
Java EE 7 Web

網路應用程式裡使用Filter

鄭安翔

ansel_cheng@hotmail.com

課程大綱

- 1) **Intercepting Filter pattern**
 - 網路元件容器 **Request** 的生命週期
- 2) **Filter API**
- 3) **開發 Filter 類別**

Intercepting Filter pattern 攔截篩選

■ Problem

- ❑ 客戶端之請求在處理前，有重複的前置處理
 - ❑ 伺服器端產生的回應，有重複的後置處理
 - ❑ 這些前置及後置處理，需有彈性、可調校 (Configurable) 且可重複使用
 - ❑ 常見之前後置處理
 - 身分驗證
 - 存取權限控管及稽核記錄
 - 效能監控評量
-
- 回應資料處理(壓縮,加密)

Intercepting Filter pattern 攔截篩選

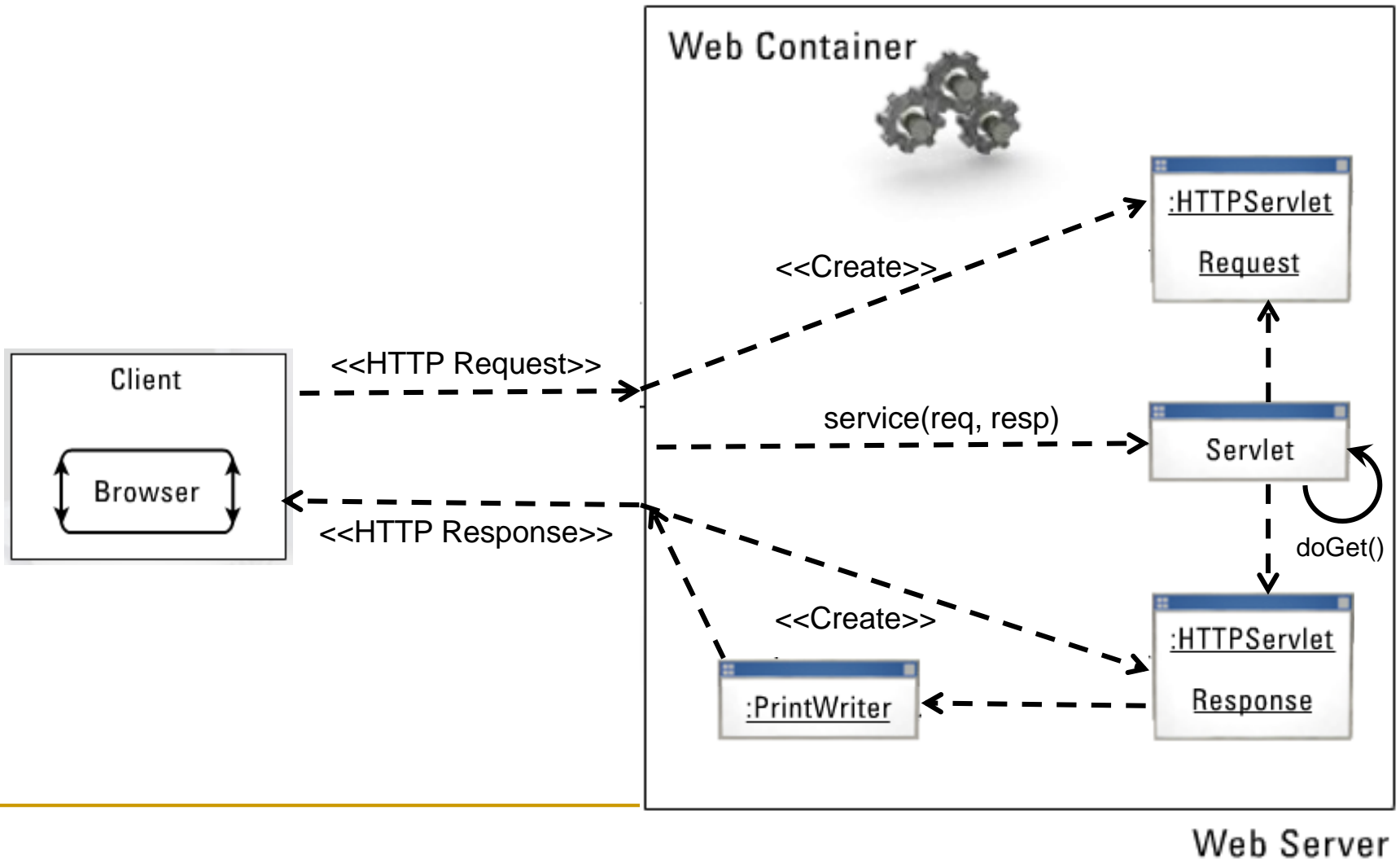
■ Solution:

- 建立可抽換的篩選器(**Filter**)來攔截客戶端的請求或伺服端的回應
- 使用描述檔來設定篩選器的啟動、關閉及執行順序

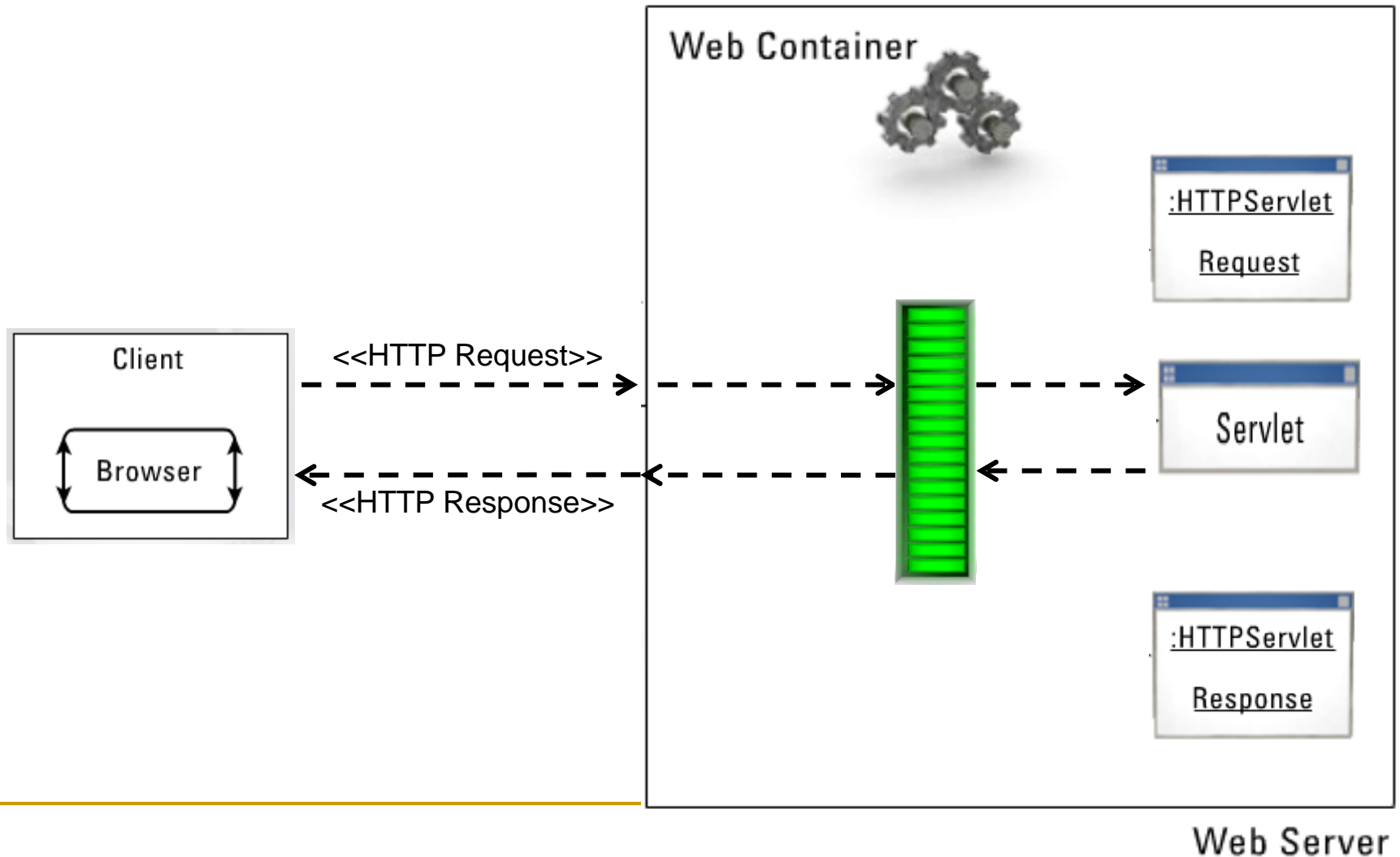
網路元件容器攔截篩選

- 網路元件容器攔截篩選模式
 - 使用者請求**Request**到達**Servlet**程式碼之前,由網路元件容器攔截,作前處理
 - 產生對應之**HttpServletRequest**物件
 - 提供其它額外的功能(ex:安全性)。
 - 元件容器在**servlet** 產生回應後,執行後處理
 - 使用者也可以使用 **filter** 的元件,自訂前/後處理
 - 元件容器前處理完成後,執行自訂**Filter**前處理
 - 元件容器執行自訂**Filter**後處理後,執行容器後處理

網路元件容器架構前 / 後處理



Filter機制 – 自訂前 / 後處理元件

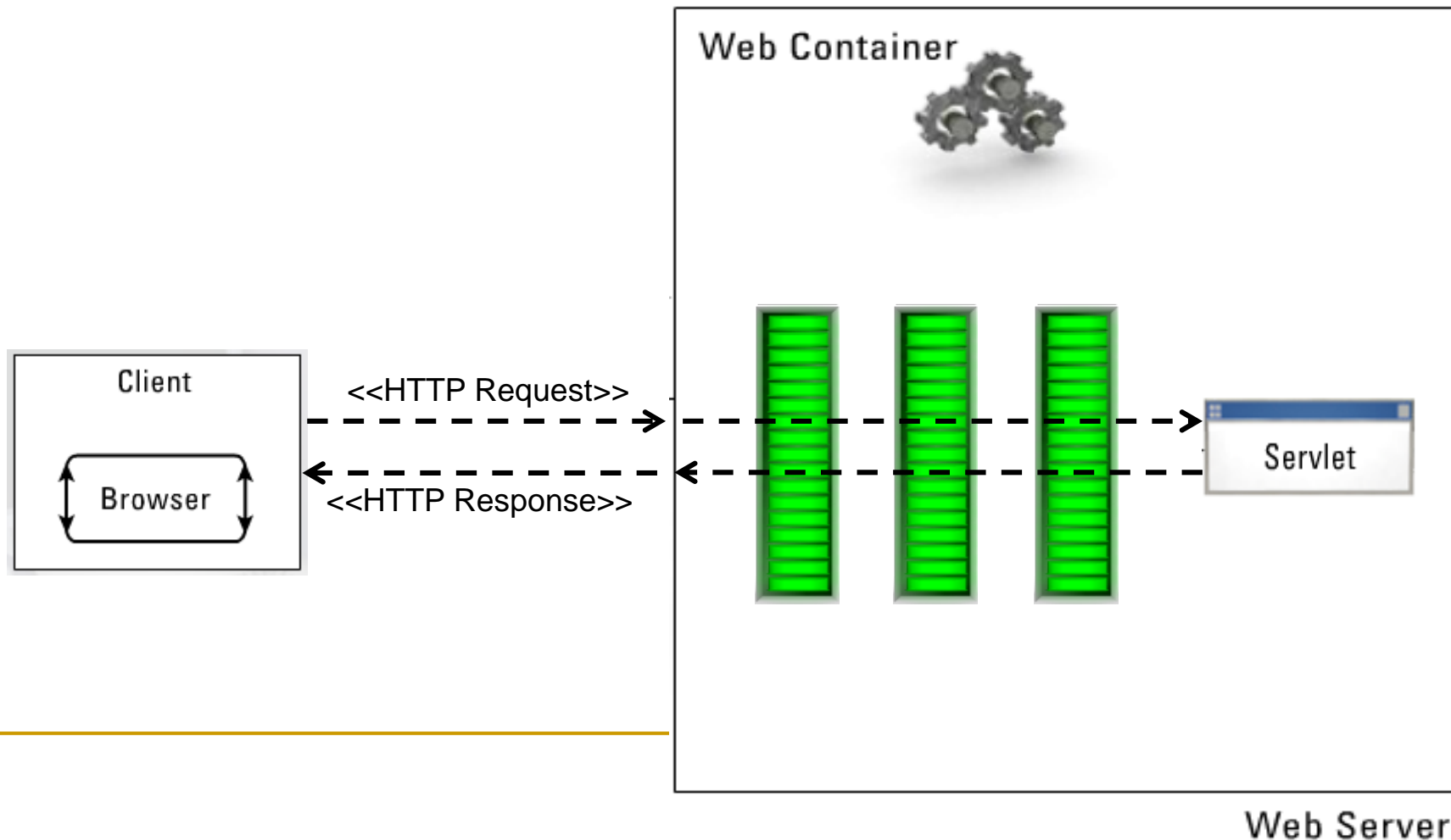


Filter 執行流程

1. 網路元件容器執行前處理工作
 - 元件容器廠商依 **Servlet / JSP** 規範實作
2. 檢查符合請求**URL** 樣式的**filter**
 - 應用程式開發者自訂**Filter**元件
3. 執行**Filter** 前處理程式碼
 - 如有多個符合**URL** 樣式的**filter**, 依部署描述中順序執行
4. **Filter**致行沒有錯誤, 呼叫目的**servlet / JSP**
5. 執行**servlet** 將**response** 傳回
6. 執行**Filter** 後處理程式碼
 - 依先前執行順序反向執行
7. 網路元件容器執行後處理

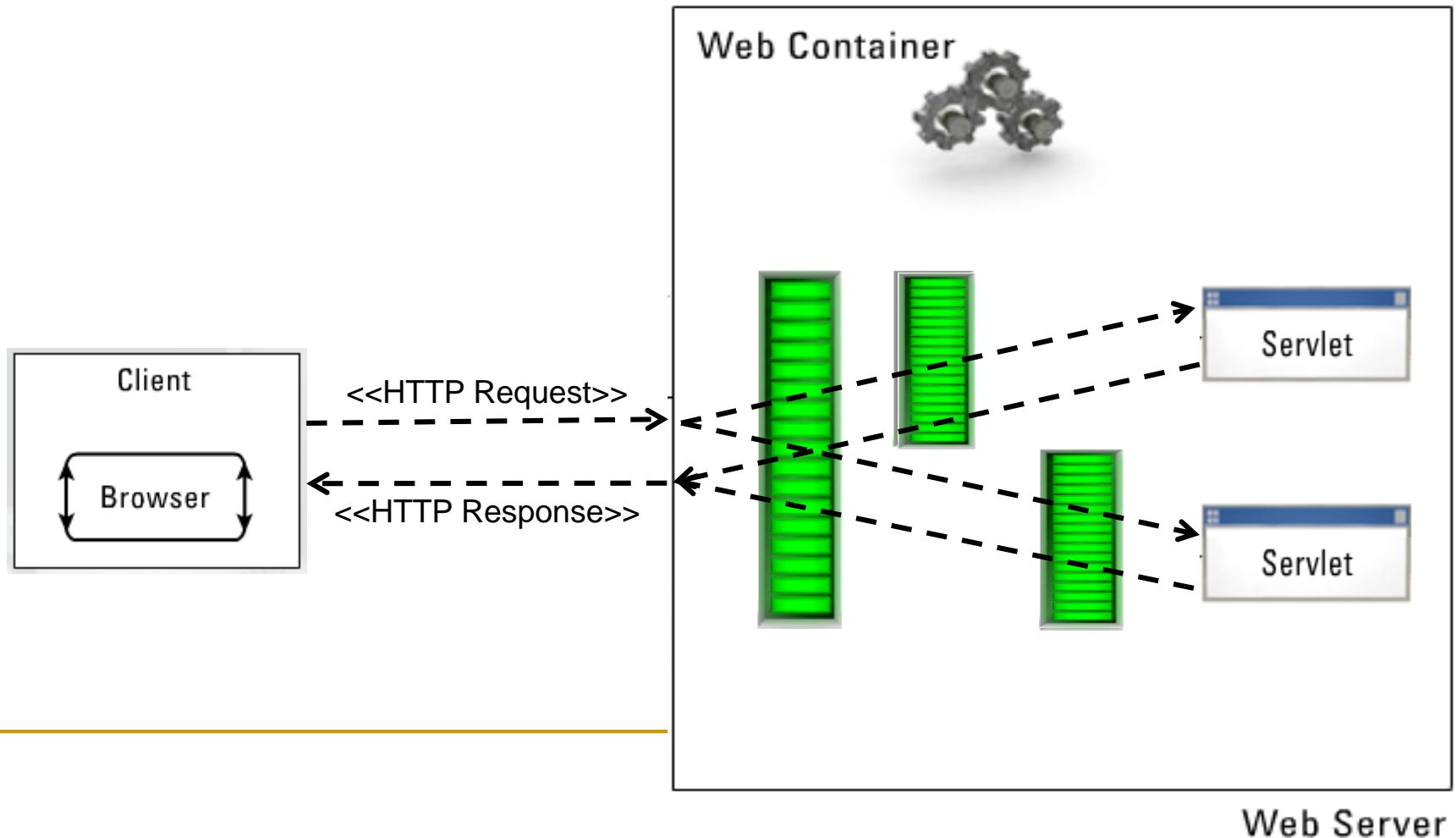
多重 Filter 鏈結

- Filter 為可重複使用模組
 - 將數個filter 鏈結起來，提供模組化的前處理和後處理



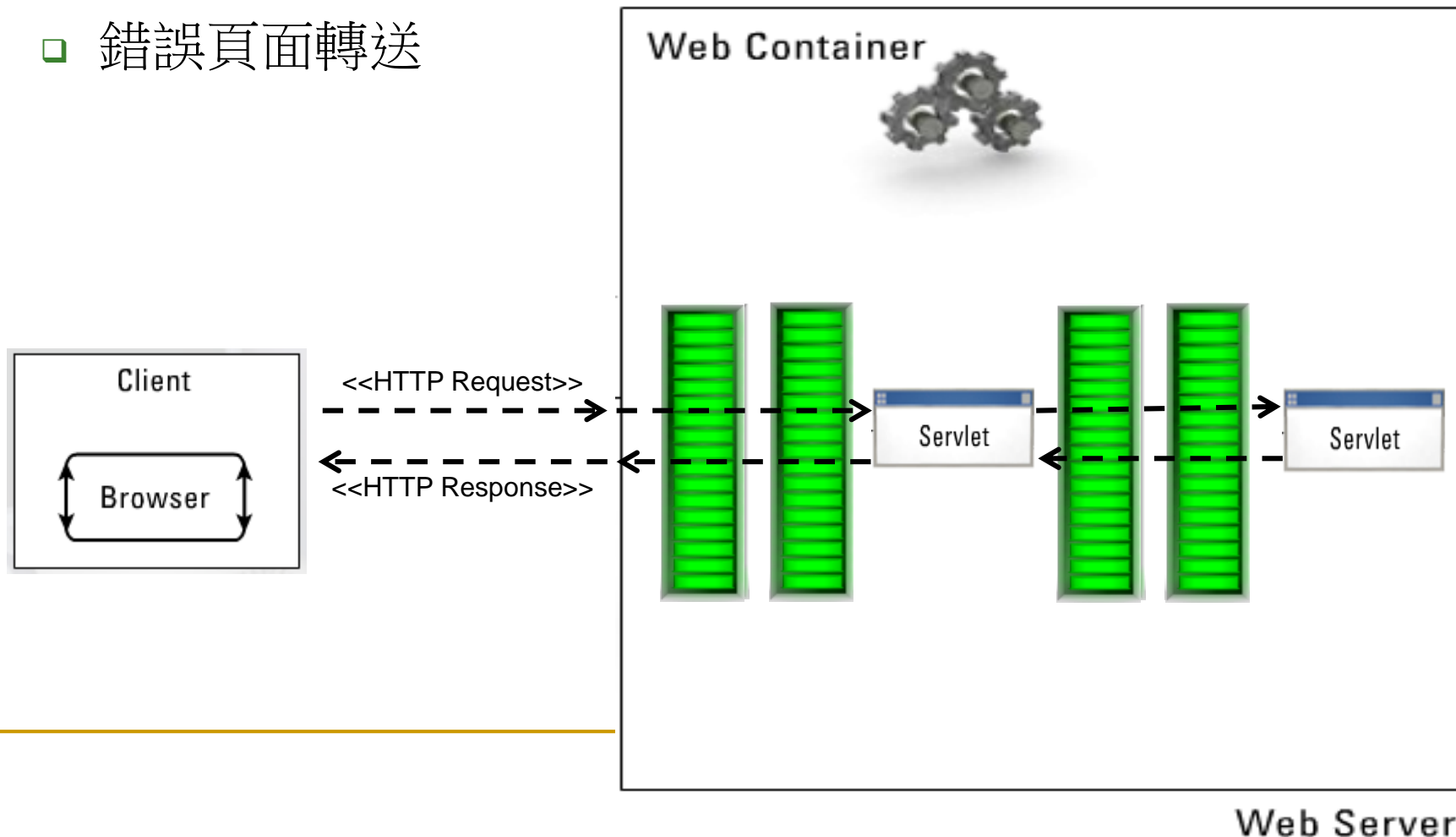
Modular Filter 模組化篩選器

- Filter 為可重複使用模組
 - 不同 request 可套用不同 filter



Filter 套用派送機制

- **Filter**也可以套用網路元件容器的派送過程
 - 用於引入(include)或轉送(forward)等派送機制
 - 錯誤頁面轉送



Filter 用途

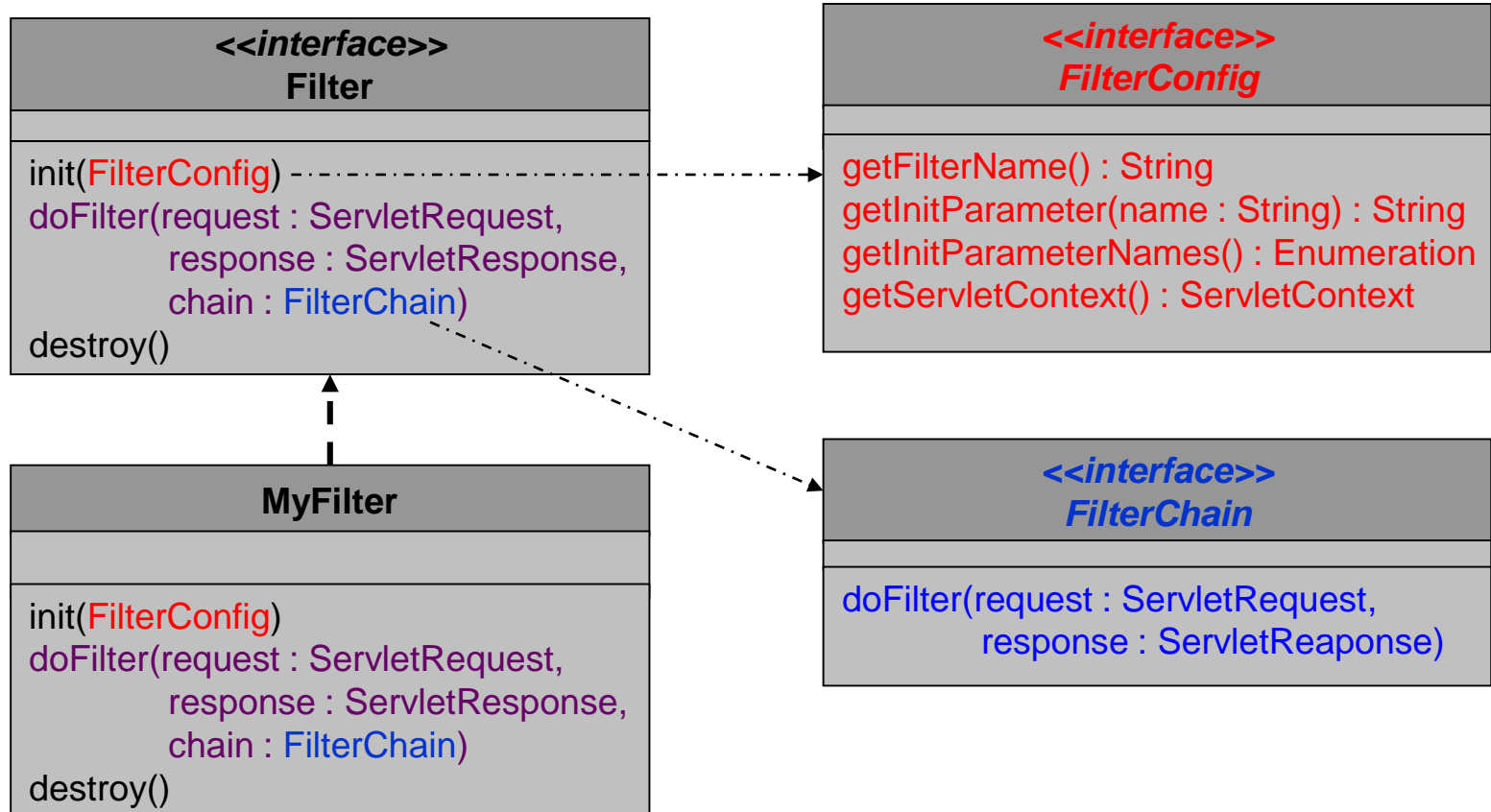
■ Filter 常用用途：

- 保護需要通過使用者認證或是會員認證的資源
- 驗證/稽核請求
- 壓縮回應資料串流
- 改變回應
- 測量並記錄servlet 效能

課程大綱

- 1) Intercepting Filter pattern
 - 2) **Filter API**
 - 3) 開發 **Filter** 類別
-

Filter API – javax.servlet 套件



Filter 方法	傳回值	說明
init(FilterConfig config)	void	當Filter初始啟動時,網路元件容器會呼叫此方法一次
doFilter(ServletRequest request, ServletResponse response, FilterChain chain)	void	當每一次的 Request/Response 進行過程中,當所請求的URL與Filter對應(Mapping)符合時,網路元件容器會呼叫此方法
destroy()	void	當Filter結束時,網路元件容器會呼叫此方法

FilterConfig 方法	傳回值	說明
getFilterName()	String	取得部署描述檔(web.xml)中定義的過濾器(Filter)名稱。
getInitParameter(String name)	String	取得指定名稱之初始化參數值,當該參數不存在時傳回null
getInitParameterNames()	Enumeration	傳回包含該Filter定義之所有初始化參數名稱的列舉物件
getServletContext()	ServletContext	取得ServletContext之參考

FilterChain 方法	傳回值	說明
doFilter(ServletRequest request, ServletResponse response)	void	觸發FilterChain(鏈結)中的下個元件,若下一個鏈結是Filter物件,呼叫其doFilter()方法,若下一個鏈結是Servlet物件,呼叫該Servlet的service()

Annotation 設定 Filter

```
@WebFilter(filterName="perfFilter", urlPatterns={"/*"},
    dispatcherTypes={ DispatcherType.FORWARD,
        DispatcherType.ERROR,
        DispatcherType.REQUEST,
        DispatcherType.INCLUDE},
    initParams={ @WebInitParam( name="Log Entry Prefix",
        value="Performance:")
    }
)
```


Deployment Descriptor – Filters

```
<web-app>
  <display-name> Web application 名稱. </display-name>
  <description> Web application 說明. </description>
  <filter>
    <filter-name> Filter 名稱 </filter-name>
    <filter-class> Filter 類別類別全名 (Fully Qualified Name). </filter-class>
    <init-param>
      <param-name> Filter 初始參數名稱 </param-name>
      <param-value> Filter 初始參數值 </param-value>
    </init-param>
  </filter>
  ....
  <filter-mapping>
    <filter-name> Filter 名稱 </filter-name>
    <url-pattern> Filter URL 對應 </url-pattern>
    <servlet-name> Filter Servlet 對應 </servlet-name>
    <dispatcher> REQUEST / FORWARD / INCLUDE / ERROR </dispatcher>
  </filter-mapping>
  <filter-mapping>..... </filter-mapping>
  <servlet>.....</servlet>
</web-app>
```

兩個標籤擇一使用
URL對應優先於
Servlet Name對應

Filter觸發範圍：客戶端直接傳來的Request/
其他Servlet Forward/
其他Servlet Include/
錯誤頁面

FilterConfig物件

FilterChain物件

Filter Chaining Example

/admin/AddLeague

執行順序：

perfFilter

auditFilter

transformFilter

```
<web-app>
  <servlet-mapping>
    <servlet-name>AddLeague</servlet-name>
    </url-pattern>/admin/AddLeague</url-pattern>
  </servlet-mapping>
  ....
  <filter-mapping>
    <filter-name>perfFilter</filter-name>
    <url-pattern>/*</url-pattern>
  </filter-mapping>
  <filter-mapping>
    <filter-name>transformFilter</filter-name>
    <servlet-name>AddLeague</servlet-name>
  </filter-mapping>
  <filter-mapping>
    <filter-name>auditFilter</filter-name>
    <url-pattern>/admin/*</url-pattern>
  </filter-mapping>
</web-app>
```

Filter Dispatch Options

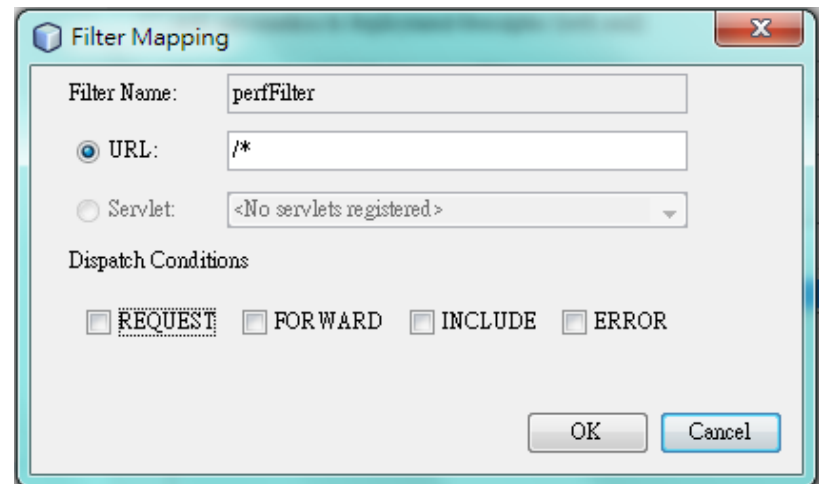
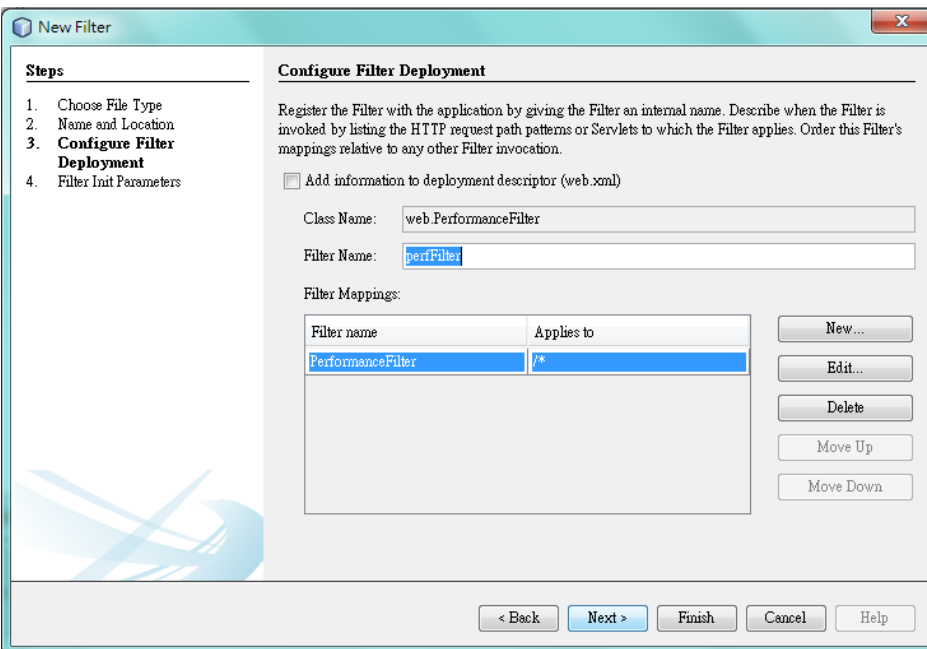
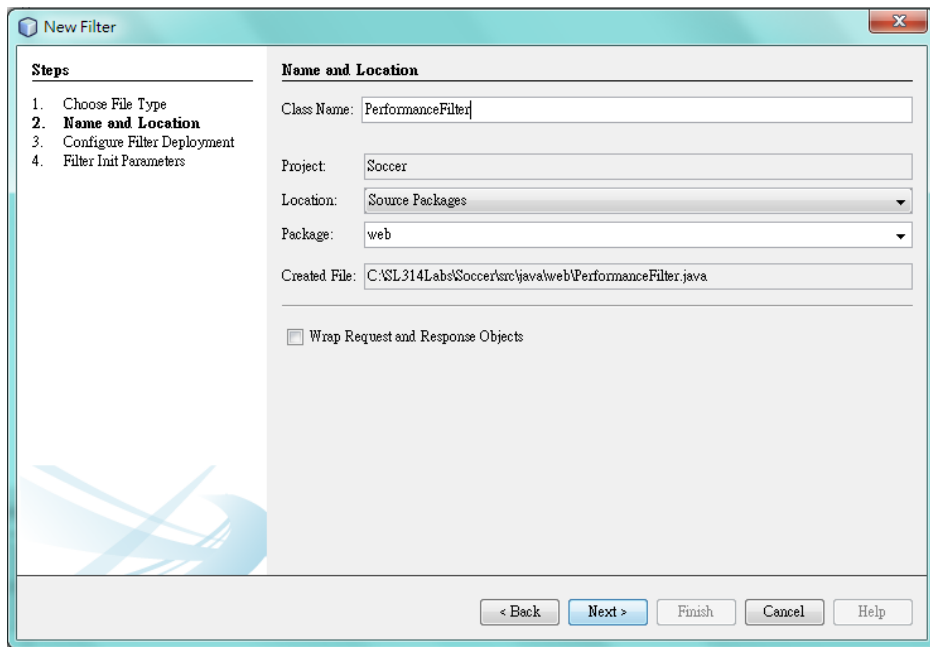
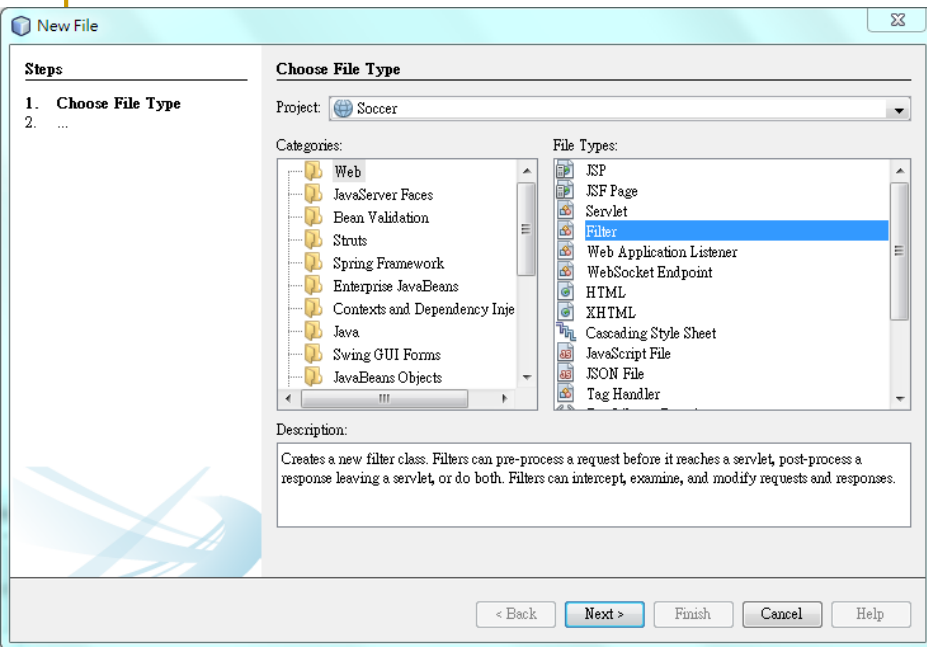
- **Filter** 一般只作用在用戶端直接來的請求
- **<dispatcher>** 可設定 **filter** 套用於內部的派送
 - 頁面引入(include)
 - 頁面轉送(forward)
 - 錯誤頁面

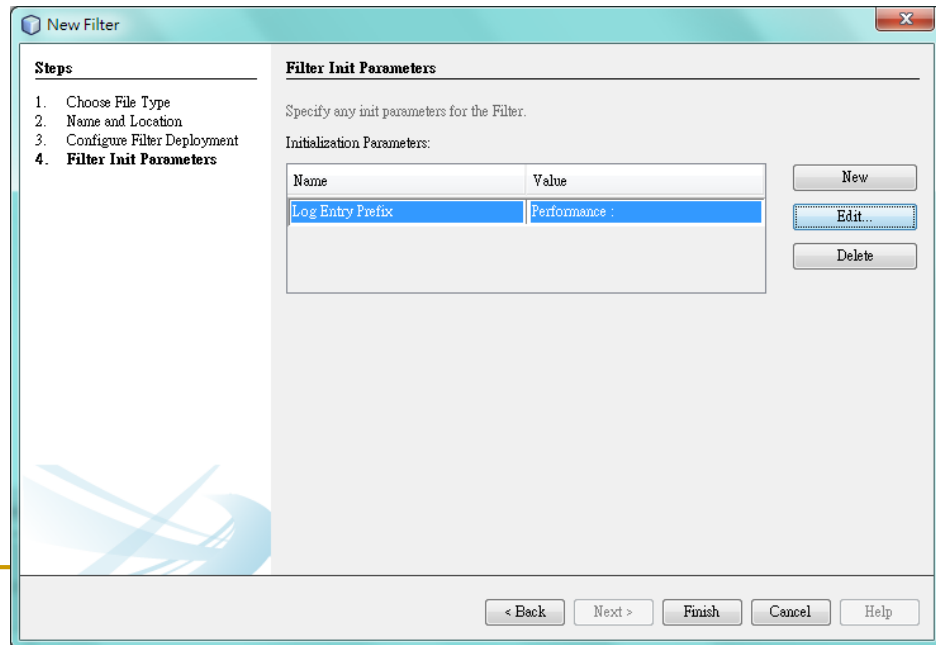
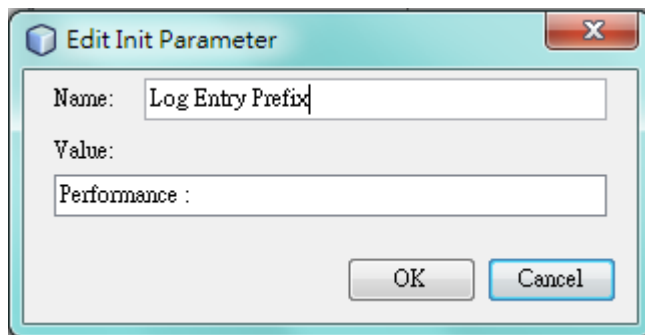
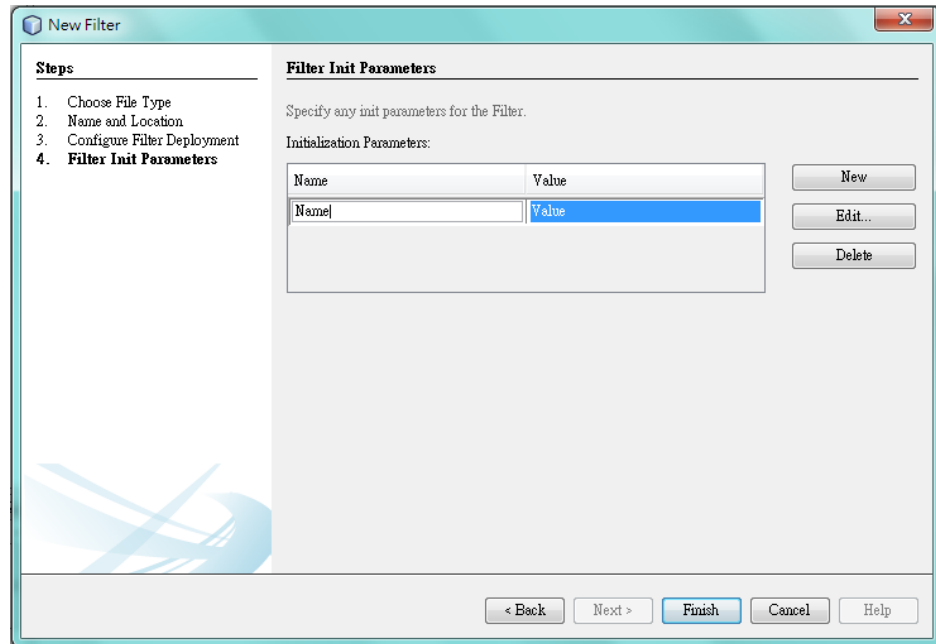
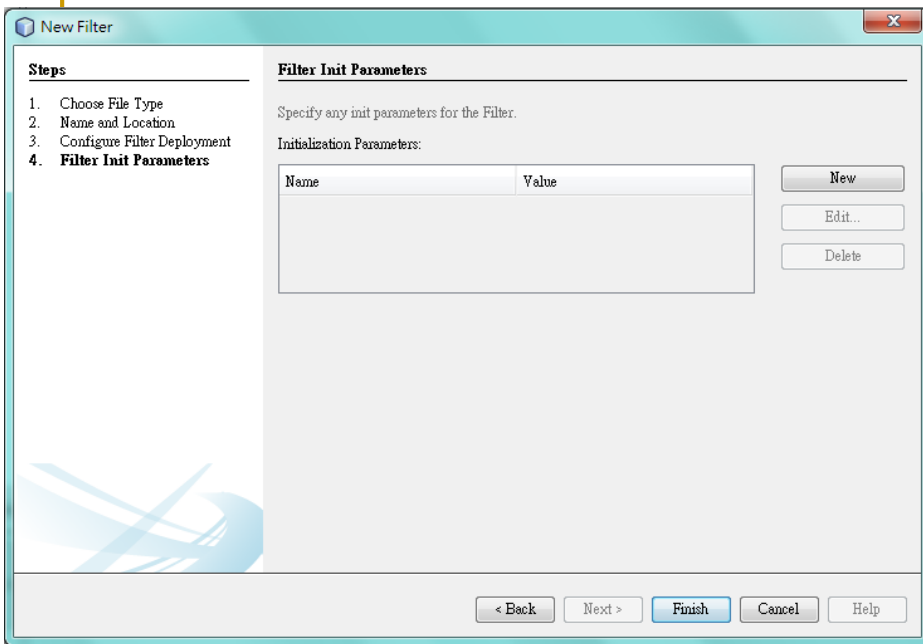
```
<filter-mapping>  
  <filter-name>auditFilter</filter-name>  
  <url-pattern>*.do</url-pattern>  
  <dispatcher>INCLUDE</dispatcher>  
  <dispatcher>FORWARD</dispatcher>  
</filter-mapping>
```

auditFilter只套用在 include及forward 至 *.do的servlet URL為*.do 的用戶端直接請求，不會套用 auditFilter

課程大綱

- 1) Intercepting Filter pattern
- 2) Filter API
- 3) 開發 **Filter** 類別





Performance

PerformanceFilter

-config : **FilterConfig**
-logPrefix : String

init(**FilterConfig**)
doFilter(request : ServletRequest,
 response : ServletResponse,
 chain : **FilterChain**)
destroy()

```
package web;
import javax.servlet.*;      import javax.servlet.http.*;
import java.io.*;           import javax.servlet.annotation.*;

@WebFilter(filterName = "perfFilter", urlPatterns = {"/*"}, initParams = {
    @WebInitParam(name = "Log Entry Prefix", value = "Performance : ")})
public class PerformanceFilter implements Filter {
    private FilterConfig config;
    private String logPrefix;

    public void init(FilterConfig config) throws ServletException {
        this.config = config;
        logPrefix = config.getInitParameter("Log Entry Prefix");
    }

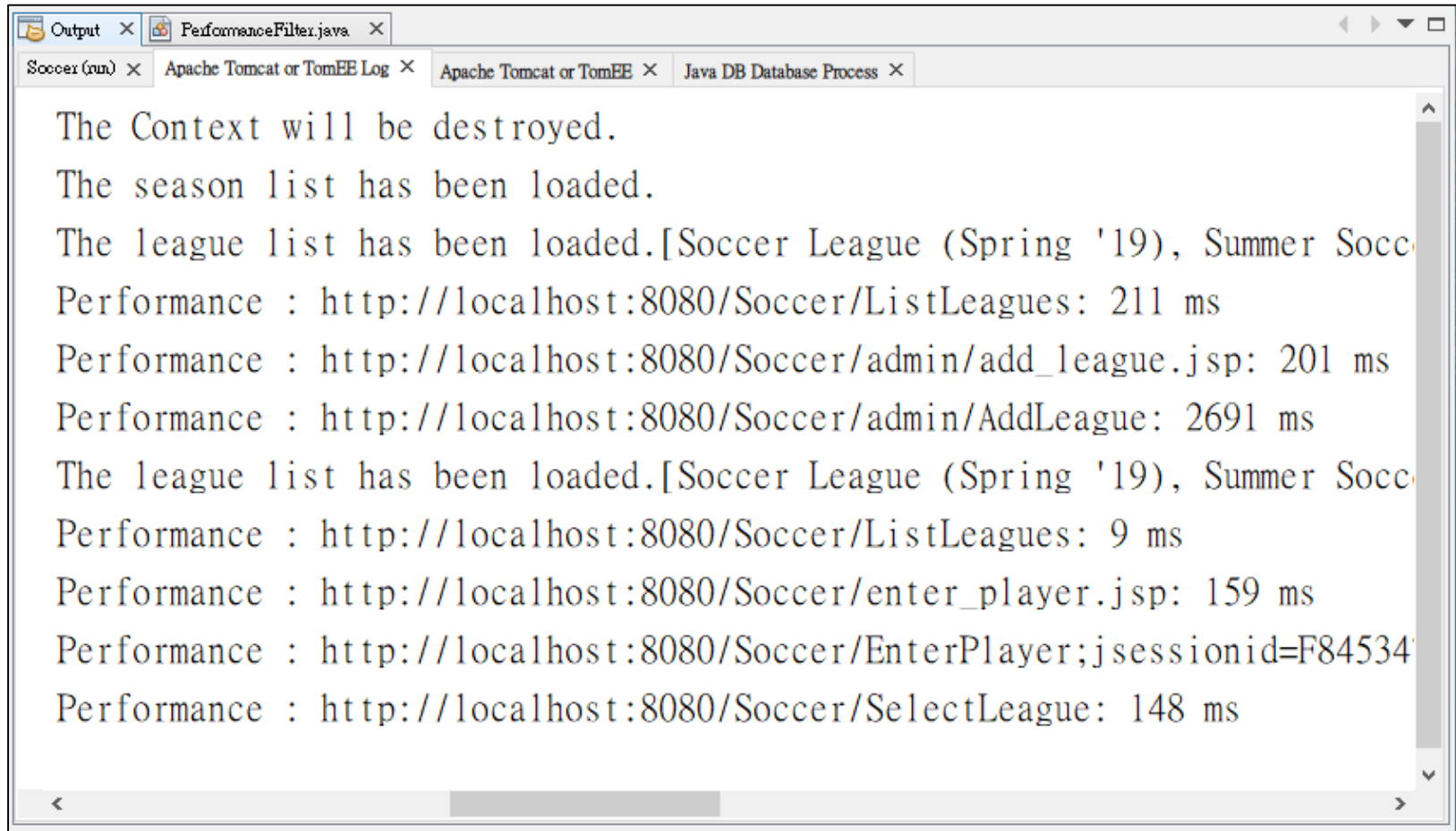
    public void doFilter(ServletRequest request, ServletResponse response,
                        FilterChain chain) throws ServletException, IOException {
        long begin = System.currentTimeMillis();
        chain.doFilter(request, response);
        long end = System.currentTimeMillis();
        StringBuffer logMessage = new StringBuffer();
        if (request instanceof HttpServletRequest) {
            logMessage = ((HttpServletRequest)request).getRequestURL();
        }
        logMessage.append(": ");
        logMessage.append(end - begin);
        logMessage.append(" ms");
        if (logPrefix != null) {
            logMessage.insert(0, logPrefix);
        }
        config.getServletContext().log(logMessage.toString());
    }

    public void destroy() {
        config = null;
        logPrefix = null;
    }
}
```

web.xml 設定

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app>
  <context-param>
    <param-name>data-directory</param-name>
    <param-value> /WEB-INF/data/</param-value>
  </context-param>
  <filter>
    <filter-name>perfFilter</filter-name>
    <filter-class>web.PerformanceFilter</filter-class>
    <init-param>
      <param-name>Log Entry Prefix</param-name>
      <param-value>Performance : </param-value>
    </init-param>
  </filter>
  <filter-mapping>
    <filter-name>perfFilter</filter-name>
    <url-pattern> /* </url-pattern>
  </filter-mapping>
  ....
</web-app>
```


PerformanceFilter



The screenshot shows an IDE window with multiple tabs. The active tab is 'PerformanceFilter.java'. Below it, there are four sub-tabs: 'Soccer (run)', 'Apache Tomcat or TomEE Log', 'Apache Tomcat or TomEE', and 'Java DB Database Process'. The main area displays the output of the 'Soccer (run)' process, which includes several log messages and performance metrics. The messages are: 'The Context will be destroyed.', 'The season list has been loaded.', 'The league list has been loaded.[Soccer League (Spring '19), Summer Soccer]', and 'Performance : http://localhost:8080/Soccer/ListLeagues: 211 ms'. This is followed by a series of performance metrics for different URLs: 'Performance : http://localhost:8080/Soccer/admin/add_league.jsp: 201 ms', 'Performance : http://localhost:8080/Soccer/admin/AddLeague: 2691 ms', 'The league list has been loaded.[Soccer League (Spring '19), Summer Soccer]', 'Performance : http://localhost:8080/Soccer/ListLeagues: 9 ms', 'Performance : http://localhost:8080/Soccer/enter_player.jsp: 159 ms', 'Performance : http://localhost:8080/Soccer/EnterPlayer;jsessionid=F84534...', and 'Performance : http://localhost:8080/Soccer/SelectLeague: 148 ms'.

```
The Context will be destroyed.  
The season list has been loaded.  
The league list has been loaded.[Soccer League (Spring '19), Summer Soccer]  
Performance : http://localhost:8080/Soccer/ListLeagues: 211 ms  
Performance : http://localhost:8080/Soccer/admin/add_league.jsp: 201 ms  
Performance : http://localhost:8080/Soccer/admin/AddLeague: 2691 ms  
The league list has been loaded.[Soccer League (Spring '19), Summer Soccer]  
Performance : http://localhost:8080/Soccer/ListLeagues: 9 ms  
Performance : http://localhost:8080/Soccer/enter_player.jsp: 159 ms  
Performance : http://localhost:8080/Soccer/EnterPlayer;jsessionid=F84534...  
Performance : http://localhost:8080/Soccer/SelectLeague: 148 ms
```

Lab

- 修改 **ContactInfo** 網路應用專案
- 新增 **Filter** 類別 **SecurityFilter.java**
 - Filter Mapping URL Patterns 設為 `/SelectCustomer`
 - Dispatch 狀態勾選 **REQUEST**
 - 初始參數：無
 - 刪除下列 **NetBeans** 實作內容
 - `getFilterConfig()`、`setFilterConfig()`、`sendProcessError()`、`log()`、`toString()`、`doBeforeProcessing()`、`doAfterProcessing()` 方法
 - `init(FilterConfig filterConfig)` 方法僅保留 `filterConfig` 初始設定
 - `destroy()` 方法修改為空白實作
 - 修改 `doFilter(...)`
 - 取得 **Session** 物件中 `"user"` 屬性
 - 若 **Session** 物件或 `"user"` 屬性不存在時, 轉送 `login.jsp` 畫面
 - **Session** 物件及 `"user"` 屬性存時, 執行 `chain.doFilter(...);`

Lab

- 新增 login.jsp
 - 輸入使用者姓名的Form表單
 - Action 送至"Login"
 - user 輸入欄位
- 新增 web.Login.java Servlet
 - 取得請求參數user並存入HttpSession屬性
 - 轉送至welcome.jsp
- 新增 welcome.jsp
 - 顯示歡迎使用者登入資訊
 - 回首頁連結
- 修改 customerList.jsp及customerView.jsp 檔案
 - 在內容之前,顯示問候user資訊
- 測試、執行
 - 顯示問候使用者的資訊