



## LẬP TRÌNH JAVA 5

### BÀI 8: VALIDATION & INTERCEPTOR

- ⊙ Hiểu được tầm quan trọng của validation
- ⊙ Triển khai validation trong Spring MVC
- ⊙ Hiểu cơ chế hoạt động của Interceptor
- ⊙ Tạo và sử dụng Interceptor
- ⊙ Ứng dụng Interceptor để bảo vệ tài nguyên riêng tư



- ❑ Dữ liệu vào không hợp lệ sẽ gây các lỗi khó lường. Vì vậy việc kiểm soát dữ liệu vào luôn đóng vai trò quan trọng của ứng dụng.
- ❑ Các lỗi thường gặp
  - ❖ Để trống ô nhập...
  - ❖ Không đúng định dạng email, creditcard, url...
  - ❖ Sai kiểu số nguyên, số thực, ngày giờ...
  - ❖ Giá trị tối thiểu, tối đa, trong phạm vi...
  - ❖ Không giống mật khẩu, đúng captcha, trùng mã
  - ❖ Không như mong đợi của việc tính toán nào đó...

## ➔ ĐĂNG KÝ THÀNH VIÊN

Tên đăng nhập : nnghiem 🔥 Mã này đã được sử dụng.

Mật khẩu : 🔥 Trường bắt buộc.

Nhập lại mật khẩu mới : abc 🔥 Giá trị nhập không giống.

Họ và tên : 🔥 Trường bắt buộc.

Giới tính : ☒ Nam ☐ Nữ

Thư điện tử : abc 🔥 Không đúng dạng email.

Điện thoại di động :

Ngày sinh : 🔥 Trường bắt buộc.

Địa chỉ :

Hình ảnh : C:\NetworkCfg.xml  🔥 Không chấp nhận loại tập tin này.

Mã bảo mật : AS 🔥 Sai mã bảo mật. **AEBG44**

Đăng ký

## ❑ Kiểm các lỗi cho form

- ❖ Không để trống họ và tên
- ❖ Không để trống điểm
- ❖ Điểm phải có giá trị từ 0 đến 10
- ❖ Phải chọn ngành

❖ Không để trống họ và tên	❖ Không để trống điểm	❖ Điểm phải có giá trị từ 0 đến 10	❖ Phải chọn ngành
<p>Quản lý sinh viên</p> <p>Vui lòng sửa các lỗi sau đây !</p> <p>Họ và tên</p> <input type="text"/> <p>Điểm</p> <input type="text"/> <p>Chuyên ngành</p> <p>Vui lòng chọn chuyên ngành !</p> <p><input type="radio"/> Ứng dụng phần mềm <input type="radio"/> Thiết kế trang web</p> <p>Update</p>	<p>Quản lý sinh viên</p> <p>Chúc mừng, bạn đã nhập đúng !</p> <p>Họ và tên</p> <input type="text"/> <p>Điểm</p> <input type="text"/> <p>Chuyên ngành</p> <p><input checked="" type="radio"/> Ứng dụng phần mềm <input type="radio"/> Thiết kế trang web</p> <p>Update</p>		

**Nhập sai**

**Nhập đúng**



# DEMO

Chạy student/validate1.htm  
+ Nhập dữ liệu không hợp lệ  
+ Nhập dữ liệu hợp lệ



```
@RequestMapping(value="validate1", method=RequestMethod.POST)
```

```
public String validate1(ModelMap model,
```

```
    @ModelAttribute("student") Student1 student, BindingResult errors) {
```

```
    if(student.getName().trim().length() == 0){
```

```
        errors.rejectValue("name", "student", "Vui lòng nhập họ tên !");
```

```
    }
```

```
    if(student.getMark() == null){
```

```
        errors.rejectValue("mark", "student", "Vui lòng nhập điểm !");
```

```
    }
```

```
    else if(student.getMark() < 0 || student.getMark() > 10){
```

```
        errors.rejectValue("mark", "student", "Điểm không hợp lệ !");
```

```
    }
```

```
    if(student.getMajor() == null){
```

```
        errors.rejectValue("major", "student", "Vui lòng chọn một chuyên ngành !");
```

```
    }
```

```
    if(errors.hasErrors()){
```

```
        model.addAttribute("message", "Vui lòng sửa các lỗi sau đây !");
```

```
    }
```

```
    else{
```

```
        model.addAttribute("message", "Chúc mừng, bạn đã nhập đúng !");
```

```
    }
```

```
    return "student1";
```

```
}
```

Đối số này nên là  
đối số cuối cùng

**rejectValue()** cho  
phép bổ sung  
thông báo lỗi cho  
thuộc tính **mark**  
của bean **student**

Phương thức  
**hasErrors()** cho  
biết có thông báo  
lỗi nào hay không?

```
{message}
<form:form action="student/validate1.htm" modelAttribute="student">
    <div>Họ và tên</div>
    <form:input path="name"/>
    <form:errors path="name"/>
    <div>Điểm</div>
    <form:input path="mark"/>
    <form:errors path="mark"/>
    <div>Chuyên ngành</div>
    <form:errors path="major" element="div"/>
    <form:radiobuttons path="major" items="{majors}"
        itemLabel="name" itemValue="id"/>
    <div>
        <button>Validate 1</button>
    </div>
</form:form>
```

Hiển thị lỗi thuộc tính **name** của bean student

Thuộc tính **element** chỉ ra thẻ chứa thông báo lỗi. Mặc định là <span>



❑ Thông báo lỗi sinh ra có dạng

❖ `<tag id="{thuộc tính}.errors">{thông báo lỗi}</tag>`

❑ Ví dụ

❖ `<span id="name.errors">...</span>`

*Hiển thị lỗi cho thuộc tính họ tên*

❑ CSS sau được sử dụng để định dạng cho các thông báo lỗi

```
*[id$=.errors]{  
    color:red; font-style: italic;  
}
```

❑ Selector `*[id$=.errors]` sẽ chọn ra các thẻ có thuộc tính `@id` kết thúc bởi chữ `".errors"`



# DEMO

Giải thích student/validate1.htm

+ Student1

+ Controller.validate1()

+ student1.jsp



- ❑ Nạp các annotation lỗi vào các thuộc tính lớp bean được sử dụng để nhận dữ liệu form

```
public class Student2 {  
    @NotBlank(message="Không được để trống họ và tên !")  
    private String name;  
  
    @NotNull(message="Không được để trống điểm !")  
    @DecimalMin(value="0", message="Điểm không hợp lệ !")  
    @DecimalMax(value="10", message="Điểm không hợp lệ !")  
    private Double mark;  
  
    @NotNull(message="Vui lòng chọn chuyên ngành !")  
    private String major;  
  
    getters/setters  
}
```

- ❑ Annotation kiểm lỗi

- ❖ @NotBlank: kiểm trường name rỗng
- ❖ @NotNull: kiểm trường mark và major null
- ❖ @DecimalMin(), @DecimalMax(): kiểm khoảng số thực

```
@RequestMapping(value="validate2", method=RequestMethod.POST)  
public String validate2(ModelMap model,  
    @Validated @ModelAttribute("student") Student2 student, BindingResult errors) {  
    if(errors.hasErrors()){  
        model.addAttribute("message", "Vui lòng sửa các lỗi sau đây !");  
    }  
    else{  
        model.addAttribute("message", "Chúc mừng, bạn đã nhập đúng !");  
    }  
    return "student2";  
}
```

- ❑ Chỉ cần bổ sung **@Validated** trước bean nhận dữ liệu form thì các thuộc tính của bean sẽ được kiểm lỗi theo các luật đã nạp vào các trường bean

❑ `<form:errors path="*" />` được sử dụng để hiển thị tất cả các lỗi

```
{message}
```

```
<form:form action="student/validate2.htm" modelAttribute="student">
```

```
<!-- Hiển thị lỗi tập trung -->
```

```
<form:errors path="*" element="ul"/>
```

```
<div>Họ và tên</div>
```

```
<form:input path="name"/>
```

```
<div>Điểm</div>
```

```
<form:input path="mark"/>
```

```
<div>Chuyên ngành</div>
```

```
<form:radiobuttons path="major" items="{m  
    itemLabel="name" itemValue="id"/>
```

```
<div>
```

```
<button>Validate 2</button>
```

```
</div>
```

```
</form:form>
```

## Quản lý sinh viên

Vui lòng sửa các lỗi sau đây !

*Không được để trống điểm !*

*Không được để trống họ và tên !*

*Vui lòng chọn chuyên ngành !*

Họ và tên

Điểm

Chuyên ngành

☐ Ứng dụng phần mềm ☐ Thiết kế trang web

Validate 2



# DEMO

Giải thích student/validate2.htm

+ Student2

+ Controller.validate2()

+ student2.jsp



- ❑ Để có thể sử dụng phương pháp kiểm lỗi bằng annotation này cần bổ sung các thư viện sau
- ❖ validation-api-1.0.0.GA.jar
  - ❖ hibernate-validator-4.2.0.Final.jar
  - ❖ log4j-1.2.16.jar
  - ❖ slf4j-api-1.7.5.jar
  - ❖ slf4j-log4j12-1.7.5.jar
  - ❖ slf4j-simple-1.6.1.jar

Annotation	Ý nghĩa	Ví dụ
NotBlank	Chuỗi không rỗng	@NotBlank()
NotNull	Không cho phép null	@NotNull()
NotEmpty	Chuỗi /tập hợp không rỗng	@NotEmpty()
Length	Độ dài chuỗi	@Length(min=5, max=10)
Max	Giá trị số nguyên tối đa	@Max(value="10")
Min	Giá trị số nguyên tối thiểu	@Min(value="0")
Size, Range	Phạm vi số nguyên tối	@Size(min=0, max=10)
DecimalMax	Giá trị số thực tối đa	@DecimalMin(value="5.5")
DecimalMin	Giá trị số nguyên tối thiểu	@DecimalMax(value="9.5")
Future	Thời gian trong tương lai	@Future()
Past	Thời gian trong quá khứ	@Past()
Pattern	So khớp biểu thức chính qui	@Pattern(regexp="[0-9]{9,10}")
Email	Đúng dạng email	@Email()
CreditCardNumber	Đúng dạng số thẻ tín dụng	@CreditCardNumber()
URL	Đúng dạng url	@URL()
SafeHtml	Không được chứa thẻ HTML	@SafeHtml()



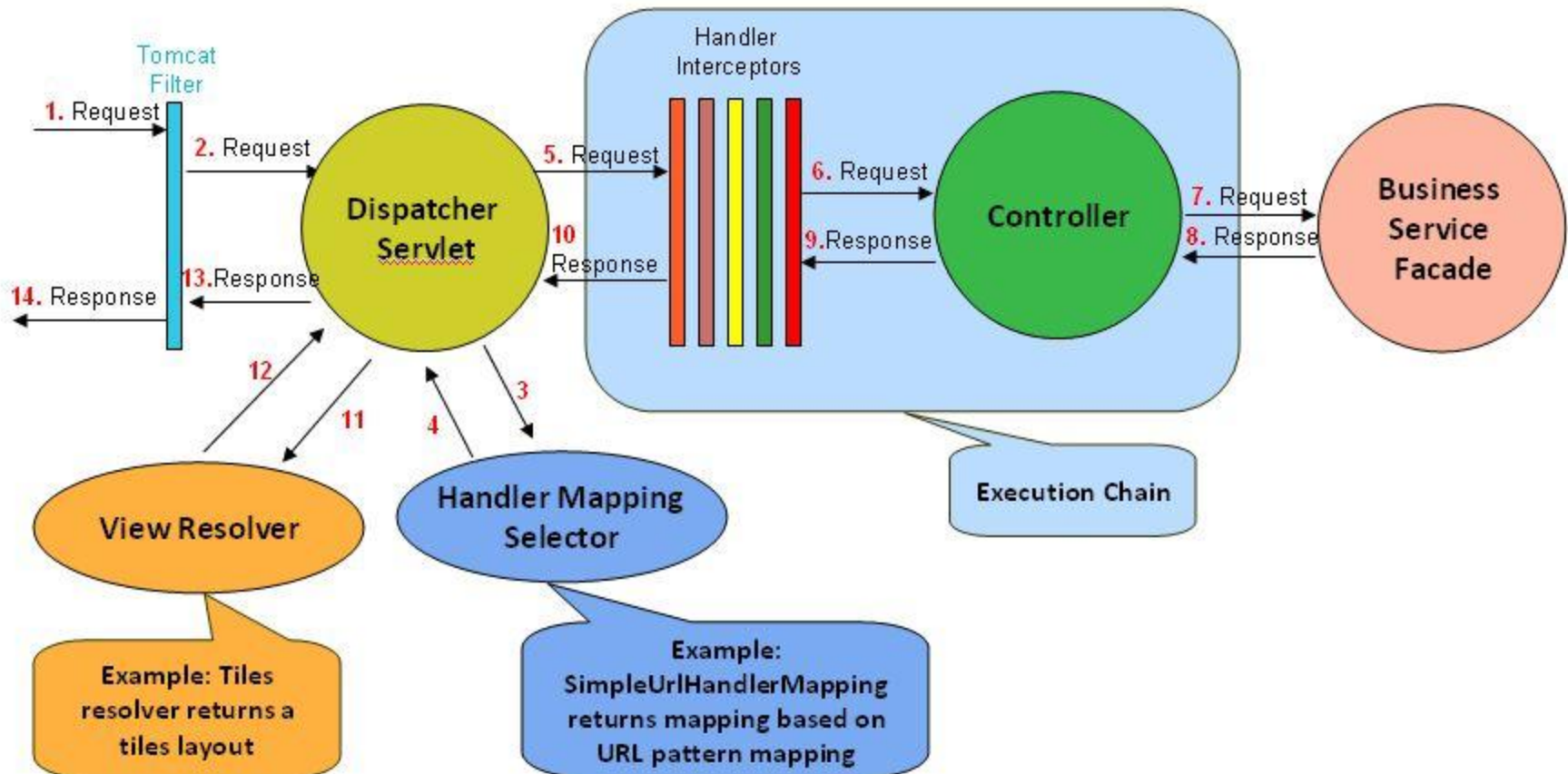


# LẬP TRÌNH JAVA 5

---

## PHẦN 2

- ❑ Interceptor là một thành phần có nhiệm vụ tiền và hậu xử lý các request đến phương thức action



# CẤU TRÚC CỦA INTERCEPTOR

Kế thừa lớp **HandlerInterceptorAdapter**

```
public class MyInterceptor extends HandlerInterceptorAdapter{
```

```
@Override
```

```
public boolean preHandle(HttpServletRequest request,  
    HttpServletResponse response, Object handler) throws Exception {  
    return true;  
}
```

Chạy **TRƯỚC** phương thức action

**false** sẽ không chuyển yêu cầu đến action

```
@Override
```

```
public void postHandle(HttpServletRequest request,  
    HttpServletResponse response, Object handler,  
    ModelAndView modelAndView) throws Exception {  
}
```

Chạy **SAU** phương thức action, **TRƯỚC** view

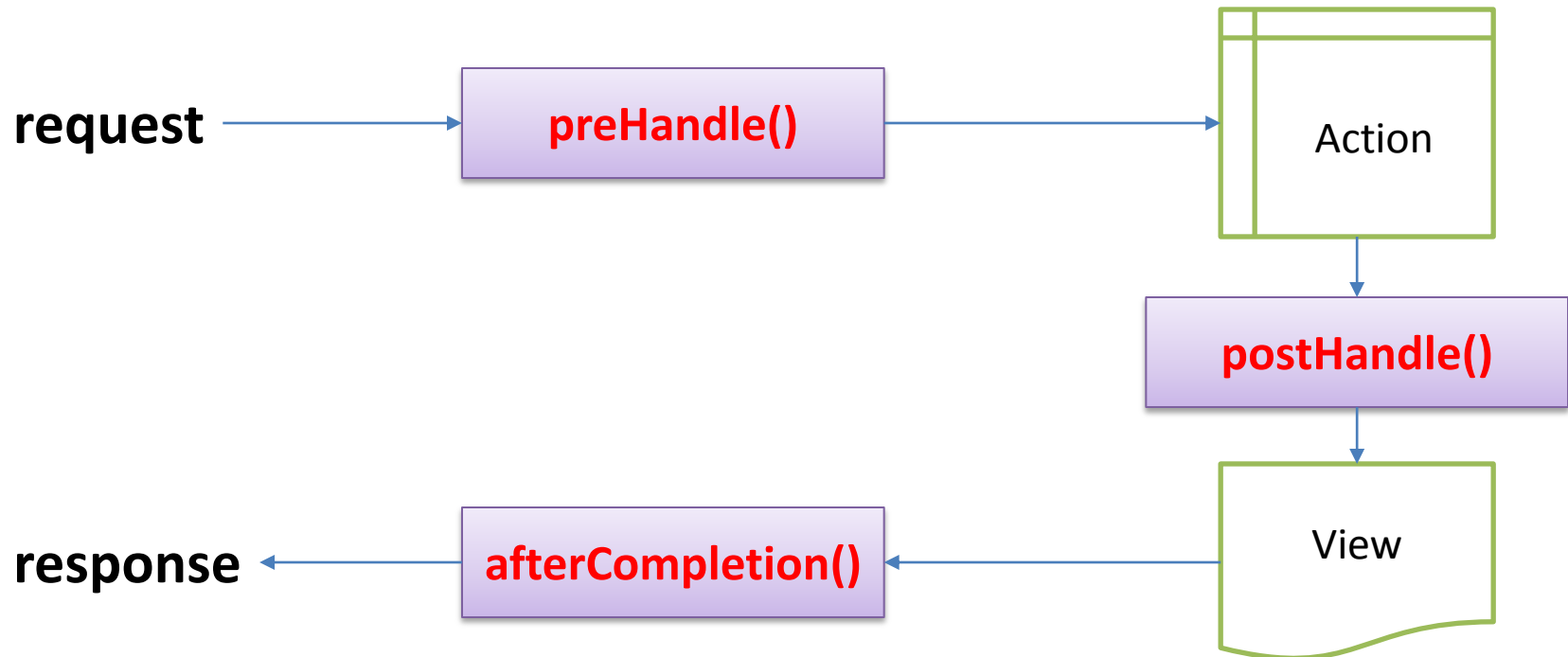
```
@Override
```

```
public void afterCompletion(HttpServletRequest request,  
    HttpServletResponse response, Object handler, Exception ex)  
    throws Exception {  
}
```

Chạy **SAU** view

```
}
```

# QUI TRÌNH XỬ LÝ CỦA INTERCEPTOR



- ❑ Nếu chúng ta muốn xử lý một công việc nào đó trước khi action thực thi thì phải viết mã ở **preHandle**.
- ❑ Nếu chúng ta muốn chuẩn bị một điều gì đó cho View thì có thể viết mã ở **postHandle()**

```
public class LoggerInterceptor extends HandlerInterceptorAdapter{
    @Override
    public boolean preHandle(HttpServletRequest request,
        HttpServletResponse response, Object handler) throws Exception {
        System.out.println("LoggerInterceptor.preHandle()");
        return true;
    }

    @Override
    public void postHandle(HttpServletRequest request, HttpServletResponse response,
        Object handler, ModelAndView modelAndView) throws Exception {
        System.out.println("LoggerInterceptor.postHandle()");
    }

    @Override
    public void afterCompletion(HttpServletRequest request,
        HttpServletResponse response, Object handler, Exception ex) throws Exception {
        System.out.println("LoggerInterceptor.afterCompletion()");
    }
}
```

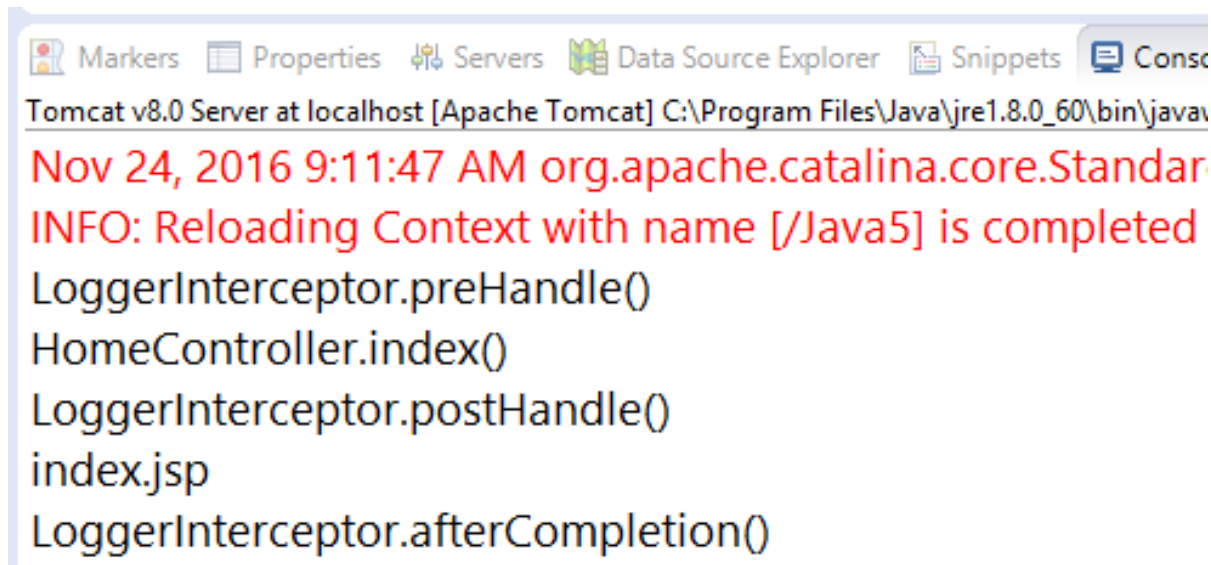
- ❑ Interceptor sau khi đã xây dựng xong cần phải khai báo với hệ thống Spring để lọc các action
- ❑ Khai báo sau đây LoggerInterceptor sẽ lọc tất cả mọi action

```
<!-- Khai báo Interceptor -->  
<mvc:interceptors>  
    <!-- Lọc tất cả các action -->  
    <bean class="poly.interceptor.LoggerInterceptor" />  
</mvc:interceptors>
```

```
@Controller
@RequestMapping("/home/")
public class HomeController{
    @RequestMapping("index")
    public String index() {
        System.out.println("HomeController.index()");
        return "home/index";
    }
    @RequestMapping("about")
    public String about() {
        System.out.println("HomeController.about()");
        return "home/index";
    }
    @RequestMapping("contact")
    public String contact() {
        System.out.println("HomeController.contact()");
        return "home/index";
    }
}
```

```
<%@ page pageEncoding="utf-8"%>
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <title>Interceptor</title>
</head>
<body>
    <h1>Interceptor</h1>
    <%
        System.out.println("index.jsp");
    %>
</body>
</html>
```

- ❑ Chạy home/index.htm và xem kết xuất từ Console



```
Tomcat v8.0 Server at localhost [Apache Tomcat] C:\Program Files\Java\jre1.8.0_60\bin\javaw.exe
Nov 24, 2016 9:11:47 AM org.apache.catalina.core.StandardContext reload
INFO: Reloading Context with name [/Java5] is completed
LoggerInterceptor.preHandle()
HomeController.index()
LoggerInterceptor.postHandle()
index.jsp
LoggerInterceptor.afterCompletion()
```

- ❑ Qua kết quả chúng ta thấy thứ tự thực hiện  
**preHandle()**=>**index()**=>**postHandle()**=>**index.jsp**=>**afterCompletion()**  
Interceptor → Action → Interceptor → View → Interceptor





# DEMO

Chạy `home/index.htm` và giải thích  
rõ hoạt động của `LoggerInterceptor`



- ❑ Đôi khi Interceptor được xây dựng ra chỉ để lọc một số action chứ không phải lọc tất cả các action
- ❑ Cấu hình sau chỉ cho phép LoggerInterceptor lọc action home/index.htm và home/about.htm

```
<!-- Cấu hình interceptor -->
<mvc:interceptors>
  <!-- Chỉ lọc các URL theo path -->
  <mvc:interceptor>
    <mvc:mapping path="/home/index.htm"/>
    <mvc:mapping path="/home/about.htm"/>
    <bean class="poly.interceptor.LoggerInterceptor" />
  </mvc:interceptor>
</mvc:interceptors>
```

- ❑ Một tình huống khác là chúng ta muốn lọc tất cả các action trong HomeController chỉ loại trừ home/index.htm

```
<!-- Cấu hình interceptor -->
<mvc:interceptors>
  <!-- Chỉ lọc các URL theo path -->
  <mvc:interceptor>
    <mvc:mapping path="/home/**"/>
    <mvc:exclude-mapping path="/home/index.htm"/>
    <bean class="poly.interceptor.LoggerInterceptor"/>
  </mvc:interceptor>
</mvc:interceptors>
```

- ❑ Ở đây chúng ta thấy <mvc:exclude-mapping> được sử dụng để loại trừ các action không muốn lọc còn \*\* là ký hiệu đại diện cho nhóm ký tự bất kỳ

- ❑ Các action màu vàng của 2 controller sau chỉ được phép truy cập sau khi đã đăng nhập

```
@Controller
@RequestMapping("/user/")
public class UserController{
    @RequestMapping("login")
    public String login() {...}
    @RequestMapping("logoff")
    public String logoff() {...}
    @RequestMapping("register")
    public String register() {...}
    @RequestMapping("activate")
    public String activate() {...}
    @RequestMapping("forgot-password")
    public String forgot() {...}
    @RequestMapping("change-password")
    public String change() {...}
    @RequestMapping("edit-profile")
    public String edit() {...}
}
```

```
@Controller
@RequestMapping("/order/")
public class OrderController{
    @RequestMapping("checkout")
    public String checkout() {...}
    @RequestMapping("list")
    public String list() {...}
    @RequestMapping("detail")
    public String detail() {...}
}
```

- ❑ Xây dựng SecurityInterceptor lọc tất cả các action của 2 controller trên loại trừ các action không tô màu vàng.
- ❑ SecurityInterceptor phải chạy trước khi request đến action và sẽ thực hiện công việc:
  - ❖ Kiểm tra xem trong session có attribute có tên là user hay chưa? Nếu chưa có thì chuyển hướng sang user/login.htm
  - ❖ Ở user/login.htm sau khi đăng nhập thành công cần tạo một attribute user trong session

# XÂY DỰNG SECURITYINTERCEPTOR

```

public class SecurityInterceptor extends HandlerInterceptorAdapter{
    @Override
    public boolean preHandle(HttpServletRequest request,
        HttpServletResponse response, Object handler) throws Exception {
        HttpSession session = request.getSession();
        if(session.getAttribute("user") == null){
            response.sendRedirect(request.getContextPath() + "/user/login.htm");
            return false;
        }
        return true;
    }
}

```

```

<mvc:interceptors>
    <mvc:interceptor>
        <mvc:mapping path="/user/**"/>
        <mvc:mapping path="/order/**"/>
        <mvc:exclude-mapping path="/user/login.htm"/>
        <mvc:exclude-mapping path="/user/register.htm"/>
        <mvc:exclude-mapping path="/user/forgot-password.htm"/>
        <mvc:exclude-mapping path="/user/activate.htm"/>
        <bean class="poly.interceptor.SecurityInterceptor"/>
    </mvc:interceptor>
</mvc:interceptors>

```



# DEMO

Chưa đăng nhập:

+ Chạy user/register.htm

+ Chạy user/change-password.htm

Đã đăng nhập

+ Chạy user/change-password.htm

Giải thích





**FPT POLYTECHNIC**  
THỰC HỌC - THỰC NGHIỆP

ĐĂNG NHẬP ĐĂNG KÝ QUÊN MẬT KHẨU

TRANG CHỦ GIỚI THIỆU LIÊN HỆ GÓP Ý HỎI - ĐÁP SẢN PHẨM ENGLISH TIẾNG VIỆT

Module giao diện này thuộc về layout nhưng lại cần nạp dữ liệu từ CSDL.

Vấn đề: Viết mã ở đâu để cấp dữ liệu cho module này?

1. Viết trong mọi action của mọi controller và view nào cũng cần dữ liệu này
2. Viết trong postHandle() của Interceptor lọc tất cả mọi action. Rõ ràng phương án này rất tối ưu

5 MẶT HÀNG  
1.150.000 VNĐ  
XEM GIỎ HÀNG

- MÁY TÍNH KHÁCH TAY
- ĐIỆN THOẠI DI ĐỘNG
- MÁY CHỤP ẢNH
- TÚI XÁCH DU LỊCH
- NỮ TRANG
- NÓN THỜI TRANG
- NƯỚC HOA
- QUẠT MÂY



- ❑ Tìm hiểu tầm quan trọng của Validation
- ❑ Thực hiện kiểm lỗi bằng tay
- ❑ Thực hiện kiểm lỗi bằng annotation
- ❑ Tìm hiểu Interceptor
- ❑ Xây dựng Interceptor
- ❑ Cấu hình Interceptor để lọc action
- ❑ Ứng dụng Interceptor để bảo vệ chức năng riêng tư.





**Cảm ơn**