# Mục tiêu:

Kết thúc bài thực hành này bạn có khả năng

* Nắm vững kỹ thuật kiểm lỗi bằng tay, bằng annotation
* Biết cách tạo và sử dụng Interceptor
* Ứng dụng Interceptor để bảo vệ tài nguyên riêng tư của người dùng

# PHẦN I: VALIDATION

Yêu cầu kiểm lỗi form sau đây



Các lỗi cần kiểm

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

Trong phần 1 bạn có 2 bài tập thực hiện 2 phương pháp kiểm lỗi khác nhau

## Bài 1 (2 điểm): Kiểm lỗi bằng tay

Hãy viết mã để kiểm các lỗi theo yêu cầu như mô tả ở trên.

Bước 1: Chuẩn bị

Bạn cần phải chuẩn bị lớp bean, view và controller cho bài tập này.

* Tạo lớp bean Student1 buộc dữ liệu với form

|  |
| --- |
| **package** poly.bean;  **public** **class** Student1 {  **private** String name;  **private** Double mark;  **private** String major;    **getters/setters**  } |

* Buộc bean với form

|  |
| --- |
| <form:form action=*"validation/validate1.htm"* modelAttribute=*"student"*>  <div>Họ và tên</div>  <form:input path=*"name"*/>    <div>Điểm</div>  <form:input path=*"mark"*/>    <div>Chuyên ngành</div>  <form:radiobutton path=*"major"* value=*"APP"* label=*"Ứng dụng phần mềm"*/>  <form:radiobutton path=*"major"* value=*"WEB"* label=*"Thiết kế trang web"* />    <div>  <button>Validate 1</button>  </div>  </form:form> |

* ValidationController

|  |
| --- |
| @Controller  @RequestMapping("/validation/")  **public** **class** ValidationController {    @RequestMapping("form1")  **public** String form1(ModelMap model) {  model.addAttribute("student", **new** Student1());  **return** "validation/form";  }    @RequestMapping(value="validate1", method=RequestMethod.***POST***)  **public** String validate1(ModelMap model,  @ModelAttribute("student") Student1 student) {  **return** "validation/form";  }  } |

Bước 2: Bổ sung mã kiểm lỗi cho action validate1()

* Thêm đối số BindingResult errors vào validate1()

|  |
| --- |
| @RequestMapping(value="validate1", method=RequestMethod.***POST***)  **public** String validate1(ModelMap model,  @ModelAttribute("student") Student1 student, BindingResult errors) |

* Viết mã kiểm lỗi cho validate()

|  |
| --- |
| **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 !");  } |

Chú ý:

* + Phương thức errors.rejectValue(prop, bean, message) được sử dụng để bổ sung một thông báo lỗi
    - Prop: tên thuộc tính có lỗi
    - Bean: tên bean (trong model) chứa thuộc tính
    - Message: thông báo lỗi
  + Phương thức errors.hasErrors() được sử dụng để kiểm tra xem có lỗi hay không

Bước 3: Hiển thị lỗi

* Bổ sung thông báo lỗi vào giao diện của view bằng cách đặt <form:errors path="prop"/> vào vị trí muốn hiển thị lỗi của thuộc tính có tên prop. Ví dụ sau hiển thị lỗi cho thuộc tính mark.

|  |
| --- |
| <form:form path=*"mark"*/> <form:errors path=*"mark"*/> |

* Để định dạng thông báo lỗi bạn cần hiệu chỉnh CSS sau

|  |
| --- |
| <style type=*"text/css"*>  \*[id$=errors]{  color:*red*;  font-style: *italic*;  }  </style> |

* Đừng quên ${message} để hiển thị thông báo

Bước 4: Chạy validation/form1.htm sau đó

* Không nhập gì và nhấp nút
* Điểm ngoài phạm vi 0..10 và nhấp nút
* Nhập đúng và nhấp nút

## Bài 2 (2 điểm): Kiểm lỗi bằng annotation

Yêu cầu kiểm lỗi không thay đổi so với bài 1 nhưng bạn phải sử dụng phương pháp annotation để kiểm lỗi các thuộc tính của bean. Thực hiện theo hướng dẫn sau:

Bước 0: Bổ sung thư viện cần thiết

* 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

Bước 1: Chuẩn bị

* Tạo lớp lean Student**2** với nội dung giống Student1
* Buộc bean form form tương tự như trên chỉ đổi thuộc tính action=”validation/validate**2**.htm”
* Bổ sung vào ValidationController hai action là form**2**() và validate**2**() tương tự như chuẩn bị của bài 1 chỉ khác
  + Form2() ánh xạ “form**2**”
  + Validate2() ánh xạ “validate**2**”

Bước 2: Bổ sung annotation kiểm lỗi vào Student**2**

|  |
| --- |
| **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**  } |

Chú ý: đọc thông báo lỗi để hiểu vai trò của các annotation

Bước 3: Hiệu chỉnh validate2()

|  |
| --- |
| @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";  } |

Chú ý: @Validated được thêm trước đối số student sẽ làm cho các annotation kiểm lỗi được đính sẵn trong lớp Student2 hoạt động. Tất cả các thông báo lỗi được bổ sung một cách tự động vào errors.

Bước 4: Hoàn toàn tương tự bước 3 của bài 1. Bạn chép giao diện đã hoàn thành trong bài 1 và sửa action=”validation/validate2.htm” là được.

Bước 5: Chạy và kiểm thử validation/form2.htm tương tự như bước 4 của bài 1

# PHẦN II: INTERCEPTOR

## Bài 3 (2 điểm): Tạo và sử dụng Interceptor

Để hiểu cơ chế hoạt động của Interceptor và biết cách cấu hình để Interceptor hoạt động trong bài này yêu cầu bạn xây dựng LoggerInterceptor để ghi nhận thời điểm thực hiện của các hoạt động trong hệ thống.

Bước 1: Xây dựng LoggerInterceptor

|  |
| --- |
|  |

Lưu ý: Interceptor kế thừa từ HandlerInterceptorAdapter và có thể override 3 phương thức preHande(), postHandle() và afterCompletion()

Bước 2: Khai báo Interceptor trong file cấu hình

|  |
| --- |
| <mvc:interceptors>  <bean class=*"poly.interceptor.LoggerInterceptor"* />  </mvc:interceptors> |

Với khai báo này thì khi người sử dụng truy cập đến bất kỳ action nào trong hệ thống thì Interceptor cũng hoạt động

Bước 3: Tạo Controller và view

Controller: HomeController

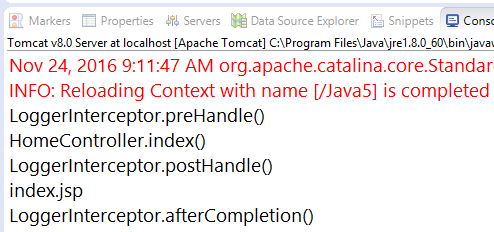
|  |
| --- |
| @Controller  @RequestMapping("/home/")  **public** **class** HomeController {  @RequestMapping("index")  **public** String index() {  System.***out***.println("HomeController.index()");  **return** "home/index";  }  @RequestMapping("about")  **public** String about() {  System.***out***.println("HomeController.about()");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

View: index.jsp, about.jsp, contact.jsp

|  |
| --- |
| <h1>Interceptor</h1>  <%  System.out.println("index.jsp");  %> |

Bước 4: Chạy home/index.htm và phân tích kết quả

Xem kết xuất từ tab Console bạn sẽ nhìn thấy như hình sau



Điều cần lưu ý duy nhất ở kết quả này là thứ tự thực hiện của các công việc trong hệ thống:

1. preHandle()
2. index()
3. postHandle()
4. index.jsp
5. afterCompletion()

Bước 5: Cấu hình Interceptor để lọc một vài action

Với cấu hình như trên thì LoggerInterceptor sẽ lọc tất cả các action. Bạn có thể chỉ lọc một số action nào đó. Ví dụ sau sẽ cấu hình LoggerInterceptor lọc home/index.htm và home/about.htm

|  |
| --- |
| <mvc:interceptors>  <mvc:interceptor>  <mvc:mapping path=*"/home/index.htm"*/>  <mvc:mapping path=*"/home/about.htm"*/>  <bean class=*"poly.interceptor.LoggerInterceptor"* />  </mvc:interceptor>  </mvc:interceptors> |

Đôi khi bạn muốn lọc tất cả mọi action trong HomeController ngoại trừ home/contact.htm là bạn không muốn lọc thì bạn cần khai báo như sau

|  |
| --- |
| <mvc:interceptors>  <mvc:interceptor>  <mvc:mapping path=*"/home/****\*\*****"*/>  <mvc:exclude-mapping path=*"/home/contact.htm"*/>  <bean class=*"poly.interceptor.LoggerInterceptor"* />  </mvc:interceptor>  </mvc:interceptors> |

Chú ý: dấu \*\* là đại diện cho action bất kỳ

Bước 6: Chạy

* home/index.htm bạn sẽ thấy LoggerInterceptor hoạt động
* home/contact.htm bạn sẽ thấy LoggerInterceptor không hoạt động

## Bài 4 (2 điểm): Bảo vệ tài nguyên riêng tư

Giả sử trong ứng dụng web của bạn có một số action chỉ được phép truy cập sau khi đã đăng nhập. Cụ thể trong website bán hàng có các chức năng sau đây (có highlight) chỉ được truy cập sau khi đã đăng nhập

|  |  |
| --- | --- |
|  |  |

Nhiệm vụ của bạn là viết AuthorizeInterceptor để ngăn chặn việc truy cập trái phép đến các action này.

Hướng dẫn:

1. Tạo lớp kế thừa từ HandlerInterceptorAdapter
2. Override preHandle()
3. Viết mã cho preHandle() như sau

|  |
| --- |
| HttpSession session = request.getSession();  **if**(session.getAttribute("user") == **null**){  response.sendRedirect(request.getContextPath() + "/user/login.htm");  **return** **false**;  }  **return** **true**; |

Đoạn mã này làm công việc: chuyển về trang đăng nhập nếu trong session không có attribute user (ý muốn nói là chưa đăng nhập)

1. Khai báo interceptor trong file cấu hình lọc
   1. “/user/logoff\*\*”
   2. “/user/change\*\*”
   3. “/user/edit\*\*”
   4. “/order/\*\*”

## Bài 5 (2 điểm): Cung cấp dữ liệu dùng chung trên layout

Vấn đề: layout được sử dụng cho tất cả mọi view. Nếu layout cần dữ liệu đọc từ CSDL thì nên viết mã ở đâu để cấp dữ liệu cho layout? Trả lời là tất cả mọi action có liên quan đến view.

Đơn giản hơn rất nhiều là bạn viết một Interceptor đọc dữ liệu từ CSDL và đặt sẵn vào request ở sự kiện preHandle(). Với cách làm này thì view nào, layout nào cũng có thể sử dụng được.

Yêu cầu:

1. Viêt GlobalInterceptor và override sự kiện preHandle(). Trong preHandle() đọc List<Major> từ CSDL và đặt vào request.

|  |
| --- |
| Session session = factory.getCurrentSession();  String hql = "FROM Major";  Query query = session.createQuery(hql);  List<Major> list = query.list();  request.setAttribute("majors", list); |

1. Cấu hình sao cho GlobalInterceptor lọc tất cả các action
2. Tạo trang major-menu.jsp

|  |
| --- |
| <c:forEach var=*"m"* items=*"*${majors}*"*>  <a href=*"major/*${m.id}*.htm"*>${m.name}</a>  </c:forEach> |

1. Trong phần layout hoặc một view bất kỳ, bạn bao hàm major-menu.jsp vào bằng <jsp:include page=”major-menu.jsp”/>
2. Chạy và kiểm tra

***Chú ý:***

* *Phần I và Phần II chỉ áp dụng cho dạy tích hợp. Sinh viên làm phần 1 và phần 2 theo 2 bài khác nhau tương ứng với 2 phần lý thuyết đã dạy trong bài học.*
* *Nếu giảng dạy theo phương pháp truyền thống thì sinh viên phải thực hiện tất cả các bài trong một buổi.*