





Bộ môn Công nghệ Phần mềm  
Viện CNTT & TT  
Trường Đại học Bách Khoa Hà Nội

## LẬP TRÌNH WEB HƯỚNG JAVA

### Bài 10: Servlet nâng cao

Giảng viên: ThS. Trịnh Tuấn Đạt  
Bộ môn CNPM  
Email: trinhthuandat.bk@gmail.com/dattt@soict.hut.edu.vn



## Nội dung


- 1. Kỹ thuật Include, Forward, Redirect
- 2. Servlet filter
- 3. Xử lý sự kiện trong vòng đời Servlet
- 4. Điều khiển tương tranh
- 5. Invoker Servlet

DatTT-DSE-SOICT-HUST 2




## 1. Kỹ thuật Include, Forward, Redirect

DatTT-DSE-SOICT-HUST 3



## 1.1. Kỹ thuật đính kèm (include)


DatTT-DSE-SOICT-HUST 4



## Khi nào cần đính kèm 1 tài nguyên Web?

- Đôi khi sẽ rất hữu ích nếu có thể chèn các nội dung tĩnh/động đã tạo trong web resource khác:
  - Ví dụ: thêm banner, copyright information vào trong response trả về từ một Web component (JSP/servlet)

DatTT-DSE-SOICT-HUST 5



## Các loại Web Resource được đính kèm

- Static resource
  - Đơn thuần thêm nội dung tĩnh vào response của servlet đang xét (servlet muốn đính kèm)
- Dynamic web component (Servlet hoặc JSP)
  - Gửi request tới Web component ĐƯỢC ĐÍNH KÈM
  - Thực thi Web component ĐƯỢC ĐÍNH KÈM
  - Đính kèm kết quả thực thi được vào response của servlet đang xét

DatTT-DSE-SOICT-HUST 6

## Web Resource được đính kèm có thể và không thể làm gì?

- Web resource được đính kèm có thể truy cập tới đối tượng request, nhưng bị hạn chế với đối tượng response
  - có thể viết vào phần body của response và commit một response
  - không thể thiết lập các headers hoặc gọi bất kỳ phương thức nào (ví dụ, setCookie) ảnh hưởng đến các headers của response

DatTT-DSE-SOICT-HUST

7

## Làm thế nào để Include Web resource?

- Lấy đối tượng `RequestDispatcher` từ đối tượng `ServletContext`
  - `RequestDispatcher dispatcher = getServletContext().getRequestDispatcher("/banner");`
- Sau đó, gọi phương thức `include()` của đối tượng `RequestDispatcher` với tham số là đối tượng `request` và `response`:
  - `dispatcher.include(request, response);`

DatTT-DSE-SOICT-HUST

8

## Ví dụ: BannerServlet - Web component được đính kèm

```
public class BannerServlet extends HttpServlet {
    public void doGet (HttpServletRequest request,
        HttpServletResponse response)
        throws ServletException, IOException {

        PrintWriter out = response.getWriter();
        out.println("<body bgcolor=\"#ffffff\">" +
            "<center>" + "<hr> <br> &nbsp;" + "<h1>" +
            "<font size=1+3\" color=\"#CC0066\">Duke's </font>" +
            "<img src=\"" + request.getContextPath() +
            "/duke.books.gif\">" +
            "<font size=1+3\" color=\"black\">Bookstore</font>" +
            "</h1>" + "</center>" + "<br> &nbsp;" + "<hr> <br> ");
    }

    public void doPost (HttpServletRequest request,
        HttpServletResponse response)
        throws ServletException, IOException {

        PrintWriter out = response.getWriter();
        out.println("<body bgcolor=\"#ffffff\">" +
            "<center>" + "<hr> <br> &nbsp;" + "<h1>" +
            "<font size=1+3\" color=\"#CC0066\">Duke's </font>" +
            "<img src=\"" + request.getContextPath() +
            "/duke.books.gif\">" +
            "<font size=1+3\" color=\"black\">Bookstore</font>" +
            "</h1>" + "</center>" + "<br> &nbsp;" + "<hr> <br> ");
    }
}
```

DatTT-DSE-SOICT-HUST

9

## Ví dụ: đính kèm "BannerServlet"

```
RequestDispatcher dispatcher =
    getServletContext().getRequestDispatcher("/banner");
if (dispatcher != null)
    dispatcher.include(request, response);
}
```

DatTT-DSE-SOICT-HUST

10

## 1.2. Kỹ thuật forward

DatTT-DSE-SOICT-HUST

11

## Khi nào cần sử dụng kỹ thuật "Forwarding" tới Web resource khác?

- Khi muốn có một Web component thực hiện xử lý sơ bộ 1 request và một component khác đảm nhiệm sinh response
  - Ví dụ: xử lý 1 phần request rồi chuyển cho component khác, tùy từng request

DatTT-DSE-SOICT-HUST

12

## Quy định khi sử dụng kỹ thuật "Forwarding"

- Nên sử dụng khi yêu cầu 1 resource khác phản hồi lại cho 1 user
  - Nếu đã truy cập vào đối tượng `ServletOutputStream` hoặc `PrintWriter` trong servlet, sẽ không forward được nữa, nếu không có ngoại lệ `IllegalStateException`

DatTT-DSE-SOICT-HUST

13

## Cách thức "Forwarding" tới một Web resource khác?

- Lấy ra đối tượng `RequestDispatcher` từ đối tượng `HttpServletRequest`
  - `RequestDispatcher dispatcher = request.getRequestDispatcher("/template.jsp");`
- Nếu cần giữ lại URL gốc để xử lý thêm, có thể lưu lại thành 1 thuộc tính tầm vực request
- Gọi phương thức `forward()` của đối tượng `RequestDispatcher`
  - `dispatcher.forward(request, response);`

DatTT-DSE-SOICT-HUST

14

## Ví dụ: Dispatcher Servlet

```
public class Dispatcher extends HttpServlet {
    public void doGet(HttpServletRequest request,
        HttpServletResponse response) {
        request.setAttribute("selectedScreen",
            request.getServletPath());
        RequestDispatcher dispatcher = request.
            getRequestDispatcher("/template.jsp");
        if (dispatcher != null)
            dispatcher.forward(request, response);
    }
    public void doPost(HttpServletRequest request,
        ...
    }
}
```

DatTT-DSE-SOICT-HUST

15

## 1.3. Chỉ định trình duyệt redirect tới Web Resource khác

DatTT-DSE-SOICT-HUST

16

## Redirect một Request

- 2 kỹ thuật điều hướng request
- Cách 1:
  - `res.setStatus(res.SC_MOVED_PERMANTLY);`
  - `res.setHeader("Location", "http://...");`
- Cách 2:
  - `public void sendRedirect(String url)`

DatTT-DSE-SOICT-HUST

17

## 2. Servlet Filters

DatTT-DSE-SOICT-HUST

18

## 2. Servlet Filters

- 2.1. Servlet filters là gì và tại sao cần?
- 2.2. Các Servlet Filters móc nối với nhau như thế nào?
- 2.3. APIs lập trình cho Servlet Filter
- 2.4. Cấu hình Servlet filter trong file web.xml
- 2.5. Các bước xây dựng và triển khai các servlet filters
- 2.6. Ví dụ

DatTT-DSE-SOICT-HUST

19

## 2.1. Servlet filters là gì và tại sao cần?

DatTT-DSE-SOICT-HUST

20

## Java Servlet Filters là gì?

- Là thành phần để chặn và sửa (intercept & modify) các requests và responses
  - Filters có thể liên kết với nhau thành 1 chuỗi và tích hợp vào hệ thống khi triển khai (deploy)
- Cho phép cấu hình:
  - Đếm, chặn các truy cập (access)
  - Caching, compression, logging
  - Authentication, access control, encryption
  - Content transformations
- Được giới thiệu trong Servlet 2.3 (Tomcat 4.0)

DatTT-DSE-SOICT-HUST

21

## Một Filter có thể làm gì?

- Kiểm tra (Examine) các request headers
- Điều chỉnh lại đối tượng request: sửa phần dữ liệu hoặc request headers
- Điều chỉnh lại đối tượng responseL sửa phần dữ liệu hoặc response headers
- Gọi đến filter kế tiếp trong chuỗi
- Kiểm tra (Examine) lại các response headers sau khi đã gọi filter kế tiếp trong chuỗi
- Tung ra ngoại lệ để thông báo có lỗi trong quá trình xử lý

DatTT-DSE-SOICT-HUST

22

## Ví dụ 1:

- Đang có nhiều servlets và các trang JSP pages cần thực hiện chung các chức năng như ghi log hoặc chuyển thành định dạng XSLT (để biểu diễn dữ liệu)
  - Không muốn thay đổi tất cả các servlets & các trang JSP này
  - Muốn xây dựng các chức năng chung như 1 module, tái sử dụng được
- Giải pháp:
  - Tạo 1 logging filter và compression filter
  - Gắn vào hệ thống khi triển khai

DatTT-DSE-SOICT-HUST

23

## Ví dụ 2

- Muốn tách biệt điều khiển truy cập (access control) khỏi phần code hiển thị-presentation (JSP pages)
  - LTV không muốn thay đổi từng trang JSP
- Giải pháp
  - Tạo một "access-control" servlet

DatTT-DSE-SOICT-HUST

24

### Ví dụ 3

- Có rất nhiều các Web resources mà chỉ thay đổi 1 số ít các giá trị (Ví dụ banners hoặc tên công ty)
  - LTV không muốn thay đổi các Web resources này mỗi lần tên công ty thay đổi
- Giải pháp
  - Tạo "banner replacement filter" hoặc "company name replacement filter"

DatTT-DSE-SOICT-HUST

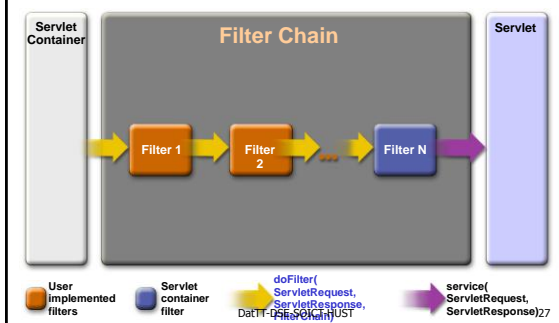
25

## 2.2. Các Servlet Filters móc nối với nhau như thế nào?

DatTT-DSE-SOICT-HUST

26

### Servlet Filter làm việc như thế nào?



DatTT-DSE-SOICT-HUST

27

### Filter Chain làm việc như thế nào?

- Các filters có thể móc nối với nhau
  - Thứ tự móc nối: chính là thứ tự các phần tử `<filter>` trong file `web.xml`
- Filter đầu tiên trong chuỗi sẽ được container gọi
  - Qua phương thức `doFilter(ServletRequest req, ServletResponse res, FilterChain chain)`
  - Filter sau khi thực hiện công việc xong, sẽ gọi filter tiếp theo trong chuỗi: gọi phương thức `chain.doFilter(..)`
- Filter cuối cùng gọi phương thức `service()` của Servlet

DatTT-DSE-SOICT-HUST

28

## 2.3. APIs lập trình cho Servlet Filter

DatTT-DSE-SOICT-HUST

29

### Giao diện `javax.servlet.Filter`

- `init(FilterConfig)`
  - Được gọi 1 lần duy nhất khi filter được khởi tạo lần đầu
  - Lấy ra đối tượng `ServletContext` từ đối tượng `FilterConfig` và lưu vào đâu đó (Vd làm 1 thuộc tính của Filter) để phương thức `doFilter()` có thể truy cập.
  - Đọc các tham số khởi tạo từ đối tượng `FilterConfig` qua phương thức `getInitParameter()`
- `destroy()`
  - Được gọi duy nhất 1 lần khi container hủy đối tượng filter
  - VD: đóng file hoặc đóng kết nối Database

DatTT-DSE-SOICT-HUST

30

## Giao diện javax.servlet.Filter

- doFilter(ServletRequest req, ServletResponse res, FilterChain chain)
  - Được gọi mỗi khi filter được kích hoạt
  - Chứa các xử lý của filter
  - Đối tượng ServletRequest được ép kiểu về HttpServletRequest nếu request là HTTP request
  - Gọi filter tiếp theo: chain.doFilter(..)
  - Hoặc chặn request lại. Cách thức:
    - không gọi phương thức chain.doFilter(..)
    - Filter phải cung cấp output cho client
  - set headers on the response for next entity

DatTT-DSE-SOICT-HUST

31

## Một số lớp khác liên quan tới Servlet Filter

- javax.servlet.FilterChain
  - Được truyền làm tham số cho phương thức doFilter()
- javax.servlet.FilterConfig
  - Được truyền làm tham số trong phương thức init()
- javax.servlet.HttpServletResponseWrapper
  - Là cài đặt tiện ích cho giao diện HttpServletResponse interface

DatTT-DSE-SOICT-HUST

32

## 2.4. Cấu hình Servlet Filter trong file web.xml file

DatTT-DSE-SOICT-HUST

33

## Cấu hình trong web.xml

- <filter>
  - <filter-name>: tên của filter
  - <filter-class>: giúp container biết được lớp xử lý tương ứng của filter
- </filter>
- <filter-mapping>
  - <filter-name>: tên của filter là gì
  - <url-pattern>: mẫu URLs (Web resources) áp dụng cho filter có tên ở trên
- </filter-mapping>

DatTT-DSE-SOICT-HUST

34

## Ví dụ: web.xml của BookStore

```

<web-app>
<display-name>Bookstore1</display-name>
<description>no description</description>
<filter>
<filter-name>OrderFilter</filter-name>
<filter-class>filters.OrderFilter</filter-class>
</filter>
<filter>
<filter-name>HitCounterFilter</filter-name>
<filter-class>filters.HitCounterFilter</filter-class>
</filter>
<filter-mapping>
<filter-name>OrderFilter</filter-name>
<url-pattern>/receipt</url-pattern>
</filter-mapping>
<filter-mapping>
<filter-name>HitCounterFilter</filter-name>
<url-pattern>/enter</url-pattern>
</filter-mapping>
<listener>
...
</listener>
<servlet>
...
</servlet>

```

DatTT-DSE-SOICT-HUST

35

## 2.5. Các bước xây dựng và triển khai các Servlet Filters

DatTT-DSE-SOICT-HUST

36

## Các bước xây dựng Servlet Filter

- Tạo một lớp thực thi giao diện Filter
  - Thực thi các xử lý trong phương thức doFilter()
  - Gọi phương thức doFilter() của đối tượng FilterChain
- Cấu hình filter cho Servlet và các trang JSP cần áp dụng
  - Sử dụng <filter> và <filter-mapping>

DatTT-DSE-SOICT-HUST

37

## 2.6. Ví dụ Servlet Filter

DatTT-DSE-SOICT-HUST

38

## Ví dụ: HitCounterFilter

```
public final class HitCounterFilter implements Filter {
    private FilterConfig filterConfig = null;

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

    public void destroy() {
        this.filterConfig = null;
    }

    // Continued in the next page...
```

DatTT-DSE-SOICT-HUST

39

## Ví dụ: HitCounterFilter

```
public void doFilter(ServletRequest request,
    ServletResponse response, FilterChain chain)
    throws IOException, ServletException {

    if (filterConfig == null) return;
    StringWriter sw = new StringWriter();
    PrintWriter writer = new PrintWriter(sw);
    Counter counter = (Counter)
        filterConfig.getServletContext().getAttribute("hitCounter");
    writer.println("The number of hits is: " +
        counter.increaseCounter());

    // Log the resulting string
    writer.flush();
    filterConfig.getServletContext().log(sw.getBuffer().toString());
    ...
    chain.doFilter(request, wrapper);
    ...
}
```

DatTT-DSE-SOICT-HUST

40

## Cấu hình HitCounterFilter

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web
Application 2.3//EN" "http://java.sun.com/dtd/web-
app_2_3.dtd">

<web-app>
  <display-name>Bookstore</display-name>
  <description>no description</description>

  <filter>
    <filter-name>HitCounterFilter</filter-name>
    <filter-class>filters.HitCounterFilter</filter-class>
  </filter>

  <filter-mapping>
    <filter-name>HitCounterFilter</filter-name>
    <url-pattern>/enter</url-pattern>
  </filter-mapping>
  ...
```

DatTT-DSE-SOICT-HUST

41

## 3. Xử lý sự kiện trong vòng đời Servlet



DatTT-DSE-SOICT-HUST

42

### 3. Xử lý sự kiện trong vòng đời Servlet

- Hỗ trợ xử lý sự kiện khi có thay đổi trạng thái trong
  - ServletContext
    - Startup/shutdown
    - Thay đổi thuộc tính (Attribute changes)
  - HttpSession
    - Creation và invalidation
    - Thay đổi thuộc tính (Changes in attributes)

DatTT-DSE-SOICT-HUST

43

### Các bước thực hiện khi xử lý sự kiện

- Quyết định phạm vi của đối tượng cần xử lý sự kiện (context hay session)
- Thực thi giao diện phù hợp
- Override các phương thức cần xử lý sự kiện
- Cấu hình file web.xml
- Cung cấp thêm các tham số khởi tạo nếu cần

DatTT-DSE-SOICT-HUST

44

### Đăng ký Listener

- Web container
  - Tạo 1 thực thể cho mỗi lớp Listener
  - Đăng ký thực thể đó sẽ xử lý sự kiện gì, tùy theo:
    - Giao diện (interfaces) mà nó thực thi
    - Thứ tự đăng ký trong file web.xml
    - (Các Listeners sẽ được gọi theo đúng thứ tự đăng ký)

DatTT-DSE-SOICT-HUST

45

### Các giao diện Listener

- ServletContextListener
  - contextInitialized/Destroyed(ServletContextEvent)
- ServletContextAttributeListener
  - attributeAdded/Removed/Replaced(ServletContextAttributeEvent)
- HttpSessionListener
  - sessionCreated/Destroyed(HttpSessionEvent)
- HttpSessionAttributeListener
  - attributeAdded/Removed/Replaced(HttpSessionBindingEvent)
- HttpSessionActivationListener
  - Xử lý sessions di trú (migrate) từ server này sang server khác
  - sessionWillPassivate(HttpSessionEvent)
  - sessionDidActivate(HttpSessionEvent)

DatTT-DSE-SOICT-HUST

46

### Ví dụ: Context Listener

```
public final class ContextListener
    implements ServletContextListener {
    private ServletContext context = null;

    public void contextInitialized(ServletContextEvent event) {
        context = event.getServletContext();

        try {
            BookDB bookDB = new BookDB();
            context.setAttribute("bookDB", bookDB);
        } catch (Exception ex) {
            context.log("Couldn't create bookstore
                database bean: " + ex.getMessage());
        }

        Counter counter = new Counter();
        context.setAttribute("hitCounter", counter);
        counter = new Counter();
        context.setAttribute("orderCounter", counter);
    }
}
```

DatTT-DSE-SOICT-HUST

47

### Ví dụ: Context Listener

```
public void contextDestroyed(ServletContextEvent event) {
    context = event.getServletContext();
    BookDB bookDB = BookDB.context.getAttribute("bookDB");
    bookDB.remove();
    context.removeAttribute("bookDB");
    context.removeAttribute("hitCounter");
    context.removeAttribute("orderCounter");
}
}
```

DatTT-DSE-SOICT-HUST

48



## Cấu hình Listener

```
<web-app>
  <display-name>Bookstore1</display-name>
  <description>no description</description>

  <filter>..</filter>
  <filter-mapping>..</filter-mapping>
  <listener>
    <listener-class>listeners.ContextListener</listener-class>
  </listener>
  <servlet>..</servlet>
  <servlet-mapping>..</servlet-mapping>
  <session-config>..</session-config>
  <error-page>..</error-page>
  ...
</web-app>
```

DatTT-DSE-SOICT-HUST

49

## Mô hình Thread và đồng bộ hóa các Servlet

DatTT-DSE-SOICT-HUST

50

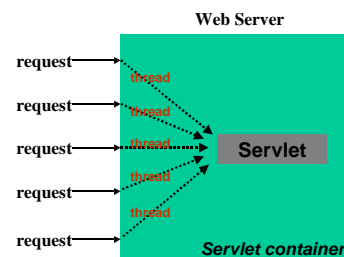
## Vấn đề điều khiển tương tranh trong 1 servlet

- Phương thức service() của một thực thể servlet có thể được gọi bởi nhiều clients (multiple threads)
- LTV lập trình Servlet phải tự giải quyết vấn đề tương tranh (concurrency)
  - Cần bảo vệ dữ liệu chung (shared data)
  - Được gọi là "**servlet synchronization**"
- 2 lựa chọn đồng bộ hóa servlet
  - Sử dụng **synchronized block**
  - Sử dụng **SingleThreadModel**

DatTT-DSE-SOICT-HUST

51

## Nhiều Threads, Một Servlet Instance



DatTT-DSE-SOICT-HUST

52

## Sử dụng synchronized block

- Synchronized blocks được sử dụng để đảm bảo **chỉ có duy nhất một thread tại một thời điểm có thể thực thi một đoạn code**

```
synchronized(this) {
  myNumber = counter + 1;
  counter = myNumber;
}
...
synchronized(this) {
  counter = counter - 1 ;
}
```

DatTT-DSE-SOICT-HUST

53

## Giao diện SingleThreadModel

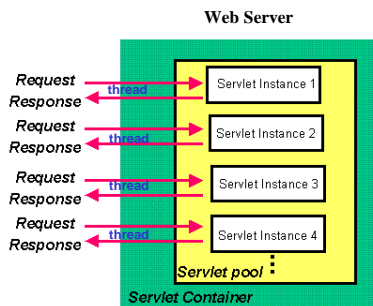
- Servlets cũng có thể thực thi javax.servlet.SingleThreadModel
- Server sẽ quản lý một kho (pool) các servlet instances
- Đảm bảo rằng chỉ có **MỘT** thread truy cập một instance
- Có thể dẫn đến bùng nổ instances

```
Public class SingleThreadModelServlet extends
HttpServlet implements SingleThreadModel {
  ...
}
```

DatTT-DSE-SOICT-HUST

54

## SingleThreadModel



DatTT-DSE-SOICT-HUST

55

## Khuyến cáo

- Nếu có thể, nên sử dụng **synchronized block**
  - SingleThreadModel có chi phí rất cao

DatTT-DSE-SOICT-HUST

56

## Invoker Servlet

DatTT-DSE-SOICT-HUST

57

## Invoker Servlet là gì?

- Thực thi các servlet không định nghĩa trong file web.xml
- Chủ yếu sử dụng cho mục đích debug và test
- Ví dụ ngữ cảnh sử dụng
  - Có 1 lượng lớn servlets (vd: 1000 servlets)
  - LTV không muốn thực hiện servlet mapping cho tất cả 1000 servlets trong file web.xml, đặc biệt là mới đang phát triển ứng dụng (chưa triển khai)
  - Nếu sử dụng invoker servlet (cung cấp bởi container), user vẫn truy cập 1000 servlets bình thường, không cần thực hiện servlet mapping trong file web.xml.

DatTT-DSE-SOICT-HUST

58

## Làm thế nào để sử dụng Invoker Servlet?

- Bỏ comment cho đoạn cấu hình sau từ file <Tomcat>/conf/web.xml và khởi động lại Tomcat

```
<!--
<servlet>
  <servlet-name>invoker</servlet-name>
  <servlet-class>
    org.apache.catalina.servlets.InvokerServlet
  </servlet-class>
  <init-param>
    <param-name>debug</param-name>
    <param-value>0</param-value>
  </init-param>
  <load-on-startup>2</load-on-startup>
</servlet>
```

DatTT-DSE-SOICT-HUST

59

## Làm thế nào để sử dụng Invoker Servlet?

- Thêm đoạn cấu hình sau vào file web.xml

```
<servlet-mapping>
  <servlet-name>invoker</servlet-name>
  <url-pattern>/myservlets/*</url-pattern>
</servlet-mapping>
```

- Từ trình duyệt, truy cập tới các servlet như sau:
  - http://localhost:8080/myservlets/newservlet2

DatTT-DSE-SOICT-HUST

60