FPT Education

FPT POLYTECHNIC

cdio™

Conceive Design Implement Operate

SCOPES, LISTENER & FILTER

J4 PROGRAMMING – SERVLET/JSP & JPA

www.poly.edu.vn

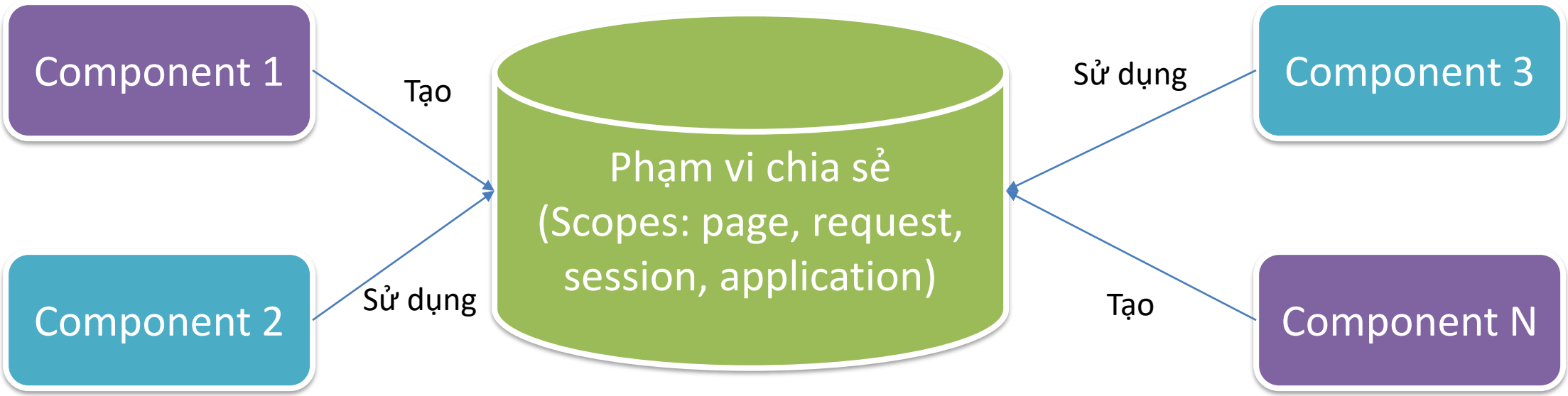
NỘI DUNG

- Phần 1: Scope và Listener
  - Scopes và Scope API
  - Listener
- Phần 2: Filter
  - Filter
  - Ứng dụng của Filter

1 Scopes và Listener

SCOPE LÀ GÌ?

- Chia sẻ dữ liệu là việc dữ liệu được tạo ra bởi một thành phần (Servlet, JSP, filter, listener) này và được sử dụng bởi một thành phần khác. Nơi duy trì dữ liệu tạm thời đó được gọi là scope (phạm vi chia sẻ).
- Trong servlet/jsp tồn tại 4 scope với tên gọi là **page**, **request**, **session** và **application**

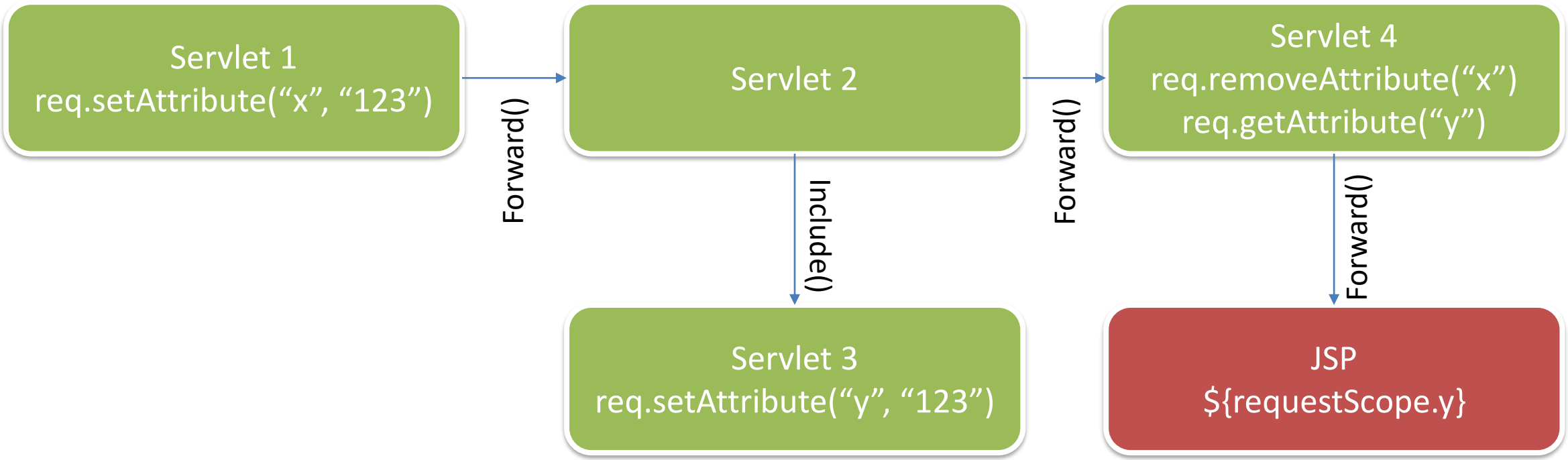


THAM CHIẾU SCOPE TRONG SERVLET

- Request (HttpServletRequest) sẵn có dưới dạng đối số của phương thức dịch vụ
- Tham chiếu đến Session và Application
  - HttpSession session = request.**getSession**()
  - ServletContext application = this.getServletContext()
  - ServletContext application = request.**getServletContext**()
  - ServletContext application = session.getServletContext()
- Scope API
  - **<<scope>>. setAttribute**(String, Object)
  - **<<scope>>.getAttribute**(String): Object
  - **<<scope>>. removeAttribute**(String)

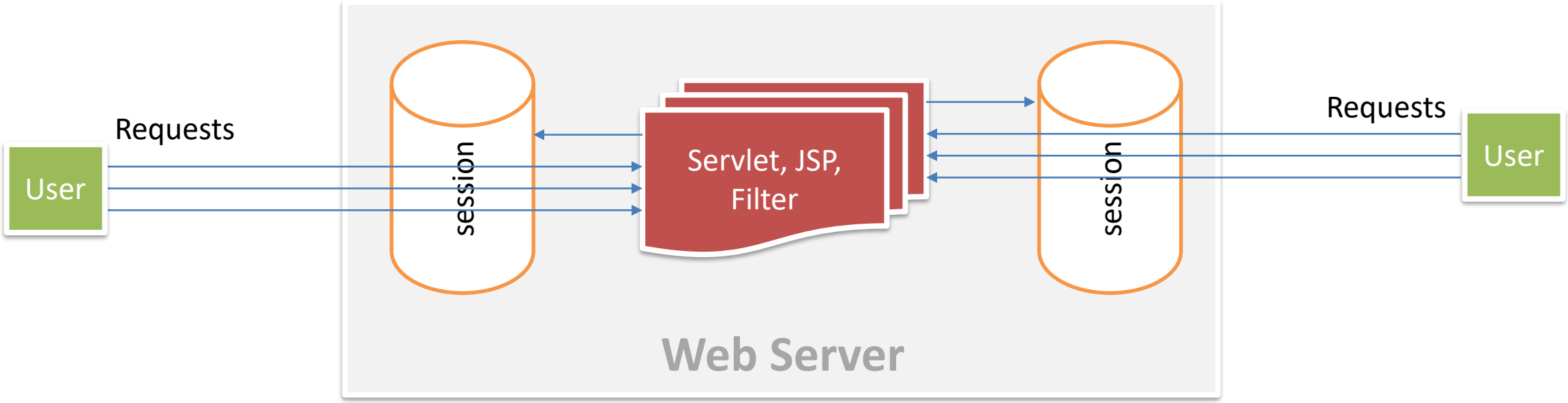
REQUEST SCOPE

- **Request** (HttpServletRequest) được sử dụng để chia sẻ dữ liệu giữa các thành phần hoạt động trên cùng một dây chuyền include() và forward()



SESSION SCOPE

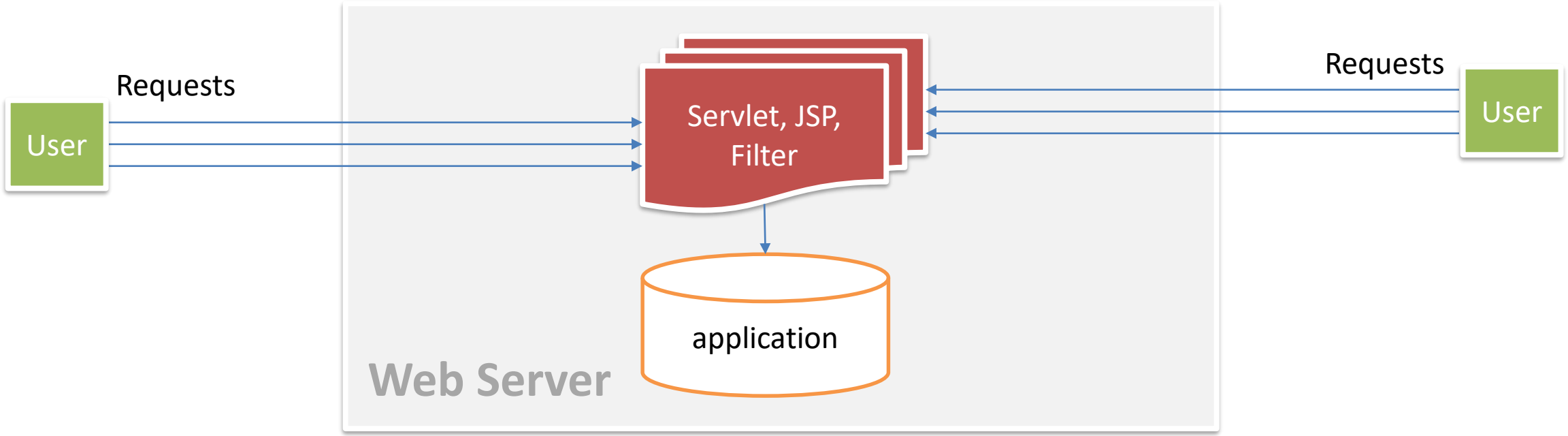
- Session (HttpSession) được sử dụng để chia sẻ dữ liệu giữa các thành phần hoạt động trong cùng một phiên làm việc.



- Có thể sử dụng session để duy trì giỏ hàng điện tử, thông tin đăng nhập, thông tin cá nhân hóa...

APPLICATION SCOPE

- Application (ServletContext) được sử dụng để chia sẻ dữ liệu cho tất cả mọi thành phần trong ứng dụng (chung toàn ứng dụng).



- Có thể sử dụng application để duy trì bộ đếm khách thăm web, Cache dữ liệu dùng chung để giảm truy vấn CSDL,...


DEMO

@WebListener

- @WebListener được tạo ra để lắng nghe và xử lý những sự kiện xảy ra bên trong ứng dụng web.
  - Khi nào ứng dụng bắt đầu, kết thúc
  - Khi nào một phiên làm việc bắt đầu, hết hạn
- ServletContextListener được sử dụng để xử lý 2 sự kiện: bắt đầu và kết thúc ứng dụng
- HttpSessionListener được sử dụng để xử lý 2 sự kiện: session được tạo ra và session hết hạn
- …

SERVLETCONTEXTLISTENER

```
@WebListener
public class AppListener implements ServletContextListener{
        @Override
        public void contextDestroyed(ServletContextEvent sce) {
                // TODO Chạy trước khi ứng dụng web bị shutdown
                ServletContext app = sce.getServletContext();
        }
        @Override
        public void contextInitialized(ServletContextEvent sce) {
                // TODO Chạy ngay sau khi ứng dụng khởi động
                ServletContext app = sce.getServletContext();
        }
}
```

HttpSessionListener

```
@WebListener
public class UserListener implements HttpSessionListener{
        @Override
        public void sessionCreated(HttpSessionEvent e) {
                // TODO Chạy ngay sau khi một phiên làm việc được tạo
                HttpSession session = e.getSession();
        }
        @Override
        public void sessionDestroyed(HttpSessionEvent e) {
                // TODO Chạy ngay trước khi phiên làm việc hết hạn
                HttpSession session = e.getSession();
        }
}
```

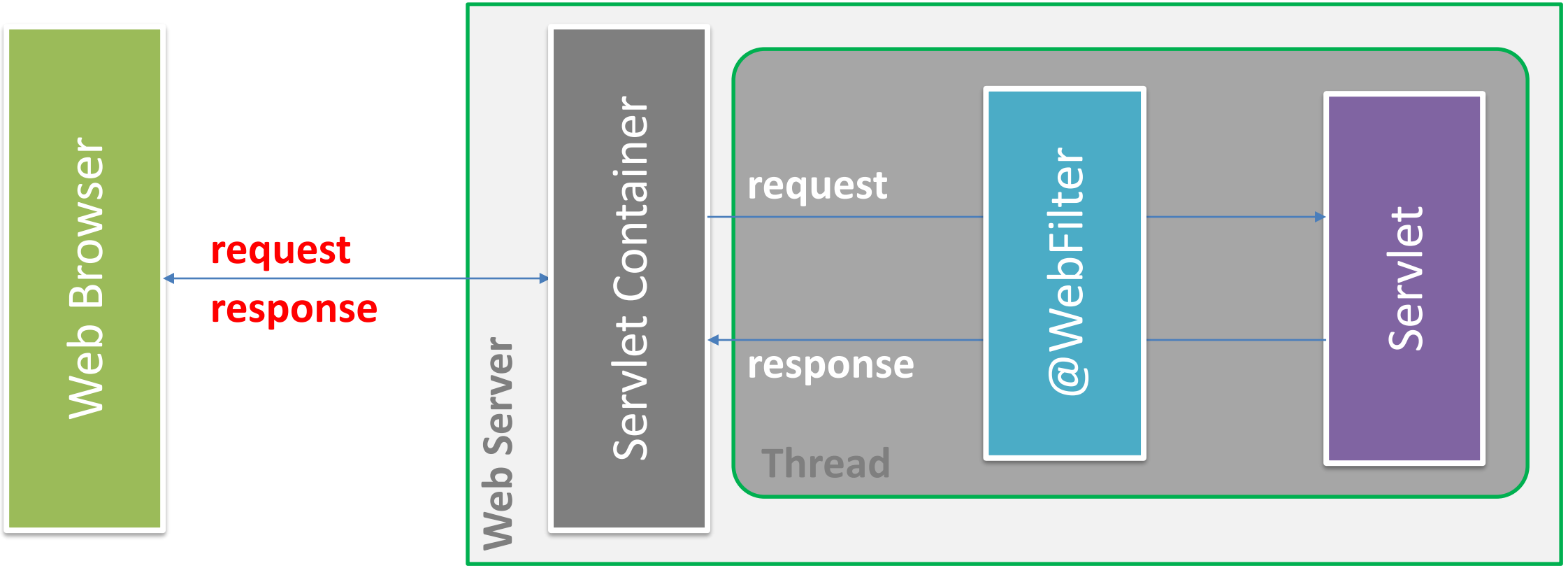
@WebListener

```
@WebListener
public class AppListener implements ServletContextListener, HttpSessionListener {
        @Override
        public void sessionCreated(HttpSessionEvent e) {
        }
        @Override
        public void sessionDestroyed(HttpSessionEvent e) {
        }
        @Override
        public void contextDestroyed(ServletContextEvent e) {
        }
        @Override
        public void contextInitialized(ServletContextEvent e) {
        }
}
```


DEMO
Online Users Counter
Visitors Counter
Shopping Cart

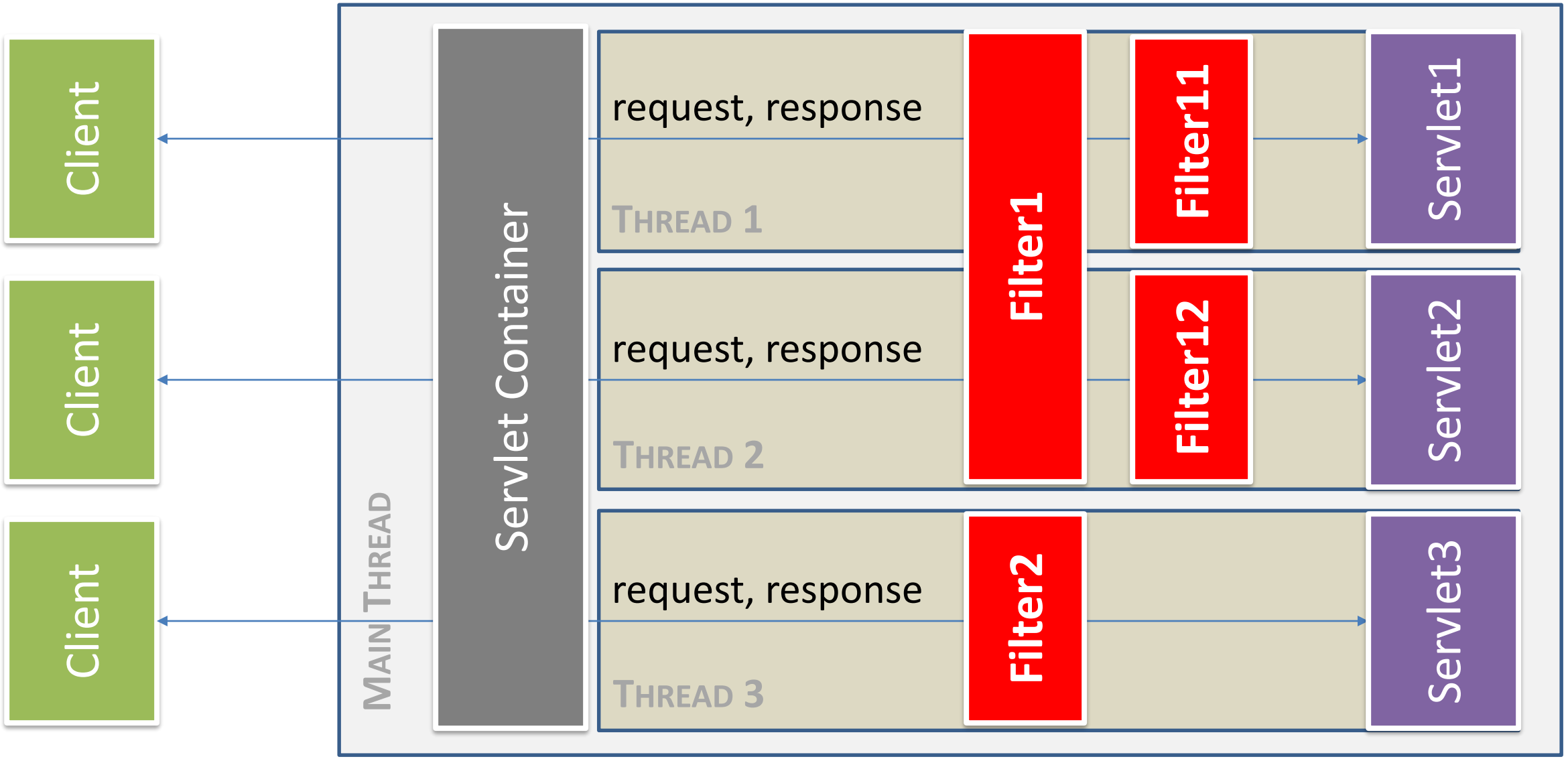
② Filter

- @WebFilter được sử dụng để thực hiện các công việc trước khi request chuyển đến servlet và sau khi servlet thực hiện xong.



Chú ý: Filter.doChain() và Servlet.service() luôn luôn hoạt động trong cùng một Thread

MULTIPLE WEBFILTER



Chú ý: một servlet/jsp có thể được lọc bởi nhiều filter và một filter có thể lọc nhiều servlet

CẤU TRÚC CỦA WEBFILTER

```
@WebFilter(urlPatterns = {"/url1", "/url2", "/urlN"}, dispatcherTypes = DispatcherType.REQUEST)
public class MyFilter implements Filter{
        @Override
        public void destroy() {
                // TODO Chạy trước khi bị giải phóng
        }
        @Override
        public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain)
                        throws IOException, ServletException {
                // TODO Chạy trước khi đến servlet/jsp
                chain.doFilter(request, response); // chuyển tiếp đến servlet/jsp
                // TODO Chạy sau khi servlet/jsp thực hiện xong
        }
        @Override
        public void init(FilterConfig filterConfig) throws ServletException {
                // TODO Chạy sau khi được nạp vào
        }
}
```

Chú ý: nếu trong doFilter() có gọi chain.doFilter() thì request mới được chuyển tiếp đến Servlet hoặc Filter kế tiếp, ngược lại thì Servlet hoặc Filter kế tiếp không được thực hiện

URLPATTERNS VÀ DISPATCHERTYPES

- urlPatterns xác định các url bị lọc. Cú pháp tương tự @WebServlet
  - ***.php**: chỉ lọc các địa chỉ url với đuôi là .php
  - **/***: lọc tất cả các địa chỉ url
  - **/abc/***: chỉ lọc tất cả địa chỉ bắt đầu với /abc/
  - **/abc/def.php**: lọc một địa chỉ /abc/def.php
- dispatcherTypes xác định các cách kích hoạt bộ lọc
  - **REQUEST**: lọc mỗi khi có request (mặc định)
  - **INCLUDE**: lọc mỗi khi được include()
  - **FORWARD**: lọc mỗi khi được forward()
  - **ERROR**: lọc mỗi khi có lỗi

Ví dụ: Utf8Filter - Xử lý tiếng việt

```
@WebFilter(urlPatterns = "/*", dispatcherTypes = DispatcherType.REQUEST)
public class Utf8Filter implements Filter{
        @Override
        public void doFilter(ServletRequest req, ServletResponse resp, FilterChain chain)
                        throws IOException, ServletException {
                req.setCharacterEncoding("utf-8");
                resp.setCharacterEncoding("utf-8");
                chain.doFilter(req, resp);
        }
        @Override
        public void init(FilterConfig filterConfig) throws ServletException {}
        @Override
        public void destroy() {}
}
```

FPT Education
FPT POLYTECHNIC

SẮP XẾP THỨ TỰ THỰC HIỆN WEBFILTER

```
@WebFilter(filterName="filter1", urlPatterns="/url1/*")
public class Filter1 implements Filter {}

@WebFilter(filterName="filter2", urlPatterns="/url2/*")
public class Filter2 implements Filter {}
```

```
<filter-mapping>
   <filter-name>filter1</filter-name>
   <url-pattern />
</filter-mapping>
<filter-mapping>
   <filter-name>filter2</filter-name>
   <url-pattern />
</filter-mapping>
```

web.xml

Filter nào mapping trước sẽ thực hiện trước

Xây dựng HttpFilter

- Vấn đề của javax.servlet.Filter
  - javax.servlet.Filter phục vụ cho nhiều công nghệ (FTP và HTTP) nên đối số của doChain() là ServletRequest và ServletResponse.
  - Trong khi lập trình web thì chúng ta cần HttpServletRequest và HttpServletResponse nên cần phải thực hiện ép kiểu.
  - javax.servlet.Filter chứa 3 phương thức init(), destroy() và doFilter(). Vì vậy khi thực thi theo interface này chúng ta phải viết cả 3 phương thức nói trên. Tuy nhiên trong thực tế đôi khi chúng ta không quan tâm đến init() và destroy() và chỉ viết mã cho doFilter()
- Giải pháp: xây dựng HttpFilter thừa kế Filter
  - Khai báo lại init(), destroy() và doFilter() với từ khóa default
  - Bổ sung doFilter(HttpServletRequest, HttpServletResponse, FilterChain) và gọi nó trong default doFilter()

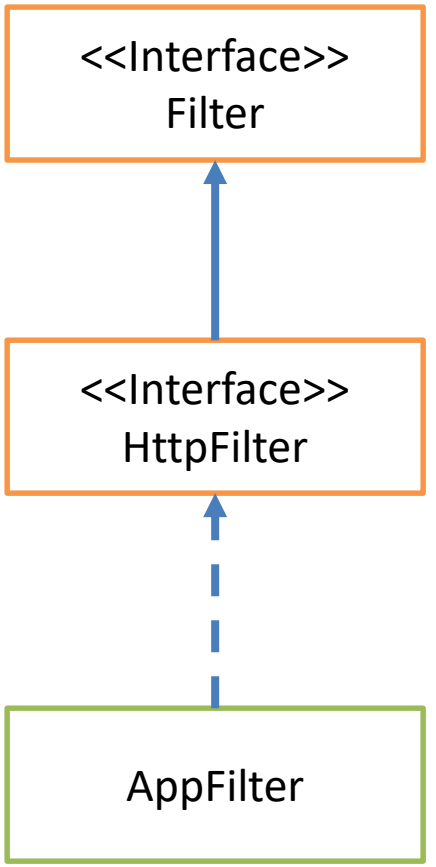
XÂY DỰNG HTTPFILTER

```
public interface HttpFilter extends Filter {
        @Override
        default void doFilter(ServletRequest req, ServletResponse resp, FilterChain chain)
                        throws IOException, ServletException {
                HttpServletRequest request = (HttpServletRequest)req;
                HttpServletResponse response = (HttpServletResponse)resp;
                this.doFilter(request, response, chain);
        }
        void doFilter(HttpServletRequest req, HttpServletResponse resp, FilterChain chain)
                        throws IOException, ServletException;

        @Override
        default void init(FilterConfig e) throws ServletException {}

        @Override
        default void destroy() {}
}
```

AppFilter

- Tạo một Servlet Filter
  - Thực thi theo interface HttpFilter thay vì Filter
  - Khi cần chúng ta có thể override init() và destroy()
- Ví dụ: Tạo AppFilter thiết lập utf-8 cho tất cả request

```
@WebFilter("/*")
public class AppFilter implements HttpFilter{
        @Override
        public void doFilter(HttpServletRequest req, HttpServletResponse resp,
                        FilterChain chain) throws IOException, ServletException {
                req.setCharacterEncoding("utf-8");
                resp.setCharacterEncoding("utf-8");
                chain.doFilter(req, resp);
        }
}
```



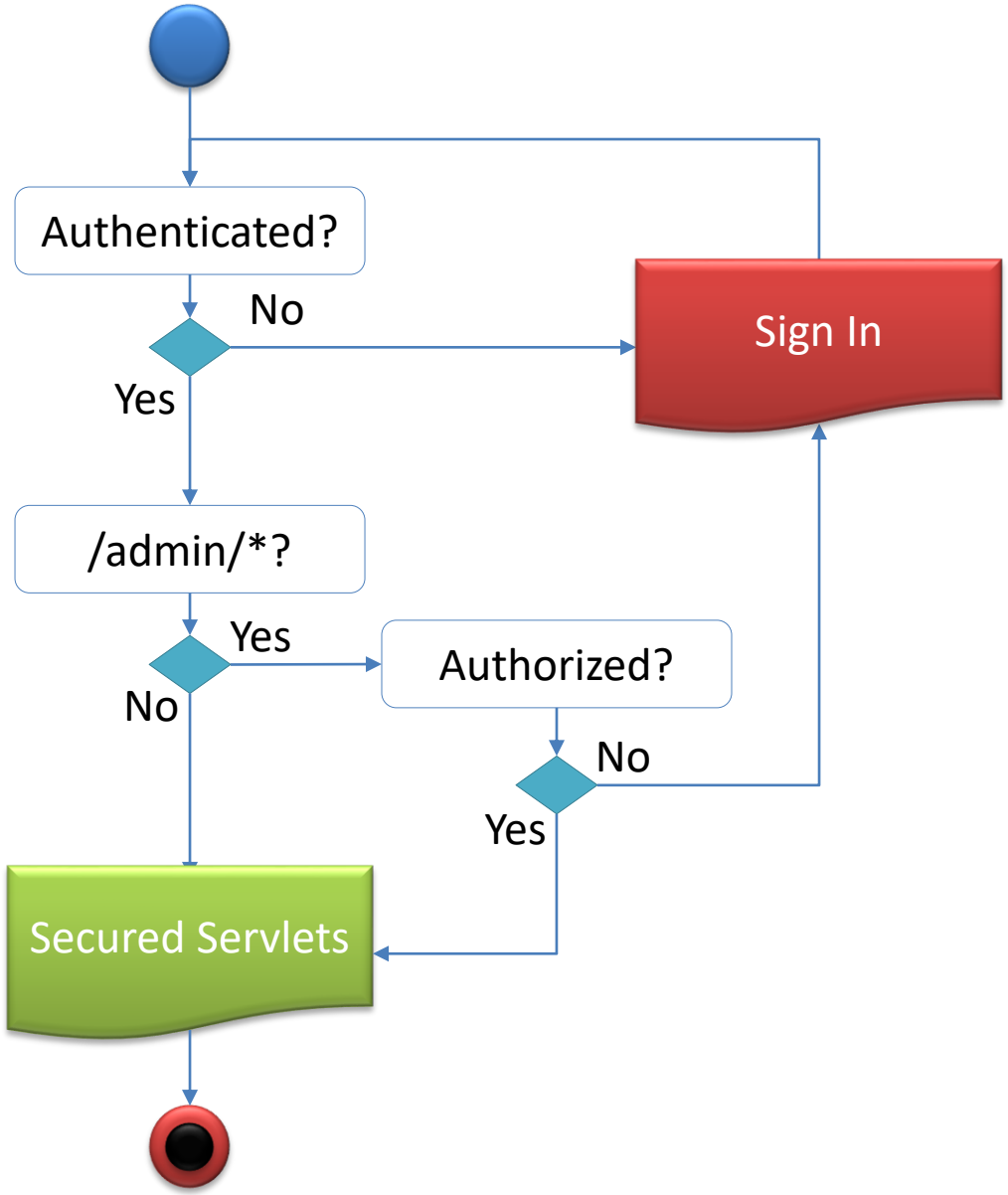
CÁC ỨNG DỤNG CỦA WEBFILTER

- **LoggerFilter** thông tin truy xuất (user, url, time)
  - Ghi nhận thông tin truy xuất đến từng trang vào CSDL phục vụ cho việc tra cứu, tổng hợp thống kê
- **AuthFilter** hạn chế quyền truy cập
  - Kiểm tra session để biết được quyền truy cập của user hiện tại
- **RRSharer** chia sẻ request và response theo Thread ID
  - Sử dụng static Map<Long, HttpServletRequest> với key là ID của Thread hiện tại. Bạn có thể tham chiếu đến HttpServletRequest hiện tại bất kỳ đâu trong ứng dụng, từ đó xây dựng thư viện tiện ích xử lý cookie, parameter… một cách tiện lợi

LoggerFilter – Ghi nhận lịch sử truy xuất

```
@WebFilter("/admin/*")
public class LoggerFilter implements HttpFilter{
        @Override
        public void doFilter(HttpServletRequest req, HttpServletResponse resp,
                        FilterChain chain) throws IOException, ServletException {
                User user = (User) req.getSession().getAttribute("user");
                String uri = req.getRequestURI();
                Date time = new Date();
                // ghi nhận user, uri, time vào CSDL hoặc file
                chain.doFilter(req, resp);
        }
}
```

AUTHFILTER – BẢO MẬT CHỨC NĂNG



B1: Request có URL được bảo mật

B2: Kiểm tra đăng nhập?

- B2.1: Chưa -> Đăng nhập (quay trở lại URL trước đó)
- B2.2: Rồi -> B3

B3: Truy xuất trang quản trị?

- B3.1: Không -> Secured Servlet
- B3.2: Đúng -> B4

B4: Kiểm tra vai trò?

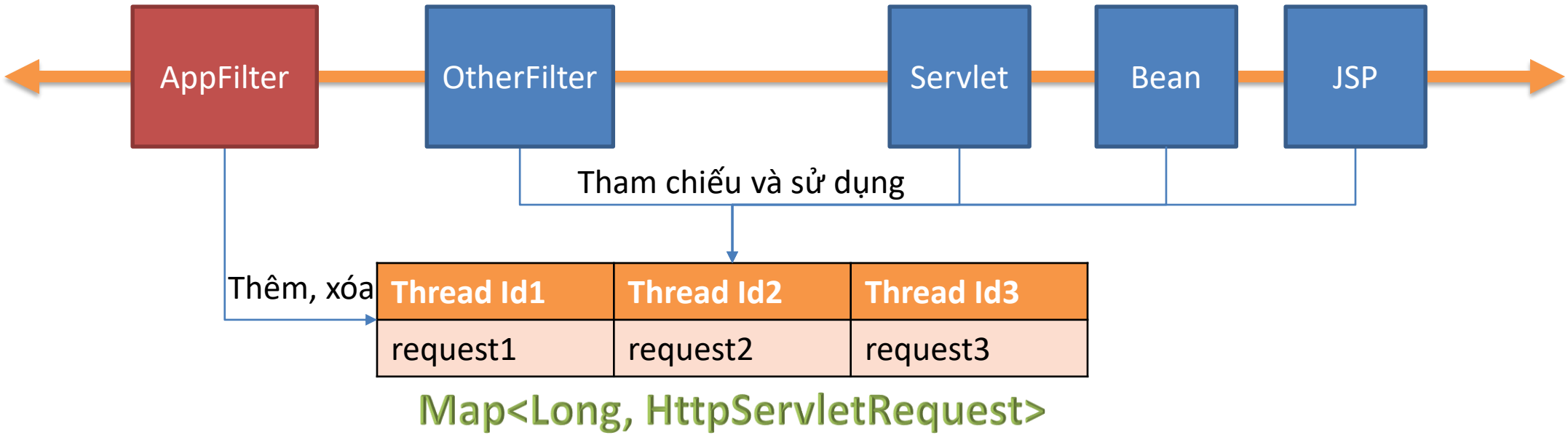
- B4.1: Admin -> Secured Servlet
- B4.2: Không -> Đăng nhập

AUTHFILTER – BẢO MẬT CHỨC NĂNG

```
@WebFilter({"/video/like/*", "/video/share/*", "/account/change-password", "/admin/*"})
public class AuthFilter implements HttpFilter{
        @Override
        public void doFilter(HttpServletRequest req, HttpServletResponse resp, FilterChain chain)
                        throws IOException, ServletException {
                String uri = req.getRequestURI();
                User user = (User) req.getSession().getAttribute("user");
                String error = "";
                if(user == null) { // chưa đăng nhập
                        error = resp.encodeURL("Vui lòng đăng nhập!");
                } else if(!user.getAdmin() && uri.contains("/admin/")) { // không phải admin
                        error = resp.encodeURL("Vui lòng đăng nhập với vai trò admin!");
                }
                if(!error.isEmpty()) { // truy cập không hợp lệ
                        req.getSession().setAttribute("securi", uri);
                        resp.sendRedirect("/fpoly/account/sign-in?error=" + resp.encodeURL(error));
                } else { // truy cập hợp lệ
                        chain.doFilter(req, resp);
                }
        }
}
```

CHIA SẺ REQUEST VÀ RESPONSE

- Trong lập trình Servlet, request và response đóng vai trò rất quan trọng. Nếu có request và response, chúng ta dễ dàng làm việc được với Parameter, Cookie, Session, Application…
- *Chú ý: tất cả những hoạt động trong một yêu cầu từ người sử dụng được thực hiện trong cùng một Thread vì vậy chúng ta có thể chia sẻ request và response theo mã của Thread này.*



RRSharer (Request Response Sharer)

```
public class RRSharer {
        private static Map<Long, HttpServletRequest> reqs = new HashMap<>();
        private static Map<Long, HttpServletResponse> resps = new HashMap<>();
        public static void add(HttpServletRequest req, HttpServletResponse resp) {
                reqs.put(Thread.currentThread().getId(), req);
                resps.put(Thread.currentThread().getId(), resp);
        }
        public static void remove() {
                reqs.remove(Thread.currentThread().getId());
                resps.remove(Thread.currentThread().getId());
        }
        public static HttpServletRequest request() {
                return reqs.get(Thread.currentThread().getId());
        }
        public static HttpServletResponse response() {
                return resps.get(Thread.currentThread().getId());
        }
}
```

SỬ DỤNG RRSHARER

```
@WebFilter("/*")
public class AppFilter implements HttpFilter{
        @Override
        public void doFilter(HttpServletRequest req, HttpServletResponse resp,
                        FilterChain chain) throws IOException, ServletException {
                req.setCharacterEncoding("utf-8");
                resp.setCharacterEncoding("utf-8");
                RRSharer.add(req, resp);
                chain.doFilter(req, resp);
                RRSharer.remove();
        }
}
```

- RRSharer.add(): Bổ sung req và resp vào nơi chia sẻ trước khi đến servlet
- RRSharer.remove(): Xóa req, resp khỏi nơi chia sẻ sau khi rời servlet
- Hãy sử dụng RRSharer.request() và RRSharer.response() để tham chiếu đến request và response đang hoạt động trong thread hiện tại khi muốn làm việc

SUMMARY

- ☑ Scopes: request, session và application
- ☑ Scope API: setAttribute(), getAttribute() và removeAttribute
- ☑ @WebListener
  - ☑ HttpSessionListener
  - ☑ ServletContextListener
- ☑ @WebFilter
  - ☑ urlParttern
  - ☑ dispatcherType
- ☑ Ứng dụng: Logger, Utf8, Security, Share Request và Response theo Thread Id


TIN HỌC VĂN PHÒNG QUỐC TẾ THEO CHUẨN ICDL
Syllabus 5 - Office 2007
LUẬT KINH TẾ
LUẬT KINH TẾ
KỸ THUẬT SOẠN THẢO VÀ TRÌNH BÀY VĂN BẢN
TIN HỌC VĂN PHÒNG QUỐC TẾ THEO CHUẨN ICDL
Syllabus 5 - Office 2007
Cảm ơn