



# Java Servlet

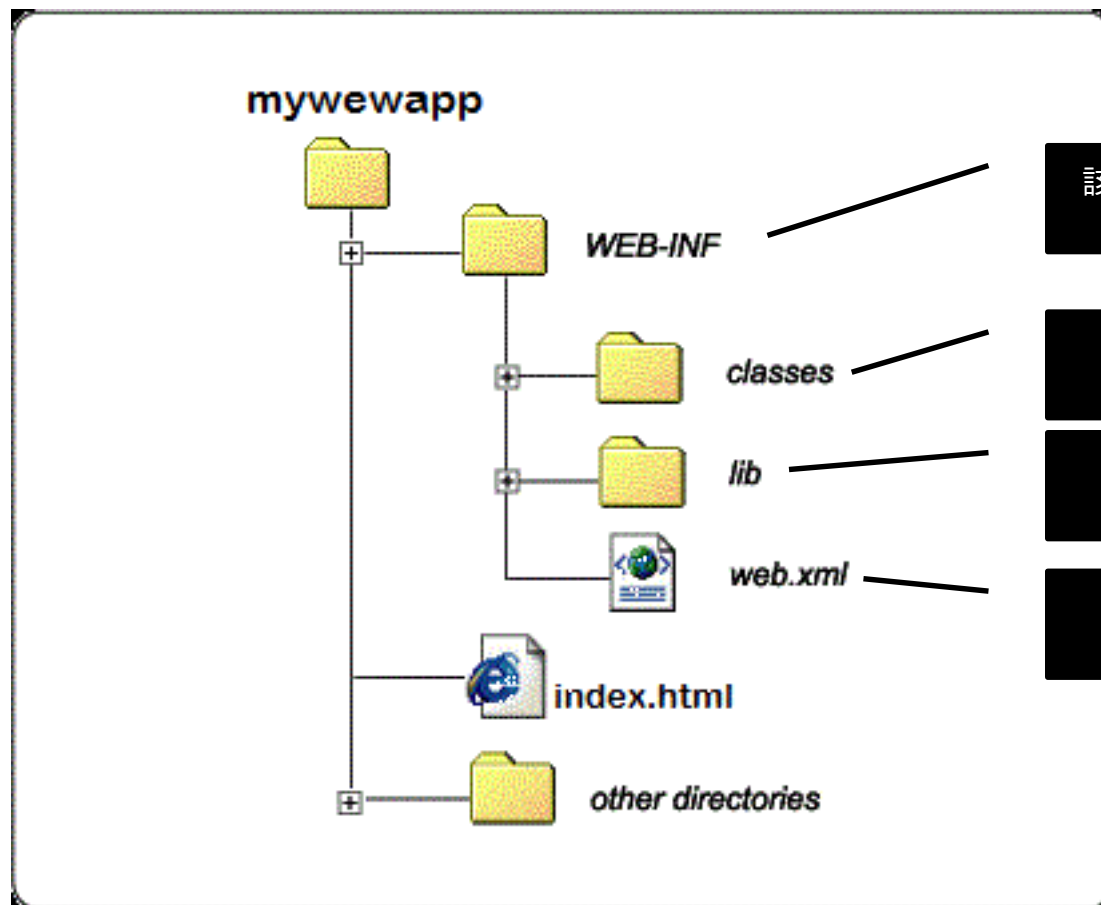
# 綱 要

- 一、送出請求 (request)
- 二、處理請求 (GenericServlet & HttpServlet)
- 三、實作我的第一個 Servlet
- 四、分析請求
- 五、送出回應
- 六、生命週期

# 綱 要

- 一、送出請求 (request)
- 二、處理請求 (GenericServlet & HttpServlet)
- 三、實作我的第一個 Servlet
- 四、分析請求
- 五、送出回應
- 六、生命週期

# Java Web 應用程序的目錄結構



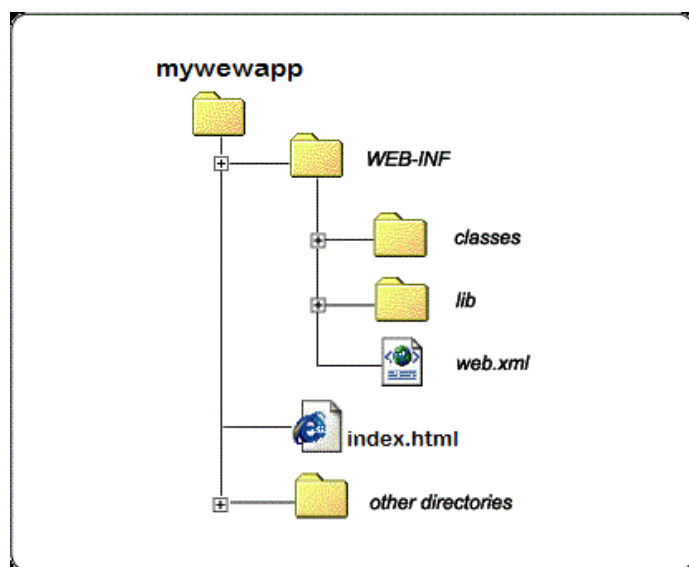
該目錄是配置設定(meta-info)目錄。存儲在此處的文件不應從瀏覽器訪問

classes 目錄包含所有已編譯的 Java 類

Web 應用程序使用的所有 JAR 資源文件

web 應用部署描述檔

# WAR 檔



mywewapp.war

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應用程式的資源。

# 一、送出請求 (*request*)

瀏覽器送出請求的事件 (*request event*)

- 超連結 *HyperLink*。
- *HTML* 表單 (*submit*)。
- 利用瀏覽器的網址列。
- *Javascript : reload() or submit()* 方法

送出請求的 *HTTP* 方法：

- *GET*、*HEAD*、*POST*、*PUT*、*DELETE*、*OPTIONS* 與 *TRACE*。

# 一、送出請求 (*request*)

使用 *Get* 的情境與限制：

- 欲得到指定資源(網頁、文件或圖片影像等)
- *Bookmark* (加入到我的最愛)
- 在不管安全性的情況下與 *Server* 有少量的互動
- 整體 *URL* 最好不要超過 240 個字元

使用 *Post* 的情境與限制：

- 傳送大量資料或檔案請求 *Server* 予以協助處理。
- 上傳的資料或檔案大小基本上沒有限制。

# 綱 要

- 一、送出請求 (request)
- 二、處理請求 (GenericServlet & HttpServlet)
- 三、實作我的第一個 Servlet
- 四、分析請求
- 五、送出回應
- 六、生命週期



## 二、處理請求

*JavaEE* 的文件中提供二種 *Servlet* 套件供開發人員使用

- *javax.servlet*

- 在 *javax.servlet* 套件中我們可以使用 *GenericServlet* 類別開發一般性的 *Servlet* 供伺服器使用而 *GenericServlet* 也將實作 *Servlet* 類別

- *javax.servlet.http*

- 若要將客戶端與伺服器端彼此能夠更緊密的溝通且需要用到 *HTTP* 協定時通常我們會使用 *HttpServlet* 類別來開發

一般來說要開發與協定有關的 *Servlet* 時應該要繼承 *HttpServlet* 類別，反之則繼承 *GenericServlet* 類別。

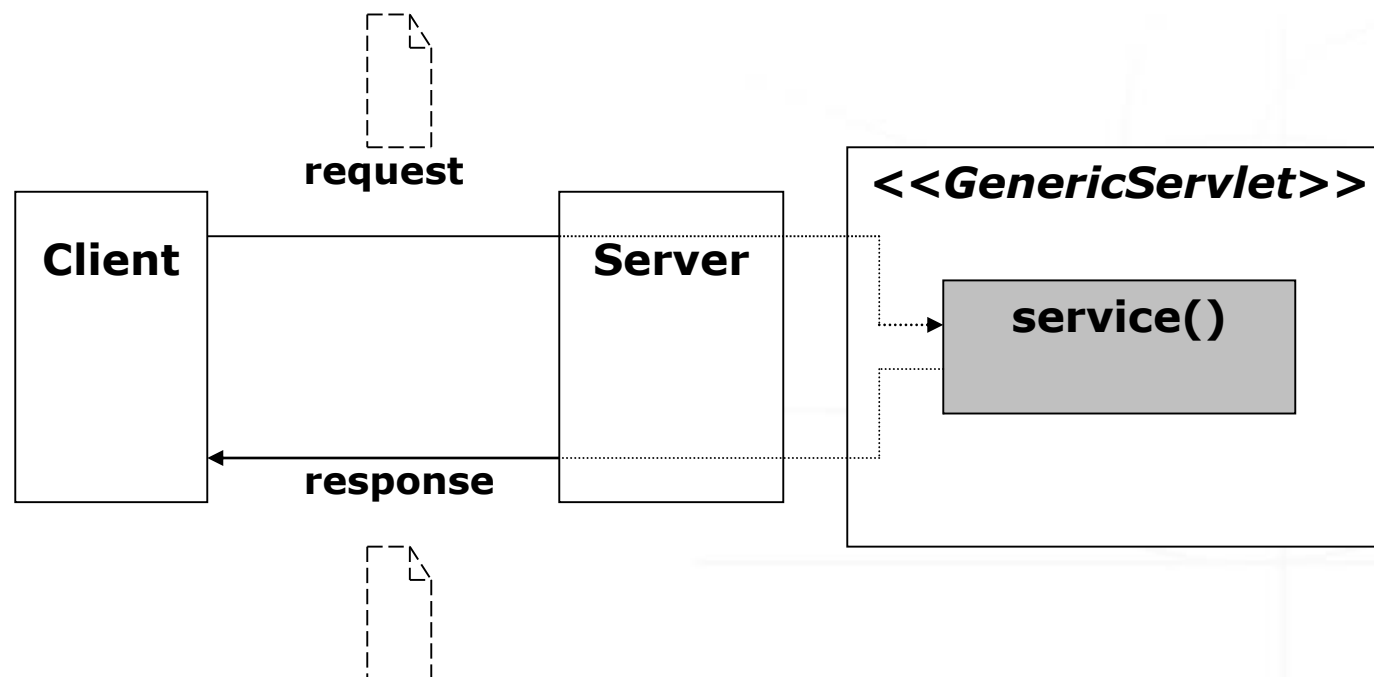
## 二、處理請求

### *GenericServlet* :

- *GenericServlet* 類別可管理 *Servlet* 的生命週期 (*init()*、*destroy()*)，*Servlet* 環境變數與*Log*。
- 使用 *GenericServlet* 時應要覆寫 *service()* 方法並處理 *client* 端的 *request*。

## 二、處理請求

*GenericServlet* 處理請求與回應  
(*request* / *response*) 的流程：



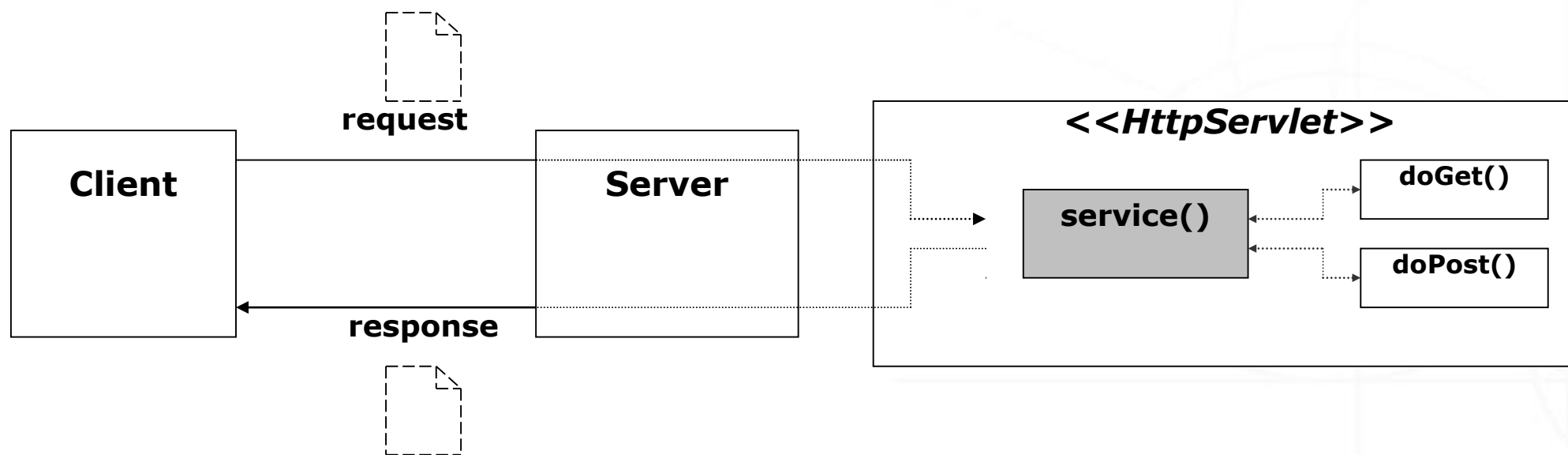
## 二、處理請求

### *HttpServlet* :

- 用到*Http*協定時可使用*HttpServlet*。
- 使用 *HttpServlet* 類別時若非必要請不要覆寫 *service()* 方法。
  - *service()* 會自動根據 *client* 端所請求的 *HTTP* 來分派 *doXXX()* 方法，*GET request* 則分派到 *doGet()* 方法，*POST request* 則分派到 *doPost()* 方法，所以有關處裡 *request* 的程式碼應寫在 *doXXX()* 方法中。

## 二、處理請求

*HttpServlet* 處理請求與回應  
(*request* / *response*) 的流程：



## 二、處理請求

- *HttpServlet* 中提供了 2 組 *service()* 方法：

<i>service(ServletRequest req, ServletResponse resp)</i>
<i>service(HttpServletRequest req, HttpServletResponse resp)</i>
請注意 <i>service()</i> 是一個覆載方法

- *Client* 端所發出的請求會先被 *HttpServlet* 中的 *service(ServletRequest req, ServletResponse resp)* 首先接到再轉交給 *service(HttpServletRequest req, HttpServletResponse resp)*，之後此方法會根據請求的HTTP的方法分派到對應的 *doXXX()* 方法進行實作。

## 二、處理請求

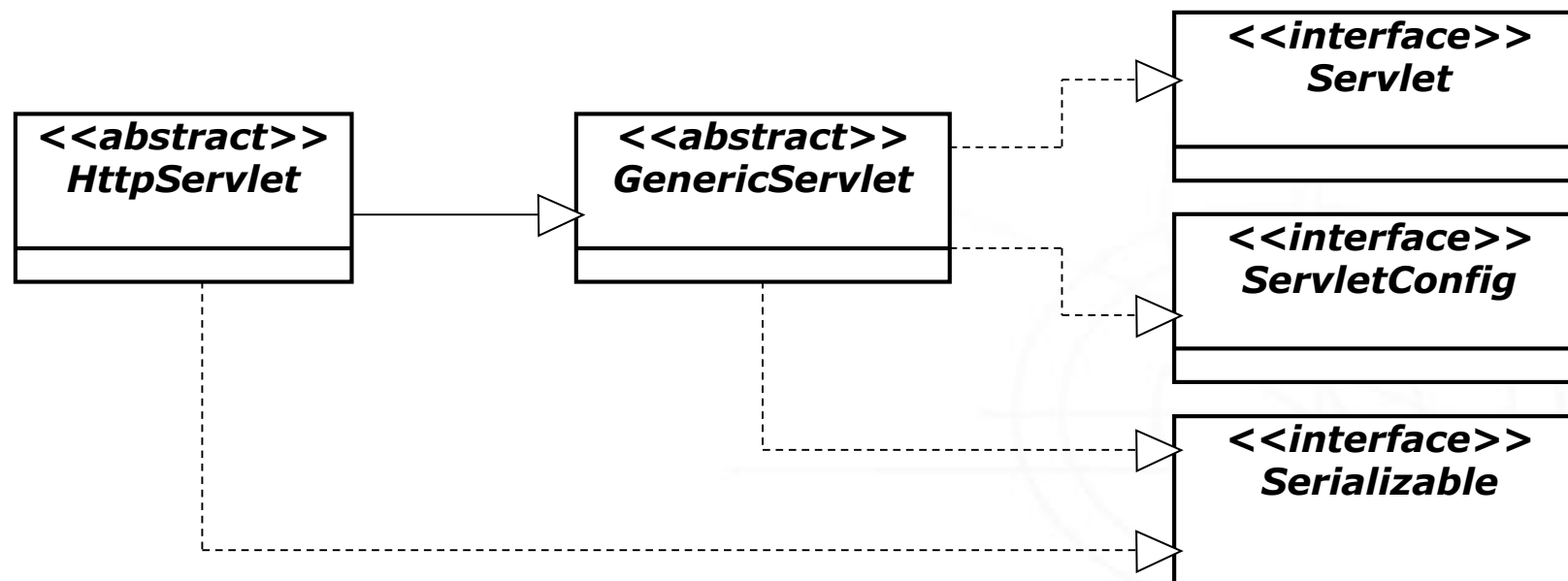
### • *GenericServlet* 與 *HttpServlet* 類別

<i>GenericServlet</i> 類別	<i>HttpServlet</i> 類別
<pre>public <u>abstract</u> class GenericServlet extends java.lang.Object implements <u>Servlet, ServletConfig,</u> java.io.Serializable</pre>	<pre>public <u>abstract</u> class HttpServlet <u>extends GenericServlet</u> implements java.io.Serializable</pre>

- *GenericServlet* 與 *HttpServlet* 類別都是 *abstract* 類別，*HttpServlet* 則會繼承 *GenericServlet* 類別。

## 二、處理請求

類別關係圖：





## 二、處理請求

**HTTP** 方法對應於 **HttpServlet** 方法：

<b>HTTP 方法</b>	<b>HttpServlet 方法</b>
<b>GET</b>	<b>doGet()</b>
<b>HEAD</b>	<b>doHead()</b>
<b>POST</b>	<b>doPost()</b>
<b>PUT</b>	<b>doPut()</b>
<b>DELETE</b>	<b>doDelete()</b>
<b>OPTIONS</b>	<b>doOptions()</b>
<b>TRACE</b>	<b>doTrace()</b>

# 綱 要

- 一、送出請求 (request)
- 二、處理請求 (GenericServlet & HttpServlet)
- 三、實作我的第一個 Servlet
- 四、分析請求
- 五、送出回應
- 六、生命週期

## 三、實作我的第一個 Servlet

- 實作三個Servlet

- 通用型 GenericServlet

- `servlet.GeroageServlet` (web.xml 部署)  
喬治

- Http協定型 HttpServlet

- `servlet.HelenServlet` (web.xml 部署)  
海倫
    - `servlet.HelloServlet` (@註釋 部署)  
哈囉

# 三、實作我的第一個 Servlet

我的第一個 *GenericServlet* 程式：

— *GeorgeServlet.java*

```
01. import javax.servlet.*;  
02. import java.io.*;  
03. public class GeorgeServlet extends GenericServlet {  
04.     public void service(ServletRequest req, ServletResponse res)  
05.         throws IOException {  
06.         PrintWriter out = res.getWriter();  
07.         out.println("HelloWorld");  
08.         out.close();  
09.     }  
10. }
```

必須要 **import javax.servlet.\*** 類別函式庫  
與 **import java.io.\*** 類別函式庫(利用  
**PrintWriter** 類別將資訊 **Response** 給  
**Client** 端)。

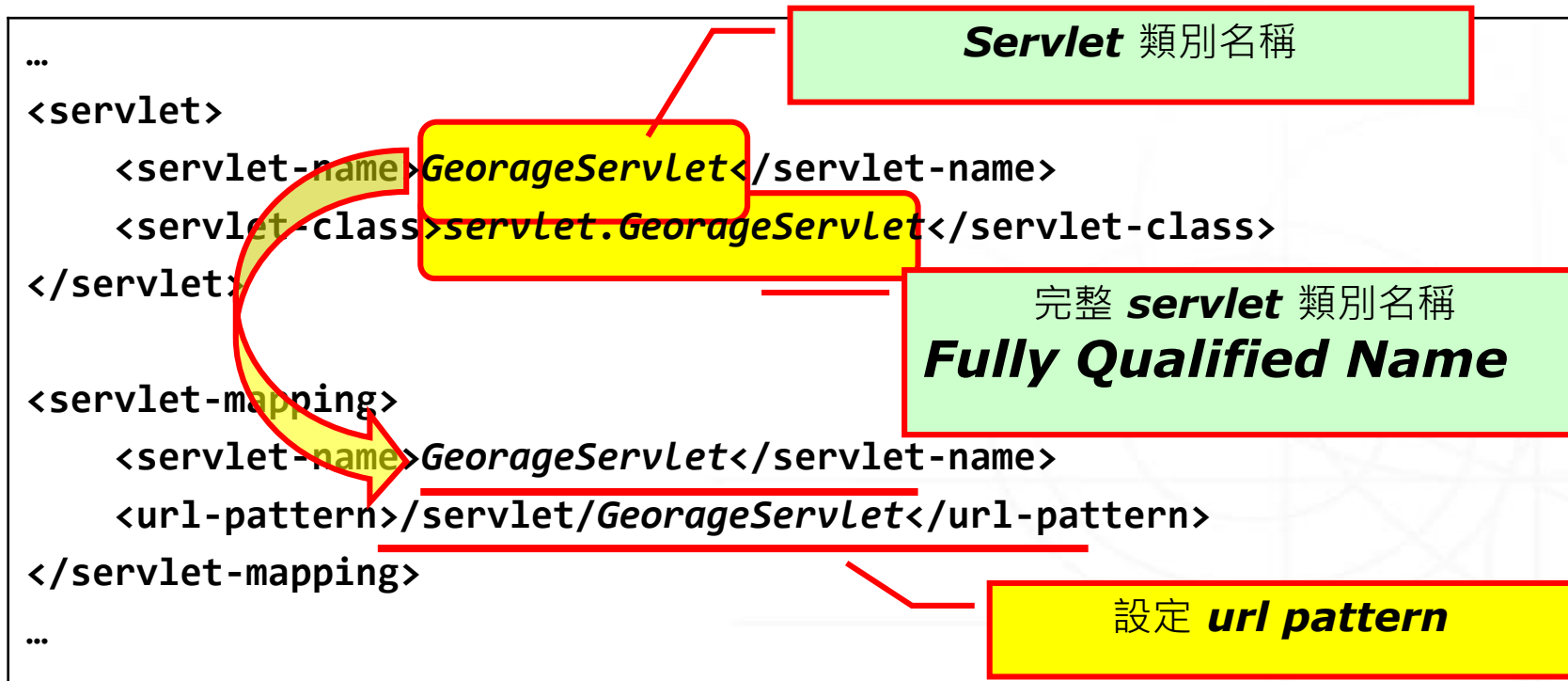
繼承 **GenericServlet** 實作 **service()** 方法

取得 **getWriter** 物件

將 " **HelloWorld** " 字串透過 **out** 物件  
回應 ( **Response** ) 給 **Client** 端

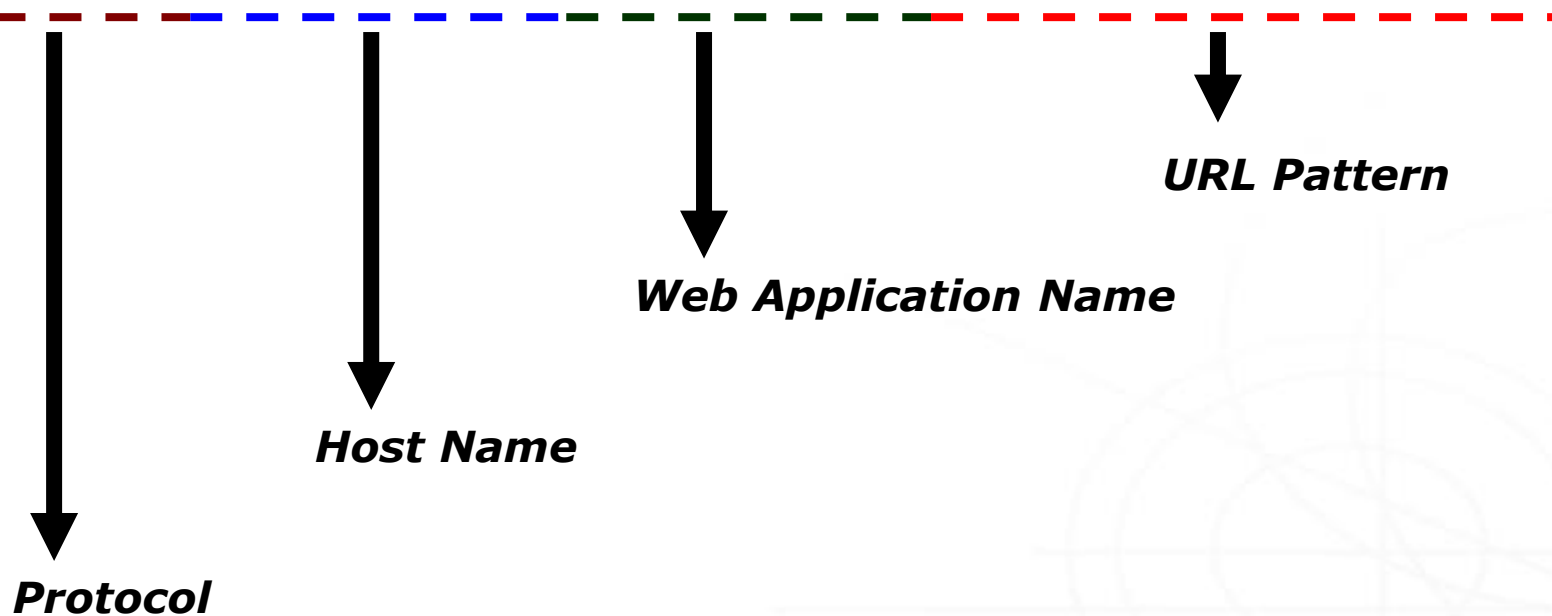
## 三、實作我的第一個 Servlet

程式部署 → DD 檔 *web.xml* :



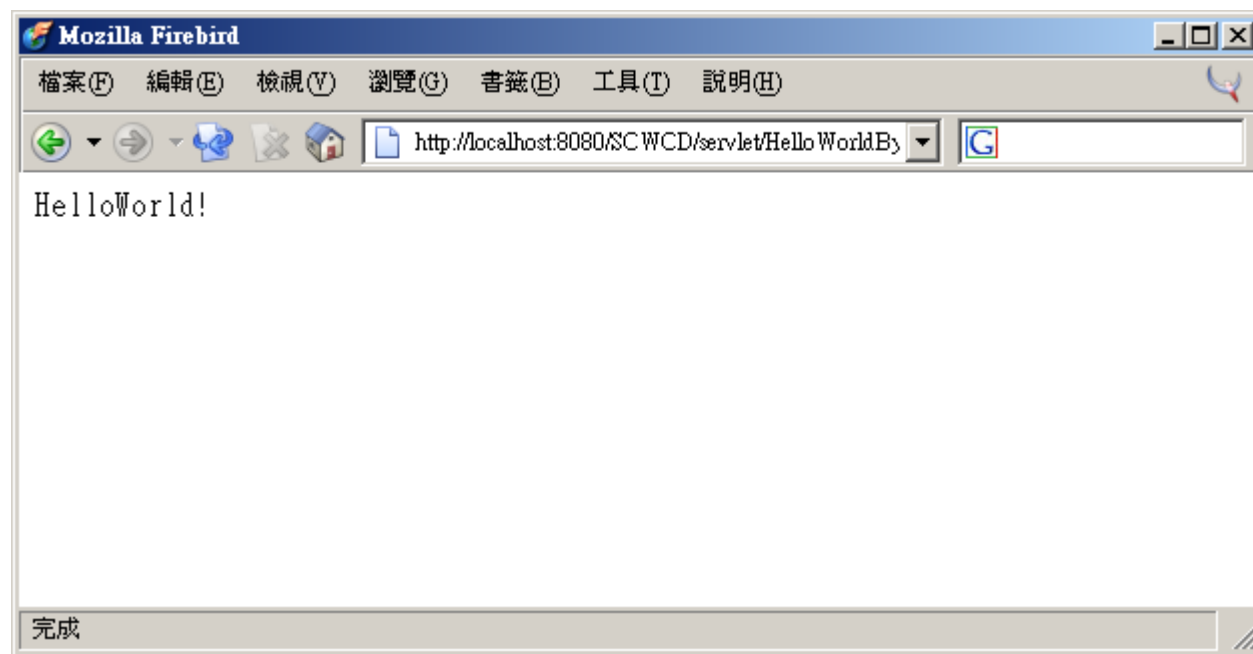
## 三、實作我的第一個 Servlet

`http://localhost:8080/JavaWebCourse/servlet/GeroageServlet`



## 三、實作我的第一個 Servlet

<http://localhost:8080/JavaWebCourse/servlet/GeroageServlet>



# 三、實作我的第一個 Servlet

我的第一個 *HttpServlet* 程式：

– *HelenServlet.java*

```
01. import javax.servlet.*;
02. import javax.servlet.http.*;
03. import java.io.*;
04. public class HelenServlet extends HttpServlet {
05.     public void doGet(HttpServletRequest req, HttpServletResponse res)
06.         throws IOException {
07.         res.setContentType("text/html; charset=UTF-8");
08.         PrintWriter out = res.getWriter();
09.         out.println("HelloWorld By Http");
10.         out.close();
11.     }
12. }
```

必須要 **import** ; 繼承 **HttpServlet class**  
類別函式庫

**doXXX()** 的參數列分別是：  
**HttpServletRequest** 與  
**HttpServletResponse** 請注意有 **Http** !



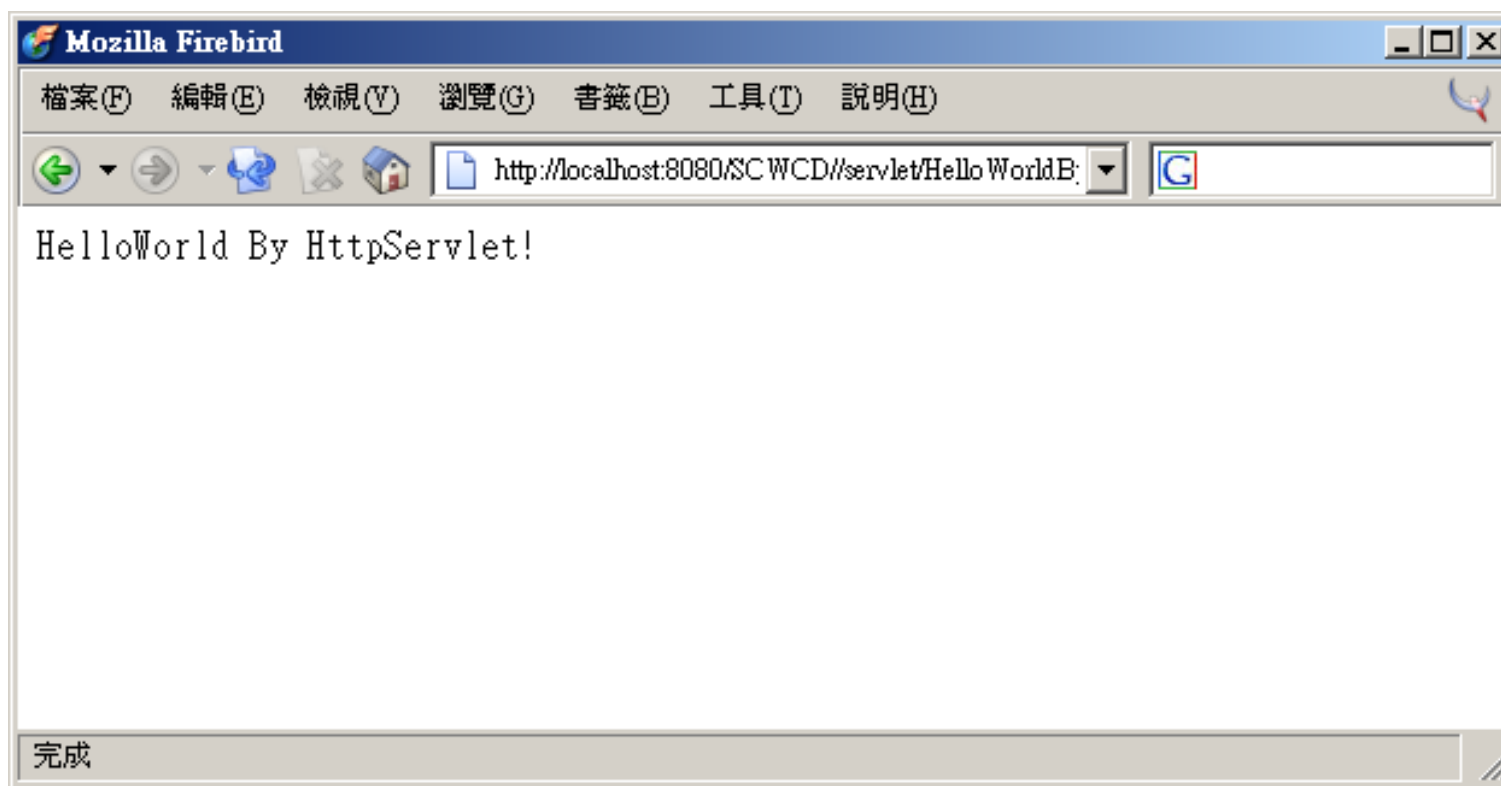
## 三、實作我的第一個 *Servlet*

程式部署 → *DD* 檔 *web.xml* :

```
...  
<servlet>  
    <servlet-name>HelenServlet</servlet-name>  
    <servlet-class>servlet.HelenServlet</servlet-class>  
</servlet>  
  
<servlet-mapping>  
    <servlet-name>HelenServlet</servlet-name>  
    <url-pattern>/servlet/HelenServlet</url-pattern>  
</servlet-mapping>  
...
```

# 三、實作我的第一個 Servlet

<http://localhost:8080/JavaWebCourse/servlet/HelenServlet>



## 三、*web.xml*

- 部署描述檔標籤 API 文件
  - [https://docs.oracle.com/cd/E13222\\_01/wls/docs81/webapp/web\\_xml.html](https://docs.oracle.com/cd/E13222_01/wls/docs81/webapp/web_xml.html)

### 三、使用註釋@配置 Servlet

- @WebServlet使用標記設定對應
- JavaEE 6 (Servlet 3.0)使支援
- 圖方便，可免去撰寫web.xml。
- `import javax.servlet.annotation.WebServlet`
- `@WebServlet(`
- `name="Servlet 名稱",`
- `urlPatterns={"URL路徑1", "URL路徑2"})`

## 三、使用註釋@配置 Servlet

servlet.HelloServlet

```
import javax.servlet.annotation.WebServlet;

@WebServlet(name="HelloServlet",urlPatterns={"/HelloServlet"})
public class HelloServlet extends HttpServlet {
    // ...
}
```

# @WebServlet 屬性列表

屬性名	類型	描述
name	String	指定Servlet 的name 屬性，等價於<servlet-name>。如果沒有顯式指定，則該Servlet 的取值即為類的全限定名。
value	String[]	該屬性等價於urlPatterns 屬性。兩個屬性不能同時使用。
urlPatterns	String[]	指定一組Servlet 的URL 匹配模式。等價於<url-pattern>標籤。
loadOnStartup	int	指定Servlet 的加載順序，等價於<load-on-startup>標籤。
initParams	WebInitParam[]	指定一組Servlet 初始化參數，等價於<init-param>標籤。
asyncSupported	boolean	聲明Servlet 是否支持異步操作模式，等價於<async-supported> 標籤。
description	String	該Servlet 的描述信息，等價於<description>標籤。
displayName	String	該Servlet 的顯示名，通常配合工具使用，等價於<display-name>標籤。

<https://tomcat.apache.org/tomcat-9.0-doc/servletapi/javax/servlet/annotation/WebServlet.html>

### 三、使用 `@WebServlet` 寫 Servlet

若web.xml與@WebServlet產生定上的衝突，則以web.xml設定為準。

```
<servlet>
  <servlet-name>HelloServlet</servlet-name>
  <servlet-class>com.HelloServlet</servlet-class>
</servlet>
<servlet-mapping>
  <servlet-name>HelloServlet</servlet-name>
  <url-pattern>/HelloServlet</url-pattern>
</servlet-mapping>
```

HelloServlet.class

```
import javax.servlet.annotation.WebServlet;

@WebServlet(name="HelloServlet",urlPatterns={"/HelloServlet"})
public class HelloServlet extends HttpServlet {
    // ...
}
```

## *metadata-complete="true"*

- 放置在web.xml文件中的描述符優先於基於@註釋的配置。
  - 甚至可以使用metadata-complete屬性完全禁用Servlet @註釋。
    - `<web-app ... version="3.1" metadata-complete="true">`
      - false (預設)
      - true 忽略@註釋與 web-fragment.xml 裡的定義  
一切以 web.xml 的設定為準



# 綱 要

- 一、送出請求 (request)
- 二、處理請求 (GenericServlet & HttpServlet)
- 三、實作我的第一個 Servlet
- 四、分析請求
- 五、送出回應
- 六、生命週期

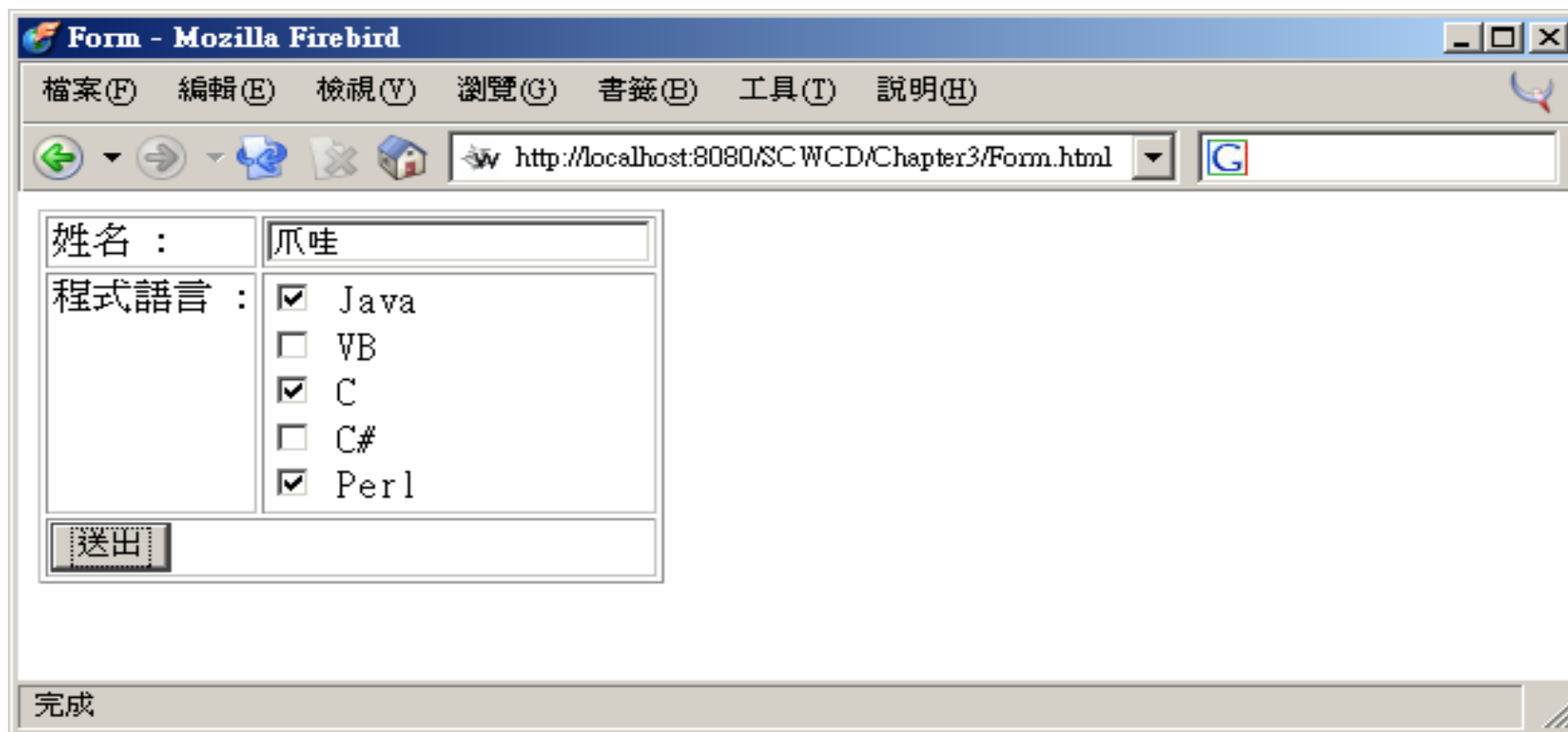
## 四、分析請求

利用 *ServletRequest* 所提供的方法進行分析：

方法名稱
<i>String</i> <i>getParameter(String name)</i> → 取得 <i>name</i> 參數的內容值。
<i>String[]</i> <i>getParameterValues(String name)</i> → 取得 <i>name</i> 參數的內容值集合。
<i>Enumeration</i> <i>getParameterNames()</i> → 取的所有 <i>client</i> 所送來的參數名稱。

## 四、分析請求

HTML Form :



The screenshot shows a Mozilla Firebird browser window titled "Form - Mozilla Firebird". The address bar displays the URL "http://localhost:8080/SC WCD/Chapter3/Form.html". The form contains the following elements:

姓名 :	<input type="text" value="爪哇"/>
程式語言 :	<input checked="" type="checkbox"/> Java <input type="checkbox"/> VB <input checked="" type="checkbox"/> C <input type="checkbox"/> C# <input checked="" type="checkbox"/> Perl
<input type="button" value="送出"/>	

The status bar at the bottom of the window displays the text "完成".

## 四、分析請求

HTML code :

```
<HTML>
<HEAD><TITLE>Form</TITLE>
<BODY>
  <Form METHOD="POST" ACTION="./servlet/QuesServlet">
    <table border="1">
      <tr><td>姓名 : </td><td><input type="text" name="myName"></td></tr>
      <tr>
        <td valign="top">程式語言 : </td>
        <td>
          <input type="checkbox" name="program" value="Java"> Java <br>
          <input type="checkbox" name="program" value="VB"> VB <br>
          <input type="checkbox" name="program" value="C"> C <br>
          <input type="checkbox" name="program" value="C#"> C# <br>
          <input type="checkbox" name="program" value="Perl"> Perl <br>
        </td>
      </tr>
      <tr> <td colspan="2"><input type="submit"></td>
    </table></Form></BODY></HTML>
```

調用 **getParameter()**

調用 **getParameterValues()**

## 四、分析請求

- 抓取 “myName” 變數內容：
  - `String myName = req.getParameter("myName");`
- 抓取 “program” 變數內容：
  - `String[] program = req.getParameterValues("program");`
    - 利用 `for-Loop` 將陣列裡面的內容依序取出。

## 四、分析請求

- 觀察提交表單Get與Post有何異同？

- `<form method="Get" ...>`
- `<form method="Post" ...>`

無論此請求中使用的特定HTTP方法如何，HttpServletRequest對像都將以相同的方式向您顯示此參數。

參數的存在並不一定意味著它實際上具有非空或有效值。因此，在使用參數值之前，首先**驗證**它們是很重要的。

HTTP協議將所有參數作為**字串傳輸**，因此您可能必須解析值並轉換數據類型。

# 綱 要

- 一、送出請求 (request)
- 二、處理請求 (GenericServlet & HttpServlet)
- 三、實作我的第一個 Servlet
- 四、分析請求
- 五、送出回應
- 六、生命週期

## 五、送出回應

- 利用 *ServletResponse* 所提供的方法來回應：
  - *setContentType()*
  - *getWriter()*
  - *getOutputStream()*
- 利用 *HttpServletResponse* 所提供的方法來回應：
  - *setHeader()*
  - *setStatus()*
  - *sendRedirect()*
  - *sendError()*



## 五、送出回應

### *ServletResponse.setContentType()*

- 設定 *response* 的內容型態與編碼方式
  - 例：*.setContentType("text/html;charset=utf-8");*
- 合法的 *content type* 包含：
  - *text/html*、*image/jpeg*、*application/jar* 等。
- *.setContentType* 預設為 *text/html*
- *.setContentType* 應宣告在 *PrintWriter* 或任何回應物件宣告之前。

## 五、送出回應

### *ServletResponse.getWriter()*

- 取得 *java.io.PrintWriter* 物件，*PrintWriter* 物件將會送出 *character* 資料到 *client* 端。
- 例：*res.getWriter();*
  - 使用時機：*Server* 端欲產生 *HTML Code* 或文字資料到 *Client* 端時。

## 五、送出回應

### `ServletResponse.getOutputStream()`

- 取得 `java.io.OutputStream` 物件，`OutputStream` 將會傳送位元組陣列到 `client` 端。
- 例：`res.getOutputStream();`
  - 使用時機：假設 `Server` 端要傳送一的檔案到 `client` 端，可以利用 `OutputStream` 來取代原先的 `PrintWriter` 物件。

## 五、送出回應

注意事項：

- 在同一個 *Servlet* 當中 *PrintWriter* 與 *OutputStream* 僅能使用一種，使用了 *PrintWriter* 就不能再使用 *OutputStream*，否則會產生 *IllegalStateException*。

## 五、送出回應

### Produce Different Content Types

Output produced by the Servlet does not have to be HTML or text.

- Set HTTP Headers to inform invoker about the nature of content your servlet is going to generate.
  - Content-Type
  - Content-Length (optional, but could be a good idea)
- Use `PrintWriter` or `ServletOutputStream` objects.

This example is artificial.

- A static file can be downloaded without the use of servlet.
- Consider generating this image dynamically.

```
...
protected void processRequest(HttpServletRequest request,
                               HttpServletResponse response)
                               throws ServletException, IOException {
    response.setContentType("image/jpeg");
    File f = new File("somePicture.jpeg");
    response.setContentLength((int)f.length());
    ServletOutputStream out = response.getOutputStream();
    Files.copy(f.toPath(), out);
    out.close();
}
...
```



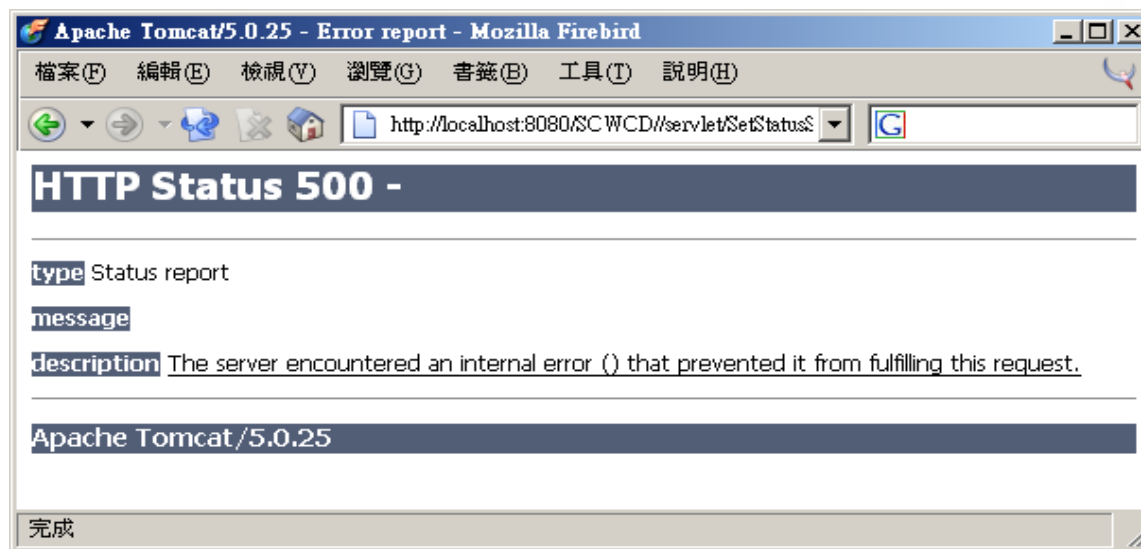
## 五、送出回應

- *HttpServletResponse* 中管理 *Header* 的方法：
  - *void setHeader(String name, String value)*
    - 設定 *header* 藉由 *name-value pair*。
  - *setIntHeader(String name, int value)* 、  
*setDateHeader(String name, Long date)* 、  
*addHeader(String name, String value)* 、  
*addIntHeader(String name, int value)* 與  
*addDateHeader(String name, Long date)* 等...
  - *Boolean containHeader(String name)*
    - 檢查所指定的 *header (name)* 是否已設定。

## 五、送出回應

`HttpServletResponse.setStatus(int sc)`

- 設定 *Status code* 給 *response* 物件。
- 例：`res.setStatus(500);`



## 五、送出回應

- *HttpServletResponse.sendRedirect()*
  - 重導到某一個網頁。
  - 例：`res.sendRedirect( "http://www.yahoo.com" );`
  - 使用上要注意：
    - 假設 *response* 已經 *committed* 完成(例如已經利用 *PrintWriter* 物件將資料送給 *client* 端)之後就不能呼叫 *sendRedirect()*，否則 *Servlet Container* 會丟出一個 *IllegalStateException* 例外。
  - *sendRedirect()* 不是由 *Server* 發動的而是由 *browser* 所發動。



## 五、送出回應

### *sendRedirect()* 錯誤的使用：

```
01. PrintWriter out;  
02. String s = req.getParameter("s");  
03. out = res.getWriter();  
04. out.println(s);  
05. out.flush();  
06. res.sendRedirect("http://www.yahoo.com");
```

**committed** 完成之後又再呼叫 **sendRedirect()**，**Servlet Container** 將會丟出 **IllegalStateException** 例外物件。

## 五、送出回應

*sendRedirect()* 正確的使用：

```
01. PrintWriter out;  
02. String s = req.getParameter("s");  
03. if (s.equals("java")) {  
04.     out = res.getWriter();  
05.     out.println(s);  
06.     out.flush();  
07. }  
08. else {  
09.     res.sendRedirect("http://www.yahoo.com");  
10. }
```

## 五、送出回應

### *HttpServletResponse.sendError()*

- 傳送所指定的狀態碼到 *Client* 端，瀏覽器會顯示適當的資訊給 *end user*。
- 例：
  - *res.sendError(res.SC\_UNAUTHORIZED)*  
*res.sendError(res.SC\_UNAUTHORIZED, "This is UNAUTHORIZED");*
- 若要自定 *error* 訊息給 *client* 端請使用 *sendError()* 而不建議使用 *setStatus()*。



## Quiz

Quiz

Why must a servlet define its URL?

- a. To provide the container with a way to access the class
- b. To provide the browser with the appropriate header information
- c. To provide the browser with a way to access the servlet
- d. To store session state



## Quiz



Q

oo.com.tw

In which directory do you place the optional deployment descriptor (`web.xml`)?

- a. `Htdocs`
- b. `WEB-INF`
- c. `WEB-INIT`
- d. `htdocs/WEB-INIT`

## Quiz



Given a situation where two parameters are passed in the request to a servlet, a string name and an integer ID, which method pair correctly reads these parameters?

- a. `String name = response.getParameter("name");`  
`int id = response.getParameter("id");`
- b. `String name = request.getParameter("name");`  
`int id = request.getParameter("id");`
- c. `String name = request.getParameter("name");`  
`int id = request.getIntParameter("id");`
- d. `String name = request.getParameter("name");`  
`int id =`  
`new Integer(request.getParameter("id")).intValue();`

# 分析 Header

## 分析 Header 請求：

方法名稱：

***String getHeader(String headerName)***

取得 *headerName* 變數名稱的 *head* 內容值。

***Enumeration getHeaders(String headerName)***

取得 *headerName* 的集合資料

***Enumeration getHeaderNames()***

取得所有 *Header* 名稱

***int getIntHeader(String headerName)***

取得 *headerName* 的整數值

# 分析 Header

- 使用 Header 的情境：
  - 在Servlet代碼中處理HTTP協議標頭非常重要，因為它們包含有關調用客戶端的有價值信息。您可以使用標頭查找客戶端使用的瀏覽器，操作系統，國家/地區或語言等。



# 分析 Header

- Servlet 客戶端 HTTP 請求
  - <https://www.runoob.com/servlet/servlet-client-request.html>
- Servlet 客戶端 HTTP 回應
  - <https://www.runoob.com/servlet/servlet-server-response.html>

# 分析 *Header*

## *Header* 範例：

```
Enumeration headers = req.getHeaderNames();  
while(headers.hasMoreElements()) {  
    String header = (String)headers.nextElement();  
    String value = req.getHeader(header);  
}
```

取得所有 **Header** 名稱

取得當下的  
**Header** 元素

判斷是否還有  
**Header** 元素

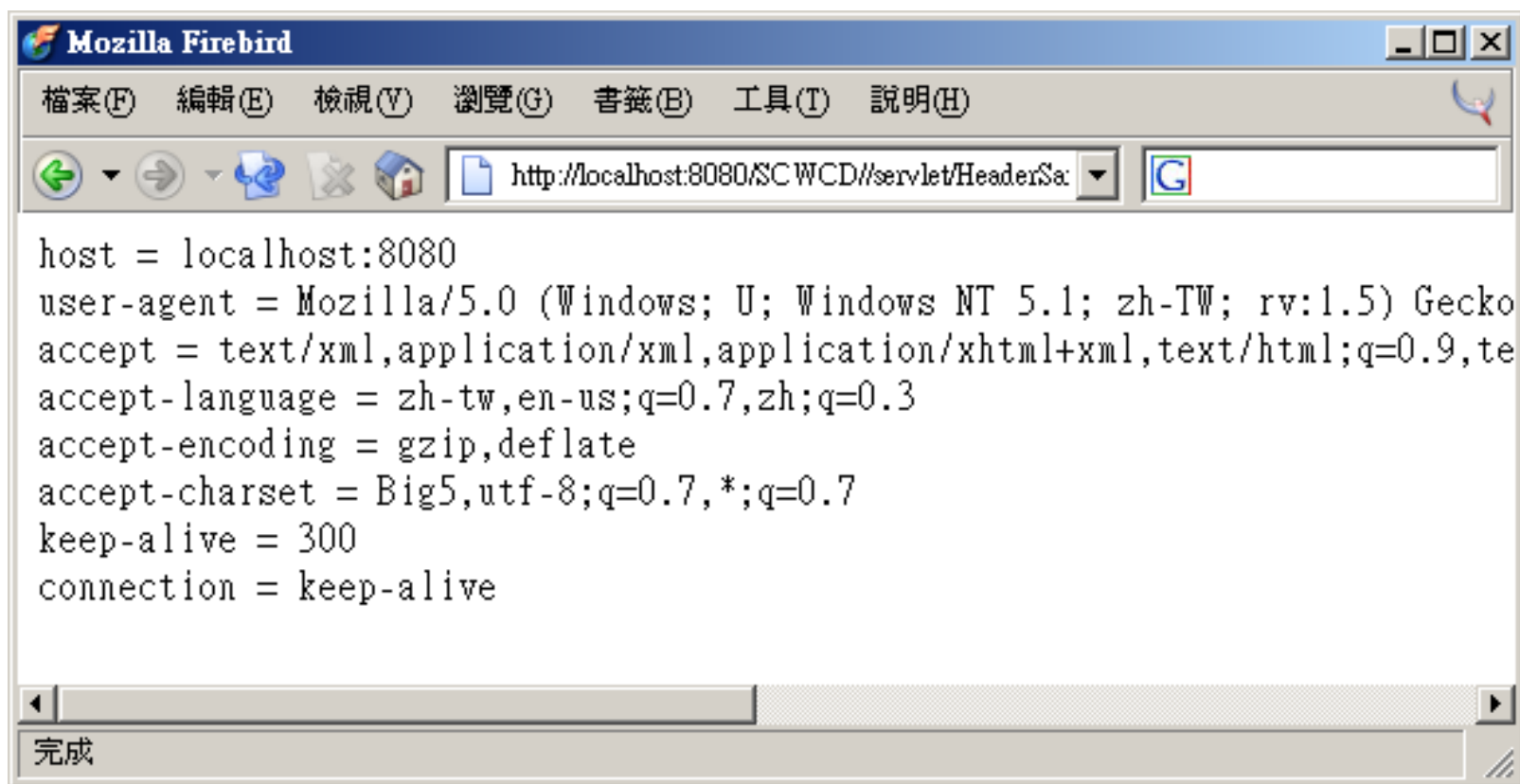
取得 **Header** 的內容值

# 分析 *Header*

- 部署 `HeaderServlet`
  - `web.xml` 或 `@註釋`

# 分析 Header

<http://localhost:8080/WebAppSource/servlet/HeaderServlet>



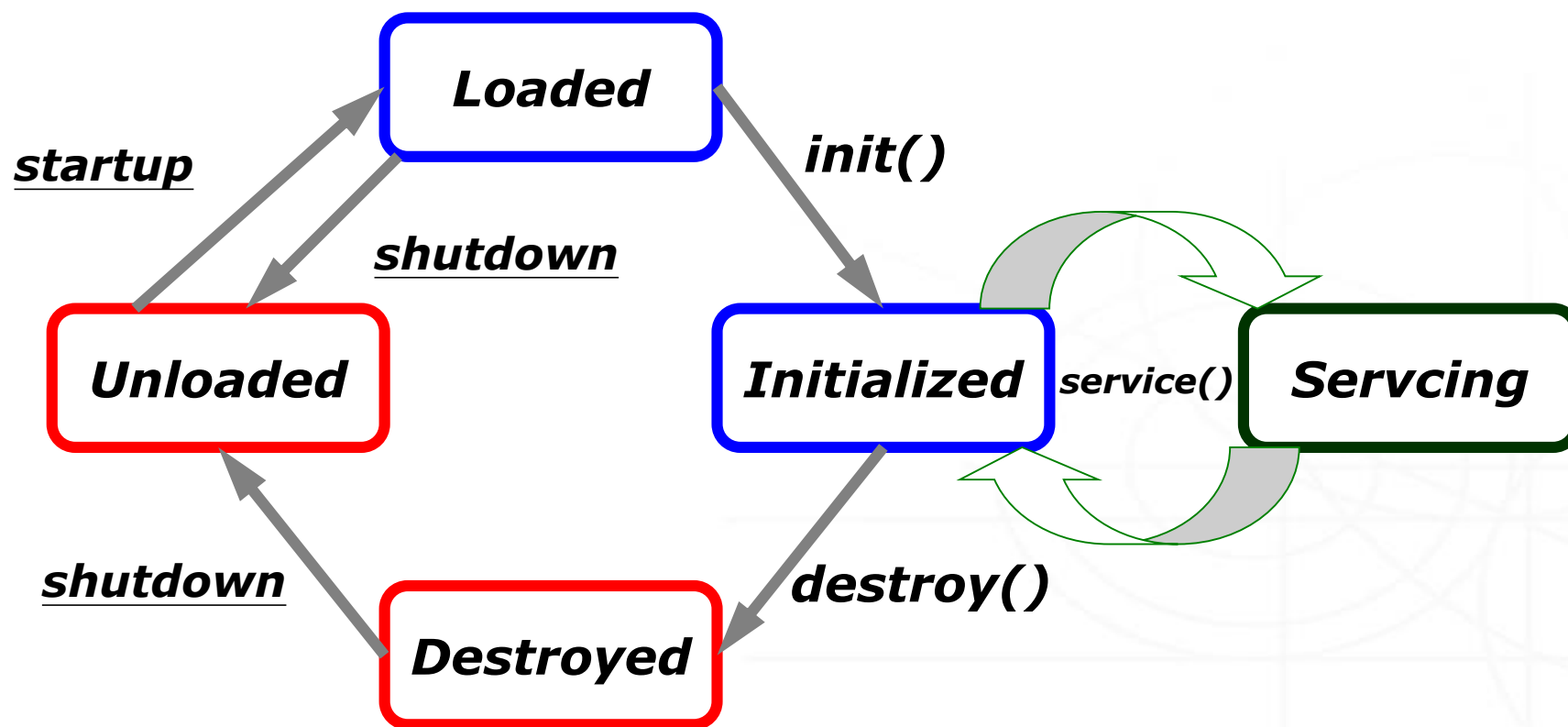
# 綱 要

- 一、送出請求 (request)
- 二、處理請求 (GenericServlet & HttpServlet)
- 三、實作我的第一個 Servlet
- 四、分析請求
- 五、送出回應
- 六、生命週期

# ***Servlet 生命週期***

- *Servlet Life cycle*
  - 載入：Load
  - 初始：Initializing
  - 執行與銷毀：Servcing & Destroy
  - 卸載：Unload

# Servlet 生命週期



# Servlet 生命週期

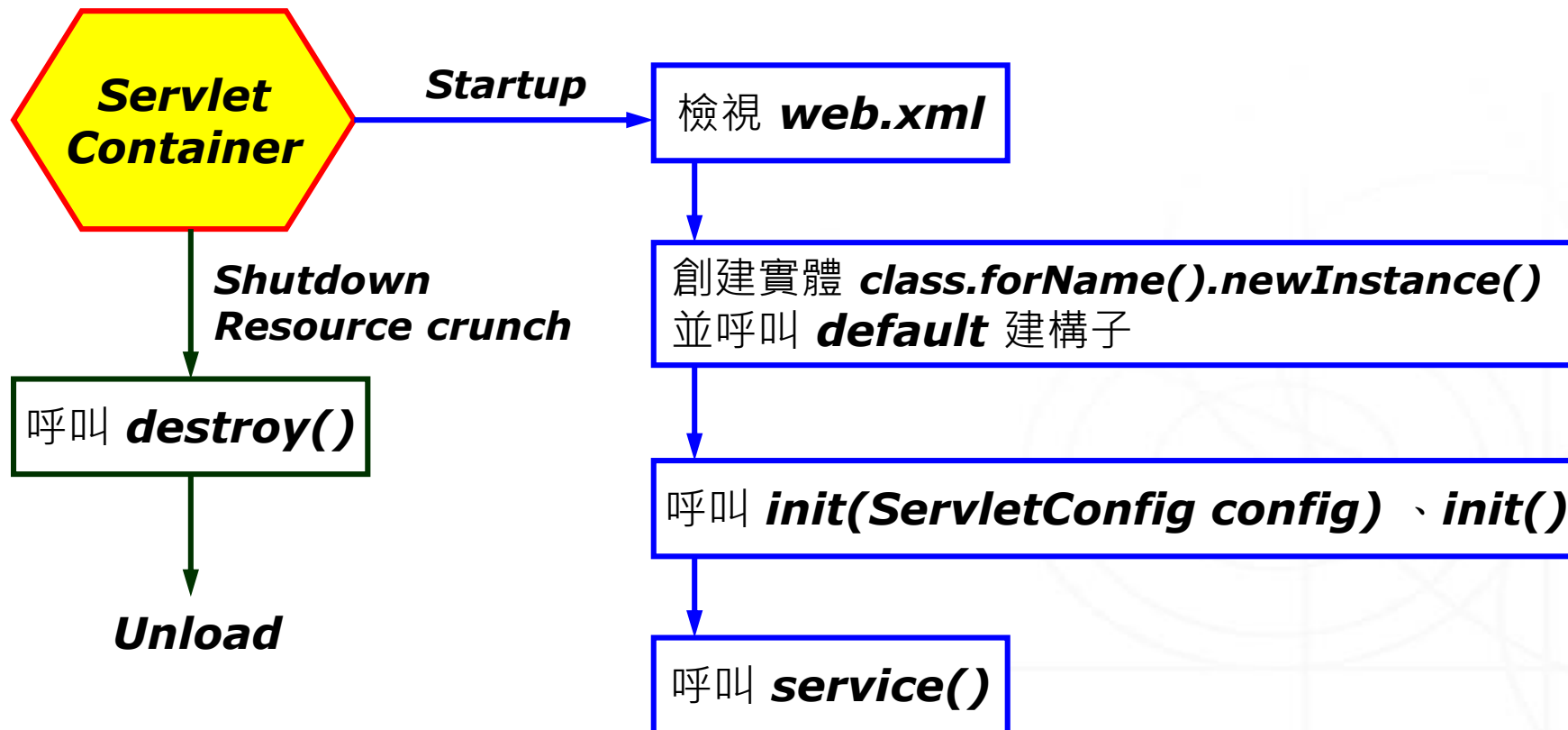
- 與 *servlet life cycle* 相關的方法：

方法名稱	簡要說明
<b><i>void init(ServletConfig config)</i></b>	<b><i>ServletContainer</i></b> 會呼叫此方法來進行 <b><i>servlet</i></b> 的初始化
<b><i>void init()</i></b>	<b><i>Developer</i></b> 可覆寫此方法自行撰寫 <b><i>servlet</i></b> 初始化時的相關實作
<b><i>void service(ServletRequest req, ServletResponse res)</i></b>	服務 <b><i>client</i></b> 端來的要求 ( <b><i>request</i></b> )
<b><i>void destroy()</i></b>	當 <b><i>servlet container</i></b> 決定要卸載( <b><i>unload</i></b> ) 該 <b><i>servlet</i></b> 前會呼叫並實作此方法



# Servlet 生命週期

- The servlet container's perspective



# 載入 : *Load*

- 當 *servlet container* 啟動時，會依照 *web.xml* 或 *@Annotation* 中的定義載入 *servlet* 實體。
  - *Servlet container* 會呼叫 *Class.forName(className).newInstance()*，以建立 *servlet* 的物件實體。
  - 接著再呼叫該 *servlet* 的預設建構子 (*public constructor with no arguments*)。
  - 這整個過程就稱為 “*servlet is loaded*”。

# 初始 : *Initializing*

- *Initializing*

- *init(ServletConfig config) v.s. init()*

- 實作上會以覆寫 *init()* 為主，若一定要覆寫 *init(ServletConfig config)* 則必須記得要呼叫 *super.init(ServletConfig config)* 為的是將現行的組態傳給 *GenericServlet*。

# 初始 : *Initializing*

- Servlet API 2.1 之後  
*init(ServletConfig config)* 與 *init()* 之關係 :

```
ServletConfig _config = null;  
void init(ServletConfig config) {  
    _config = config // servlet 組態  
    init();  
}
```

```
void init(){  
    // Block of Code  
}
```

# 初始 : *Initializing*

- *Servlet* 物件會產生幾個？
  - 在一般的情況下 *ServletContainer* 只會在初始時呼叫 *init(ServletConfig config)* 一次，因而僅會有一個物件產生，另外 *init()* 也因此僅會呼叫一次（註：呼叫 *init(ServletConfig config)* 就是為了要定義組態）
    - 例外狀況（產生多個相同的 *Servlet* 物件）：
      - *web.xml* 中定義多個相同的 *<servlet>* 元素
      - *Servlet* 實作 *SingleThreadModel* 介面  
*implements SingleThreadModel (STM)*

# 初始 : *Initializing*

- *Preinitializing*

- *Servlet container* 於啟動的過程中並不一定會把所有 *servlet* 初始化，而會把這份初始化的工作丟給最先收到的請求(*request*)來進行，這種行為稱之為 *Lazy Loading*。

- 解決方法：於 *web.xml* 中的 `<servlet>` 定義中加上 `<load-on-startup>` 元素。

- 例：`<load-on-startup>1</load-on-startup>`  
數值越小越先載入  
0 或 “負數” 表示任何時間點皆可載入。

# 執行與銷毀 : *Servcing & Destroy*

- *Serving* :

- *Servlet container* 接收到 *request* 時將會分派他們到指定的 *servlet instance* 並且直接呼叫 *service(ServletRequest req, ServletResponse res)*

# 執行與銷毀 : *Servcing & Destroy*

- *Destroy* :

- 當 *servlet container* 認為某一個 *servlet* 已經很久沒有被使用了就會自行呼叫該 *servlet* 的 *destroy()* 方法，將資源歸還給系統。

- 另外當 *servlet container* 正在卸載(*shutting down*)時也會呼叫 *servlet* 的 *destroy()* 方法。



# 執行與銷毀 : *Servcing & Destroy*

- 覆寫 `init()` 與 `destroy()` 時機
  - 一般而言我們會將 `servlet` 在 `init()` 所宣告使用的資源，並於 `destroy()` 中釋放
    - 例如：I/O、Socket、資料庫連線物件等。

# 卸載 : *Unload*

- *UnLoad* :

- 當 *servlet* 的 *destroy()* 被呼叫時代表 *servlet* 的生命週期結束並且系統會針對該 *servlet* 物件進行 *garbage collection* 的動作，整個過程完成之後我們可以說該 *servlet* 已經 *unLoaded*。
- 所以不論是該 *servlet* 因閒置過久而遭 *servlet container* 處分或是因 *servlet container* 自身進行 *shut down*，*servlet* 都將會被 *unload*。

