

# 【目錄】

## 第一章(Servlet)

1-1 Servlet簡介	004
1-2 Server環境安裝	015
1-3 撰寫Servlet入門	031
1-4 初識請求與回應	039
1-5 Servlet的生命週期	044
1-6 取得Server&client的相關資訊	052
1-7 Response Header相關應用	060
1-8 Session Tracking	064
1-9 傳遞資訊	071
1-10 資料庫連結	075

## 第二章(JSP)

2-1 JSP介紹	082
2-2 EL表達式語言	096
2-3 JSTL標準標籤函式	102

## 第三章(完善專案)

3-1 MVC架構	115
3-2 Listener	121
3-3 Filters	125
備註	129

# JAVAAEE WEB元件開發 SERVLET/JSP

洪子敬

2

# 第一章

# SERVLET

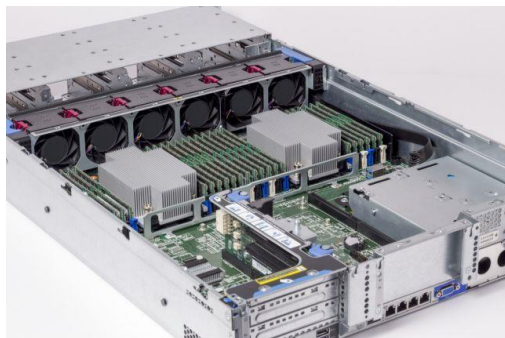
# 第一節

# **SERVLET**簡介

- Server是「向Client(客戶端)提供特定服務的電腦」
- Servlet是「在Server(伺服器)上運行的程式」



華碩 ASUS RS700-E9-RS4 1U機架, 熱抽  
支援 2 x Socket P (LGA 3647) Intel Xeon Scalable Processors Family (145W/165W)  
支援DDR4 2666/2400 RDIMM插槽x24, (每顆CPU支援6通道,12個插槽),最大至 3072GB LR-DIMM  
DVDRW 燒錄機  
支援4個3.5"熱抽硬碟槽(硬碟\*選購), 2 x M.2  
SATA 控制器: Intel Lewisburg PCH, 9 x SATA3 6Gb/s + 2 x M.2連接埠 (支援軟體RAID 0,1,5,10) (SAS Raid選購)



## 何謂SEVLET?

Servlet(Server Applet), 全稱Java Servlet。是用Java編寫的伺服器端程式。其主要功能在於互動式地瀏覽和修改資料, 生成動態Web內容。狹義的Servlet是指Java語言實現的一個介面, 廣義的Servlet是指任何實現了這個Servlet介面的類別。一般情況下, 人們將Servlet理解為後者。

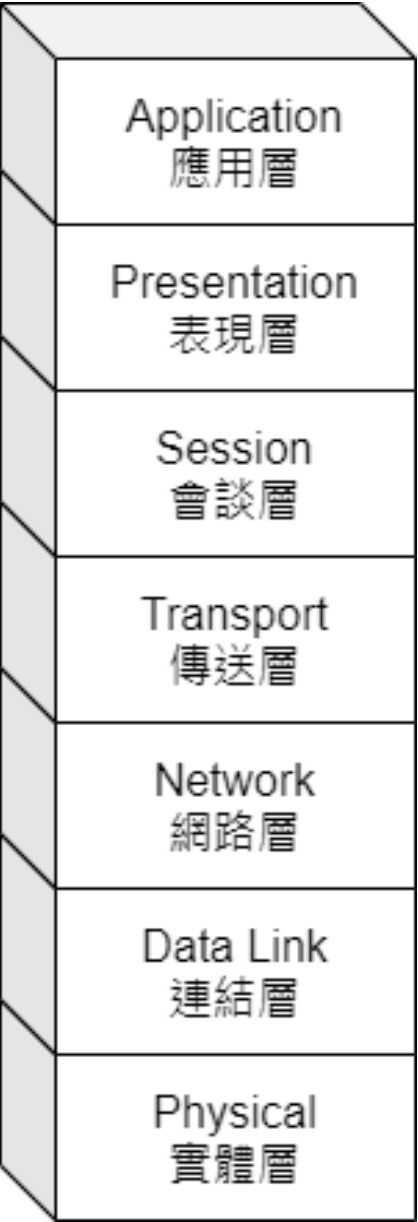
Servlet執行於支援Java的應用伺服器中。從實現上講, Servlet可以回應任何類別型的請求, 但絕大多數情況下Servlet只用來擴充基於HTTP協定的Web伺服器。

最早支援Servlet標準的是JavaSoft的Java Web Server。此後, 一些其它的基於Java的Web伺服器開始支援標準的Servlet。

<維基百科>



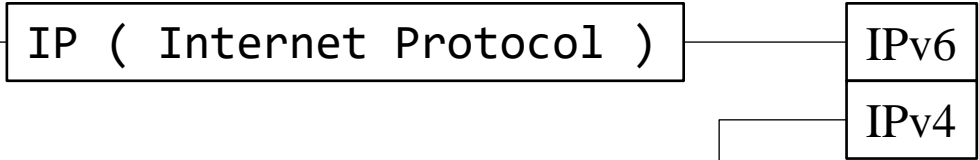
# OSI 7 Layer



HTTP (HyperText Transfer Protocol)、FTP (File Transfer Protocol)、DNS(Domain Name System)、SMTP (Simple Mail Transfer Protocol)

	TCP	UDP
特性	可靠連線	非可靠連線
速度	慢	快
穩定性	高	低
範例	檔案傳送	影片串流

TCP (Transmission Control Protocol)、UDP (User Datagram Protocol)



等級	範圍	私有網段
A	0.0.0.0~126.0.0.0 /8	10.0.0.0~10.255.255.255
B	128.0.0.0~191.255.0.0 /16	172.16.0.0~172.31.255.255
C	192.0.0.0~223.255.255.0 /24	192.168.0.0~192.168.255.255

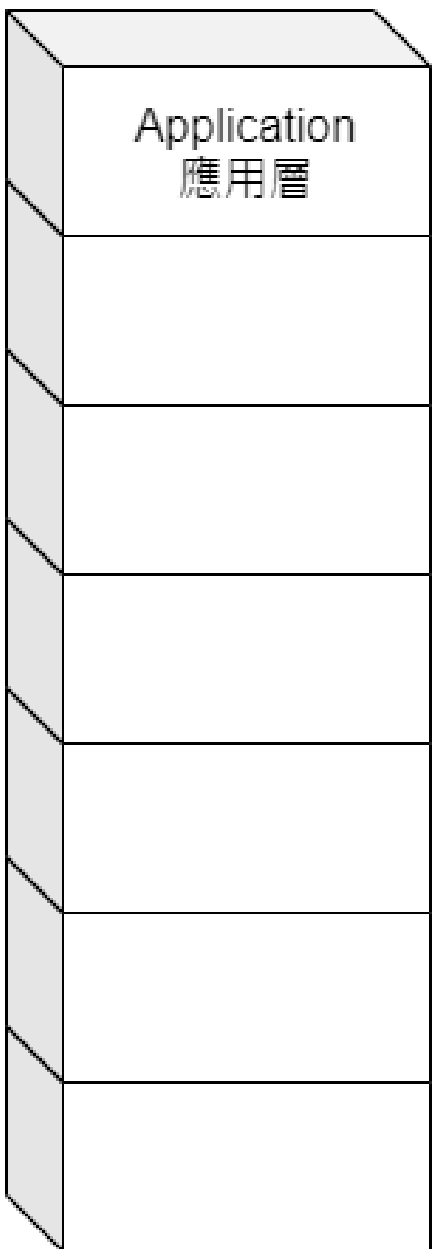
# OSI模型

開放式系統互聯模型（英語：Open System Interconnection Model，縮寫：OSI；簡稱為OSI模型）是一種概念模型，由國際標準化組織提出，一個試圖使各種電腦在世界範圍內互連為網路的標準框架。定義於ISO/IEC 7498-1。

該模型將通訊系統中的資料流劃分為七個層，從分散式應用程式資料的最高層表示到跨通訊媒介傳輸資料的物理實現。每個中間層為其上一層提供功能，其自身功能則由其下一層提供。功能的類別通過標準的通訊協定在軟體中實現。

開放式系統互聯模型的開發始於上世紀70年代後期，用以支援各種電腦聯網方法的出現。在上世紀80年代，該模型成為國際標準化組織（ISO）開放系統互連小組的工作產品。

<維基百科>



### 【HTTP】超文本傳輸協定

1. 是一種**伺服器(Server)**與**客戶端(Client)**溝通的通訊協定
2. 現行版本為HTTP1.1、HTTP2.0
3. 明文傳輸
4. 兩大基本特性：**基於請求與回應的模型**、**無狀態協定**

兩大基本特性:

- ▶ 基於請求(Request)/回應(Response)模型  
沒有請求，就沒有回應。
- ▶ 無狀態(Stateless)協定  
回應結束後，不會記得使用者的狀態。

通訊協定:

通訊協定是為兩台電腦(裝置)之間的對話方式；  
根據不同的連線方式與使用的網路服務而定，會有不同的通訊協定。

# 超文本傳輸協定 HTTP

超文本傳輸協定 ( 英語 :HyperText Transfer Protocol，縮寫:HTTP ) 是一種用於分佈式、協作式和超媒體訊息系統的應用層協定。HTTP是全球資訊網的數據通信的基礎。

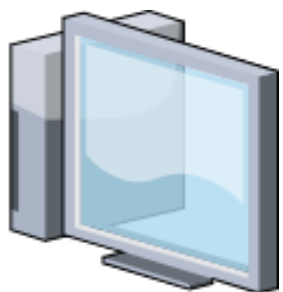
設計HTTP最初的目的是為了提供一種發布和接收HTML頁面的方法。透過HTTP或者HTTPS協定請求的資源由統一資源識別碼 ( Uniform Resource Identifiers，URI ) 來標識。

HTTP的發展是由提姆·柏內茲-李於1989年在歐洲核子研究組織 ( CERN ) 所發起。HTTP的標準制定由全球資訊網協會和網際網路工程任務組進行協調，最終發布了一系列的RFC，其中最著名的是1999年6月公佈的 RFC 2616，定義了HTTP協定中現今廣泛使用的一個版本—HTTP 1.1。

2014年12月，網際網路工程任務組 ( IETF ) 的 Hypertext Transfer Protocol Bis ( httpbis ) 工作小組將HTTP/2標準提議遞交至IESG進行討論，於2015年2月17日被批准。HTTP/2標準於2015年5月以RFC 7540正式發表，取代HTTP 1.1成為HTTP的實作標準。

〈維基百科〉

## 主從式架構(Client-Server)



Client端  
電腦、手機

使用http通訊協定溝通



Server端  
伺服器、雲端

JDBC



DataBase

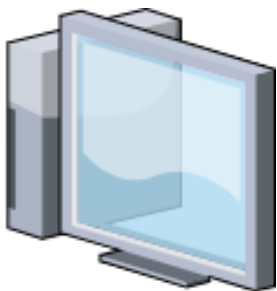
HTML  
CSS  
JAVASCRIPT

Servlet  
JSP

MS SQL SERVER



Client



使用http通訊協定溝通

Http Request



Http Response



Server



請求一張圖片  
請求一個網頁  
請求執行一支程式

回應一張圖片  
回應一個網頁  
執行一支程式，回應執行結果

現在思考一個問題：如何定位資源？

Ans：使用URI (Uniform Resource Identifier)

**Q: 伺服器的資源這麼多，如何找出所需要的資源？**

**A: 使用URI**

什麼是URI？那URL、URN又是什麼？

URI: Uniform Resource Identifier(統一資源識別碼)

URL: Uniform Resource Locator(統一資源定位符)

URN: Uniform Resource Name(統一資源名稱)

URL的標準最先出現，U原先代表Universal(萬用)，在標準化後則代表Uniform(統一)。

URN則表示某個資源獨一無二的名稱。

URI規範出現後，URL、URN成為URI子集。

URI範例：

http://i.imgur.com/mJc3wn7.mp4



使用協定



請求網域



請求資源

簡言之，URI就類似網路地址，告知瀏覽器要去哪裡找到資源。

## 統一資源標識符 URI

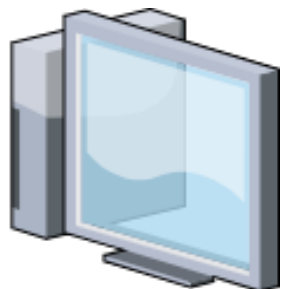
Uniform Resource Identifier

在電腦術語中是用於標識某一網際網路資源名稱的字串。

該種標識允許使用者對網路中（一般指全球資訊網）的資源通過特定的協定進行互動操作。URI的最常見的形式是統一資源定位符（URL），經常指定為非正式的網址。更罕見的使用法是統一資源名稱（URN），其目的是通過提供一種途徑。用於在特定的命名空間資源的標識，以補充網址。

<維基百科>

Client



使用http通訊協定溝通

Http Request



Http Response

Server



回應一張圖片

回應一個網頁

執行一支程式，回應執行結果

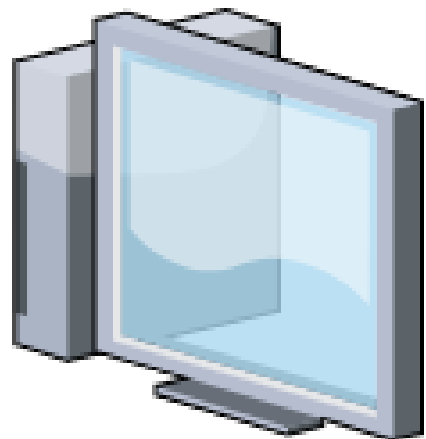
`http://192.168.30.200/apple.jpg`

`http://localhost/hello.html`

`http://127.0.0.1/MyProject/getName.do(servlet)`

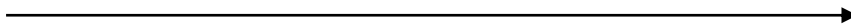


使用URI定位資源

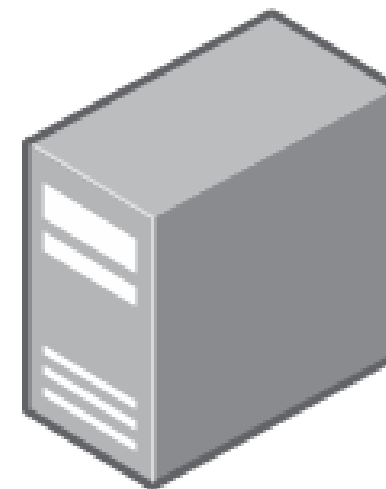


Client端

Http Request



Http Response



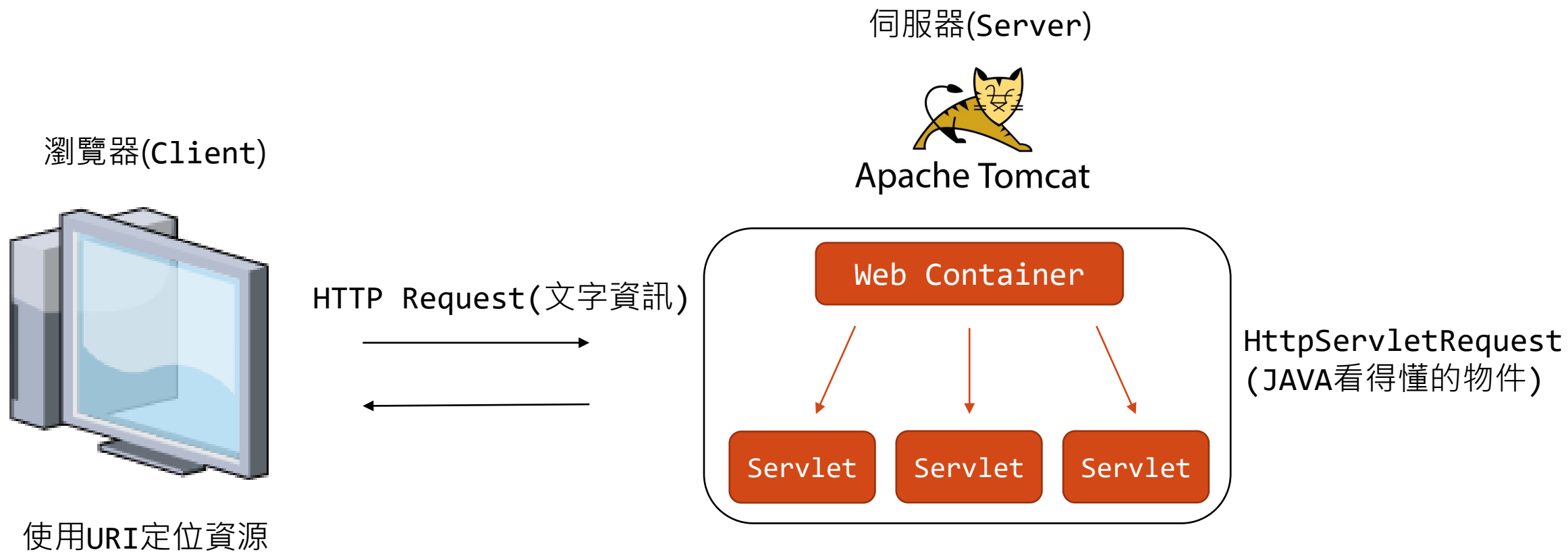
Server端

Http方法	請求的URI	協定/版本(Protocol)
POST	http://localhost/myproject/login	HTTP/1.1
accept: text/html, application/xhtml+xml, */*		
accept-language: zh-TW		
user-agent: Mozilla/5.0 (compatible; MSIE 10.0; Windows NT 6.2; WOW64)		
accept-encoding: gzip,deflate		
host: localhost:8080		
connection: keep-alive		
cookie: JSESSIONID=5EFD2E8A360A535B63DAF78180F3571C		
user=amy&password=123456		

思考一個問題，JAVA端(Servlet)如何讀取以上資訊?

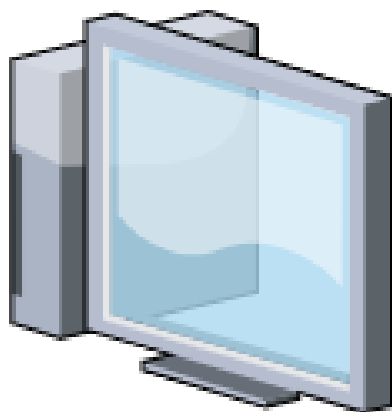


# 需要有個環境把資訊轉成JAVA可使用的物件



...但好像還不夠？  
後端的語言不只有JAVA！

瀏覽器(Client)



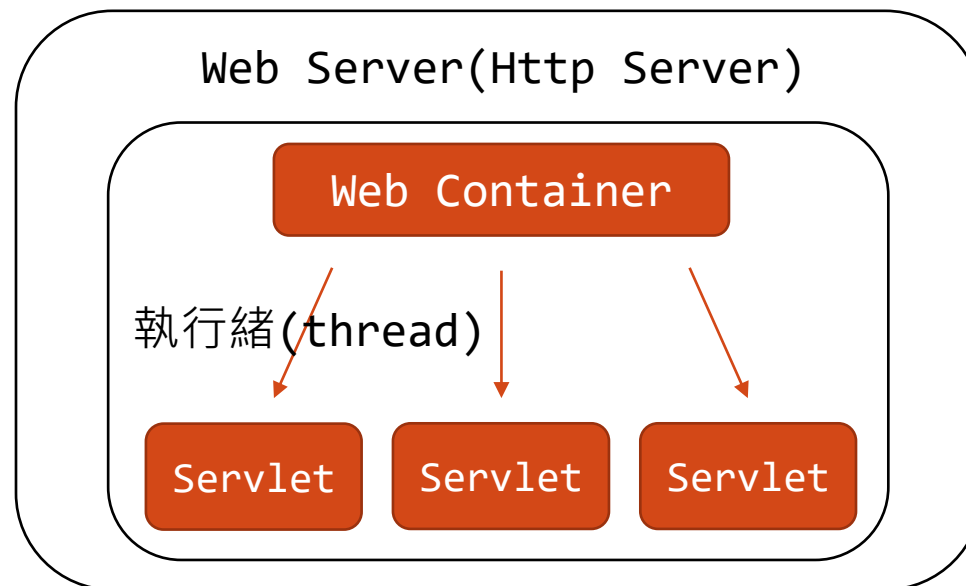
使用URI定位資源

HTTP Request  
→  
根據HTTP協定溝通  
←  
HTTP Response

伺服器(Server)



Apache Tomcat



在此，我們使用的Http Server為Tomcat(同時也是Web Container)。  
之後依據工作需求，可能改用其他Server如:JBoss、GlassFish、Oracle Weblogic等等

# 第二節

## SERVER環境安裝

依序準備以下項目：

- Java JDK開發工具
- Tomcat Web Server
- Eclipse IDE整合開發工具
- MS SQL Server資料庫



# Java JDK開發工具安裝與設定

## Open JDK安裝

1. 下載參考網站：<https://github.com/ojdkbuild/ojdkbuild>
2. 依照以下步驟：控制台→系統→進階→環境變數→系統變數
3. 設定新增環境變數：  
環境變數名稱：JAVA\_HOME  
環境變數值：JDK路徑(Ex.C:\DataSource\openjdk-xxx)
4. 找到變數path點選編輯，加入路徑：%JAVA\_HOME%\bin
5. 完成後重新開機
6. 開啟命令列(Command Line)輸入java指令測試是否安裝成功

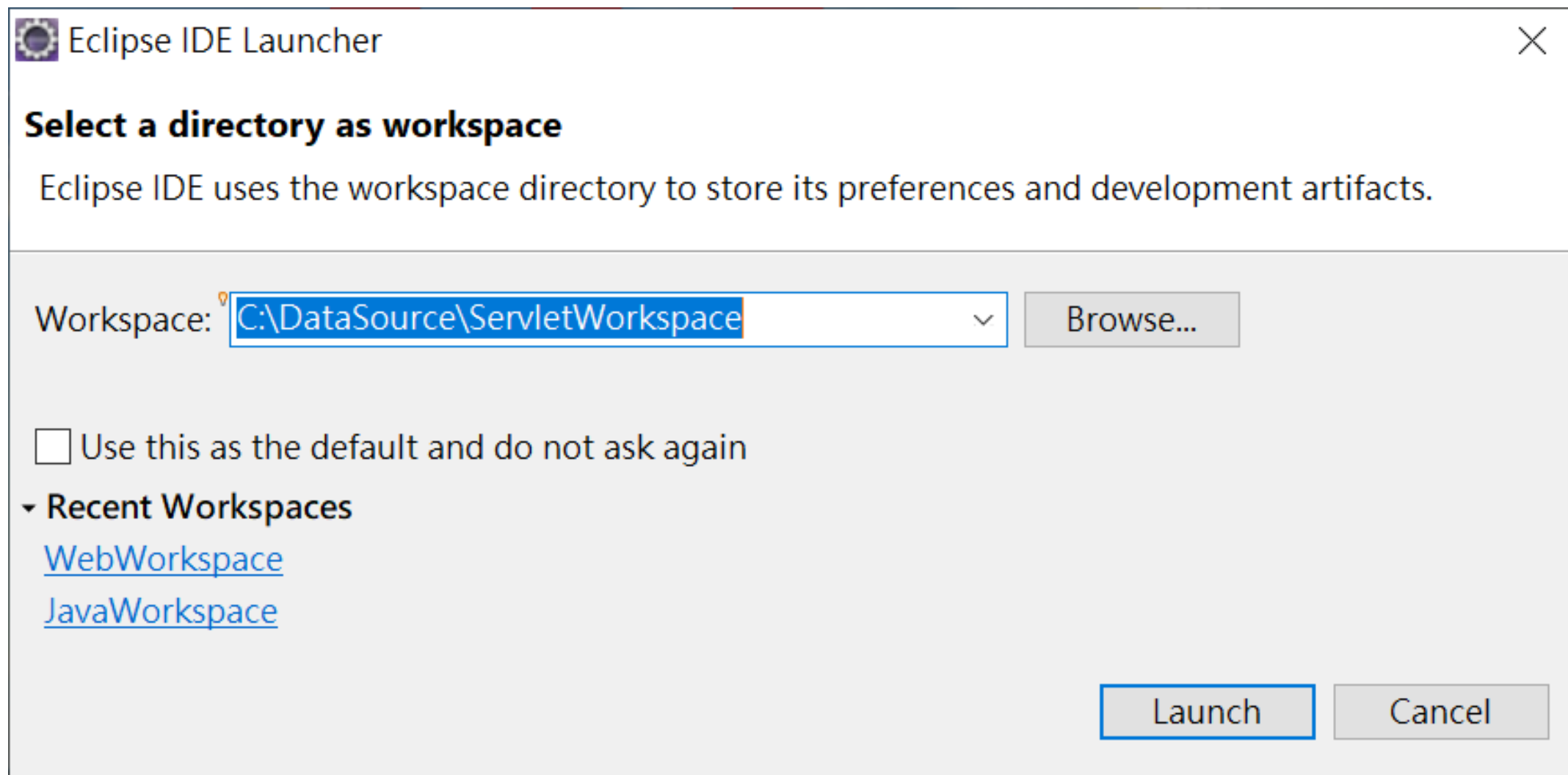
# Tomcat Web Server安裝

1. 下載參考網站：<https://tomcat.apache.org/>
2. 依據JDK版本選擇適合Tomcat版本下載，此處下載Tomcat 9
3. 下載Binary Distributions → Core → zip (執行核心)
4. 下載Source Code Distributions → zip (說明文檔)
5. 解壓縮apache-tomcat-9.x.x.zip
6. 執行bin/startup.bat
7. 連入<http://localhost:8080/> (測試)
8. 在CMD中使用Ctrl+C，終止伺服器

# Eclipse IDE整合開發工具安裝與設定

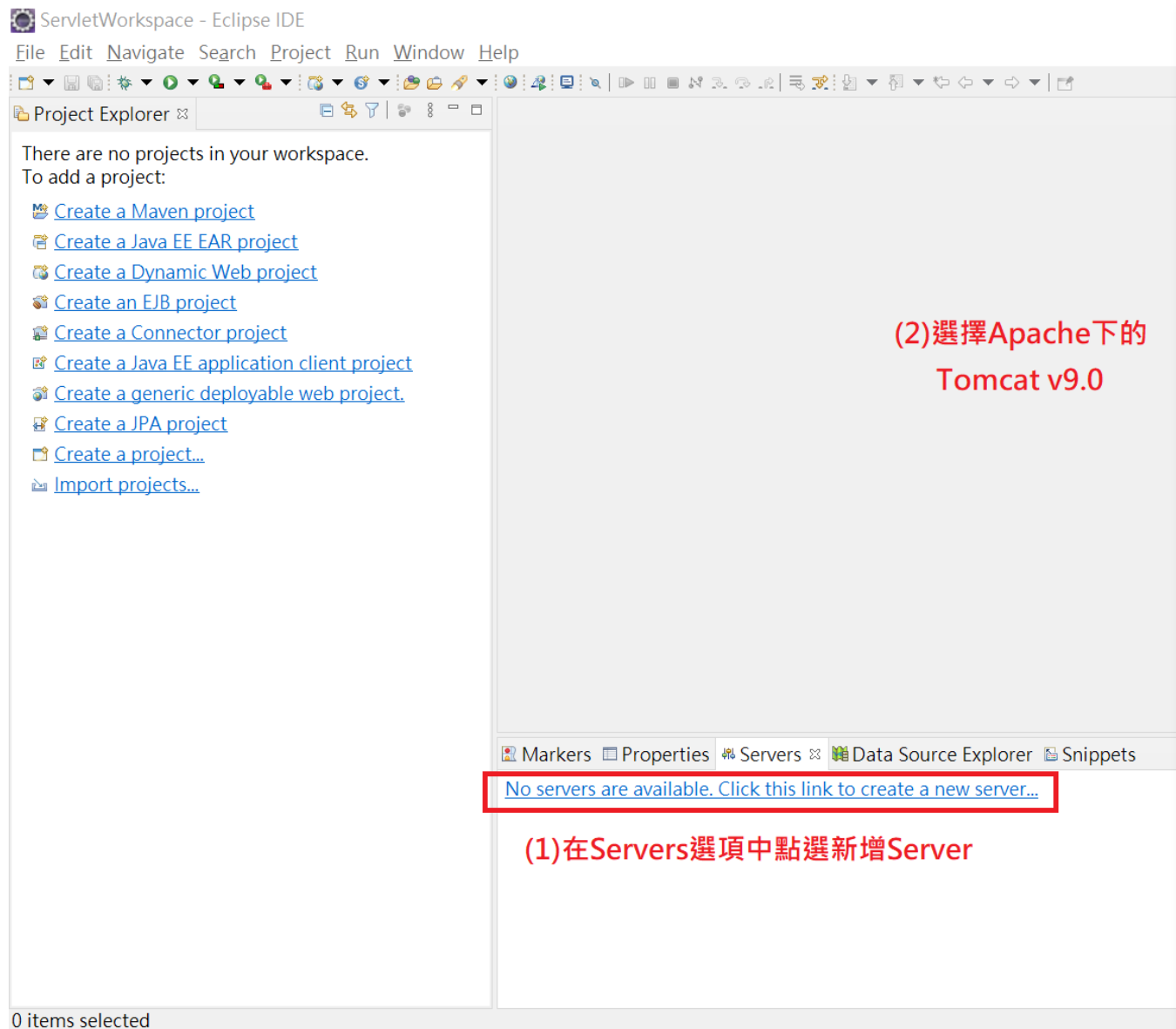
1. 下載參考網站：<https://www.eclipse.org/downloads/>
2. 安裝完成後，第一次啟動須設定工作區域workspace  
例如：C:\DataSource\xxxworkspace
3. 建立伺服器(Server)
4. 建立動態Web專案(Dynamic Web Project)
5. 設定開發環境內碼(UTF-8)
6. 撰寫程式進行測試

# 第一次啟動Eclipse時必須建立工作區域

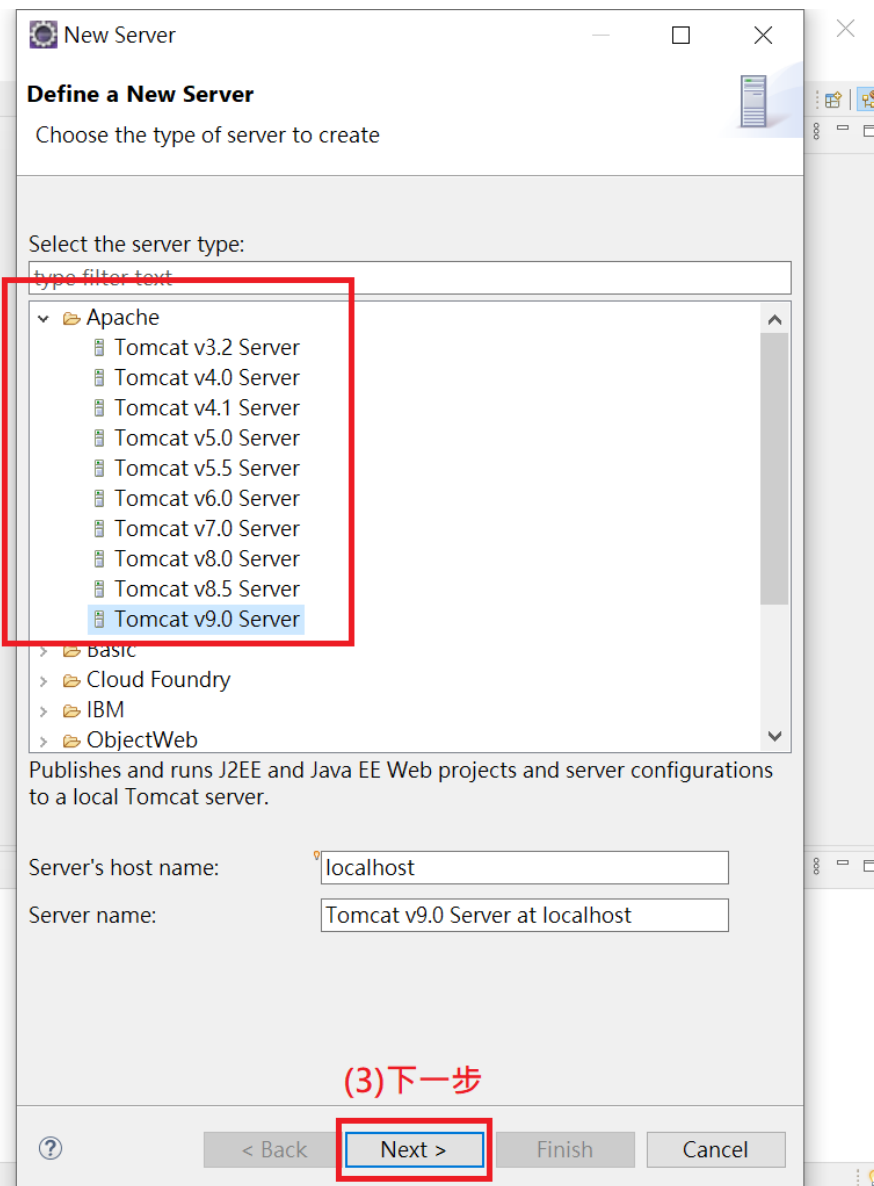





# 建立伺服器(Server)(1)



(2) 選擇Apache下的  
Tomcat v9.0



# 建立伺服器(Server)(2)

 **New Server**

**Tomcat Server**

Specify the installation directory

Name:  
Apache Tomcat v9.0

Tomcat installation directory:  
C:\DataSource\apache-tomcat-9.0.39 **(1)選擇Tomcat的資料夾**

JRE:  
Workbench default JRE

**(2)點選完成**

# 變更伺服器設定

ServletWorkspace - Tomcat v9.0 Server at localhost - Eclipse IDE

File Edit Navigate Search Project Run Window Help

Project Explorer

Servers

Tomcat v9.0 Server at localhost

### Overview

#### General Information

Specify the host name and other common settings.

Server name: Tomcat v9.0 Server at localhost

Host name: localhost

Runtime Environment: Apache Tomcat v9.0

Configuration path: /Servers/Tomcat v9.0 Server at localhost- Browse...

[Open launch configuration](#)

#### Server Locations

Specify the server path (i.e. catalina.base) and deploy path. Server must be published with no modules present to make changes.

☒ Use workspace metadata (does not modify Tomcat installation)

☐ Use Tomcat installation (takes control of Tomcat installation)

☐ Use custom location (does not modify Tomcat installation)

Server path: .metadata\plugins\org.eclipse.wst.server.core\tmj Browse...

[Set deploy path to the default value \(currently set\)](#)

Deploy path: wtpwebapps Browse...

#### Server Options

Enter settings for the server.

☐ Serve modules without publishing

☐ Publish module contexts to separate XML files

☒ Modules auto-reload by default

#### Publishing

##### Timeouts

Specify the time limit to complete server operations. 啟動、關閉超過指定秒數不執行

Start (in seconds):	45
Stop (in seconds):	15

##### Ports

Modify the server ports.

Port Name	Port Number
Tomcat admin port	8005
HTTP/1.1	8080

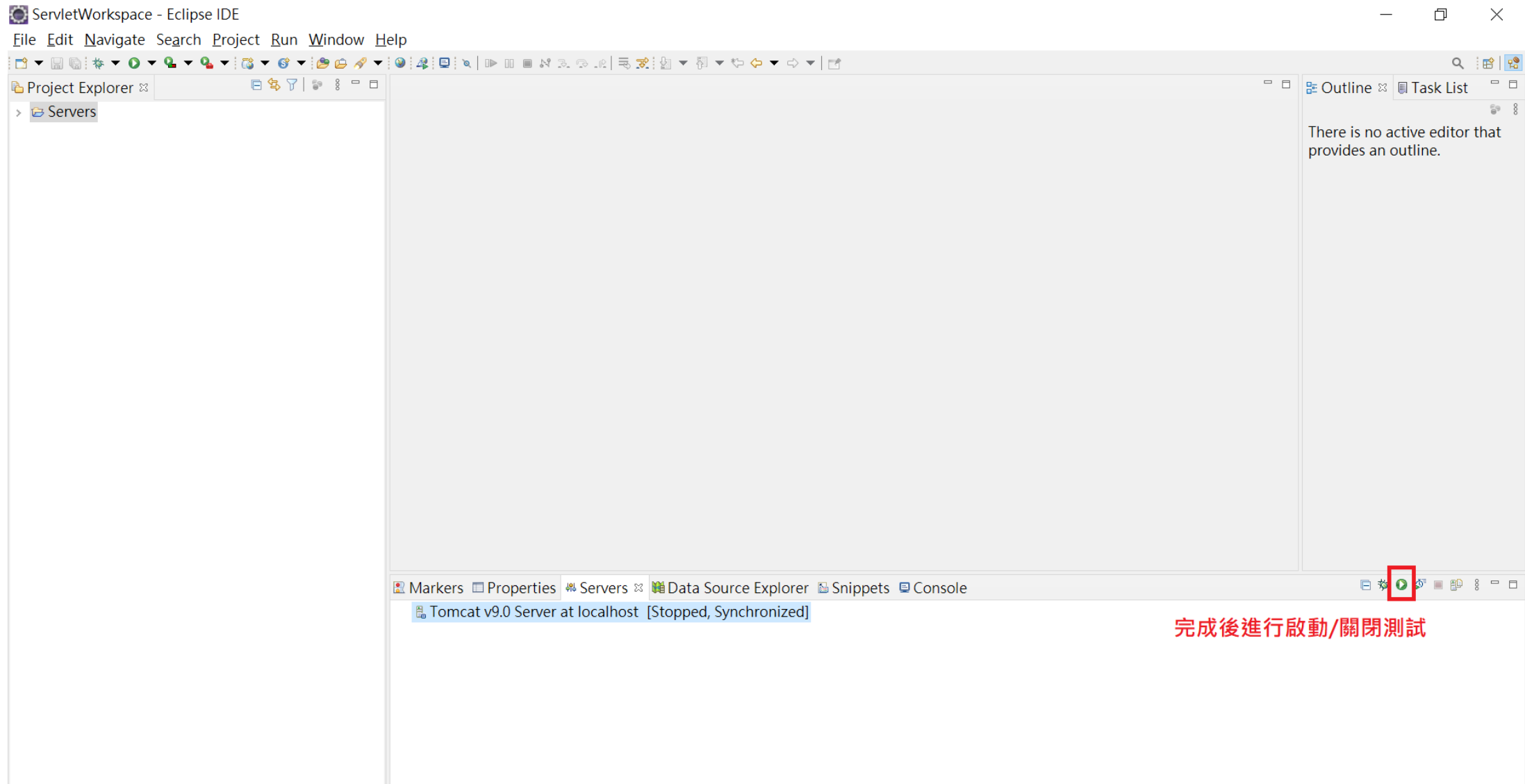
伺服器開啟的port號碼

#### MIME Mappings

Markers Properties Servers Data Source Explorer Snippets Console

Tomcat v9.0 Server at localhost [Stopped, Republish]

# 測試伺服器



完成後進行啟動/關閉測試



# 建立動態web專案(Dynamic Web Project)(1)

ServletWorkspace - Eclipse IDE

File Edit Navigate Search Project Run Window Help

New Alt+Shift+N > Maven Project  
Open File... Enterprise Application Project  
Open Projects from File System... Dynamic Web Project  
Recent Files (1) 建立動態Web專案 > EJB Project  
Close Editor Ctrl+W Connector Project  
Close All Editors Ctrl+Shift+W Application Client Project  
Save Ctrl+S Static Web Project  
Save As... JPA Project  
Save All Ctrl+Shift+S Project...  
Revert CSS File  
Move... JavaScript File  
Rename... F2 Servlet  
Refresh F5 Session Bean (EJB 3.x)  
Convert Line Delimiters To > Message-Driven Bean (EJB 3.x)  
Print... Ctrl+P Folder  
Import... Example...  
Export... Other... Ctrl+N

Properties Alt+Enter  
Switch Workspace >  
Restart  
Exit

Markers Properties Servers Data Source  
Tomcat v9.0 Server at localhost [Stopped, Repu]

New Dynamic Web Project

Dynamic Web Project

Create a standalone Dynamic Web project or add it to a new or existing Enterprise Application.

Project name JspProject (2) 輸入專案名稱

Project location  
☒ Use default location  
Location: C:\DataSource\ServletWorkspace\JspProject Browse...

Target runtime  
Apache Tomcat v9.0 New Runtime...

Dynamic web module version  
4.0 (3) 選擇版本

Configuration  
Default Configuration for Apache Tomcat v9.0 Modify...  
A good starting point for working with Apache Tomcat v9.0 runtime. Additional facets can later be installed to add new functionality to the project.

EAR membership  
☐ Add project to an EAR  
EAR project name: EAR New Project...

Working sets  
☐ Add project to working sets New...  
Working sets: Select...

(4) 點選下一步

< Back Next > Finish Cancel

# 建立動態web專案(Dynamic Web Project)(2)

New Dynamic Web Project

**Java**  
Configure project for building a Java application.

Source folders on build path:

src

Add Folder...  
Edit...  
Remove

Default output folder:  
build\classes

(1)點選下一步

< Back Next > Finish Cancel



New Dynamic Web Project

**Web Module**  
Configure web module settings.

Context root: JspProject  
Content directory: WebContent

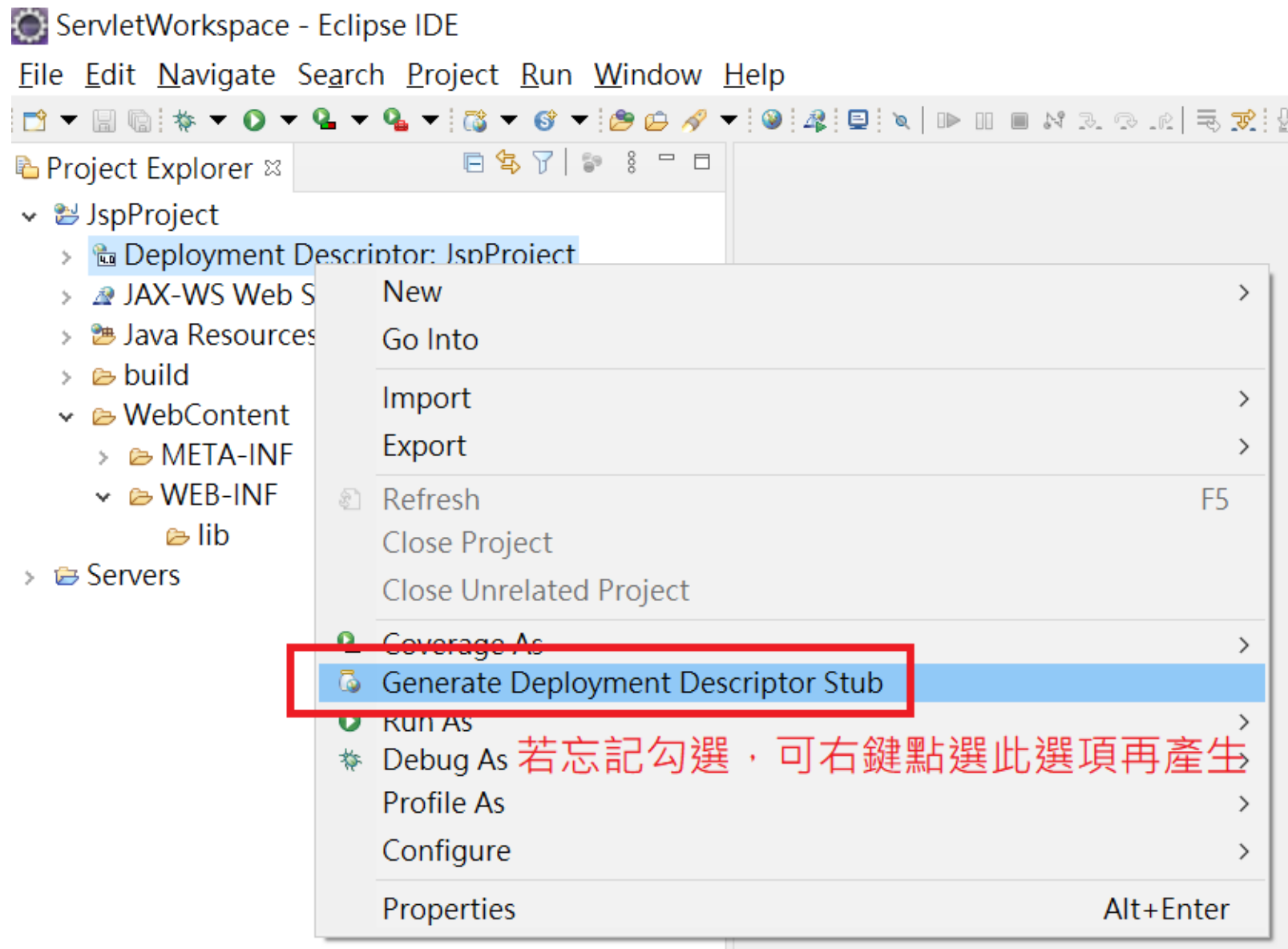
☒ Generate web.xml deployment descriptor

(2)勾選以產生部署描述檔web.xml

(3)點選完成

< Back Next > Finish Cancel

# 備註:手動產生部署描述檔



# 專案組件介紹

ServletWorkspace - Eclipse IDE

File Edit Navigate Search Project Run Window Help

Project Explorer

- ▼ JspProject
  - > Deployment Descriptor: JspProject 部署描述檔web.xml
  - > JAX-WS Web Services
  - ▼ Java Resources
    - > src 放置xx.java與xxx.jar檔案的資料夾
    - > Libraries
  - > build
  - ▼ WebContent
    - > META-INF
    - > WEB-INF 放置html、jsp網頁圖檔、資料檔及部署描述檔的資料夾
  - > Servers

# 設定工作環境編碼

The screenshot shows the Eclipse IDE interface with the 'Preferences' dialog box open. The 'Window' menu is highlighted in the top-left corner, and the 'General' category is selected in the left sidebar of the 'Preferences' dialog. The 'Workspace' sub-category is also selected. In the 'Workspace' section, the 'Text file encoding' is set to 'UTF-8'. The 'Apply' button is highlighted in the bottom-right corner of the dialog.

(1)點選window→Preferences

(2)找到General下的Workspace

(3)設定文字編碼為UTF-8

(4)按下應用



# 設定JSP檔案編碼

The screenshot shows the Eclipse IDE interface with the following elements and annotations:

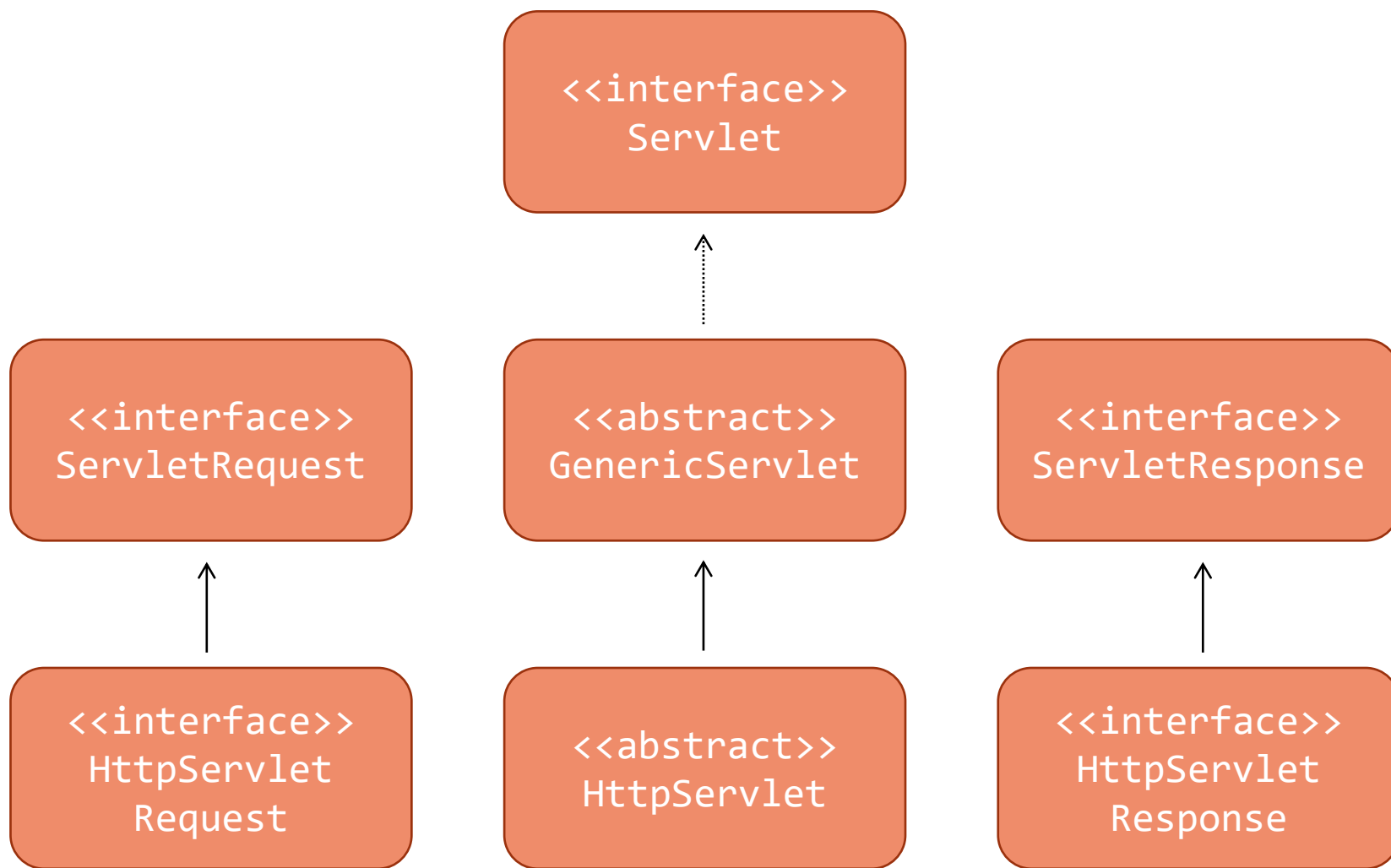
- Window Menu:** The 'Window' menu is open, and the 'Preferences' option is highlighted with a red box. Below it is the annotation: (1) 找到window下的 Preferences.
- Preferences Dialog:** The 'Preferences' dialog is open. In the left sidebar, 'Web' is expanded, and 'JSP Files' is selected with a red box. Below it is the annotation: (2) 找到Web下的 JSP Files.
- JSP Files Settings:** In the 'JSP Files' section, the 'Encoding' dropdown is set to 'ISO 10646/Unicode(UTF-8)' and is highlighted with a red box. Below it is the annotation: (3) 設定JSP編碼為UTF-8.
- Buttons:** The 'Apply and Close' button at the bottom right of the dialog is highlighted with a red box. Below it is the annotation: (4) 應用且關閉.

The bottom status bar shows 'Tomcat v9.0 Server at localhost [Stopped, Republish]'.

# 第三節

## 撰寫SERVLET入門

- 定義在`javax.servlet`與`javax.servlet.http` package
- Tomcat 10 & Servlet 5 以上開始由Jakarta取代javax



# 開發SERVLET程式

## 【狹義的Servlet】

在Server上執行的程式(Applet)

## 【廣義的Servlet】

是指實作了Servlet API所有類別。Servlet 本身是一個介面，實作的部分則由Web Container完成，如Tomcat、GlassFish、JBoss、WebSphere、WebLogic等等。

所有Servlet程式都需繼承自  
`javax.servlet.Servlet`介面  
但從Tomcat 10開始改為繼承自  
`Jakarta.servlet.Servlet`介面

【範例】 extends **GenericServlet**

```
public class HelloWorld extends GenericServlet {  
    @Override  
    public void service(ServletRequest req, ServletResponse res){  
        //略...  
    }  
}
```

【範例】 extends **HttpServlet**

```
public class HelloWorld extends HttpServlet {  
    @Override  
    protected void doGet(HttpServletRequest req, HttpServletResponse resp){  
        // 略...  
    }  
    @Override  
    protected void doPost(HttpServletRequest req, HttpServletResponse resp){  
        // 略...  
    }  
}
```

## 撰寫第一支SERVLET

最初Servlet設計並非針對於網路應用程式，只要繼承GenericServlet介面並改寫其service( )，理論上也能實作出其他協定的Servlet(如FtpServlet、SmtplibServlet)，但實際上不會有人這樣使用。提到Servlet，所有的Java web工程師都是以Http協定為前提討論的。

在撰寫Servlet時，除非有特殊需求否則不應繼承GenericServlet，應一律繼承**HttpServlet**。



客戶端使用者

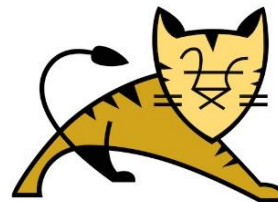


請求執行「登入」的程式

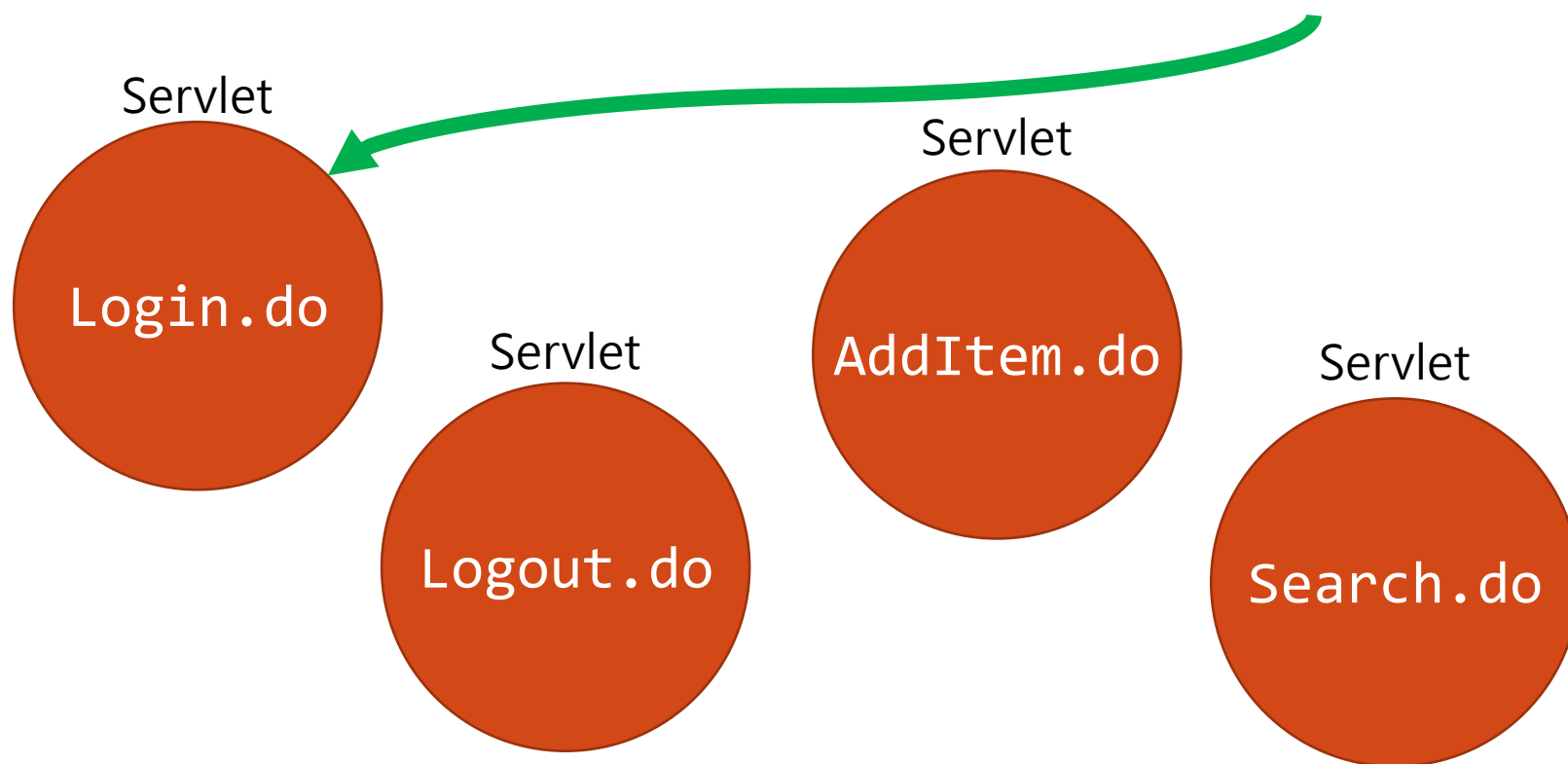
Login.do

http://localhost/shop/login.do

你的貓Server



Apache Tomcat



## 命名你的程式：註冊

Servlet並非桌面應用程式，沒有main方法(程式進入點)。

寫好的程式要如何執行？由誰來執行？誰想要執行？

### 【基本流程】

客戶端(網頁使用者)  
請求一支程式資源



發送Http Request



Web Server接收到請求  
轉發給Web Container  
(在此都是Tomcat)



Web Container根據資源名稱  
執行對應的程式

將程式「命名」的行為，稱之為註冊。

那麼，程式要在哪裡註冊？有兩種方法：**web.xml** 與 **annotation**

```
@WebServlet("/HelloWorld.do")
public class HelloWorld extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp){
        // 略...
    }
}
```

```
@WebServlet(urlPatterns="/HelloWorld.do")
public class HelloWorld extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp){
        // 略...
    }
}
```

```
@WebServlet(urlPatterns="/HelloWorld.do", loadOnStartup = 3)
public class HelloWorld extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp){
        // 略...
    }
}
```

# 註冊：ANNOTATION

**Servlet 3.0(2009/09)新增的功能**

提供一個簡便的方法讓程式設計師能輕鬆的註冊Servlet，簡化開發流程並增加程式可讀性。

如欲註冊一支Servlet，只需在其Class上方撰寫@WebServlet即可。

最簡單的寫法為

**@WebServlet("/ServletName")**

務必注意名稱不可重複，且一定要加上"/"，如果忘了加上"/"，則整個Server會無法起動。(Server起動時會掃描所有Servlet註冊的名稱)

@WebServlet之中可以加上其他屬性如loadOnStartup、urlPatterns、description、name等。

另外一個註冊方法會寫在web.xml裡

標準設定如下：

Web.xml第一行宣告XML版本：

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

第二行指定此檔案所用的DTD(Document Type Definition)或XMLSchema檔，所有的部署描述檔都以這兩行(或類似的宣告)做為開始：

```
<web-app  
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-  
  instance" //...中略  
  version="4.0">
```

註：各個版本的Servlet頂層標籤略有不同

其餘在<web-app>與</web-app>間的文字為檔案主體，用來提供該Web App的資訊給伺服器。

## web.xml 部署描述檔

web.xml檔案是Java Web專案中的一個配置檔案。

當伺服器啟動時，伺服器就會預先讀取每一個Web App的web.xml檔，用來提供該Web App的資訊給伺服器。

此檔案包含它所屬的Web App組態資訊，可讓你完全控制所屬的Web App行為，如Servlet、Filter、Listener的註冊、URL對應、歡迎檔、錯誤頁面、預設App起始參數...等。

此檔案不是Web App必需的，但可以增加Web App的彈性。



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

```
//...前略
```

一支Servlet程式可以有多個註冊名稱，  
每個註冊名稱會對應到一個Servlet實體

```
<web-app>
```

```
  <servlet>
```

內部註冊名稱

```
    <servlet-name>myServlet</servlet-name>
```

```
    <servlet-class>tw.ch1.HelloWorld</servlet-class>
```

```
  </servlet>
```

Servlet實體

```
  <servlet-mapping>
```

```
    <servlet-name>myServlet</servlet-name>
```

```
    <url-pattern>/Hello</url-pattern>
```

```
  </servlet-mapping>
```

```
</web-app>
```

url-pattern則根據註冊名稱尋找實體，  
同一個名稱可有多個url-pattern

## SERVLET註冊：XML

一支Servlet程式可以有多個內部註冊名稱(name)，每個內部註冊名稱會對應到一個Servlet實體。

### 【url-pattern命名】

務必注意要加入斜線"/"，且不可重複命名；若有以上情況，則伺服器會無法啟動。

```
<servlet-mapping>
```

```
    <servlet-name>myServlet</servlet-name>
```

```
    <url-pattern> /Hello/* </url-pattern>
```

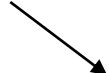
```
</servlet-mapping>
```

```
<servlet-mapping>
```

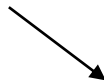
```
    <servlet-name>myServlet</servlet-name>
```

```
    <url-pattern> /*.aa </url-pattern>
```

```
</servlet-mapping>
```



前置路徑對應: /HelloServlet/parame



副檔名對應(延伸檔名對應): /DoSomething.aa

# URL路徑對應

有三種對應設定，提供更彈性的處理方式，如附加參數、Restful風格、同類路徑過濾等等。

## 【完全對應】

/Hello/DoSomethingServlet

## 【前置路徑對應】

/Hello/\*

(沒有 /AAA/\*/BBB 這種寫法)

## 【副檔名對應(延伸檔名對應)】

/Hello/\*.aa

若以上三種都有設定，則以

【完全對應】→【前置路徑】→【副檔名】  
為執行順序。

# 第四節

## 初識 請求與回應

# HTTP通訊協定

通訊協定是指「電腦間互相通訊的方式」。

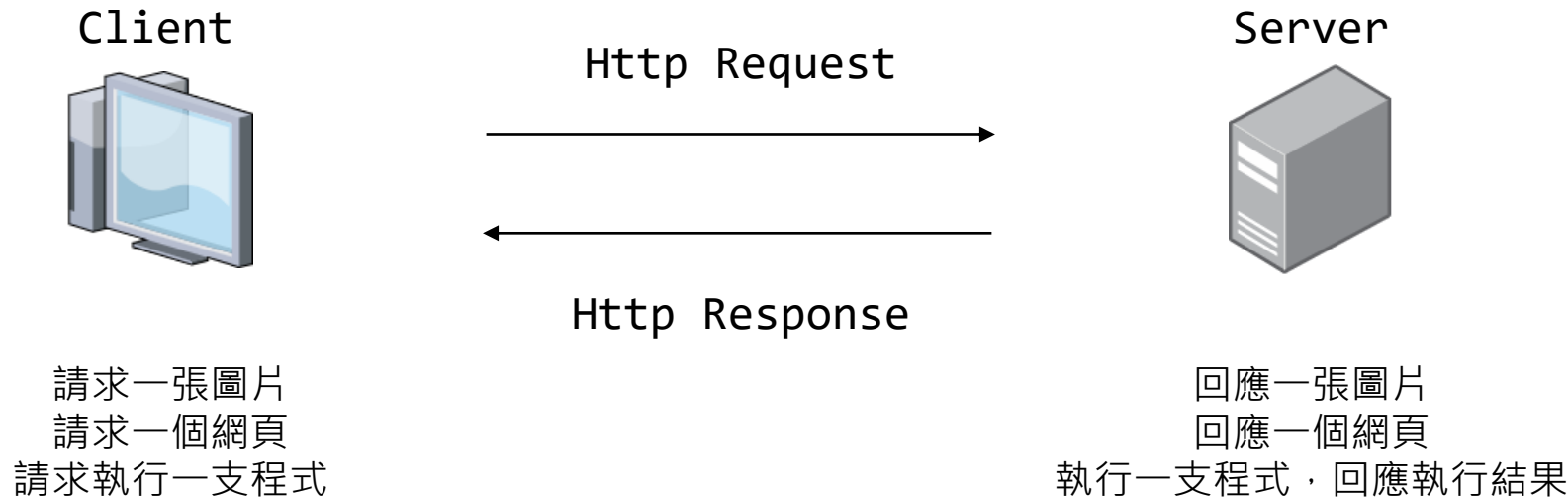
HTTP通訊協定則是「客戶端與伺服器溝通的方式」。

## 【HTTP通訊協定的兩大特性】

1. 無狀態協定
2. 基於請求/回應的模型

## 【主從式架構】

也稱為客戶端/伺服器架構，由多台Client對一台Server提出請求(Request)。與之對應的為P2P(peer to peer)架構，客戶端同時也扮演伺服器的角色。



試想，請求執行以下程式，需要附上什麼資訊？會得到什麼回應？

1. 請求執行「取得所有商品資訊」的程式。
2. 請求執行「登入」的程式。
3. 請求執行「把我的PDF存在雲端」的程式。
4. 請求執行「查詢這期統一發票中獎號碼，順便幫我看一下輸入的這組號碼有沒有中獎」的程式。

那麼，要如何「接收使用者(客戶端)的請求資訊」  
以及，要如何「回應客戶端訊息」呢？



## 【GET】

請求執行指定程式

```
http://localhost:8080/MyProject/GetSomething.do  
<a href='DoSomething.do'>
```

帶上參數(QueryString)的GET寫法

```
http://localhost:8080/MyProject/Login.do?userName=allan
```

## 【POST】

```
<form action='Login.do' method='post' >  
    姓名:<input type='text' name='userName'>  
    年齡:<input type='number' name='userAge'>  
</form>
```

補充：使用JavaScript送出

1. 使用document.createElement('form')創建<form>元素。
2. 使用AJAX發送非同步請求

# HTTP八種請求方法

HTTP 1.1定義了8種請求方法，  
分別是GET、POST與其他6種。

### 【GET請求】

1. 在網址列直接輸入資源URI、使用<a href='URI'>，或使用<form method='GET'>都可送出GET請求。
2. 歷史紀錄會記錄送出的請求參數(Query String)。
3. 請求參數的長度有限制，超出會截斷或報錯，每個瀏覽器的長度限制都不同

### 【POST請求】

1. 使用<form method='POST'>或使用JavaScript送出。
2. 長度沒有限制，上傳檔案請使用POST。
3. 與其他所有方法都屬於明文傳輸(未加密)

### 【其他六種】

HEAD、PUT、DELETE、OPTIONS、TRACE、CONNECT

1. `String` `getParameter(String name)`:  
使用Client設定好的參數名稱(name)，取得「字串」參數值。
2. `Map<String, String[]>` `getParameterMap( )`:  
取得所有參數名稱、參數值，以Key、Value方式存成Map物件。
3. `Enumeration` `getParameterNames( )`:  
取得所有參數名稱，存成Enumeration物件。  
【Enumeration介面】  
在JDK 5 時加入的舊介面，較不容易使用，建議轉型成ArrayList型別。  
`ArrayList<T> list = Collections.list(enumeration);`
4. `String[]` `getParameterValues(String name)`:  
使用參數名稱取得Client端傳過來的所有參數值。  
也可使用`List<String> list = Arrays.asList(String[]);`  
取得List物件。
5. `void` `setCharacterEncoding(String encoding)`:  
設定request的字元編碼。

# HttpServletRequest 介面基本方法介紹

HttpServletRequest繼承自  
ServletRequest

介面中定義了許多實用方法，可使用這些方法取得客戶端送過來的param(參數)。

實作由Container(伺服器)完成，於建立ServletRequest實體後，呼叫HttpServletRequest的  
`service(ServletRequest, ServletResponse)`，並將其當作引數傳入，其後進行強制轉型成  
HttpServletRequest，以供HttpServletRequest使用。

# HttpServletResponse 介面基本方法介紹

## HttpServletResponse繼承 ServletResponse

介面中定義了許多實用方法，可使用這些方法對客戶端進行回應。

實作由Container(伺服器)完成，於建立ServletResponse實體後，呼叫HttpServlet的service(ServletRequest, ServletResponse)，並將其當作引數傳入，其後進行強制轉型成HttpServletResponse，以供HttpServlet使用。

### 1. void setContentType(String type):設定回應類型

【引數範例】 "text/html;charset=UTF-8"

【常用型別】 --Google關鍵字: 型別+MimeType

"text/plain"; //純文字

"text/xml"; //XML格式文件

"text/html" ; //HTML格式文件

"image/jpeg"; //jpg格式圖片

"application/json"; //JSON格式文件

### 2. PrintWriter getWriter( ):取得文字輸出物件

【PrintWriter類別方法】

write(多載參數); //輸出字元或字串

print(多載參數); //先呼叫String.valueOf(i)

### 3. ServletOutputStream getOutputStream( ):

取得資料流輸出物件

【ServletOutputStream類別方法】

write(多載參數); //輸出二進制資料流

print(多載參數); //輸出字元



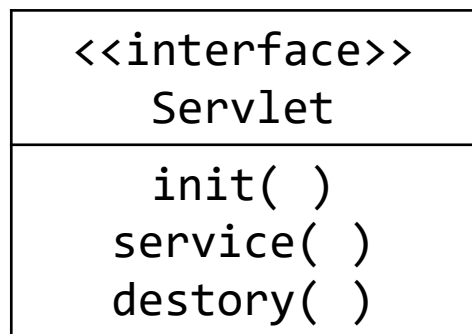
# 第五節

## **SERVLET**的生命週期

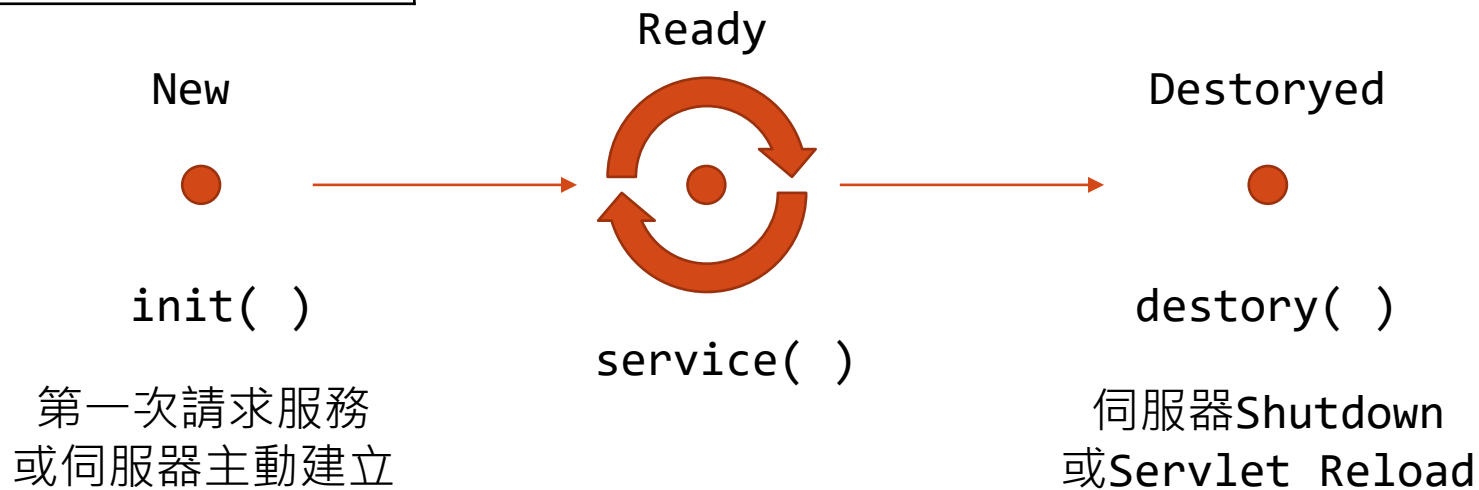
Servlet container(伺服器)替我們建立Servlet實體，並由其呼叫Servlet的init( )、service( )、destroy( )等方法，以管理Servlet的生命週期。

程式設計師藉由實作上述方法，予以控制該Servlet物件及該物件的相關資源。

javax.servlet.Servlet



Servlet生命週期



## Servlet 生命週期 基本概念

如無特別設定，伺服器啟動時不會建立Servlet物件實體。

當客戶端第一次對Servlet進行請求時，伺服器會建立Servlet物件實體，同時也建立ServletConfig、ServletRequest、ServletResponse等物件實體，並將其當作引數傳入Servlet。

其後伺服器會呼叫Servlet內以下方法：

```
init(getServletConfig()); //勿改寫
```

↓

```
init(); //改寫目標
```

↓

```
service(); //轉型成doGet()、doPost()
```

當伺服器被中止時(非意外)，或Servlet Reload時，會呼叫Servlet的destroy()方法。

# Servlet container 執行步驟與相關技術

1. 當Servlet第一次被客戶端請求時，container會載入該Servlet，建立實體  
例外情況:Servlet Reloading 與 load-on-startup
2. container呼叫Servlet的`init( )`方法，以進行Servlet的初始化工作  
衍生技術:`getInitParameter( String name )` 方法
3. container呼叫Servlet的`service( )`方法，處理請求  
衍生技術:執行續同步synchronization處理
4. container關閉時，呼叫每一個Servlet的`destroy( )`方法  
衍生技術:釋放之前`init( )`內引用的資源或儲存一些永續性資料

# Servlet Reloading(Servlet 重載入)

- 更改Web-INF/classes底下之Servlet類別檔(class)後，不必重新啟動伺服器，伺服器會自動重新載入該Servlet，稱之為Servlet Reloading
- 有些Servlet Container因考慮執行時整體效能，可能會將此功能改為手動方式
  - 若被改為手動載入，則必須重新啟動Server重新載入。

server.xml:

```
<Context docBase="MyWebApp" path="/MyWebApp" reloadable="true">
```

## Load on Startup(啟動時載入)

- 經由設定Servlet的web.xml檔案，讓伺服器在啟動時就自動載入Servlet，並呼叫其init( )的方法

```
<servlet>  
    <servlet-name>LoadOnStartupTest</servlet-name>  
    <servlet-class>pers.allen.servlet.TestServlet</servlet-class>  
    <load-on-startup>10</load-on-startup>  
</servlet>
```

數字越小，執行優先度越高，最小為0；為負值時無效

或者annotation寫法

```
@WebServlet(urlPatterns = "/myServlet",loadOnStartup = 10)
```

# Init Parameter (初始參數)

經由設定Servlet的web.xml檔案，讓伺服器在啟動時就自動載入Servlet，並呼叫其init( )的方法

XML寫法

```
<servlet>
  <servlet-name>InitParameter</servlet-name>
  <servlet-class>pers.allen.servlet.TestServlet</servlet-class>
  <init-param>
    <param-name>MyAge</param-name>
    <param-value>18</param-value>
  </init-param>
</servlet>
```

可以在Servlet中使用  
getInitParameter("MyAge")取得參數值

或者annotation寫法

```
@WebServlet(urlPatterns = "myServlet",
initParams = @WebInitParam(name="MyAge", value = "18") )
```

【補充：系統參數xml設定法】

```
<context-param>
  <param-name>MyAge</param-name>
  <param-value>18</param-value>
</context-param>
```

【取法】

於Servlet中，先使用getServletContext()取得context物件，再使用物件的getInitParameter()方法取值

## 多執行緒環境

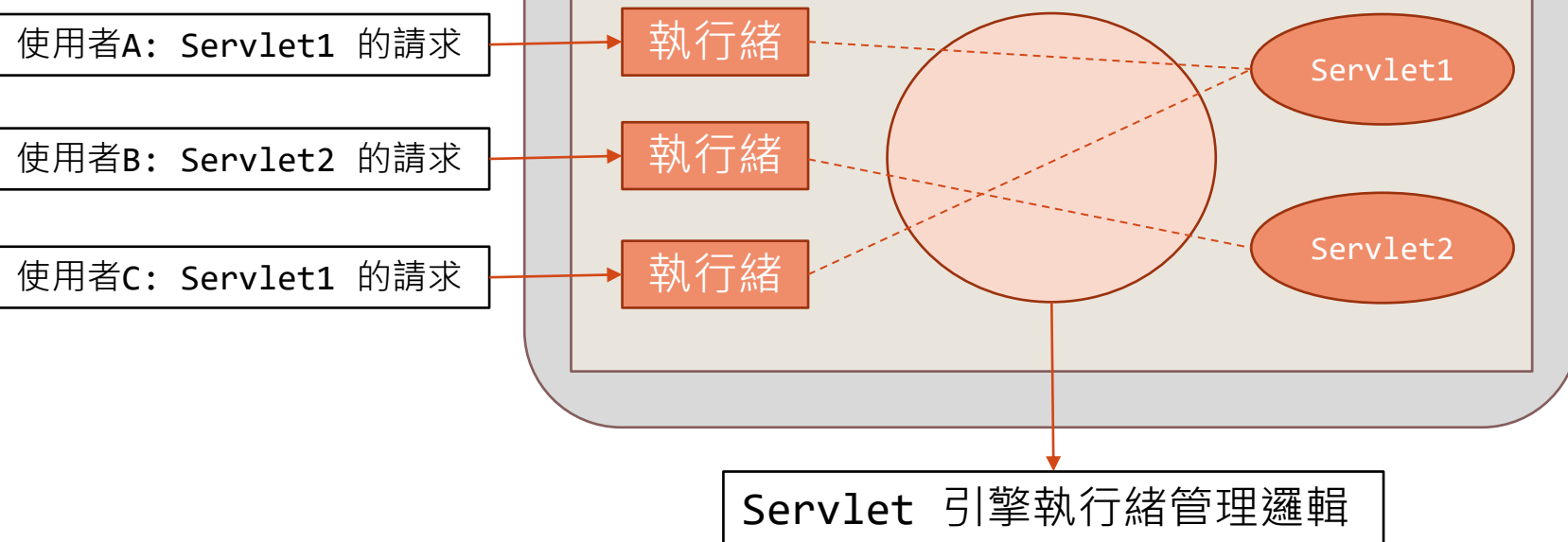
**Servlet**執行時，是處於一個多執行緒的環境下。

如果**Servlet**中設有全域變數，則須注意資源共享的問題(不同使用者存取到同樣的變數)。

若沒有宣告全域變數，基本上可忽略此節，有此概念即可。

### Web Container

#### Main Process





## 解決方法

### 1. 直接鎖定service( )

```
synchronized public void doGet( ... ){ ... }
```

優點:簡單直觀

缺點:一次只能處理一個請求，效率低

### 2. 於doGet內處理臨界區段

```
synchronized (this) {  
    count++;  
    out.write(count);    //將執行輸出的部分鎖定  
}
```

### 3. 於doGet內處理Critical section(臨界區段)

```
int localCount;  
synchronized (this) {  
    localCount = ++count; //建立區域變數紀錄當下的全域變數值  
}
```

### 4. 如果程式目的不需要持續性(Persistence)時，可將所有變數設定為區域變數(區域變數的存活期限無法持續至下一次Request)

## Synchronization(同步化)

同步可以想像成排隊，第一位執行完後，才輪第二位執行；非同步(異步Async)則是同時執行，不用等其他人完成。

非同步時可能會遇到以下問題：

- Servlet共用同一變數，在第一個執行緒尚未處理完畢時，第二個執行緒已經開始運作，導致變數存取異常。
- 資料庫連線時，前一個請求執行緒還未完成時，後一個請求執行緒已經commit，導致資料庫存取異常。

同步與非同步各有優劣，須依情境使用。

同步有「執行緒安全」優點，缺點是效率可能較差。

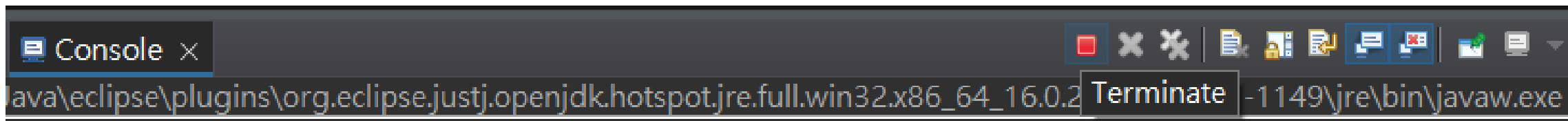
非同步有「效率高」優點，但使用時要注意共用變數存取的問題。

此處進行同步的方法是於service( )中建立臨界區段([Critical section](#))，將變數鎖定。

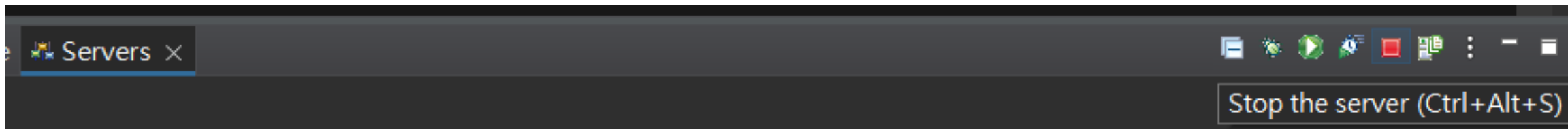


# 關於destroy( )的執行

- 目的:讓Servlet的狀態能保存至下一次的存活期
- 流程:在Servlet的destroy( )裡進行狀態保存，待下次init( )時取回保存狀態
- 注意事項:
  - 儲存狀態，可使用JAVA IO保存資料，或者存入資料庫，視需要而定
  - 如果Server因為當機而停止，則不會呼叫servlet的destroy( )方法



點選Console下的Terminate，屬於異常中止，不會呼叫destroy( )方法。



點選Servers下的Stop the server，屬於正常中止，會呼叫destroy( )方法。

# 第六節

## 取得SERVER&CLIENT 的相關資訊

# 取得Server資訊

取得Server主機的資訊

- `String request.getServerName( )` : 取得伺服器名稱
- `int request.getServerPort( )` : 取得伺服器通訊埠號
- `ServletContext request.getServletContext()` : 取得ServletContext物件，詳見下一頁  
或直接使用`getServletContext( )`↑

▪ 取得Servlet初始參數

- `String getInitParameter(String name)` : 取得指定初始參數值
- `Enumeration getInitParameterNames( )` : 列舉所有初始參數名稱

▪ 取得Servlet名稱

- `String getServletName( )` : 取得Servlet註冊的名稱

使用 `ServletContext context = getServletContext( );` 取得 context 物件

- `void context.setAttribute(String name, Object value)`  
將物件存入 context 中，可以存入「任意物件」，但取出時須 **強制轉型**。
- `Object context.getAttribute(String name)`  
取出指定物件，使用 `(Class) object`，進行強制轉型。
- `void context.removeAttribute(String name)`  
從 context 移除指定物件。
- `String context.getInitParameter(String name)`  
取出定義好的預設參數 (定義在 `web.xml` 或 `ServletContextListener` 中)。  
【`Web.xml` 預設參數寫法】p48 的補充  

```
<context-param>  
    <param-name>MyAge</param-name>  
    <param-value>18</param-value>  
</context-param>
```

第三章會提到
- `Enumeration context.getAttributeNames( )`  
取得所有 `ServletContext` 中的屬性名稱  
註: "`javax.servlet.context.tempdir`" 是伺服器強制規定一定要有的 Attribute 名稱

## ServletContext(上下文)

`Context` 並沒有適合的中文翻譯，根據語意可視為「情境」、「背景」、「脈絡」、「框架」、「上下文」等等。而 `Content` 則單純指內容。

舉個例子，一包薯條，它的 `Content` 就單純指向薯條，內容物就是薯條；但它的 `Context` 可能是麥X勞的薯條、漢X王的薯條，或者是你家隔壁早餐店的薯條，其價值就不同了，要根據情境決定。

`ServletContext` 就字面理解即是 `Servlet` 所屬的 `Context`，是故「**每一個 `Servlet` 的 `Context` 都是相同的**」(只要是在同一個專案)。

在 `Servlet` 中使用 `getServletContext( )` 取得 context 物件 (定義在 `GenericServlet` 中)。

context 物件實作了數種實用方法，有需要可查詢 API，在此介紹幾個較為常用方法。

- `String context.getRealPath(String virtualPath)`  
轉換指定的虛擬路徑為真實檔案路徑。
- `Java.net.URL context.getResource(String URIPath)`  
傳回指定的URIPath(URIPath必須以/開頭)的URL物件  
取得物件後，可使用其`openStream()`方法，再使用IO做下一步的處理
- `InputStream context.getResourceAsStream(String URIPath)`  
與上方方法相似，但直接取得InputStream物件。  
儲存InputStream的內容請使用`InputStream.readAllBytes()`;  
儲存Bytes陣列而非InputStream，儲存水而非水龍頭。
- `String context.getMimeType(String file)`  
根據file檔案的副檔名回傳所對應的MIME Type。(或直接Google)  
常見的MIME Type有

```
context.getMimeType("xxx.html")=text/html
context.getMimeType("xxx.txt")=text/plain
context.getMimeType("xxx.gif")=image/gif
context.getMimeType("xxx.jpg")=image/jpeg
```

Server內的web.xml通常會有一組「副檔名-MIME Type」的對照表，我們也可以在自己web app內的web.xml中對其做修改或擴充。

## ServletContext其他方法

在進行「檔案存取」、尤其是「圖片處理」時，要特別注意路徑的寫法。

Eclipse在執行Servlet時，執行專案的環境不會是在你的workspace下的專案位置；Eclipse會在更深層的目錄，如下

```
\workspace\專案名稱\
.metadata\plugins\org.eclipse.wst.
server.core\tmp0\wtpwebapps\
中建立一個暫存的主案來執行測試。
```

是故，如果你臨時生成在專案下的檔案是寫死「絕對路徑」，則會發生存取不到圖片的問題。

要解決這個問題，可使用「相對路徑」的寫法存取圖片，或使用context物件下的`getRealPath()`方法取得執行當下的路徑。

如`context.getRealPath("")`;

或使用

```
context.getAttribute("javax.servlet
.context.tempdir");
```

也可取得暫存專案的路徑。



# 取得Client資訊

取得Client主機的資訊

- `String request.getRemoteAddr( )`: 取得Client端主機的IP Address
- `String request.getRemoteHost( )`: 取得Client端主機的名稱
- 取得Request參數
  - `String request.getParameter(String name)`  
使用參數名稱取得Client端傳送過來的單一參數值
  - `Map<String, String[ ]> request.getParameterMap( )`  
取得所有參數名稱、參數值，以Key、Value方式存成Map物件
  - `Enumeration<String> request.getParameterNames( )`  
列舉所有參數名稱，存成Enumeration物件
  - `String[] request.getParameterValues(String name):`  
使用參數名稱取得Client端傳過來的所有參數值
  - `String request.getQueryString( )`  
取得查詢字串，較少使用

# 取得路徑資訊

【URI】 `http://localhost:8080/MyWebApp/PathTest/myServlet?name=all&age=18`

1. `request.getScheme( )=http`
2. `request.getServerName( )=localhost`
3. `request.getServerPort( )=8080`
4. `request.getContextPath( )=/MyWebApp`      【環境路徑Context Path】
5. `request.getServletPath( )=/PathTest`      【Servlet路徑Servlet Path】
6. `request.getPathInfo( )=/myServlet`      【額外路徑資訊Extra Path Information】
7. `request.getPathTranslated( )=C:\apache-tomcat-9.0.45\webapps\MyWebApp\myServlet`      【前者的絕對路徑】
8. `request.getRequestURI( )=/MyWebApp/PathTest/myServlet`      【URI路徑】
9. `request.getQueryString( )=name=all&age=18 || null`
10. `request.getProtocol( )=HTTP/1.1`
11. `request.getMethod( )=GET || POST`
12. `request.getContentType( )=null || application/x-www-form-urlencoded`
13. `request.getContentLength( )=null || 17`

## 檔案上傳：Part介面

在Servlet 3.0之前沒有Part介面，那時處理檔案需使用request物件中的getInputStream()方法，取得完整的資料流，之後再撰寫判斷式區分檔案名稱、檔案本體(非常麻煩)。

Servlet 3.0中新增了Part介面，提供程式設計師方便的進行檔案上傳，你只需要下對正確的標註即可。

- **Part** request.getPart(String name)  
其中參數name代表檔案上傳欄位的name屬性，此方法會回傳一個Part物件  
part物件的getInputStream()方法，可取得java.io.InputStream物件
- **Collection<Part>** request.getParts()  
如果有多個檔案要上傳，可使用getParts()方法，會回傳一個Collection<Part>物件
- HTML表單(form)必須符合下列格式  

```
<form action="xxxServlet" method="post" enctype="multipart/form-data">  
  <input type="file" name="xxxName">  
  <input type="submit" name="上傳">  
</form>
```

【enctype屬性】指定表單編碼，如enctype="text/plain"即代表不行進行編碼
- Servlet上方要有此項設定**@MultipartConfig**(或寫在XML)，才可使用Part API  
【屬性解釋】
  - location**  
String，檔案儲存目錄，預設值為空字串
  - fileSizeThreshold**  
int，設定檔案讀取大小，超過設定值會寫入location，預設值為0
  - maxFileSize**  
long，設定單筆檔案大小限制，單位是byte，預設值是-1L，表示無限制，超過會拋出IllegalStateException。
  - maxRequestSize**  
long，總和檔案大小限制(若有多筆input)，預設值是-1L



## Part介面方法

關於檔案命名，實務上要盡量避免讓使用者自行命名檔案。惡意使用者可能會在上傳檔案名稱中添加特定路徑或特定命名，企圖覆蓋系統檔案；存入資料庫也需注意SQL Injection問題。

上傳檔案後，Server檔案命名建議使用自己的命名規則，Client端顯示可由使用者決定(同一張資料表有兩個「名稱」欄位)。

此外，關於取得檔名，part物件的`getHeader("content-disposition")`中夾帶的filename資訊，在不同瀏覽器可能會有不同的回傳值。要使用正規表達式或String物件的方法來過濾想要的資訊。

- `Collection<String> part.getHeaderNames( )`  
回傳part物件的所有header名稱。因Collection物件特性，可直接使用`System.out.println(part.getHeaderNames( ))`，印出所有內容。
- `String part.getHeader(String name)`  
根據屬性名稱取得Part的Header屬性值，其中content-disposition含有上傳檔案名稱；content-type表示上傳檔案Mime type。
- `Collection<String> part.getHeaders(String name)`  
若屬性值有多筆則使用此方法，回傳指定屬性名稱的屬性值集合。
- `String part.getName( )`  
回傳此項part在Html頁面中設定的name名稱，可用來過濾是否為檔案(而非文字或數值，但要注意name的命名規則)。
- `long part.getSize( )`  
回傳檔案的大小，單位是byte。
- `String part.getContentType( )`  
回傳檔案的Mime type。
- `String part.getInputStream( )`  
取得輸入資料流物件，可用來做進一步的操作，如存入資料庫、後端傳遞等等。
- `String part.write(String fileName)`  
根據location設定的目錄，以傳入的字串引數作為檔名，進行寫入操作。

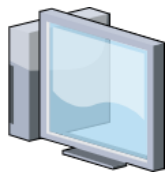
# 第七節

## RESPONSE HEADER

### 相關應用

# 傳送HTML資訊的基本觀念

- 「Response Content(回應內容)」是回應客戶端時的主要內容
  - 對HTML網頁而言，Response Content就是HTML本身文字檔
  - 對圖像而言，Response Content則是構成其影像的位元組(byte)
- 以底層的觀點來看，Web伺服器其實是將「整個回應」當成一連串的位元組傳給客戶端。所以HTTP協定規定伺服器必須：
  - 先送出「Status Line」(狀態列，包含協定、狀態碼、結果訊息)和「Response Header」(回應標頭，包含Content-Type、Content-Length)
  - 後才能送出「Response Content」(回應內容)
  - 以便客戶端能先解讀「Status Line」和「Response Header」，才能接續解讀之後送過來的「Response Content」位元組



Client

Request Line	Post /index.html HTTP/1.1
Request Header	Accept: image/gif,... Accept-Language: zh-TW, en-US... ... Content-Type: application/x-www-form-... Content-Length: 27
Request Content	name=allen&age=18

Status Line	HTTP/1.1 200 OK
Response Header	Accept-Ranges: bytes ... Content-Type: text/html Content-Length: 251
Response Content	<html>...</html>

Server



## HTTP 通訊協定

進行HTTP通訊之前，必須先建立TCP連線，建立TCP連線有三次握手，與中斷連線的四次揮手；簡言之，建立TCP連線是很消耗資源的。

HTTP通訊協定經歷了很多版本，如

### 【HTTP/0.9】

已過時，對於每個請求建立一個TCP連線，且只有GET請求方法、只能請求HTML文件、沒有回應標頭(head)、沒有狀態碼與錯誤碼，出現錯誤會回傳包含錯誤訊息的HTML文件。

### 【HTTP/1.0】

引入標頭(head)，對於每個請求建立一個TCP連線，可以傳送圖片

### 【HTTP/1.1】

現在主流版本，定義八種請求方法，預設長連線(keep-alive)，擴展更多標頭(head)，支援斷點續傳(而非每次重新下載)

### 【HTTP/2】

截至2021年10月，全球有46.5%的網站支援了HTTP/2。對標頭欄位壓縮、伺服器主動推播、資料傳輸採多路復用，讓多筆請求合併在一條TCP連線當中。

### 【HTTP/3】

棄用TCP連線，改基於UDP。  
截至2022年2月，HTTP/3仍然是草案狀態。



常見狀態碼：

【200】OK

請求已成功，請求所希望的回應頭或資料體將隨此回應返回。

【301】Moved Permanently (永久移動)

某資源永久被移動，當瀏覽器請求原網址時，告訴瀏覽器要重新導向新網址，並要求未來連結時也使用新網址(有利於搜尋引擎最佳化、SEO排名)。

【302】Found (臨時重新導向)

要求客戶端執行臨時重新導向，同時也是之後會介紹的`response.sendRedirect()`方法中會自帶的狀態碼。

【304】Not Modified(未修改)

表示請求資源沒有被修改(檢查標頭中屬性If-Modified-Since或If-None-Match)，在客戶端(瀏覽器快取)中存有副本，無須再次請求。

【400】Bad Request

由於明顯的客戶端錯誤，伺服器不能或不會處理該請求。

如格式錯誤的請求語法，過大的檔案大小，無效的請求訊息或欺騙性路由請求。

【404】Not Found

找不到請求資源，檢查你的URI路徑。

【405】Method Not Allowed

錯誤的請求方法，常見於檔案上傳指定錯誤的`method="GET"`。

【500】Internal Server Error (內部伺服器錯誤)

伺服器端錯誤，通常是程式有錯誤。

## 狀態碼(Status)

**1xx:**

資訊性，指出客戶端應繼續回應其他動作，比較少見。

**2xx:**

成功完成客戶端請求

**3xx:**

重導客戶端的請求至他處，進行下一步動作

**4xx:**

客戶端錯誤

**5xx:**

伺服器端錯誤

【設定狀態碼】

`response.setStatus(int sc)`

設定HTTP狀態碼

`resp.sendError(int sc,String msg)`

與上述方法類似，但專門用於servlet在處理Request的過程中發生的錯誤。伺服器會代為產生一個錯誤訊息網頁。

# 回應標頭設定

- 藉由Servlet設定Response Header，可提供其他有關的額外資訊，HTTP1.1的各項header設定如下
  - `void response.setHeader(String name, String value)`
    - 以上會將指定的name設定標頭值
    - 若指定的name已有標頭值，則新值會覆蓋舊值
  - `void response.addHeader(String name, String value)`
    - 同一個head需要具有多個不同值的時候，新值附加於舊值之後
- Header值應用
  - `response.setHeader("Location", String url)`:網頁重導
  - `response.setHeader("Refresh", String second)`:指定秒數後刷新頁面
  - `response.setHeader("Refresh", "3;URL=http://...")`:3秒後導向指定頁面
  - `response.setHeader("Cache-Control", "no-store")`:禁止此文件被快取 HTTP1.1
  - `response.setHeader("Pragma", "no-cache")`:禁止此文件被快取 HTTP1.0
  - `response.setDateHeader("Expires", "0")`:指該文件失效的時間，預設值為零表示立即失效

# 第八節

# SESSION TRACKING



## 【隱藏欄位】

使用HTML內的Hidden屬性

實作方式：

```
<Form action="xxxServlet">
<input type="hidden" name="sessionId" value="12345">
<input type="submit" value="送出">
</Form>
```

```
Servlet: String userSession=request.getParameter("sessionId");
```

優點：

主流瀏覽器皆支援隱藏欄位，瀏覽器不顯示其值，簡單直觀。

缺點：

檢視原始碼即可看到隱藏欄位，有安全疑慮。

## 【URL重寫】

在URL尾端加上額外的參數來達到Session Tracking目的

實作方式：

Http://local:8080/project/URLRewriting/12345

額外路徑資訊：

Http://local:8080/project/URLRewriting?sessionId=12345

```
Servlet: String userSession=request.getParameter("sessionId");
```

優點：

主流瀏覽器、伺服器皆支援，此方法對不支援(或禁用)cookie的瀏覽器有所幫助。

缺點：

資訊暴露在URL上，有安全疑慮。

# Session Tracking

網路應用程式(Web App)的運作是基於HTTP協定，而HTTP的兩大特性之一便是無狀態協定(Stateless)。這表示基於HTTP的你的網站無法「記住」使用者；對於網站來說，每一次的請求都被視為「新的請求」，每一次回應完成後都會遺忘掉所有資訊。

但一個功能正常的網站應該要有能力記住使用者，如撰寫購物車功能、登入功能等等。要如何讓網站有能力記住使用者？以目前的階段而言，總共有四種實作的方式，分別是「隱藏欄位」、「URL重寫」、「Cookie」、「Session API」。其中前兩項有安全性與實作上的問題，後兩項為現在主流的設計模式。本節重點會放在「Session API」。

**Stateful(有狀態)的例子【FTP】**

屬於Stateful(有狀態)的通訊協定

從使用者登入建立連線後，Server會一直記住使用者，直到登出

**Stateless(無狀態)的例子【HTTP】**

Server無法記住使用者，它只關心請求與回應，一旦回應結束，Server與Client的連線也同時結束。

- `Cookie c = new Cookie(String key, String value)`  
使用new方法建立一個cookie，並將key、value當成建構子參數傳入。
- `String c.getName()`  
取得此cookie的key(name)值。
- `String c.getValue()`  
取得此cookie的Value值。
- `void c.setMaxAge(int second)`  
設定此cookie的存活時間，單位為秒。預設值為-1。  
引數為-1：當瀏覽階段結束時銷毀cookie。(離開網域即算，不用關瀏覽器)  
引數為 0：立刻銷毀cookie。(cookie沒有delete方法)
- `void c.setHttpOnly(boolean httpOnly)`  
設定此cookie只給http協定使用，無法被JavaScript呼叫。預設值為false。  
開放cookie被JavaScript呼叫會有資安風險，視情況使用即可。
- `void response.addCookie(Cookie c)`  
將cookie加入回應標頭(Header)當中，若有同名cookie則取代之。  
你無法在同名cookie中新增value字串，只能取得全部value，改寫再覆蓋。
- `Cookie[] request.getCookies()`  
取得所有此請求中附加的cookie，回傳cookie陣列。  
沒有`request.getCookie(String cookieName)`這種方法。  
要操作cookie請撰寫迴圈，並用`c.getName()`判斷對象；或採用第三方jar。

## Cookie

小餅乾，儲存在客戶端的瀏覽器裡，不同的瀏覽器在相同的頁面會拿到不同的Cookie，不同的網站(domain)無法存取彼此的Cookie。不同瀏覽器、不同網站對於Cookie的處理是獨立的。

Cookie若存在客戶端的記憶體中，會「自動」附加在每一次Http的請求當中，無須自己處理。你要做的只有建立Cookie，並把它附加在Response的Header裡，客戶端在收到含有Cookie的Header時，會自動在瀏覽器的儲存空間建立Cookie。

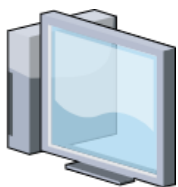
Cookie是純文字檔，以Key、Value的方式儲存資訊，儲存大小大約4KB左右(依瀏覽器決定)。因Cookie是儲存在客戶端，只要能操作客戶端，所有資訊一覽無遺，故不適合儲存敏感資料，如密碼、信用卡資訊等等。

只有同網域的Cookie會被附在Http請求之中。



http://localhost:8080/ShoppingCart/add/book

帶有請求參數「book」的GET請求



Client



Server

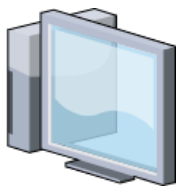
回應OK，且將cookie加入回應標頭中

#### 【新增商品的Servlet】

1. 使用API **建立** Session (自動建立ID、cookie)
2. 將「book」加入Session屬性中
3. 回應使用者新增商品成功

http://localhost:8080/ShoppingCart/getList

使用GET請求，取得購物車內容清單  
(自動附帶cookie資訊)



Client



Server

回應商品清單(文字，非物件)

#### 【取得清單的Servlet】

1. 使用API **取得** Session (自動根據Cookie取得Session)
2. 從Session裡拿出商品清單
3. 將物件轉成文字回應給Client

## Session API

主流追蹤使用者的手段，基於**cookie**實現追蹤能力。

API會自動在客戶端儲存一筆**cookie** Key值為「識別ID」、Value值為「自動產生的長字串」。以Tomcat伺服器為例，自動產生的**cookie**名稱為「JSESSIONID」，其值假設為「ID#123」。概念類似發放號碼牌。

API也會在伺服器中紀錄一筆Session ID，Key為「ID#123」(即客戶端**cookie**的value)，其值可由程式設計師進行增刪改查。

每次使用者進行Http請求時，會自動附上**cookie**中的「識別ID」(JSESSIONID)與其值(value)，伺服器可使用此value存取後端存儲的屬性。

因為Session存在於後端，故可存入物件、文字、二進字資料等等，但取出時要記得強制轉型。

# HttpSession介面方法介紹

- `HttpSession session = request.getSession( )`  
便利的方法，自動判斷Session是否存在，  
若無則產生Session ID、建立cookie、將cookie加入回應標頭(header)；  
若Session存在則回傳對應的Session(依據ID)。  
可傳入引數true或false，若傳入引數false，則判斷Session不存在時直接回傳null。  
也就是說`request.getSession( ) == request.getSession(true)`。  
預設存活時間，cookie => 瀏覽器關閉；Session => 30分(Tomcat)。
- `void session.setMaxInactiveInterval(int seconds)`  
設定此項(非全部)Session的最大存活時間，單位為秒。引數為0或負數表示永久有效，應盡量避免，會讓Server增加許多負擔(Server關閉才會移除)。若要使用務必記得撰寫主動銷毀Session的方法。
- `long session.getCreationTime( )`  
回傳Session建立的時間，以 1970/1/1 00:00 GMT為基準。單位是毫秒(千分之一秒)。
- `long session.getLastAccessedTime( )`  
回傳Session最後一次被存取的時間，以 1970/1/1 00:00 GMT為基準。單位是毫秒(千分之一秒)。
- `long System.currentTimeMillis( )`  
定義在java.lang.System下的方法，回傳呼叫當下的時間，以1970/1/1 00:00 UTC為基準。單位是毫秒(千分之一秒)。  
【延伸閱讀】[什麼是GMT、UTC?](https://pansci.asia/archives/84978) <https://pansci.asia/archives/84978>
- `boolean session.isNew( )`  
若此項Session是剛被建立，回傳true。

# HttpSession介面方法介紹2

- `void session.setAttribute(String name, Object value)`  
在這項Session中設定新屬性，第二引數的接受Object型別的物件，故可放入所有物件。  
類似的方法有`request.setAttribute(String name)` & `context.setAttribute(String name)`。  
此三項方法主要是作為Server端資訊共享用途。
- `Object session.getAttribute(String name)`  
依引數name回傳指定物件，因型別為Object，故取出時記得強制轉型。  
類似的方法有  
`request.getAttribute(String name)`，  
別跟`request.getParameter(String name)`搞混了。  
`context.getAttribute(String name)`，  
別跟`context.getInitParameter(String name)`搞混了。
- `void session.invalidate( )`  
直接銷毀此項Session，包含裡面儲存的所有屬性(Attribute)。

# 設定Session的存活時間

- 每次進行`request.getSession( )`時，都會重置Session的存活時間。
- 如未設定，Tomcat伺服器預設Session存活時間為30分鐘，cookie為離開網站時消滅。
- 個別的Session存活時間，可使用`session.setMaxInactiveInterval(int seconds)`，參數單位是秒。
- 個別Cookie存活時間，需使用`request.getCookies( )`，再撰寫迴圈判斷JSESSIONID(Tomcat預設值)，使用`c.setMaxAge(int seconds)`重設存活時間，並使用`response.addCookie(c)`加入標頭。
- 對於所有Session，可使用web.xml檔進行設定：

```
<web-app>
  <session-config>
    <session-timeout> 120 </session-timeout>  <!-- 單位為分鐘 -->
    <cookie-config>
      <max-age> 120 * 60 </max-age>           <!-- 單位為秒 -->
    </cookie-config>
  </session-config>
</web-app>
```



# 第九節

## **SERVER**資訊共享

- 主要分為兩大方法：資訊共享、控制權共用
- 資訊共享
  - 將一個資訊以屬性-屬性值的方式存入ServletContext或HttpSession之中，以實現資訊共享。
  - ServletContext:存入的屬性，所有Servlet都能存取。
  - HttpSession:存入的屬性只有同一次Session能夠存取。
- 控制權共用(調派請求)
  - 控制權共用可讓兩個以上的Servlet共用同一個request的控制權，有forward(轉送)、include(包含)兩項方法。
  - 使用這兩項方法前，須先取得RequestDispatcher物件(調派請求物件)
  - RequestDispatcher rdp=request.getRequestDispatcher(String path)
    - 回傳調派請求物件，此物件下實作了兩個方法forward()、include()
    - forward()  
Servlet可以轉送整個request給其他Servlet，再由其他Servlet進行回應，request的控制權在於後者。
    - include()  
Servlet可以在自己的程序當中，包含其他Servlet產生的回應，request的控制權在於前者。
  - 基本上使用request.getRequestDispatcher(String path).forward()

## Server資訊共享

一個架構完善的Web App，不可能只靠一支Servlet完成所有操作與功能，故勢必得讓多支Servlet(或JSP)協同合作，為了方便所有、或者彼此相關的Servlet能夠存取到相同資訊，Server必須使用些方法進行後端資訊共享。

主要方法是將「欲共享資訊」放入context、session、request區間之中。

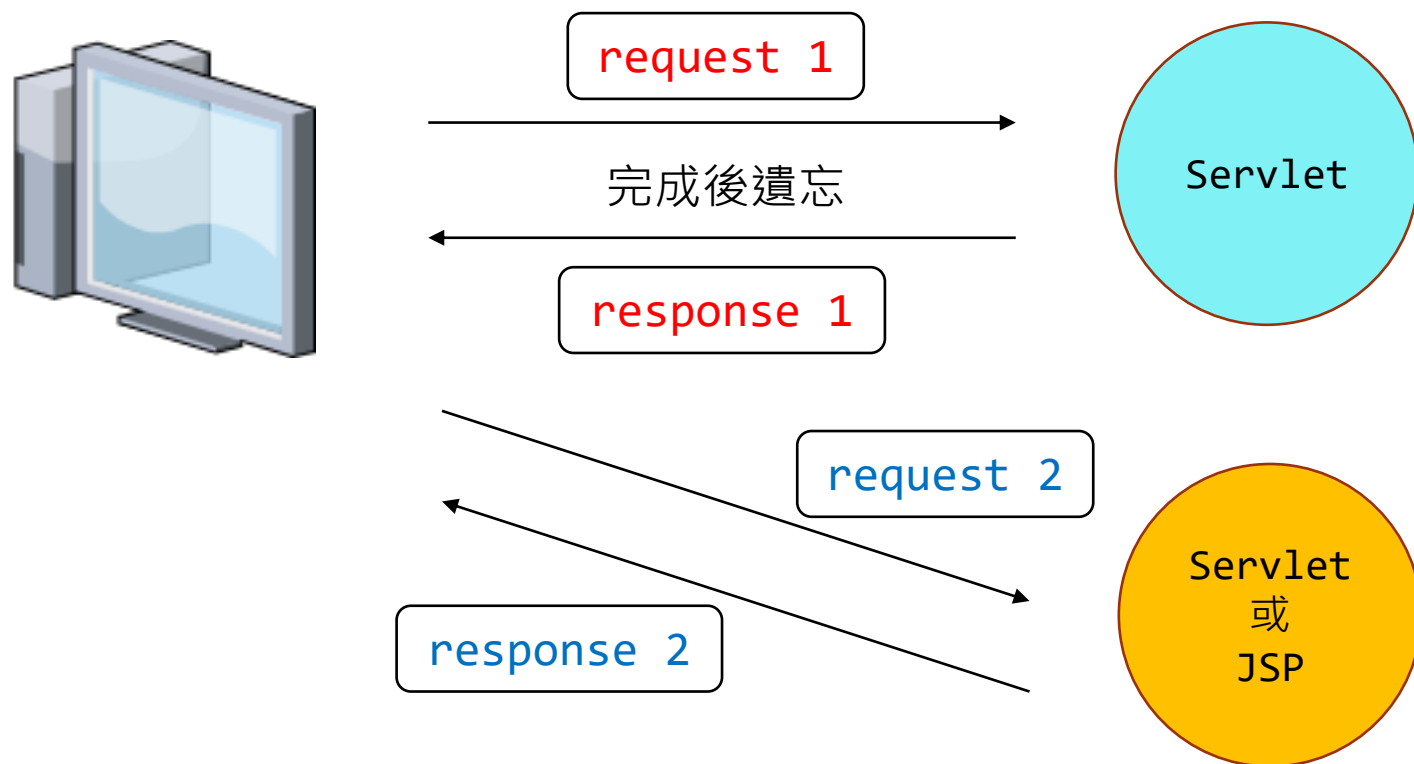
放入context表示Server關閉之前所有Servlet都能共享資訊。

放入Session則表示在使用者離開網站之前(預設，取決於你如何設定session、cookie存活時間)，所有Servlet都能共享資訊。

放入request則表示「只有這一次的請求與回應」可以共享資訊。

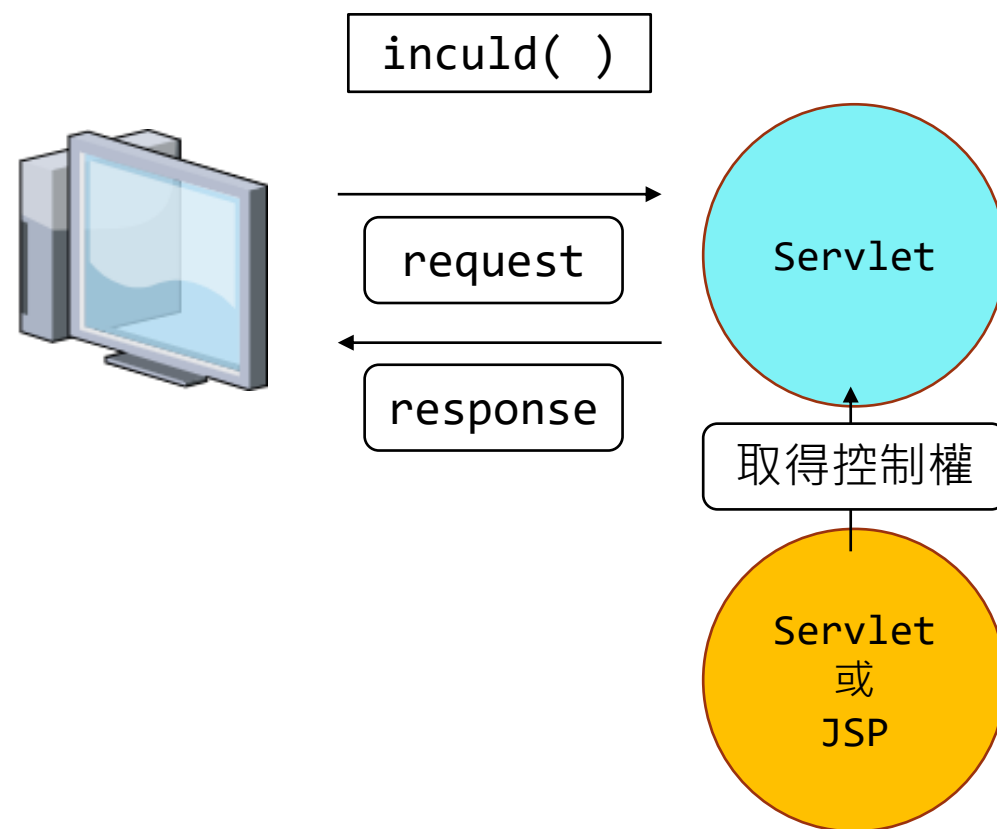
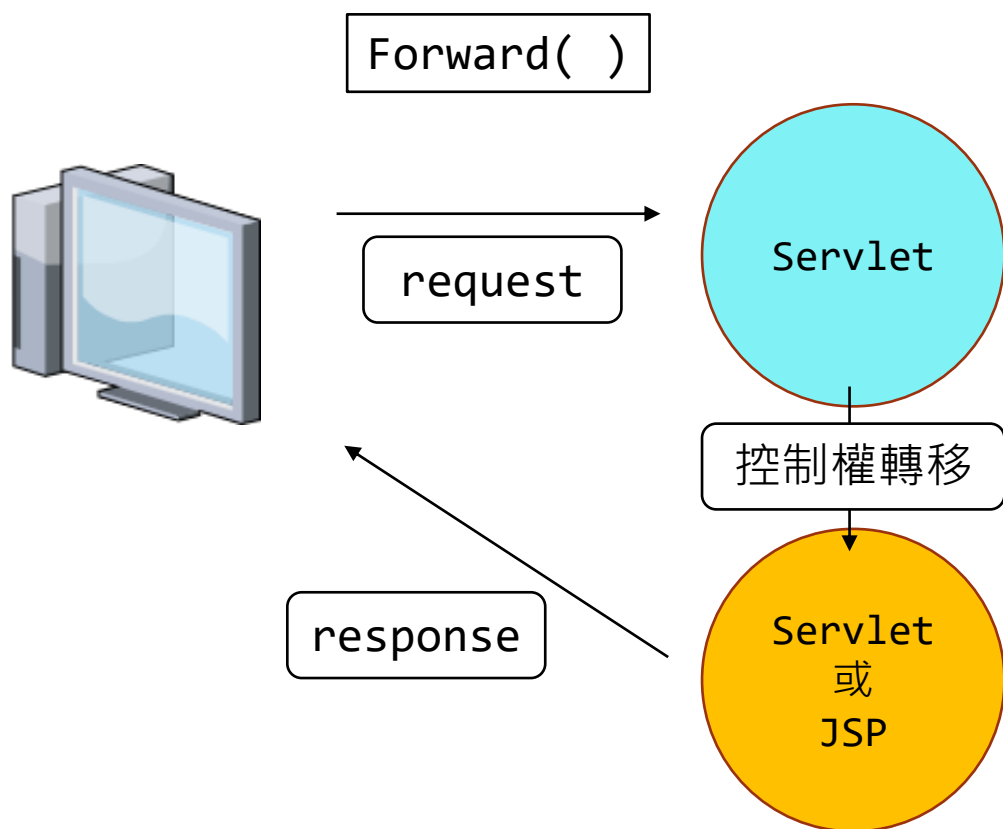
# 頁面重導

- 使用`response.sendRedirect(String path)`
  - 會產生兩次請求，兩次回應。第一次請求後，回應302暫時重新導向；客戶端收到回應後，對新URI發出第二次請求。
  - 理論上速度較慢，但差異不大。
  - 網址列會變動。(因對新URI做請求，資源位置不同。)
  - 存在`request`區間的屬性會無法存取，因第一次完成請求/回應後即銷毀`request`。(無狀態協定)
  - 要共享資訊請使用`session`或`context`。
  - 重新整理(F5)不會造成太大影響(新URI地址)。



# 調派請求

- 使用`request.getRequestDispatcher(String path).forward(request, response);`
  - 僅有一次請求與回應，客戶端不會意識到後端是哪支程式回應的。
  - 網址列不會變動。對客戶端而言，只會看到第一次請求的對象進行回應。(即使你後面forward了10次也是)
  - 存在request區間的屬性，會因為request的控制權被移轉而跟著移轉(相同的request)。
  - 共享資訊可使用三個區間request、session、context。
  - 重新整理(F5)會對同一支Servlet做出相同請求，千萬要注意。



# 第十節 資料庫連結



# 兩種連線的基本模式(使用純JDBC)

- 將JDBC連線寫在doGet/doPost內，作為區域變數

優點:不會有Transaction(交易)問題

缺點:每次請求都會重啟連線，耗費大量資源

- 將JDBC寫在Servlet內，作為全域變數

- 使用init()建立連線實體

- 使用destroy()關閉連線

- 優點:只建立一次連線

- 缺點:

會持續佔用一個連線直到Servlet被銷毀，若此Servlet的極少被執行，則造成佔用浪費。

- 若此Servlet的執行頻率極高，程式內又使用Transaction，可能會造成多個執行緒同時共用一個資料庫連線的不正常情況(使用Transaction時資料庫連線不可共用)

## 資料庫連結

Server不可能把資料都存在Context裡，不切實際也不方便管理資料，與資料庫進行連線、存取資料是必須的。

與DataBase(DB，資料庫)進行連線，是基於TCP/IP協定，而建立TCP連線非常消耗資源。(【補充】為什麼資料庫連線很消耗資源?)

在一個單日可能擁有上千、上萬次請求的Web App裡，不可能針對每一個請求都建立一次資料庫連線。是故，便有了連線池(Connection pool)的概念出現。

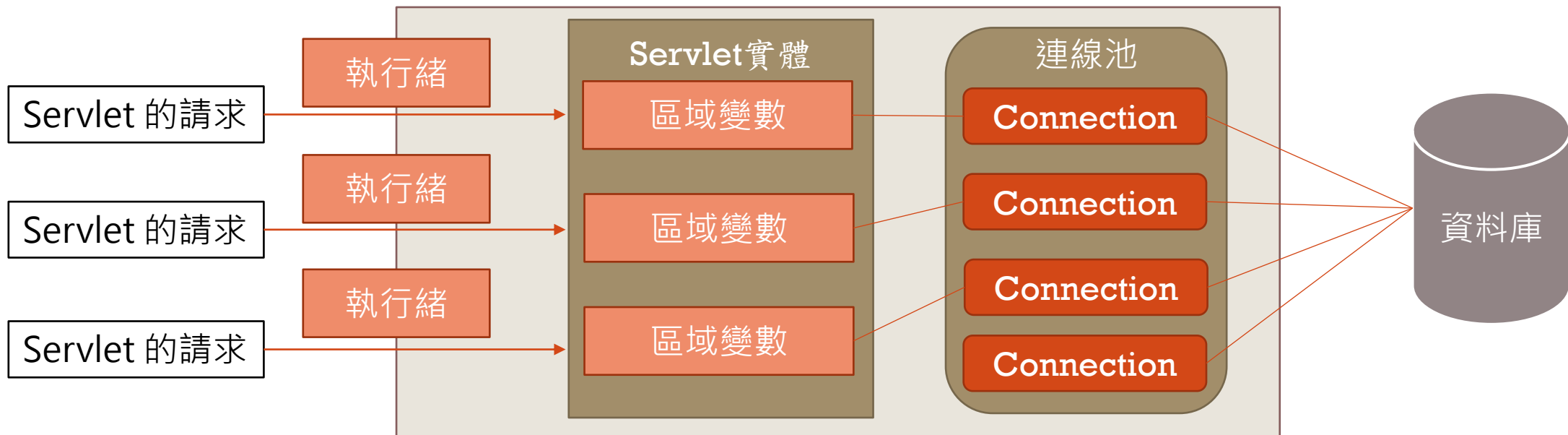
連線池即是在Server啟動時，預先建立數條連線，當有需要時從池中(pool)取出連線，使用完後歸還。

建立連線池後，取得連線物件需仰賴JNDI(Java Naming and Directory Interface)，Java 命名與目錄介面。



## 連線池Connection pool基本觀念

- 為了解決上述兩種方法的缺點，最佳的方式是使用Connection pool(連線池)
- Connection pool是一種對資料庫連線(Connection物件)的管理機制，它可以在Server啟動時，預先建立起「若干個資料庫連線」，並分配這些連線給有需要的Servlet。
- Servlet在有需要的時候從Connection pool裡取出連線物件，使用完後歸還



<Resource

name="資源名稱" type="實作類別"

auth="取用方式" username="使用者名稱"

password="使用者密碼" driverClassName="使用的驅動程式類別"

url="資料庫連結URL" />

設定context.xml

## 參數說明

name: 用於參考資料庫資源的JNDI路徑，路徑為java:comp/env/資源名稱

type: 用於尋找JNDI資源時所使用的實作類別，通常為javax.sql.DataSource

auth: JNDI資源取用方式，若提供Servlet使用引數為Servlet，若提供JSP使用則引數為Container。

## 設定部署描述檔web.xml 建立JNDI資源參考

<resource-ref>

<description>部署描述說明資訊</description>

<res-ref-name>資源名稱</res-ref-name>

<res-type>實作類別</res-type>

<res-auth>取用方式</res-auth>

</resource-ref>

設定web.xml

## 說明

- description: 給部署者看的說明性文字
- res-ref-name: 用於參考資料庫資源的JNDI路徑，路徑為java:comp/env/資源名稱
- res-type: 用於尋找JNDI資源時所使用的實作類別，通常為javax.sql.DataSource
- res-auth:  
JNDI資源取用方式，若提供Servlet使用引數為Servlet，若提供JSP使用則引數為Container。

# 使用連線池

在此項例子，連線池的實作由Tomcat完成，若有興趣研究，也可以使用第三方jar。

## 使用java.sql.DataSource介面

一個DataSource物件就代表一個資料庫，應用程式透過這個介面來取得資料庫連線物件(Connection)

需進行兩項XML設定

context.xml

web.xml

設定完成後，在程式中使用JNDI去尋找並存取DataSource

## 透過JNDI取得資料庫連結

- 透過JNDI取得資料庫連結

```
InitialContext context = new InitialContext();  
DataSource ds = (DataSource)context.lookup("java:comp/env/資源名稱");  
Connection conn1 = ds.getConnection();  
//當Servlet程式中自訂使用者身分驗證時，  
//需設定<res-auth>Servlet</res-auth>  
Connection conn2 = ds.getConnection("user", "password")
```

# 設定範例

**Tomcat server**下的【context.xml】

```
<Resource
    name="jdbc/TestDB(名稱可自訂)" type="javax.sql.DataSource"
    database="myDb" auth="Container" username="sa" password="123456"
    driverClassName="com.microsoft.sqlserver.jdbc.SQLServerDriver"
    url="jdbc:sqlserver://localhost:1433;"/>
```

**你的專案**下的【web.xml】

```
<resource-ref>
    <description>My MSSQL Connection Pool</description>
    <res-ref-name>jdbc/TestDB(名稱可自訂)</res-ref-name>
    <res-type>javax.sql.DataSource</res-type>
    <res-auth>Container</res-auth>
</resource-ref>
```

【在程式中的取法】

```
InitialContext context = new InitialContext();
DataSource ds = (DataSource)context.lookup("java:comp/env/jdbc/TestDB");
Connection conn = ds.getConnection();
```

# 第二章

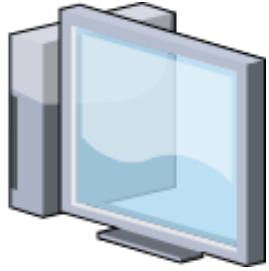
## JSP

# 第一節

# JSP介紹



Client



Http Request



Server



Http Response

請求執行一支「查詢所有商品」的程式  
/GetAllProducts.do

Servlet(GetAllProducts.do)接收到請求  
→ Servlet使用JDBC跟資料庫要「所有商品」的資料  
→ Servlet取得「所有商品」的資料  
→ Servlet使用response.getWriter()物件回應資料

客戶端使用者



給我所有商品的頁面



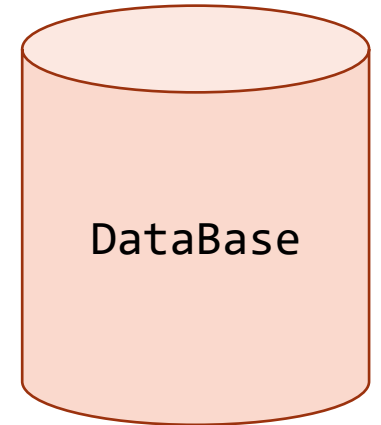
GetAllProduct.do

你要的頁面：  
`out.write("<!Doctype html>");`  
`out.write("<html> ");`  
`out.write("<head>");`  
`out.write ...`

使用JDBC取得所有商品資料



DataBase



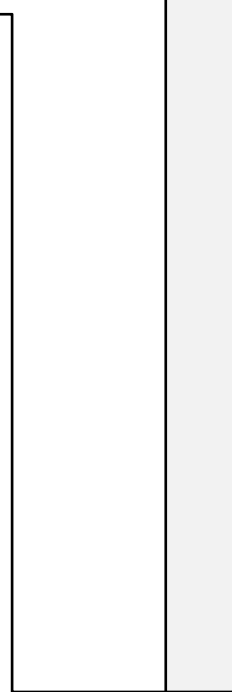
客戶端使用者A



給我所有商品的頁面



你要的所有商品頁面



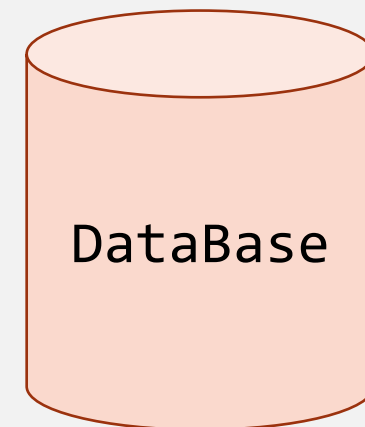
盒子



使用JDBC取得所有商品資料



Server



這個盒子裡面有所有商品的資料  
幫我回應給客戶端使用者



```
protected void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
    String userName = "Amy";

    PrintWriter out = response.getWriter();

    out.write("<!DOCTYPE html>");
    out.write("<html>");
    out.write("<head>");
    out.write("<meta charset=\"UTF-8\">");
    out.write("<title>Welcome!</title>");
    out.write("</head>");
    out.write("<body>");
    out.write("<div>Hello! " + userName + "</div>");
    out.write("</body>");
    out.write("</html>");

    out.close();
}
```

你不會想用這麼天才的寫法寫網頁的

這種寫法有缺陷，  
但的確是JSP的一種展現

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Welcome!</title>
</head>
<% String userName="Amy"; %>

<body>
    <div>Hello! <%= userName %> </div>
</body>
</html>
```

## JSP(Jakarta Server Pages)

原叫做Java Server Pages，後因甲骨文公司(Oracle)將Java EE轉讓給Eclipse基金會，而在2018/02改名成Jakarta Server Page(因商標問題)。

為了簡化動態網頁的建立，昇陽公司(Sun Microsystems)提出了JSP這項技術標準。

只要在Html中加入Java程式片段和JSP標籤，且副檔名使用.jsp，即可構成JSP網頁。

JSP存放位置和HTML存放位置相同，但要注意JSP是在後端執行的。JSP會轉換成Servlet，最後輸出成HTML。

JSP生命週期與Servlet類似：

```
_jspInit()→
_jspService()→
_jspDestroy()。
```

白話點說，JSP就像是個挖很多洞的模板頁面(Template Page)，需仰賴其他程式(Servlet)賦予值。

客戶端使用者A



給我所有商品的頁面

你要的所有商品頁面  
(最後使用者收到的會是HTML頁面)  
(但資源名稱還是.jsp或.do)

盒子

GetAllProduct.do

使用JDBC取得所有商品資料

Server

DataBase

JSP

裡面寫好了很多固定的HTML樣板，  
與很多待填入資料的「洞」。  
把盒子裡面的物件取出後，放入挖好的洞，  
最後全部轉為HTML輸出給客戶端。

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Welcome!</title>
</head>
<% String userName="Amy"; %>
<body>
    <div>Hello! <%= userName %> </div>
</body>
</html>
```

# JSP

## 【Element】

Container會解析、處理的部分

### 1. Scripting Element(描述語言元素)

- `<% ... %>`
- `<%! ... %>`
- `<%= ... %>`

### 2. Directive Element(指令元素)

- `<%@page ... %>`
- `<%@include ... %>`
- `<%@taglib ... %>`

### 3. Action Element(動作元素)

- `<jsp:forward>`
- `<jsp:include>`
- `<jsp:useBean>`
- 其他

## 【Template】

HTML的部分(固定的樣板)

## Template & Element

JSP 程式通常會包括 Template Data & Element兩部分。

Template Data是指其中HTML的部分，Web Container不會處理這部分，會直接當成文字輸出給 Client。

Element指的是Container會處理的部分，即為`<%...%>`標籤隔開的部分。

Element 又分成三個部分：Scripting、Directive、Action。



## Scripting Element (描述語言元素)

此類元素作為JSP最初使用的元素，可讓你在JSP的頁面中撰寫JAVA程式碼，用以存取其他物件的屬性、方法

但一個JSP頁面若都使用此類元素來撰寫，則會讓程式碼變得太過凌亂（混和了JAVA與HTML兩種程式碼），且往後會難以維護。

是故在一個良好的MVC架構下，此類元素不應該出現在任何JSP頁面中，此小節理解即可。

### 1. `<%! Int i=0;%> <%! 方法 %>`

- 宣告(Declaration)
- 此為實體變數、實體方法

### 2. `<% 任意JAVA程式碼 %>`

- 小腳本(Scriptlets)
- 如內有宣告變數，則其為區域變數

### 3. `<%= a+b+c %>`

- 運算式(Expressions)
- 不可使用「；」結尾
- 結果會以字串方式輸出在HTML上

## Directive Element (指令元素)

此類元素通常用於整個JSP頁面的說明。

功能為宣告頁面屬性、載入某個類別或某項標籤，你通常會在JSP頁面的開頭看見它。

不需要記下所有<%@page>的屬性，需要使用時再查詢即可。

### 1. <%@page ... %>

- 如:<%@pagecontentType="text/html;charset=UTF-8"pageEncoding="UTF-8"import="java.util.\*" %>
- 有10多種屬性(其餘屬性)

### 2. <%@include ... %>

- 如:<%@ include file="aaa/xxx.xx">
- 目錄aaa下的檔案xxx.xx，先包含至此JSP後，才被轉譯成.java檔(Servlet)，最後輸出成HTML

### 3. <%@taglib ... %>

- 主要用於宣告使用「標籤」，在JSTL章節會提到。

## Action Element (動作元素)

此類元素以標籤格式撰寫 (類HTML 標籤)，根據語法不同會執行不同動作，但幾乎已被淘汰不使用。

唯一可能會用上的標籤是 `<jsp:include>`，其他不是被取代就是被淘汰，知道有此類元素存在即可。

### 1. `<jsp:forward>`

將頁面控制權轉發到指定對象，但正常情況下不應使用此標籤，你不該在一個JSP中把控制權轉移到另一個JSP中，會讓程式凌亂且難以維護。

### 2. `<jsp:include>` 將指定對象內容納入此頁JSP。

### 3. `<jsp:param>` 結合以上兩個標籤，傳遞參數使用。

### 4. `<jsp:useBean>`

若Bean存在，則使用；

若不存在則建立，已被EL表達式取代。

### 5. `<jsp:setProperty>` 結合`<jsp:useBean>`使用

### 6. `<jsp:getProperty>` 結合`<jsp:useBean>`使用

### 7. `<jsp:plugin>`

上古時代(1990)用於執行Java Applet程式。

### 8. `<jsp:fallback>`

當你的plugin或useBean載入失敗，可使用此標籤向客戶端顯示錯誤訊息。但你根本沒機會用那兩個標籤。

## <jsp:includ> 與 <%@include ... %>

懶人包結論：用<jsp:includ>

- <%@include ... %>
  - 指令元素(Directive Element)
  - 在編譯時間就會把目標檔案納入自身，效率稍微高一點點。
  - 在目標檔案的所有「元素」都會被納入編譯，意指目標檔案的「元素」會影響自身。
  - 【範例】<%@include file="navigate.jsp" %>
- <jsp:includ>
  - 動作元素(Action Element)
  - 在編譯時期不會把目標檔案納入自身，代客戶端請求抵達後才納入且編譯，較為彈性。
  - 【範例】把product.jsp的頁面納入自身，並給予屬性。(略有component的意味)  
組件  

```
<jsp:include page="product.jsp">  
    <jsp:param name="productName" value="豆花" />  
    <jsp:param name="productPrice" value="30塊" />  
</jsp:include>
```

# JavaBean

- JavaBeans是一個用來包裝資料、重複使用的軟體元件(資料載體)
- 一個標準的JavaBean必須有以下3個特性：
  - 必須是一個公開的(**public**)類別，必須擁有一個不帶參數的建構子
  - 其屬性不可直接被存取，必須透過**get**、**set**方法來取用與設定
  - 必須是一個可序列化(**Serializable**)的類別，必須 implements **java.io.Serializable**
- JSP使用<jsp:useBean>時，可將JavaBean直接嵌入JSP網頁中



## <jsp:useBean> (過時了!)

- useBean標籤語法

```
<jsp:useBean id="beanName"  
  class="packageName.classname"  
  scope="page|request|session|application" />
```

- useBean標籤的語法簡介
- 在指定的範圍內尋找以id名稱為識別名稱，型別為packageName.classname的屬性物件實體，並將此物件實體存入變數BeanName內。如果不存在，會建立新的javaBean物件，並以id名稱為識別名稱，放置於指定的scope範圍內提供存取。
  - id:javaBean的識別名稱
  - class:javaBean fully-qualify classname(完整類別名稱包含套件及類別名稱)
  - scope範圍：指定javaBean物件的存取範圍，共有四個範圍(page、request、session及application)，預設為page存取範圍

# <jsp:setProperty> (過時了!)

- setProperty標籤語法

```
<jsp:setProperty name="beanName"
  property="propertyname"
  value="propertyValue" />
```
- setProperty標籤的語法說明
- 表示設定bean物件的某個property(特性)的內含值
  - name: 指定bean物件的名稱，此名稱對應到useBean中的id值
  - property: 設定bean物件的property(特性)名稱
  - value: 設定property(特性)的值(直接給值)
  - param: 表單控制項名稱(從網頁表單中取得欄位值)

# JSP隱含物件

- JSP有9個隱含物件(Implicit Object)，不用宣告即可直接使用

Implicit Object	Type	Scope
request	HttpServletRequest	request
response	HttpServletResponse	Page
pageContext	PageContext	Page
session	HttpSession	Session
application	ServletContext	Application
out	PrintWriter	Page
config	ServletConfig	Page
page	Object	Page
exception	Throwable	Page

- 有屬性的隱含物件:pageContext、request、session、application(存取範圍小→大)

# 第二節

## PL表達式語言

1. 語法: `${expression}`
2. 若要讓JSP頁面忽略EL表達式的運算，則可使用以下方法
  - `<%@ page isELIgnored="true" %>` → 整個頁面都忽視
  - `\${expression}` → 忽視單筆EL
3. EL的保留字

true	false	and	or
not	eq	gt	lt
ge	ne	le	mod
div	null	empty	instanceof

#### 4. EL的運算子

類型	運算子	範例
算術運算子	<code>+</code> <code>-</code> <code>*</code> <code>/</code> (div) <code>%</code> (mod)	<code>\${10/3}</code> <code>\${6 mod 2}</code>
關係運算子	<code>==(eq)</code> <code>!=(ne)</code> <code>&gt;(gt)</code> <code>=(ge)</code> <code>&lt;=(le)</code>	<code>\${(6*2)&gt;=15}</code> <code>\${(6*2) ge 15}</code>
邏輯運算子	<code>&amp;&amp;(and)</code> <code>  (or)</code> <code>!(not)</code>	<code>\${name!="mary"}</code>
empty運算子	empty判斷值是否為空	<code>\${empty name}</code>
條件運算子	<code>A?B:C</code>	<code>\${price&gt;=1000?"便宜":"貴"}</code>

## EL表達式語言 (Expression Language)

EL表達式在JSP2.0、Tomcat5以上皆有支援。其被設計用於取代Scripting, Directive, Action等元素。

在JSP頁面中使用EL表達式，可簡潔的取出物件屬性；傳統的MVC架構中會大量使用EL表達式

EL表達式與下一節會提到的JSTL(JSP標準標籤庫)有緊密的關聯。

EL表達式僅能取出單一物件的屬性，若想取出List或Set集合的所有物件屬性，則需使用JSTL(JSP標準標籤庫)。



# 使用EL讀取字串

語法:\${識別名稱}

- 【範例】

<%

```
String title="This is a JSP Page";  
request.setAttribute("contentTitle",title);
```

%>

- 使用EL取得方式

Page Title:\${contentTitle}

# 讀取Map物件與陣列物件

- Map物件: \${識別名稱.key}

- 【範例】

```
<% Map map = new HashMap();  
    map.put("username", "Allen");  
    map.put("password", "1234");  
    request.setAttribute("login", map);%>
```

- 使用EL取得方式

```
帳號:${login.username}<br>  
密碼:${login.password}<br>
```

- 陣列物件: \${識別名稱[序號]}

- 【範例】

```
<%  
    String[] fruit={"banana","pineapple","longan"};  
    request.setAttribute("taiwanFruit", fruit);  
%>
```

- 使用EL取得方式

```
台灣水果: ${taiwanFruit[0]}<br>  
台灣水果: ${taiwanFruit[1]}<br>
```

# 讀取JavaBean物件

- 【範例】

```
<%  
    Users u = new Users();  
    u.setName("Allen");  
    u.setAge(18);  
    request.setAttribute("myUserBean", u);  
%>
```

使用EL取得方式：

```
${ myUserBean.getName() }  
${ myUserBean.age }
```

# 讀取Scope物件、 param物件、 cookie物件

- **【Scope物件】**
  - **pageContext**: 相對於JSP的隱含物件pageContext
  - **pageScope**: 取得page範圍的某屬性名稱所對應的值
  - **requestScope**: 取得request範圍的某屬性名稱所對應的值
  - **sessionScope**: 取得session範圍的某屬性名稱所對應的值
  - **applicationScope**: 取得application範圍的某屬性名稱所對應的值
- **【param物件】** 相當於使用request.getParameter("參數名稱");
  - 例: `${param.參數名稱}`
- **【cookie物件】**
  - 例: `${cookie.name.value}` 取得儲存於cookie內name的變數值

## 【備註】

EL也可以直接存取JSP的所有隱含物件，其中可使用 `${pageContext.request.contextPath}` 取得執行的專案名，以利於絕對路徑的撰寫。

# 第三節

# JSTL標準標籤函式庫



## 【JSTL所提供的標籤函式庫主要分為五大類】

### 1. 核心功能標籤庫(Core tag library)

提供條件判斷、屬性存取、URL 處理及錯誤處理等標籤。

### 2. I18N格式標籤庫(I18N-capable formatting tag library)

提供數字、日期等的格式化功能，以及區域 ( Locale )、訊息、編碼處理等國際化功能的標籤。

### 3. SQL標籤庫(SQL tag library):

提供基本的資料庫查詢、更新、設定資料來源 ( DataSource ) 等功能之標籤。

### 4. XML處理標籤庫(XML tag library):

提供XML剖析、流程控制、轉換等功能之標籤。

### 5. 函式功能標籤庫(Functions tag library):

提供常用字串處理的自訂EL函式標籤庫。

## 【下載&安裝】

### 1. JSTL並不在JSP的標準規範中，若想使用JSTL，需另行下載JSTL的實作。

- apache-tomcat-7.0.6x 以前的版本，需下載  
jstl.jar 、 standard.jar
- apache-tomcat-7.0.6x 以後的版本，需下載  
taglibs-standard-impl-1.2.5.jar  
taglibs-standard-spec-1.2.5.jar

並放置於webapp\Web-INF\lib 的目錄內。

### 2. 去哪下載?

- tomcat\webapps\examples\WEB-INF\lib 的目錄內取得
- Tomcat官網:<https://tomcat.apache.org/download-taglibs.cgi>

# JSTL(JavaServer Pages Standard Tag Library)

JSP標準標籤函式庫，以HTML標籤的形式封裝了常用的Java功能，比如for迴圈、if條件式try catch偵錯等等。讓不懂JAVA的前端工程師也可以撰寫JSP頁面(在以前)。

JSTL主要分五大類標籤，分別負責不同的功能，此次課程中只會使用到**核心功能標籤庫(Core tag library)**，若對於其他標籤感興趣，可自行查詢用法。

# JSTL五大類標籤庫宣告

JSTL標籤庫	網址資源URI參考	慣用前置碼Prefix
Core標籤庫	<a href="http://java.sun.com/jsp/jstl/core">http://java.sun.com/jsp/jstl/core</a>	c
XML標籤庫	<a href="http://java.sun.com/jsp/jstl/xml">http://java.sun.com/jsp/jstl/xml</a>	x
I18N格式標籤庫	<a href="http://java.sun.com/jsp/jstl/fmt">http://java.sun.com/jsp/jstl/fmt</a>	fmt
Database標籤庫	<a href="http://java.sun.com/jsp/jstl/sql">http://java.sun.com/jsp/jstl/sql</a>	sql
Functions標籤庫	<a href="http://java.sun.com/jsp/jstl/functions">http://java.sun.com/jsp/jstl/functions</a>	fn

► 所有使用JSTL的JSP網頁都必須加入與標籤庫有關的敘述

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

```
<%@ taglib prefix="x" uri="http://java.sun.com/jsp/jstl/xml" %>
```

```
<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>
```

```
<%@ taglib prefix="sql" uri="http://java.sun.com/jsp/jstl/sql" %>
```

```
<%@ taglib prefix="fn" uri="http://java.sun.com/jsp/jstl/functions" %>
```

# JSTL核心標籤庫簡介<c:...>

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

功能分類	標籤名稱
運算式操作	out
	set
	remove
	catch
流程控制	if
	choose
	when
	otherwise
遍歷操作	forEach
	forTokens
URL操作	import
	url
	redirect

<c:標籤名稱>

## <c:out> 列印

- 語法

<c:out

value="expression" [default="expression"] [escapeXml="boolean"]

/>

屬性	說明
value	將運算結果列印在HTML中
default	若運算結果為null或空字串，則列印此項屬性值
escapeXml	若為true，則替換以下字元  < → &lt; > → &gt; & → &amp;

## <c:set> 設定變數

- 語法

<c:set var="string" value="expression" scope="scope"/>

屬性	說明
var	宣告的變數名稱
value	存入變數的值
scope	變數存取範圍，可設定為 page(預設) request session application

## <c:remove> 移除變數

- 語法

<c:remove var="string" scope="scope"/>

屬性	說明
var	刪除指定名稱的變數
scope	表示待刪除的變數來自哪個範圍 page(預設) request session application



## <c:catch> 錯誤處理

- 語法

<c:catch [var="string"]>

...欲捕捉錯誤的部分

</c:catch>

屬性	說明
var	將捕捉到的錯誤存入指定變數中

## <c:if> 條件控制

- 語法

```
<c:if test="條件式" [var="string"] [scope="scope"]>  
    body本體(條件式為真時，才執行本體)  
</c:if>
```

屬性	說明
test	根據條件式的布林值，決定是否顯示本體
var	存入條件式運算後的結果，為true或false
scope	可存取var變數的範圍 page(預設) request session application

# <c:forEach> 迴圈、遍歷

■ 語法

```
<c:forEach items="collection" [var="string"] [begin="int"]  
[end="int"] [step="int"] [varStatus="可填入以下值"] >
```

```
body本體  
</c:forEach>
```



屬性	說明
items	執行迴圈的集合(collection)
var	當次迴圈目標的代表名稱
varStatus	當次迴圈的狀態
begin	開始的位置
end	結束的位置
step	迴圈的間隔數

屬性	說明
current	當次迴圈指向的目標
index	當次迴圈目標的索引
count	第幾次迴圈
first	此目標是否為迴圈起始值?
last	此目標是否為迴圈最末值?
begin	此迴圈的"begin"屬性值
end	此迴圈的"end"屬性值
step	此迴圈執行間距

# <c:forTokens> 文字分割

## ■ 語法

```
<c:forTokens items="string" delims="符號" [var="string"]  
[begin="int"] [end="int"] [step="int"] [varStatus="可填入以下值"]>  
    body本體  
</c:forEach>
```

屬性	說明
items	文字分割對象
var	當次迴圈目標的代表名稱
delims	使用什麼符號分割
varStatus	當次迴圈的狀態
begin	開始的位置
end	結束的位置
step	迴圈的間隔數



屬性	說明
current	當次迴圈指向的目標
index	當次迴圈目標的索引
count	第幾次迴圈
first	此目標是否為迴圈起始值?
last	此目標是否為迴圈最末值?
begin	此迴圈的"begin"屬性值
end	此迴圈的"end"屬性值
step	此迴圈執行間距

## <c:import> 載入指定URL的網頁或文件

### ■ 語法

```
<c:import url="url" [context="context"] [var="varName"]  
[scope="scope"] [charEncoding="charEncoding"]>  
    body本體  
</c:import >
```

屬性	說明
url	載入目標的URL，可為網頁或文件 與<jsp:include>類似，但可載入外部資源
context	相同container的其他context(同Server中的不同專案) 必須以/開頭
var	載入的網址或文件的內容，存入變數
scope	可存取var變數的範圍 page(預設) request session application
charEncoding	設定被載入的網址或文件的字元編碼



## <c:redirect> 將目前頁面導向指定的URL

### ■ 語法

```
<c:redirect url="url" [context="context"]>  
  (參數傳遞)<c:param name="name" value="value"/>  
</c:redirect >
```

屬性	說明
url	導向的目標網址
context	相同container的其他context(同Server中的不同專案) 必須以/開頭
c:param	傳遞參數標籤，以GET形式傳遞

## <c:url> 根據標籤屬性設定，產生URL字串存入變數

### ■ 語法

```
<c:url value="url" [context="context"] [var="varName"]  
[scope="scope"]/>
```

(參數傳遞)<c:param name="name" value="value"/>

```
</c:url >
```

屬性	說明
value	目標網址URL
context	相同container的其他context(同Server中的不同專案) 必須以/開頭
var	將產生的URL字串存入變數
scope	可存取var變數的範圍 page(預設) request session application
c:param	傳遞參數標籤，以GET形式傳遞

# 第三章 完善專案

# 第一節

## MVC架構

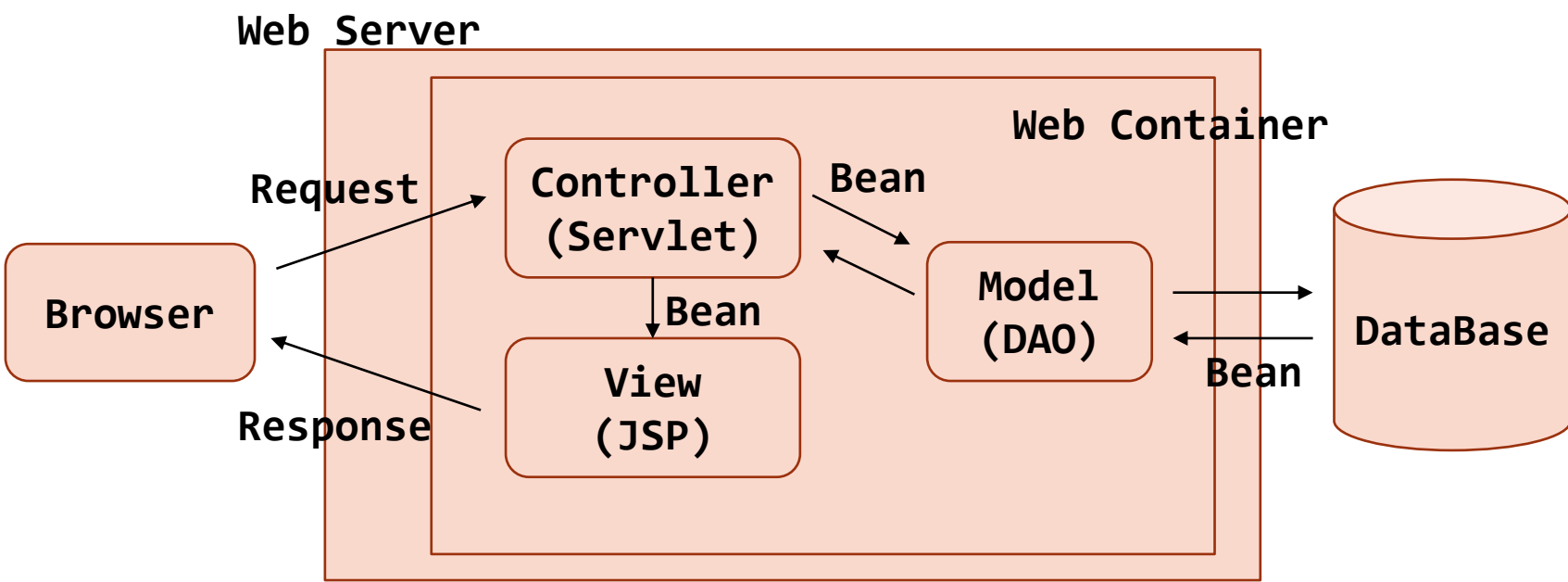
# MVC架構

MVC架構是一種軟體架構。MVC模式的目的是實現一種動態的程式設計，使後續對程式的修改和擴充簡化，並且使程式某一部分的重複利用成為可能。

除此之外，此模式透過對複雜度的簡化，使程式結構更加直覺。軟體系統透過對自身基本部分分離的同時也賦予了各個基本部分應有的功能。

<維基百科>

在MVC架構的網路應用系統中，Controller負責處理使用者提出的請求，根據情求呼叫適當的Service元件，使用Model元件對資料進行封裝，最後根據處理的結果來呼叫適當的View元件渲染頁面輸出給使用者。



- Model: 模型 (JavaBean 資料載體 + DAO 資料存取物件)
  - JavaBean: 資料載體 (資料容器)
  - DAO (Data Access Object): 資料存取物件
  - Service: 服務，商業邏輯主要撰寫處
- View: 視圖 (Template 模板頁面，即為 JSP)
- Controller: 控制器 (呼叫 Service 商業邏輯、控制導向)

# Model

- 主要負責管理資料與邏輯。
- 裡面通常會有「資料載體」JavaBean、「資料存取物件」DAO、「商業邏輯處理」Service。不過在嚴格定義下，DAO跟Service應該分割出來。但現階段而言都放在同一個package下即可。
- 關於package的命名，有依系統分割或依功能分割的規劃法，如：
  - 【依系統】
    - `tw.com.eeit.shop.model.bean.Product`
    - `tw.com.eeit.shop.model.dao.ProductDAO`
    - `tw.com.eeit.shop.model.dao.ProductDAOImpl`
  - 【依功能】
    - `tw.com.eeit.model.bean.shop.bean.Product`
    - `tw.com.eeit.model.dao.shop.ProductDAO`

包裹 + 類別



# View

- 主要負責渲染(回應)畫面給使用者，直白的可理解成「使用者看到的畫面」，不論這個畫面是由JSP、HTML或Servlet產生。
- 負責產生「使用者看到的畫面」的程式稱之為View。
- 在Java web中，通常是由JSP扮演此角色。

## 【渲染】

- 原指在電腦繪圖中，以軟體由模型生成圖像的過程。
- 在網頁設計中，也被拿來指「把超文本(HTML)的文字轉成畫面，呈現在瀏覽器上」的行為。
- 在JSP中，後端渲染是指「把資料填入挖好洞的JSP頁面」，完成後回應成HTML給客戶端。

# Controller

- 負責接受使用者的動作，讀取使用者輸入的資料、進行資料的檢核、呼叫Model中元件執行處理並選擇所要呈現的View畫面等等。
  - 讀取使用者輸入的資料
    - `String name = request.getParameter("name");`
  - 進行資料的檢核
    - 檢查使用者輸入的資料是否有錯誤，若資料正確則送至後端元件處理，若資料不正確則將頁面導向錯誤頁面的View。
  - 呼叫Model中元件執行處理
    - 進行資料的資料的判讀並選擇呼叫對應的Model元件。
  - 選擇所要呈現的View畫面
    - 根據執行的結果選擇適當的View元件。
- 在前後端分離的設計模式中，會直接回應客戶端資料，而非導向View。

# 第二節

# LISTENER

# HttpSessionBindingListener介面

- 介面用法：
  - 實作此介面為物件，在此物件「綁定bind」或「脫離unbind」自某個session時，即會得到通知、並能夠進行某種動作。
    - implements HttpSessionBindingListener
- 可實作以下兩個方法
  - public void valueBound(HttpSessionBindingEvent event)
    - 當此物件被綁定(bind)到session時，此方法會啟動
  - public void valueUnBound(HttpSessionBindingEvent event)
    - 當此物件脫離(unbind) session時，此方法會啟動
    - 不管是此物件自session中被移除；或此物件消失、被取代；或session本身失效，都算是脫離session

# ServletContextListener介面

- 介面用法：
  - 實作此介面為物件，在Servlet Context被「初始化」或「銷毀時」，即會得到通知並進行某特定動作。
  - implements ServletContextListener
- 可實作以下兩個方法
  - public void contextInitialized(ServletContextEvent sce)
    - 當context啟動時，此方法會啟動
  - public void contextDestroyed(ServletContextEvent sce)
    - 當context關閉時，此方法會關閉
- 必須使用XML或annotation註冊

```
<listener>  
  <listener-class>package.class</listener-class>  
</listener>
```

## 其他Listener

- ServletContextAttributeListener
  - ServletContext某屬性(Attribute)加入、移除、被取代時的Listener
- HttpSessionAttributeListener
  - HttpSession某屬性(Attribute)加入、移除、被取代時的Listener
- HttpSessionListener
  - HttpSession建立或銷毀時的Listener



# 第三節

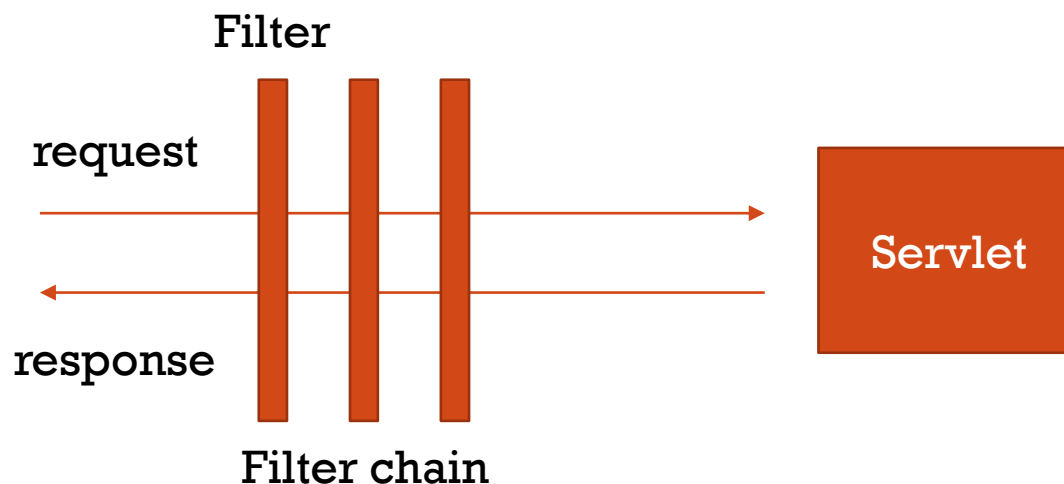
# FILTERS

# 何謂Filters

- Servlet中十分實用的功能，過濾器Filters
  - 可對request做前置處理：
    - Filters可在request抵達Servlet之前，對request做一些前置處理
    - 例如身分驗證、過濾字元...等
  - 可對request做後置處理：
    - Filters可在request離開Servlet之後，對response做一些後置處理
    - 例如對輸出進行壓縮、對XML的呈現進行轉換...等
- javax.servlet.Filter並不是Servlet，它是一種可以轉換request或修改response的Java物件

# Filter API

- `init(FilterConfig fig)`
  - 此方法只在該Filter初始化時執行一次
  - `FilterConfig`的`getInitParameter(String name)`方法，能夠取得web.xml中的初始參數值
- `doFilter(ServletRequest req, ServletResponse resp, Filter chain)`
  - 此方法執行實際的過濾動作
  - `FilterChain`是關於Filter執行順序的物件，這個順序設定在web.xml中
  - `FilterChain`的`doFilter(req, resp)`方法，會將程式控制權交給後續的過濾器(如沒有後續的Filter，則是要請求的目標網頁)，返回後，再繼續執行原來的程式
- `destroy()`
  - 銷毀Filter時呼叫



## web.xml設定

```
<filter>
  <filter-name>Filter註冊名稱</filter-name>
  <filter-class>package.class</filter-class>
</filter>
<filter-mapping>
  <filter-name>Filter註冊名稱</filter-name>
  <url-pattern>/*</url-pattern>
  <servlet-name>*</servlet-name>
</filter-mapping>
```

可透過指定URL執行filter

也可透過指定註冊的servlet名字執行filter  
若兩者都設定，則優先比對URL，再比對servlet

若某URL或Servlet套用多個Filter，則以web.xml裡  
出現順序為執行順序(上方的Filter先執行)

# 備註

# WEB APPLICATION 環境部署



## 基本觀念

安裝、設定Web Application時，應遵循Servlet/JSP規格書之規定，幾個要點如下：

1. 網站應用系統(Web Application，簡稱Web App)
2. 目錄結構
3. 部署描述檔web.xml(Deployment Descriptor)
4. 網站備存檔.war(Web Application Archive)

當一個Web應用系統被部署到Web容器時，目錄架構必須符合以下格式

Webapps/

- | - projectName/
  - xxx.html or default.jsp
  - 自訂目錄/(Ex:images目錄、網頁目錄)
  - WEB-INF/
    - web.xml:部署描述檔DD(Deployment Descriptor)
    - classes/
      - packagename/  
xxx.class:放置類別檔的位置
    - lib/  
xxx.jar:放置Jar Library函數庫檔案的位置

## Web App(網路應用程式)

一個Servlet Container可以包含多個Web App環境，每個環境即稱為一個Servlet Context。

以Tomcat為例，資料夾webapps之下的每個子目錄都被視為一個Web App。

- 將寫好的程式輸出成war檔，以便部署到其他Server。
- 可使用Eclipse，對專案點選右鍵 → Export → WAR file。
- 也可使用jdk的jar.exe命令建立：  
到Web App目錄>使用cmd輸入 `jar cvf MyWebApp.war *`
- 放置到Web Container之下的webapps目錄下，當啟動Server時會自動被解壓縮。
- 輸出時要注意Source Files有沒有需要輸出。
- Server在執行期間，請勿刪除war檔案，會連專案一併移除。

## 網頁應用程式檔(.war) Web Application Archive

在軟體工程中，WAR檔案（Web應用程式歸檔，Web application ARchive）是一種JAR檔案，其中包含用來分發的JSP、Java Servlet、Java類、XML檔案、標籤庫、靜態網頁（HTML和相關檔案），以及構成Web應用程式的其他資源。

### 【優缺點】

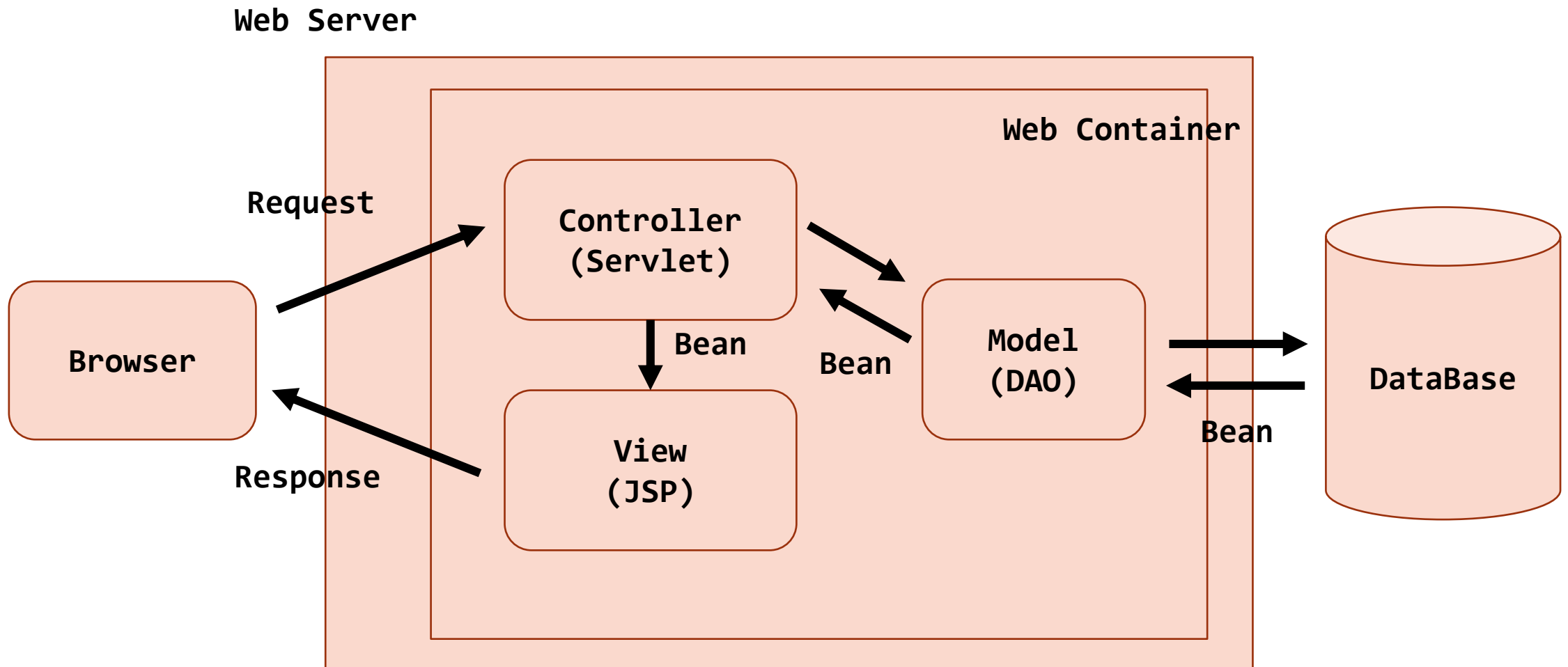
WAR檔案有如下優點：

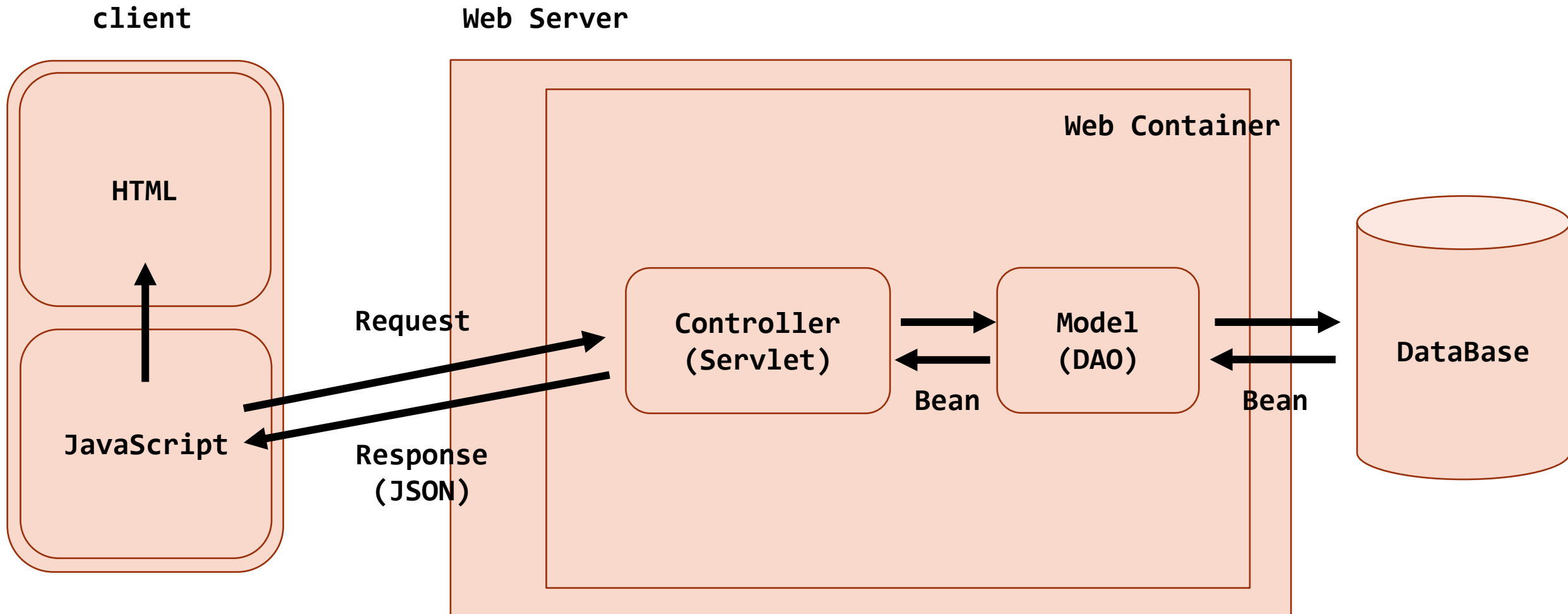
- 易於部署和測試
- 已部署的應用程式，其版本很容易辨別
- 所有的Java EE容器都支援.WAR檔案

使用WAR檔案進行Web部署的一個缺點是，即便是細微的修改，也不能在程式執行時進行。任何修改都需要重新生成和部署整個WAR檔案。

<維基百科>

# MVC架構與前後端分離







# 頁面路徑問題

```
<img src='${pageContext.request.contextPath}/img/doge.jpg'>
```

```
<c:set var="root" value="${pageContext.request.contextPath}/">  
<img src='${root}/img/doge.jpg'>
```

```
@WebListener  
public class Initialize implements ServletContextListener {  
    public void contextInitialized(ServletContextEvent sce) {  
        ServletContext context = sce.getServletContext();  
        context.setAttribute("root", context.getContextPath());  
    }  
}
```

```
}  
<img src='${root}/img/doge.jpg'>
```

**Thank  
you**