListener** attributeAdded attributeRemoved attributeReplaced	ServletContextAttributeEvent
每當有請求進來時，都想要知道，以便加以記錄。	**ServletRequestListener** request**Initialized** requestDestroyed	ServletRequestEvent
希望知道 Request 屬性何時被新增、移除、修改。	**ServletRequestAttributeListener** attributeAdded attributeRemoved attributeReplaced	ServletRequestAttributeEvent
希望知道目前線上有多少使用者。	**HttpSessionListener** session**Created** sessionDestroyed	HttpSessionEvent
希望知道 Session 屬性何時被新增、移除、取代	**HttpSessionAttributeListener** attributeAdded attributeRemoved attributeReplaced	HttpSession**Binding**Event
當屬性類別的物件被繫結到 Session 或者從 Session 中移除時，你想要它們**本身**收到通知。	**HttpSessionBindingListener** value**Bound** valueUnbound	httpSessionBindingEvent
當屬性類別的物件所繫結的 Session 被移到另一個 JVM 或者從另一個 JVM 移過來時，希望它們**本身**得到通知。	**HttpSessionActivationListener** session**Did*Activate*** session**Will*Passive***	**HttpSessionEvent**

不需要在 DD 中組態，多半由屬性類別實作。

屬性類別：其物件會被設定成某屬性之值。

讓屬性在遷移前有機會對它們的實體變數進行**序列化**。Container 必需遷移可序列化的屬性。

HttpSessionEvent

getSession()

■ see also：**HttpServletRequest**

HttpSessionBindingEvent

getSession()
getName()
getValue() 傳回的是事件觸發前的舊值

JSP 語法

Directive（指令）

<%@ ***page*** import = ”foo.* ,java.util.*” %>
isThreadSafe 預設 true；不實作 SingleThreadModel
contextType
isElIgnored
isErrorPage
errorPage

<%@ ***taglib*** tagdir=”/WEB-INF/tags/cool”
prefix=”cool” %>

<%@ ***include*** file=”xx.html” %>

轉譯期插入

Scriptlet

<% out.prlntln(Counter.getCount()); %>

Expression（運算式）

<%= Counter.getCount() %>

不得在此宣告變數。

Declaration（宣告）

<%! Int count=0; %>

將會宣告出 Servlet 類別中的**成員變數**或**成員方法**。

(不同於 scriptlet 將宣告出_jspService() 方法中的**區域變數**或**方法操作**)

Comment（註解）

<%-- 這是註解 --%>

Action（動作）

執行期插入

<jsp:*include* page=”xx.html” />

■ See also：**<c:import>**

<c:*set* var=”rate” value=”32” />

EL expression

${ applicationScope.mail }

API	隱含物件
JspWriter	out
HttpServletRequest	request
HttpServletResponse	response
HttpSession	session
ServletContext	application
ServletConfig	config
Throwable	exception
PageContext	**pageContext**
Object	page

Page 指令的設定高於 DD。

<<interface>>
javax.servlet.jsp.JspPage

jspInit()
jspDestroy() **可以覆寫**(透過宣告)

<<interface>>
javax.servlet.jsp.HttpJspPage

_jspService(HttpServletRequest, HttpServletResponse)

不可以覆寫

?jsp_precompile：伺服器預先轉譯及編譯

JspContext

getOut()： JspWriter
getAttribute(String name)
getAttribute(String name, int scope)
getAttributeNames**InScope**(int scope)
getAttrubutesScope(String name)：int
findAttribute(String name)
//other methods…

針對任何作用域：
Page→Request→Session→Application

PageContext

APPLICATION_SCOPE
PAGE_SCOPE
REQUEST_SCOPE
SESSION_SCOPE
//other fields…
---
getRequest()
getServletConfig()
getServletContext()
getSession()
//other methods…

DD

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

Standard Action

```
<jsp:useBean  id="obj" class="foo.Father" scope="request" />
```

```
<jsp:useBean  id="obj" class="foo.Father" scope="request">
    <jsp:setProperty  name="obj" property="name" value="Distiny" />
</jsp:useBean>
```

僅在找不到 bean 且新的 bean 建立時，才會執行。

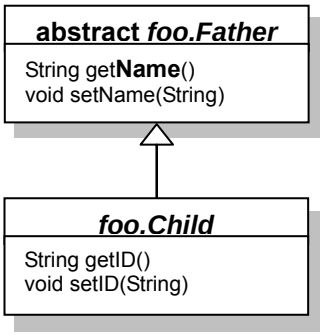
```
<jsp:useBean  id="obj" type="foo.Father" class="foo.Child" scope="page" />
```

選用，預設為 page。

```
foo.Father obj = null;
if(obj==null)
   obj = new foo.Child();
```

至少要有一個：

- 只有 type：bean 物件必需已經存在指定的作用域中。
- 有 class： **type** x = new **class**()。 class 必需滿足 Bean 法則。




```
<jsp:getProperty  name="obj" property="name" />
```

■ If nil，Exception！

```
<jsp:setProperty  name="obj" property="name" value="Distiny" />
```

```
<jsp:setProperty  name="obj" property="name" param="userName" />
```

對應至請求參數 userName

```
<jsp:setProperty  name="obj" property="name" />
```

可換成 property="*"，將自動設定所有參數。

當**請求參數名稱與 bean 的特性名稱相同**(都為"name")，不需要在標籤中指定。

(包含**字串**、**基本型別**的強制轉型，但若**使用 Scripting，將不會自動轉型**)

```
<jsp:include  page="header.jsp" />
```

```
<jsp:include  page="header.jsp">
    <jsp:param  name="subTitle" value="This is title." />
</jsp:include>
```

header.jsp

```
${param.subTitle}
```

```
<jsp:forward  page="xxx.jsp" />
```

千萬別在 forward 前作 flash，否則 forward 將失效。

EL expression

${ **firstThing** . secondThing }

EL 隱含物件 — **屬性**

Map 型別

- **pageScope** — 在 Page 作用域
- **requestScope** — 在 Request 作用域
- **sessionScope** — 在 Session 作用域
- **applicationScope** — 在 Application 作用域

用來取得屬性

- **param**
- **paramValues** — 請求參數
- **header**
- **headerValues**
- **cookie**
- **initParam** — Context 的初始參數
- **pageContext** — pageContext 實際參考，用來取得特性(getter)

Map 或 bean 物件

Map 的鍵(key) 或 Bean 的特性。

名稱必需符合 Java 識別名稱命名規則。

EL 算術運算子

- **+**
- **-**
- *****
- **/ 與 div** — N div 0 = **INFINITY**
- **% 與 mod** — N mod 0 = **Exception**

EL 關係運算子

- **== 與 eq**
- **!= 與 ne**
- **< 與 lt**
- **> 與 gt**
- **<= 與 le**
- **>= 與 ge**

EL 邏輯運算子

- **&& 與 and**
- **|| 與 or**
- **! 與 not**

Others

- **empty** — ${empty A} = **true/false**

■ EL 對未知的變數的處理，
算術表達式： **0**
邏輯表達式： **false**

${ **someList** ["something"] }

Map
bean
List
陣列

Map key
bean 特性
List 陣列 **index**

字串型別的索引值會被強制轉型為 int。
若**不是字串實字，就會被拿來評算**。
"a" 或 'a'

```
Map map = new HashMap();
map.put("A","1");
map.put("B","2");

request.setAttribute("B","A");

則$[ map[ B ] ] 將會被評算為
  $[ map[ "A" ] ] = "1"

另外，也可在括號內
使用巢狀的 EL 表達式。

$[ map[ map2[3] ] ]
```

EL 函式

function class

```
package foo;
public class ch8{
  public static int getInt() { return 5566; }
}
```

TLD

```
…
<taglib …>
  <tlib-version>1.2</tlib-version>
  <uri>myMethod</uri>
  <function>
      <name>do</name>
      <function-class>foo.ch8</function-class>
      <function-signature> int getInt() <function-signature>
  </function>
</taglib>
```

JSP

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

${ mine:do() }
```

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

```
<c:out value='${ pageContent.tip }' escapeXml='true' />
```
選用，預設為 true，進行 XML 實體轉換

防止 XSS 攻擊

```
<c:out value='${ user }' default='guest' />
```
若 value 屬性被評算為 null，
這個值將作為輸出。

≡ `<c:out value='${ user }'> guest </c:out>`

```
<c:forEach var='${ list }' items='${ movieLists }' varStatus='${ loopCount }' >
  ${ loopCount.count }
  <c:forEach var='${ movie }' items='${ list }'>
    ${ movie }
  </c:forEach >
</c:forEach >
```
選用，javax.servlet.jsp.jstl.core.LoopTagStatus

```
<c:forEach var="i" begin="1" end="20" step="1" >
    <c:out value="${i}" />
</c:forEach>
```

```
<c:if test="${ type eq 'ok' }" >
  doSomething
</c:if >
```

```
<c:choose>
  <c:when test="${ type == 'A' }" > A </c:when>
  <c:when test="${ type == 'B' }" > B </c:when>
  <c:otherwise> Others… </c:otherwise>
</c:choose>
```
← 選用

```
<c:set var="level" value="3" scope='session' />
```
不一定是字串，也可以是物件。　選用，預設 Page

```
<c:set var="level" scope='session' >
  3, 9, ${ a.level }
</c:set>
```

設定屬性變數 var

假如 value 被評算為 null，該變數會被移除！
(若沒指定 scope，由 page 開始找，接著 Request…)

假如 var 命名的屬性不存在，且 value 不是 null，屬性將會被建立。

```
<c:set target="${Person}" property="name" value="Distiny" />
```
必需是物件。　特性或鍵

```
<c:set target="${Person}" property="name" >
  ${foo.name}
</c:set>
```

設定 bean 或 Map 的特性或值

以下情況將會丟出例外
1. **target 被評算為 null**
2. **target 評算結果不是 bean 或 Map**
3. **target 是 bean，但找不到相符的 property 或找不到 value**

target 必需被評算為物件，故不能直接輸入字串實字。

target 屬性值可以 EL、Scripting 運算式、<jsp:attribute>表示。

若無指定 scope，只會在 Page 作用域中尋找屬性。

```
<c:remove  var="level" scope='session' />

<c:import  url='http://www.xxx.com/xx.html' />
```

see also：include 指令、<jsp:include>動作
不同於前兩種，url 可參照到 container 以外的資源。

動態插入

```
<c:import  url="header.jsp">
   <c:param  name="subTitle" value="This is title." />
</c:import>
```

header.jsp

```
${param.subTitle}
```

```
<c:url  value="/xxx.jsp" var="inputURL" />
```

URL 重寫

選用

```
<c:url  value="/xxx.jsp" >
   <c:param  name="firstN" value="${first}" />
   <c:param  name="lastN" value="${last}" />
</c:url>
```

URL 重寫 + 查詢字串編碼

■ See also：**<c:redirect>**

---

errPage.jsp

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

${pageContext.exception}
```

只有在指定 isErrorPage="true"的頁面中才有效

DD

```
<error-page>
   <exception-type>java.lang.Throwable</exception-type>
   <location>/errPage.jsp</location>
</error-page>

<error-page>
   <exception-type>java.lang.ArithmeticException</exception-type>
   <location>/arithmeticErrPage.jsp</location>
</error-page>

<error-page>
   <error-code>404</error-code >
   <location>/notFoundErr.jsp</location>
</error-page>
```

<location>**必需**相對於 web 應用程式的根目錄

badPage.jsp

```
<%@ page errorPage="errPage.jsp" %>
...

<c:catch  var="myException" >
   <% int x = 10/0; %>
   will not see this line.
</c:catch>

${myException.message}
```

只有在指的頁面中才能取得例外物件

將在 Page 作用域中建立例外屬性

※能呼叫 HttpServletResponse 的 sendError()方法來產生自己的錯誤代碼。

Ex：response.**sendError**(**HttpServletResponse.SC_FORBIDDEN**);
　　或 response.**sendError**(403);

TLD

```
<taglib …>
  <tlib-version> 1.2 </tlib-version>   必要
  <short-name>RandomTags</short-name>
  <function>
    … EL 函式
  </function>
  <uri>functions</uri>
  <tag>
    <description>doSomeThing</description>  選用
    <name>do</name>
    <tag-class>foo.myTagHandler </tag-class>
    <body-content>empty</body-content>
    <attribute>  選用
      <name>input</name>
      <type>xxx</type>  選用
      <required>true</required>
      <rtexprvalue>true</rtexprvalue>  選用
    </attribute>
  </tag>
</taglib>
```

Runtime Expression Value
True：可用 EL、Scripting 運算式、<jsp:attribute>
False：只能用字串實字

Java

```
package foo;
import javax.servlet.jsp.JspException;
import javax.servlet.jsp.tagext.SimpleTagSupport;
import java.io.IOException;

public class myTagHandler extends SimpleTagSupport{
  private String input;
  public void setInput(String input){ this.input = input; }

  public void doTag()
      throws JspException, IOException {
    getJspContext().getOut().write("xxx");
  }
}
```

JSP

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

<mine:do input="${someInput}"/>

<mine:do>
  <jsp:attribute name="input"> ${somInput}</jsp:attribute>
</mine:do>
```

DD (選用)

```
<web-app…>
  …
  <jsp-config>
    <taglib>
      <taglib-uri>functions</taglib-uri>
      <taglib-location>/WEB-INF/functions.tld</taglib-location>
    </taglib>
  </jsp-config>
</taglib>
```

可用 empty、scriptless、tagdependent(標籤主體當純文字處理)、JSP

Tag File

header.tag

```
<%@ attribute name="subTitle" required="true" rtexprvalue="true" %>
<img src="images/xx.jpg">
${subTitle}
```

只適用在 Tag File

JSP

```
<%@ taglib prefix="myTags" tagdir="/WEB-INF/tags" %>
<myTags:header subtitle="xxx" />
```

header.tag

```
<%@ tag body-content="tagdependent" %>
<img src="images/xx.jpg">

<jsp:doBody/>
```

JSP

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

Tag File 主體中絕不允許 Scripting。

預設為 scriptless，
可選擇 empty 或 tagdependent。

※Tag File 中無法使用 page 指令。
tag、<jsp:doBody>、<jsp:invoke>只能用在 Tag File 中。

Tag 隱含物件：

request
response
jspContext
session
application
out
config

(沒有 pageContext、page 與 exception，多了 jspContext)

使用標籤時須注意的保留字：

jsp、jspx、
java、javax、
servlet、sun、sunw

A123Hello World!B Hello World!

<jsp:invoke>、<jsp:doBody>用法

testT.tag

```
<%@ attribute name="frag" fragment="true" %>
<%@ variable name-given="code" scope="application" %>

<jsp:doBody var="code" />  將 body 內容設定給 code 變數
                           這裡也能宣告 scope

123
<jsp:invoke fragment="frag"/>
```

JSP

```
<%@ taglib prefix="mine" tagdir="/WEB-INF/tags" %>
A
<mine:testT>
   <jsp:attribute name="frag">
      ${ code }
   </jsp:attribute>

    <jsp:body> Hello World! </jsp:body>
</mine: testT>
B
${code}   code 變數被宣告為 application，故可在此使用。
```

Dynamic Attributes

Java

```
package foo;
…
import javax.servlet.jsp.tagext.DynamicAttributes;

public class myTagHandler implements DynamicAttributes{
   private String name;
   public void setName(String value){ this.name=value; }

   private Map<String,Object> tagAttrs =
                              new HashMap<String,Object>();
   public void setDynamicAttributes(String uri,
                                    String name, String value){
      tagAttrs.put(name, value);
   }
   …
```

TLD

```
<taglib …>
   <tlib-version> 1.2 </tlib-version>
   <short-name>RandomTags</short-name>
   <uri>functions</uri>
   <tag>
      <name>do</name>
      <tag-class>foo.myTagHandler </tag-class>
      <body-content>empty</body-content>
      <attribute>
         <name>name</name>
         <required>true</required>
      <attribute>
      <dynamic-attributes>true</dynamic-attributes>
   </tag>
</taglib>
```

或

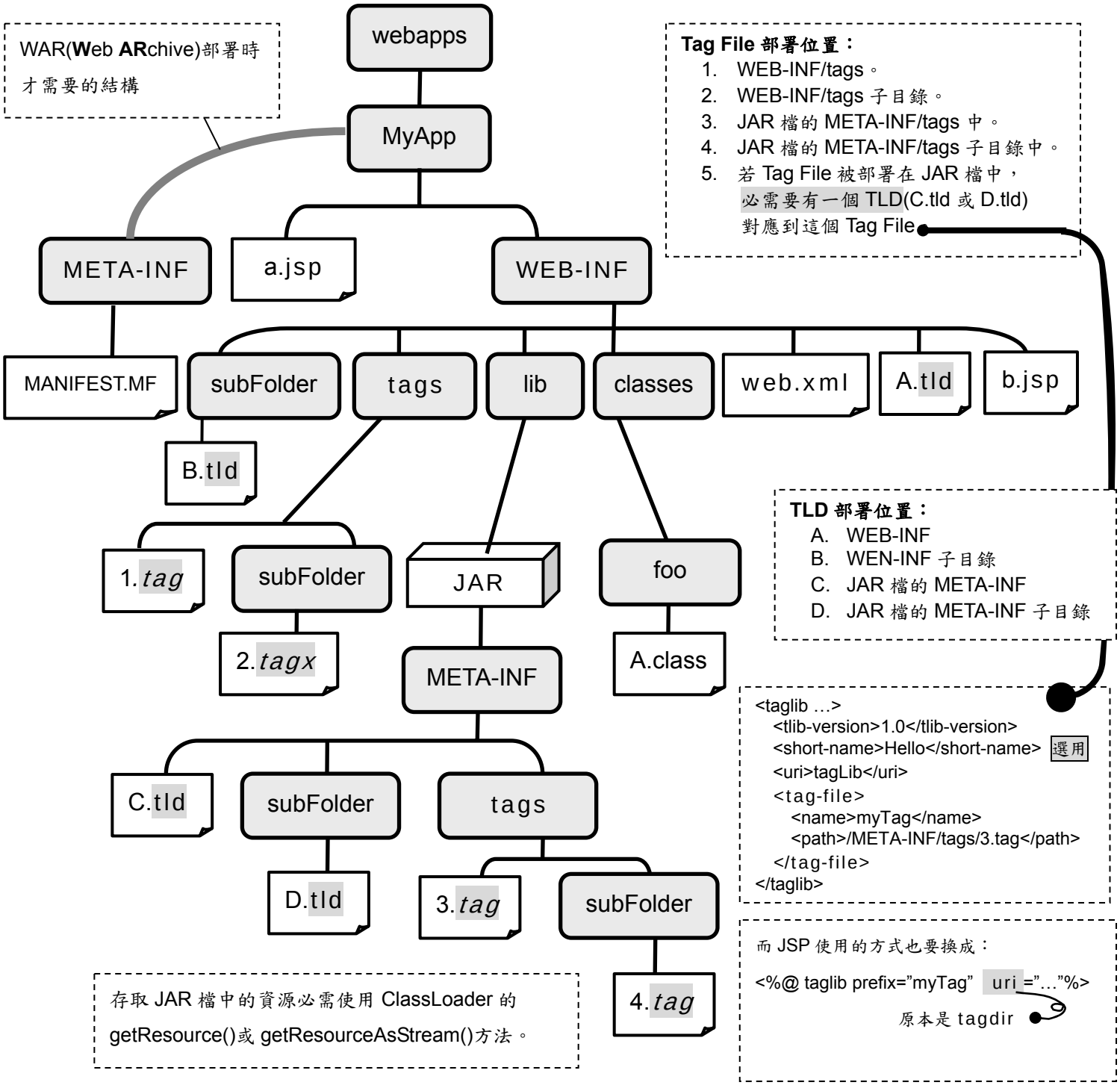
Tag File

```
<%@ tag body-content="empty" dynamic-attributes='tagAttrs' %>
<%@ taglib uri="..." prefix="c" %>          將產生 Page 作用域的 hashmap

<c:forEach var="attrEntry" items="${ tagAttrs}" >
   ${attrEntry.key} = '${attrEntry.value}'
</c:forEach >
```

Web 應用程式部署目錄結構

在 WEB-INF **或** META-INF 目錄下
沒有任何東西可被直接存取！



Simple Tag

JSP

```
<%@ taglib prefix="mine" uri="functions" %>
…
<table>
  <mine:do movieList="${movieCollect}">
    <tr>
      <td>${movie.name}</td>
      <td>${movie.price}</td>
    </tr>
  </mine:do>
</table>

This line will not show.
```

TLD

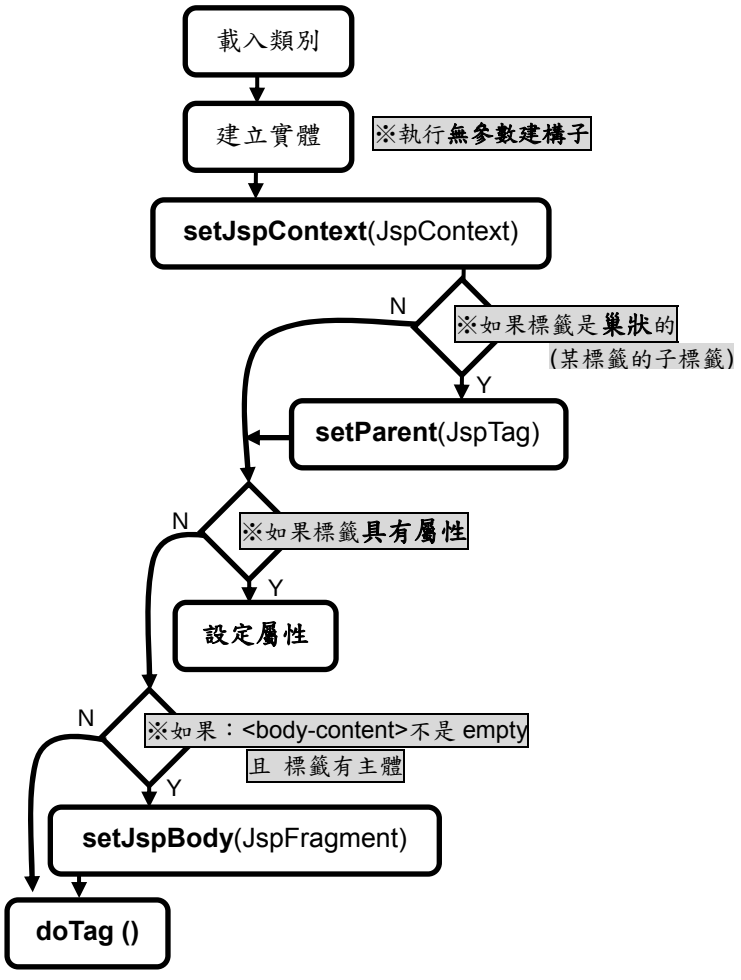
```
<taglib …>
  …
  <tag>
    <name>do</name>
    <tag-class>foo.myTagHandler </tag-class>
    <body-content>scriptless</body-content>
    <attribute>
      <name>movieList</name>
      <required>true</required>
      <rtexprvalue>true</rtexprvalue>
    <attribute>
  </tag>
</taglib>
```

JAVA

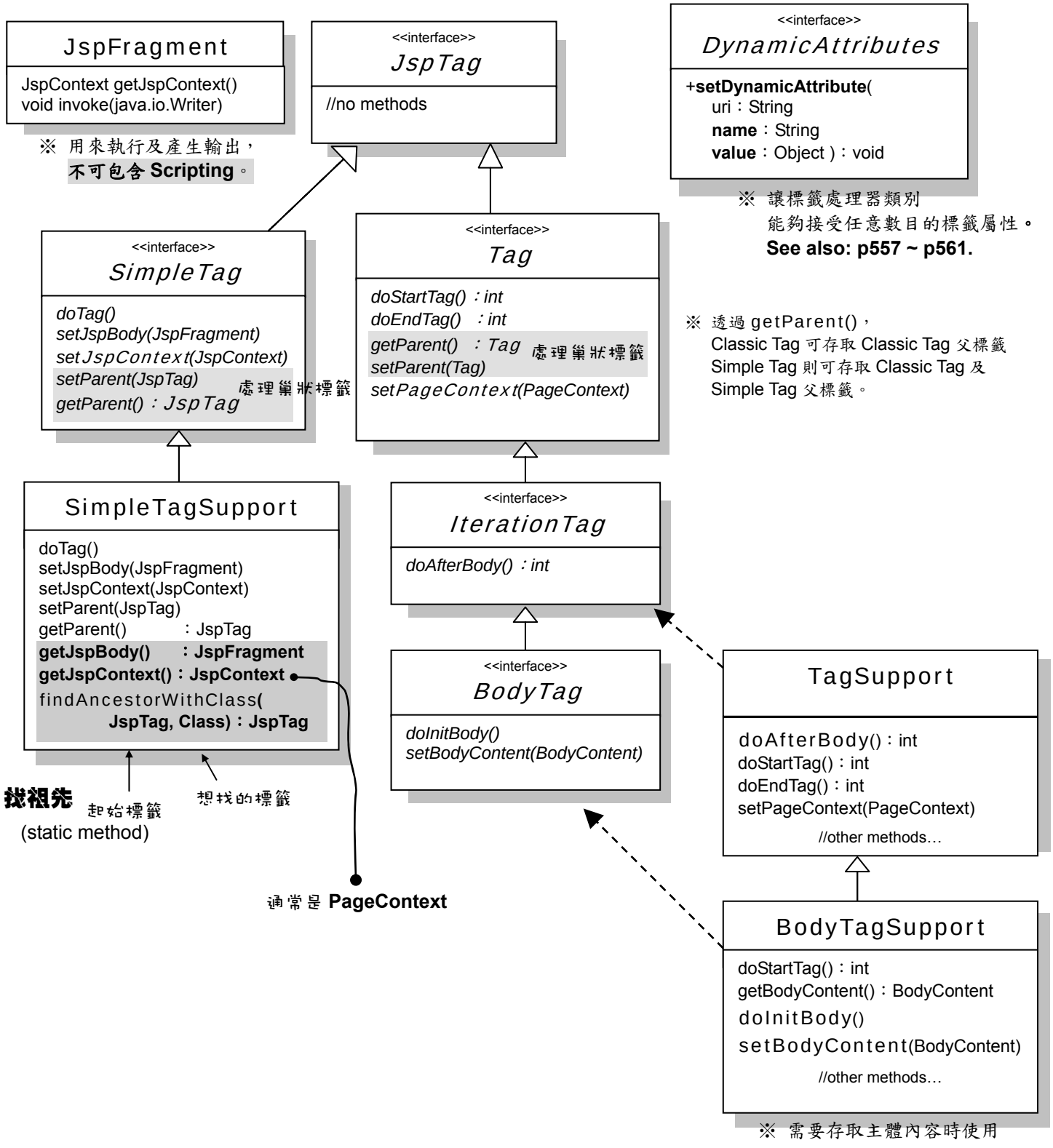
```
package foo;
import javax.servlet.jsp.tagxt.SimpleTagSupport;
…
public class myTagHandler extends SimpleTagSupport{
  private List movieList;
  public void setMovieList(List movieList){
    this.movieList = movieList;
  }

  public void doTag()   throws JspException, IOException {
    Iterator i = movieList.iterator();
    while(i.hasNext()){
      Movie movie = (Movie) i.next();
      getJspContext().setAttribute("movie", movie);
      getJspBody().invoke(null); 將標籤主體輸出到回應;
    }
    throw new SkipPageException();
  }
}
```

※每個標籤處理器實例只負責一次呼叫。



(Simple Tag 處理器的生命週期)

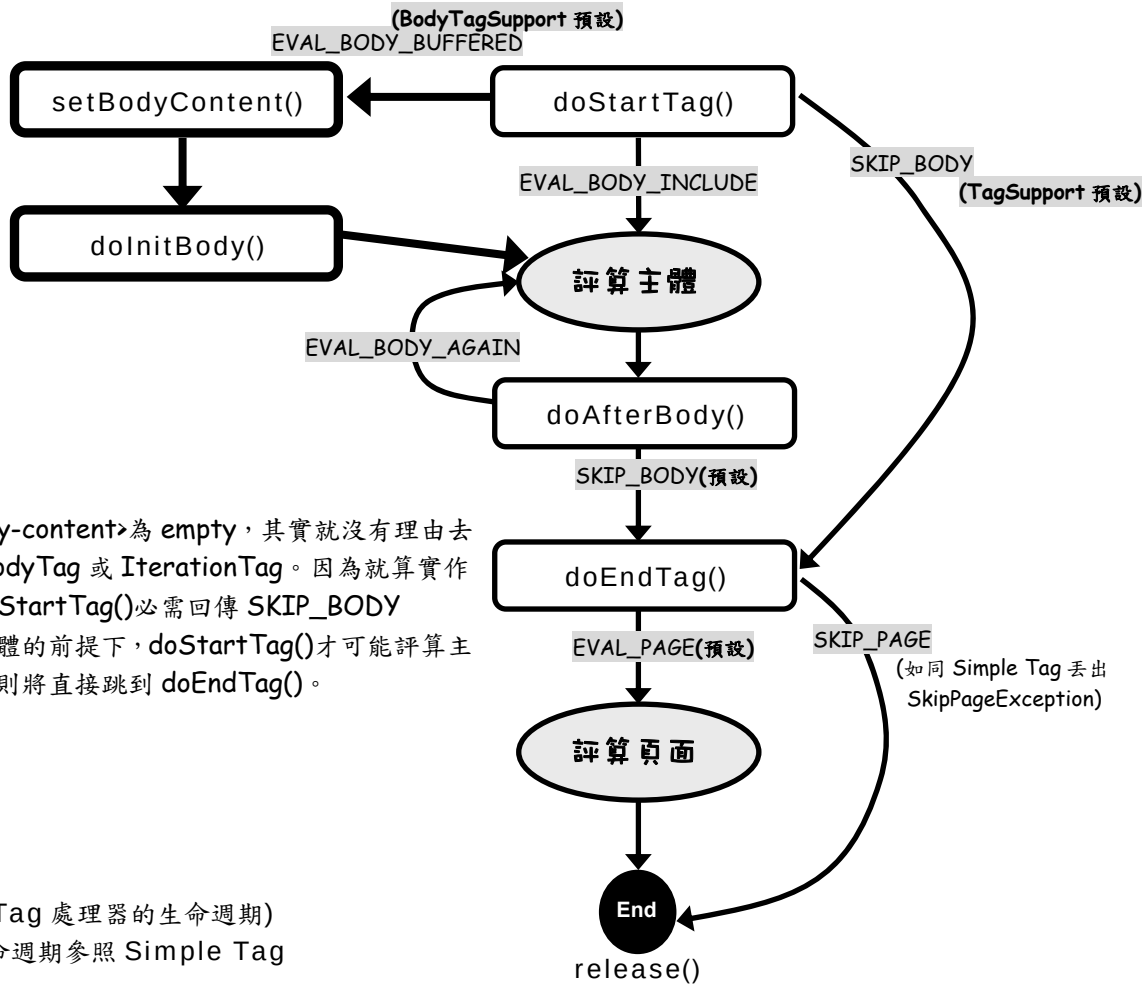

JspFragment
JspContext getJspContext()
void invoke(java.io.Writer)
※ 用來執行及產生輸出，
不可包含 Scripting。
<<interface>>
JspTag
//no methods
<<interface>>
DynamicAttributes
+setDynamicAttribute(
uri：String
name：String
value：Object)：void
※ 讓標籤處理器類別
能夠接受任意數目的標籤屬性。
See also: p557 ~ p561.
<<interface>>
SimpleTag
doTag()
setJspBody(JspFragment)
setJspContext(JspContext)
setParent(JspTag)
getParent()：JspTag
處理巢狀標籤
<<interface>>
Tag
doStartTag()：int
doEndTag()：int
getParent()：Tag
setParent(Tag)
setPageContext(PageContext)
處理巢狀標籤
※ 透過 getParent()，
Classic Tag 可存取 Classic Tag 父標籤
Simple Tag 則可存取 Classic Tag 及
Simple Tag 父標籤。
SimpleTagSupport
doTag()
setJspBody(JspFragment)
setJspContext(JspContext)
setParent(JspTag)
getParent() ：JspTag
getJspBody() ：JspFragment
getJspContext()：JspContext
findAncestorWithClass(
JspTag, Class)：JspTag
<<interface>>
IterationTag
doAfterBody()：int
<<interface>>
BodyTag
doInitBody()
setBodyContent(BodyContent)
TagSupport
doAfterBody()：int
doStartTag()：int
doEndTag()：int
setPageContext(PageContext)
//other methods…
BodyTagSupport
doStartTag()：int
getBodyContent()：BodyContent
doInitBody()
setBodyContent(BodyContent)
//other methods…
※ 需要存取主體內容時使用
找祖先
起始標籤
想找的標籤
(static method)
通常是 PageContext

Classic Tag

※Container 可能會在之後重新利用此物件。
最好在每次標籤被呼叫時，重新設定實例變數。

```
package foo;
import javax.servlet.jsp.tagxt.;
…
public class myTagHandler extends TagSupport{
   JspWriter out;
                                          相較 SimpleTag，沒有 IOException
   public void doStartTag()   throws JspException{
      out = pageContext.getOut();  相較 SimpleTag，getJspContext()
      try{
         out.println("doStartTag()");
      }catch(IOException ex){
         throw new JspException("IOException-"+ex.toString());
      }
      return EVAL_BODY_INCLUDE;   相較 SimpleTag，getJspBody().invoke(null)
   }

   public int doEndTag() throws JspException{
      try{
         out.println("doEndTag()");
      }catch(IOException ex){
         throw new JspException("IOException-"+ex.toString());
      }
      return EVAL_PAGE;
   }
}
```



※若<body-content>為 empty，其實就沒有理由去實作 BodyTag 或 IterationTag。因為就算實作了，doStartTag()必需回傳 SKIP_BODY

※在有主體的前提下，doStartTag()才可能評算主體，否則將直接跳到 doEndTag()。

(Classic Tag 處理器的生命週期)
前半段生命週期參照 Simple Tag

WEB 應用程式部署（以下將是一個不錯長的 web.xml 檔）

```
<web-app ...>

    <welcome-file-list>
      <welcome-file>index.html</welcome-file>
       <welcome-file>default.jsp</welcome-file>
    </welcome-file-list>
```

將依序指定檔案順序找尋各目錄。
Ex: http://localhost/**test/search**
則將在 **test/search** 中依序找尋 index.html、default.jsp

裡面元素不得以斜線為開頭或結束！

```
    <error-page>
      <exception-type>java.lang.Throwable</exception-type>
      <location>/errPage.jsp</location>
    </error-page>

<error-page>
    <exception-type>java.lang.ArithmeticException</exception-type>
    <location>/arithmeticErrPage.jsp</location>
</error-page>

<error-page>
    <error-code>404</error-code>
    <location>/notFoundErr.jsp</location>
</error-page>
```

```
    <context-param>
      <param-name>adminEmail</param-name>
      <param-value>admin@www.com</param-value>
    </context-param>

    <servlet>
      <servlet-name>ch5</servlet-name>
      <servlet-class>com.example.paramTest</servlet-class>
      <init-param>
          <param-name>email</param-name>
          <param-value>name@www.com</param-value>
      </init-param>
      <load-on-startup> 2 </load-on-startup>
    </servlet>

    <servlet-mapping>
        <servlet-name>ch5</servlet-name>
        <url-pattern>/ch5.do</url-pattern>
    </servlet-mapping>

<servlet>
    <servlet-name>ch3</servlet-name>
    <servlet-class>com.example.web.BeerSelect</servlet-class>
    <load-on-startup> 0 </load-on-startup>
</servlet>

<servlet-mapping>
    <servlet-name>ch3</servlet-name>
    <url-pattern>/ch3.do</url-pattern>
</servlet-mapping>
```

Servlet 對應規則：

1.先依照以下順序找尋符合條件者

A.完全相符：**/**Beer/Select.do

B.目錄相符：**/**Beer**/***

C.副檔名相符：***.do**

2.若請求符合一個以上的<url-pattern>，越明確的符合狀況 Container 將會優先選擇。

Ex：針對**/foo/bar/xx.do** 的請求將會對應到**/foo/bar/***，即使它也符合**/foo/***

選用。讓 ServletServlet 在部署期間
依照指定的優先權(0:最優先~n)被載入，而非第一次被請求時。
Ex： ch3 → ch5

```
    <listener>
      <listener-class>com.example.myServletContextListener</listener-class>
    </listener>
```

```
    <session-config>
      <session-timeout> 15 </session-timeout>
    </session-config>
```

```
<servlet>
    <servlet-name>ch7</servlet-name>
    <jsp-file>/ch7.jsp</jsp-file>
    <init-param>
        <param-name>email</param-name>
        <param-value>name@www.com</param-value>
    </init-param>
</servlet>

<servlet-mapping>
    <servlet-name>ch7</servlet-name>
    <url-pattern>/ch7.jsp</url-pattern>
</servlet-mapping>
```

針對 JSP 頁面的部署。

```
<jsp-config>
    <jsp-property-group>
        <url-pattern>*.jsp </url-pattern>
        <el-ignored> false </el-ignored>
    </jsp-property-group>
</jsp-config>
```

```
<ejb-local-ref>
    <ejb-ref-name>ejb/Customer</ejb-ref-name>
    <ejb-ref-type>Entity</ejb-ref-type>
    <local-home>com.xx.CustomerHome</local-home>
    <local>com.xx.Customer</local>
</ejb-local-ref>

<ejb-ref>
    <ejb-ref-name>ejb/LocalCustomer</ejb-ref-name>
    <ejb-ref-type>Entity</ejb-ref-type>
    <home>com.xx.CustomerHome</home>
    <remote>com.xx.Customer</remote>
</ejb-ref>
```

```
<env-entry>
    <env-entry-name>rates/discountRate</env-entry-name>
    <env-entry-type>java.lang.Integer</env-entry-type>
    <env-entry-value>10</env-entry-value>
</env-entry>
```

接受任何型別，只要它有建構式 xxClass(String)。
或在<env-entry-type>是 java.lang.Character 的情況下，有建構式 xxxClass(Character)

```
<mine-mapping>
    <extension> mpg </extension>
    <mine-type> video/mpeg </mine-type>
</mine-mapping>
```

WEB 應用程式的安全性

啟用認證。 使用者經過認證(authentication)後，才能夠被授權(authorization)！

in 傳輸層。

```
<login-config>
    <auth-method> BASIC </auth-method>
<login-config>
```

可用 BASIC(使用 Base64)、CLIENT-CERT(使用 PKC)、DIGEST、FORM(可自訂登入畫面、無加密)

```
<login-config>
    <auth-method> FORM </auth-method>
    <form-login-config>
        <form-login-page>/login.html</form-login-page>
        <form-error-page>/loginErr.html</form-error-page>
    </form-login-config>
<login-config>
```

login.html

```
<form method="POST" action="j_security_check">
    <input type="text" name="j_username">
    <input type="password" name="j_password">
    <input type="submit" value="Enter">
</form>
```

使用 FORM 務必打開 SSL 或 Session Tracking！

授權，定義角色。

```
<security-role><role-name>Admin</role-name></security-role>
<security-role><role-name>Member</role-name></security-role>
<security-role><role-name>Guest</role-name></security-role>
```

對映到廠商特定的使用者設定檔
ex: tomcat-users.xml

授權，定義資源/方法的存取限制。

```
<security-constraint>                                  零至多個。
  <web-resource-collection>
    <web-resource-name>someNameForTheKit</web-resource-name>
    <url-pattern>/foo/update/*</url-pattern>           一或多個。
    <url-pattern>/foo/delete/*</url-pattern>
    <http-method>GET</http-method>                     零至多個。
    <http-method>POST</http-method>                    未指定的方法不受限制！
  </web-resource-collection>                           一或多個。
  <auth-constraint>
    <role-name>Admin</role-name>
    <role-name>Member</role-name>                      角色名稱大小寫有差!
  </auth-constraint>
  <user-data-constraint>
    <transport-guarantee>CONFIDENTIAL</transport-guarantee>
  </user-data-constraint>                              資料完整性與機密性。
</security-constraint>
```

當有多個<security-constraint>交互作用…

A.以下情況結合任何其他東西，表示任何人皆可存取資源：
1.沒有<auth-constraint>元素
2.<role-name>*</role-name>

B.<auth-constraint/>空標籤結合任何其他東西，表示沒有人可存取。

C.不為 A 或 B 的前提，所有聯集的角色都可存取。

NONE、INTEGRAL、CONFIDENTIAL 叫 Container 在送出請求前，先切換成 SSL

(多半使用 SSL)

```
<servlet>
  <security-role-ref>
    <role-name>Manager</role-name>
    <role-link>Admin<role-link>
  </security-role-ref>
  ...
<servlet>
```

對映到<security-role-ref>定義的角色

Servlet

```
if( request.isUserInRole("Manager"){  … }
else{ … }
```

HttpServletRequest 中，與程式安全性有關的三個方法：
getUserPrinciple()：EJB
getRemoteUser()：認證判斷
isUserInRole()：授權判斷

過濾器就像 Servlet!

```
<filter>
   <filter-name>xxxRequest</filter-name>
   <filter-class>com.example.xxxRequestFilter</filter-class>
   <init-param>
      <param-name>logFileName</param-name>
      <param-value>userLog.txt</param-value>
   <init-param>
</filter>

<filter-mapping>
   <filter-name>xxxRequest</filter-name>
   <url-pattern>*.do</url-pattern>
</filter-mapping>

<filter-mapping>
   <filter-name>xxxRequest</filter-name>
   <servlet-name>xxxServlet</servlet-name>
   <dispatcher>REQUEST</dispatcher>
</filter-mapping>
```

過濾器的功能：
- 請求部分：
  - 執行安全性檢查
  - 重新格式化請求的標頭或主體
  - 稽查或記錄請求
- 回應部分：
  - 壓縮回應串流
  - 附加或修改回應串流
  - 建立不同的回應

至少要有一個。先比對<url-pattern>依照 DD 的宣告順序放入過濾器鏈，接著才比對<servlet-name>。

零至四個。

</web-app>

過濾器可用在請求委派器或錯誤處理器：
REQUEST(預設)、INCLUDE、FORWARD、ERROR。

Filter

```
package com.example;
…

public class xxxRequestFilter implements Filter{
   private FilterConfig fc;

   public void init(FilterConfig config) throws ServletException{
      this.fc = config;
   }

   public void doFilter(ServletRequest req,
                        ServletResponse resp,
                        FilterChain chain)
                  throws ServletException, IOException{
      String name = ((HttpServletRequest)req).getRemoteUser();
      if(name!=null) fc.getServletContext().log("user "+name);
      chain.doFilter(req, resp);
   }

   public void destroy(){ }
}
```

呼叫下一個過濾器或 Servlet！

將過濾器想成「概念性的呼叫堆疊」

想要建立自訂請求或回應物件時

```
class CompressionResponseWrapper
        extends HttpServletResponseWrapper{
   //覆寫任何想要客制化的方法
}
```

```
class MyCompressionFilter implements Filter{
   public void init(FilterConfig fc){}

   public void doFilter(request, response, chain){
      CompressionResponseWrapper crw =
         new CompressionResesopnseWrapper(response);
      //在此進行壓縮處理
   }

}
```

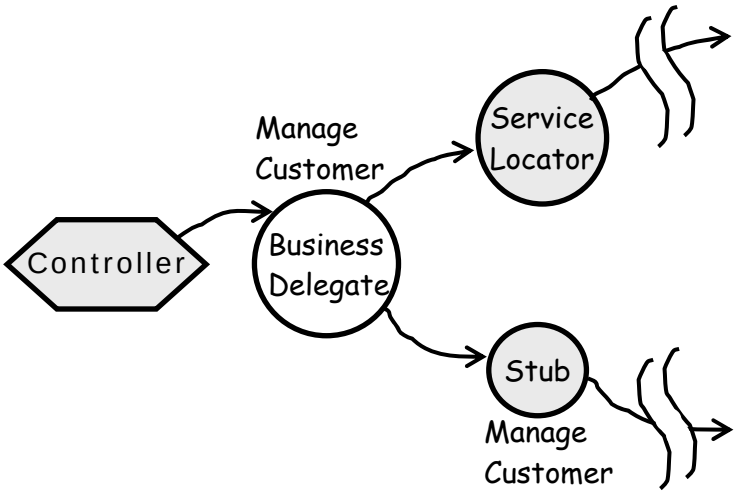
JSP Document (p.629)

	一般 JSP 頁面語法	JSP Document 語法
指令 (不含 taglib)	<%@ page import=”java.util.*” %>	<jsp:directive.page import=”java.util.*”/>
宣告	<%! int y = 3; %>	<jsp:declaration> int y = 3; </jsp:declaration>
Scriptlet	<% list.add(“A”); %>	<jsp:scriptlet>list.add(“A”); </jsp:scriptlet>
文字	Line A.	<jsp:text>Line A</jsp:text>
Scripting 運算式	<%= it.next() %>	<jsp:expression>it.next()</jsp:expression>

J2EE Design Patterns

Business Delegate

讓 WEB 層的 Controller 元件
免於直接與應用程式的遠端 Model 元件進行溝通。



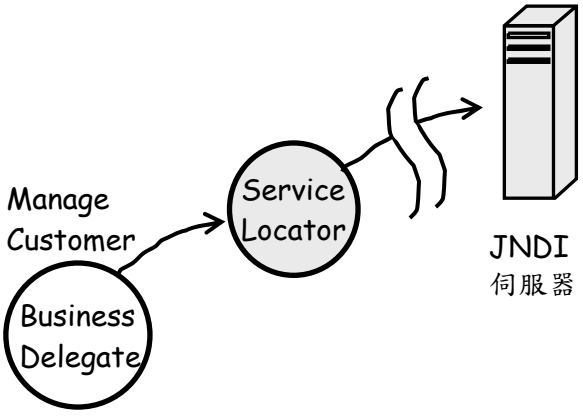
- 作為代理人，實作遠端服務的介面。
- 開啟與遠端服務間的溝通。
- 處理**溝通的細節**與**例外**。
- 轉化 Controller 取得的請求，
  並將它轉交給商業服務(透過 Stub)。
- 轉化回應並將它回傳給 Controller。
- 處理遠端元件查詢與溝通的細節，
  讓 Controller 更有內聚力(cohesion)。

★ 奠基於：
- 隱藏複雜度
- **針對介面撰寫程式碼**
- 降低隅合度
- 關注點分離

- 最小化商業層發生變動時，對 Web 層產生的衝擊。
- 降低各分層間的隅合性。
- 在應用程式中增加分層，**複雜度也將增加**。
- 方法呼叫必需是**粗粒度的(coarse-grained)**，
  以降低網路的負擔。

Service Locator

簡化所有需要進行 JNDI(或其他註冊機制)查詢的其他元件。



- 取得 InitialContext 物件。
- 查詢註冊機制。
- 處理溝通的細節與例外。
- **藉由快取增進執行效能**。
- 可搭配多種註冊機制，
  如 JNDI、RMI、UDDI 和 COS naming。

★ 奠基於：
- 隱藏複雜度
- 關注點分離

- 最小化**遠端元件改變位置或 Container** 時對 Web 層產生的衝擊。
- 降低各分層之間的隅合性。