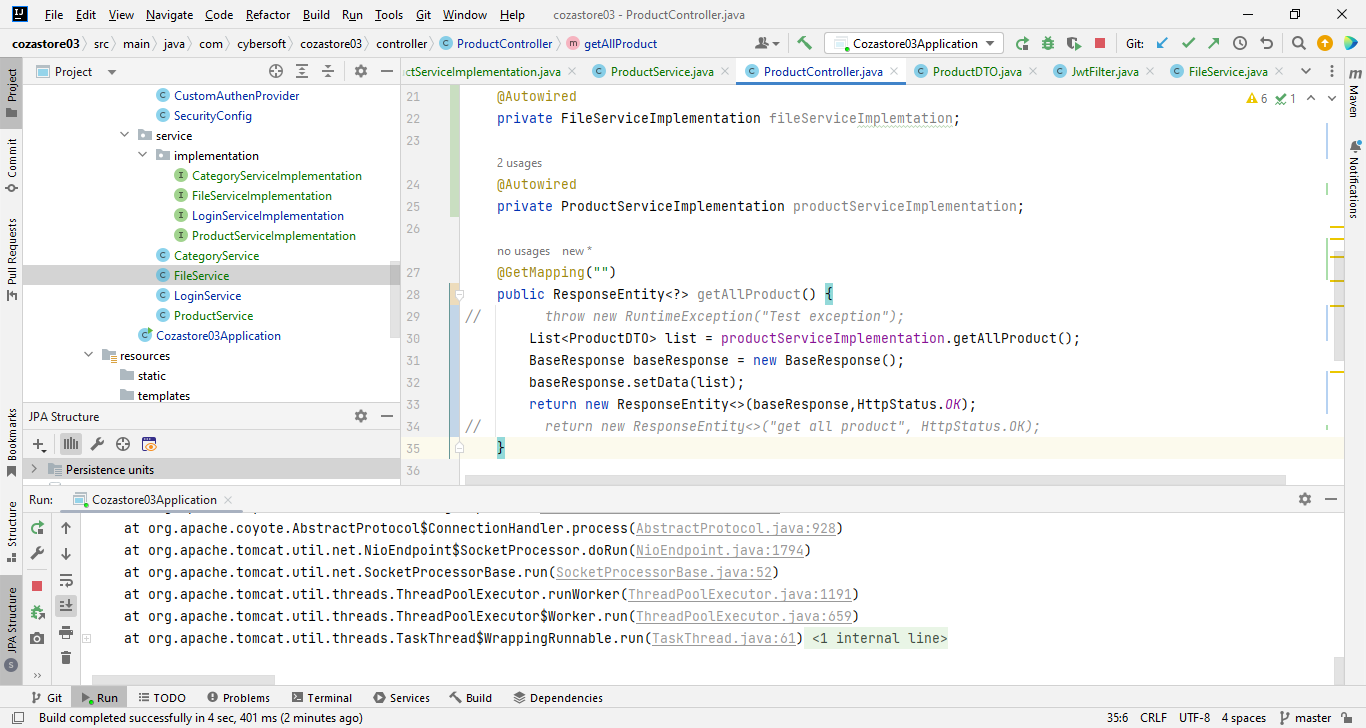
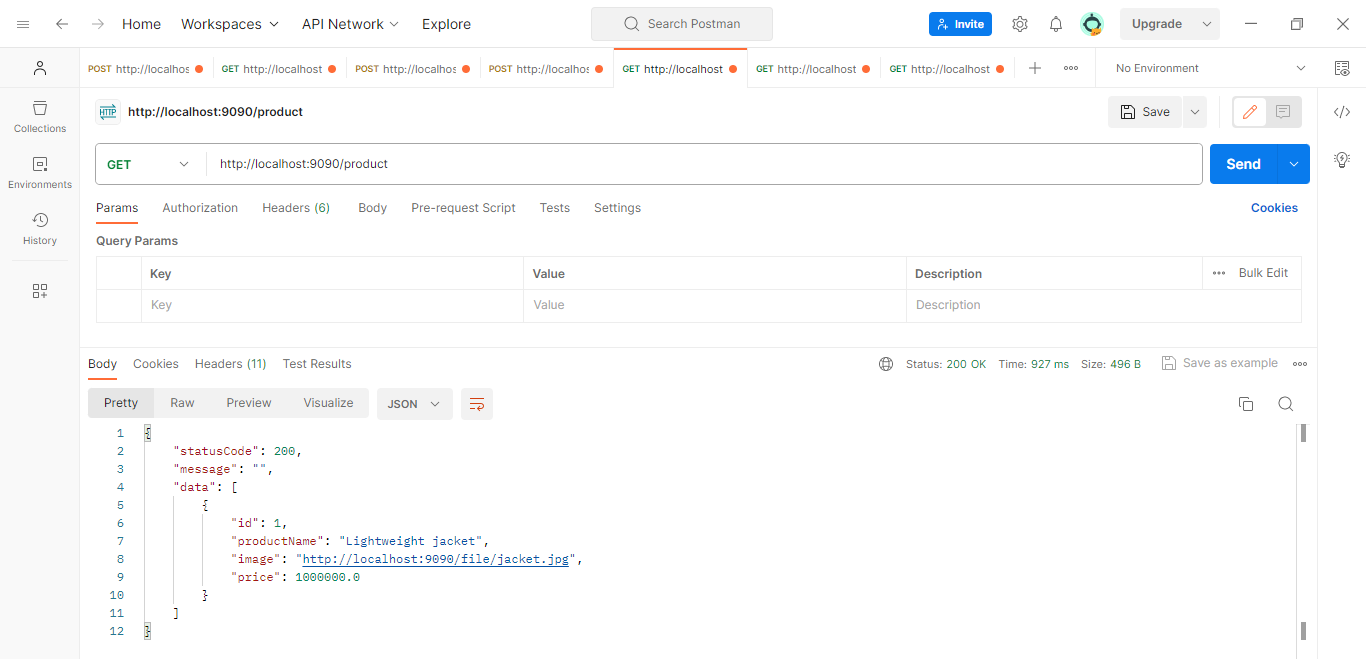
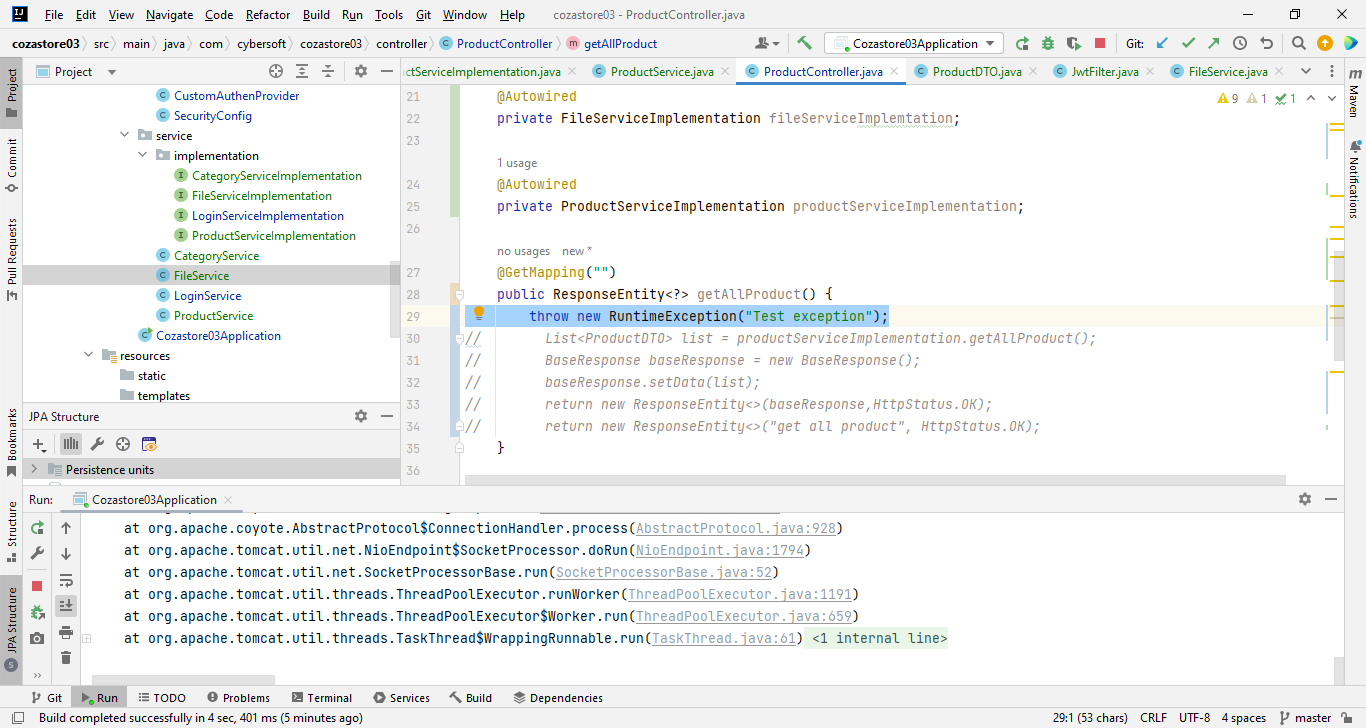
RestAPI chạy okay

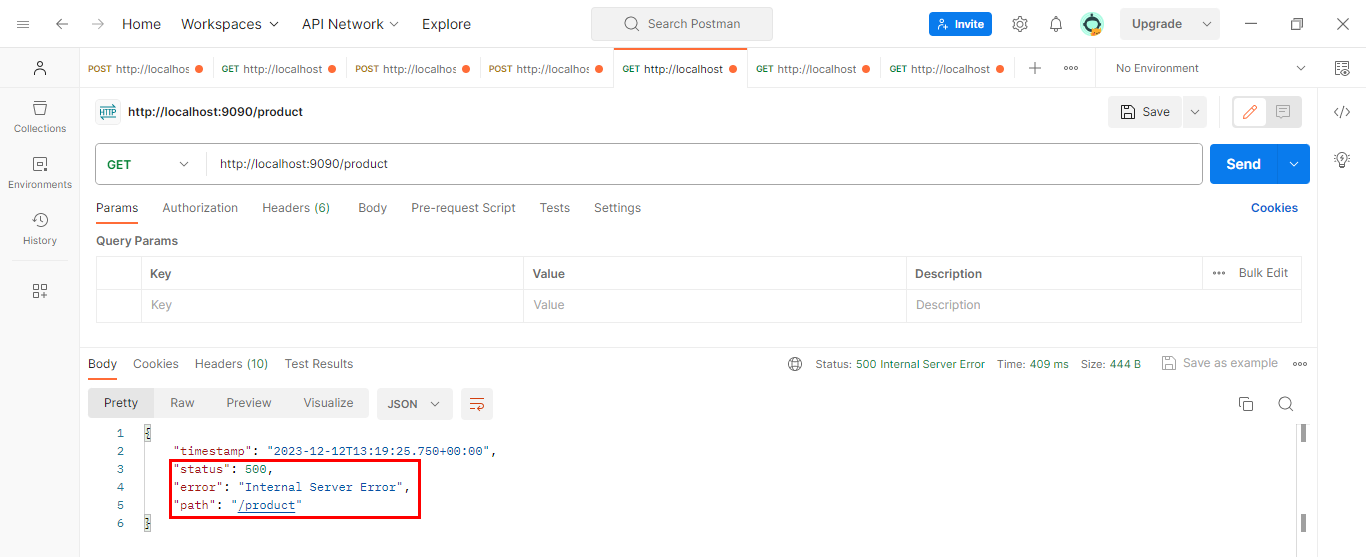




Khi REST server throw exception

VD: throw new RuntimeException("Test exception");

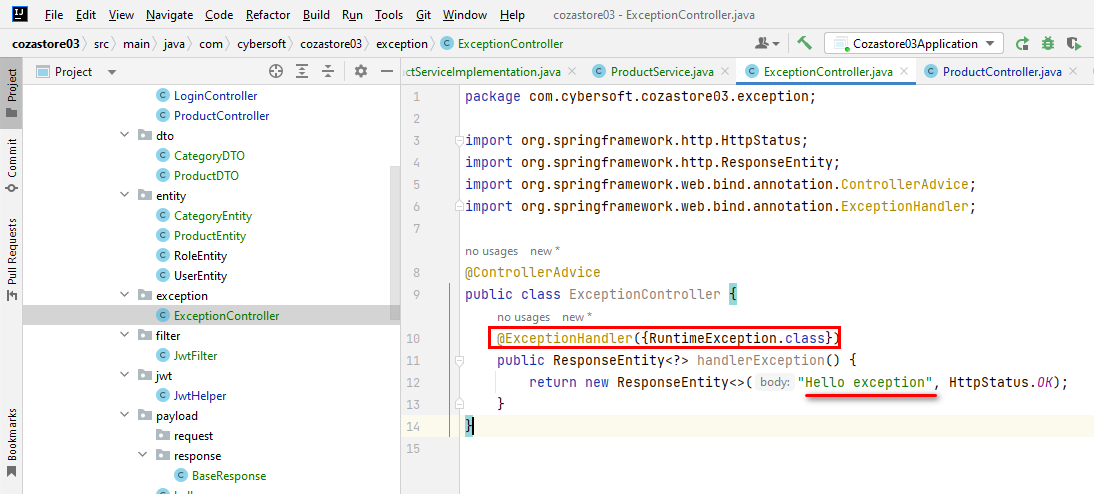




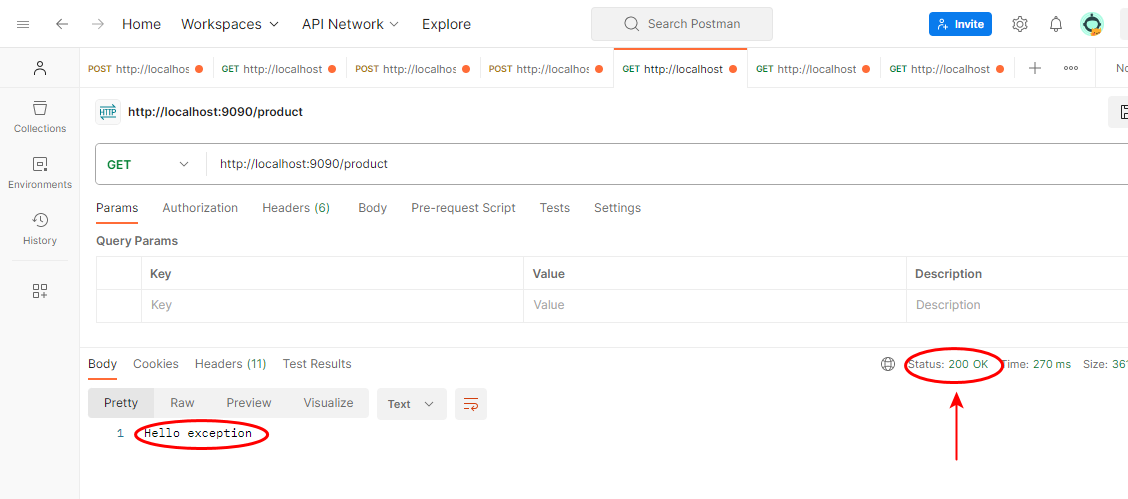
Chúng ta muốn customize khi REST ném ra cái exception nhưng mà REST server vẫn có thể control cái exception đó thì trong cái return response nên trả về status: 200, thay vì status là 500

Để lắng nghe Runtime Exception thì dùng

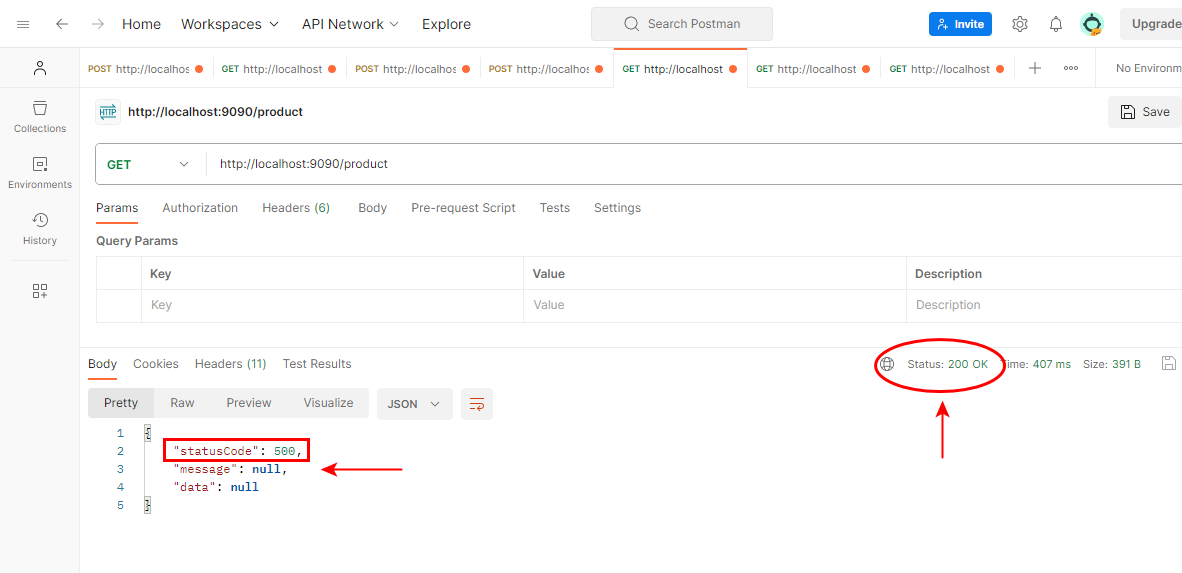
@ExceptionHandler({RuntimeException.class})



package com.cybersoft.cozastore03.exception;  
  
import org.springframework.http.HttpStatus;  
import org.springframework.http.ResponseEntity;  
import org.springframework.web.bind.annotation.ControllerAdvice;  
import org.springframework.web.bind.annotation.ExceptionHandler;  
  
@ControllerAdvice  
public class ExceptionController {  
 @ExceptionHandler({RuntimeException.class})  
 public ResponseEntity<?> handlerException() {  
 return new ResponseEntity<>("Hello exception", HttpStatus.*OK*);  
 }  
}



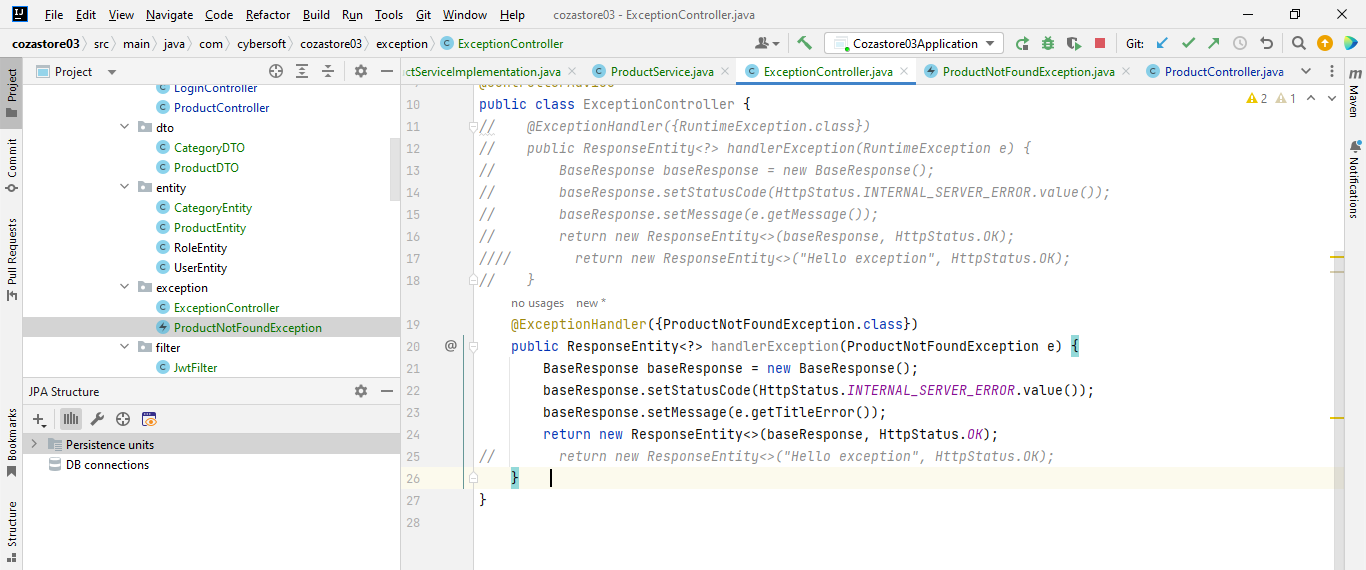
package com.cybersoft.cozastore03.exception;  
  
import com.cybersoft.cozastore03.payload.response.BaseResponse;  
import org.springframework.http.HttpStatus;  
import org.springframework.http.ResponseEntity;  
import org.springframework.web.bind.annotation.ControllerAdvice;  
import org.springframework.web.bind.annotation.ExceptionHandler;  
  
@ControllerAdvice  
public class ExceptionController {  
 @ExceptionHandler({RuntimeException.class})  
 public ResponseEntity<?> handlerException(RuntimeException e) {  
 BaseResponse baseResponse = new BaseResponse();  
 baseResponse.setStatusCode(HttpStatus.*INTERNAL\_SERVER\_ERROR*.value());  
 baseResponse.setMessage(e.getMessage());  
 return new ResponseEntity<>(baseResponse, HttpStatus.*OK*);  
*// return new ResponseEntity<>("Hello exception", HttpStatus.OK);* }  
}

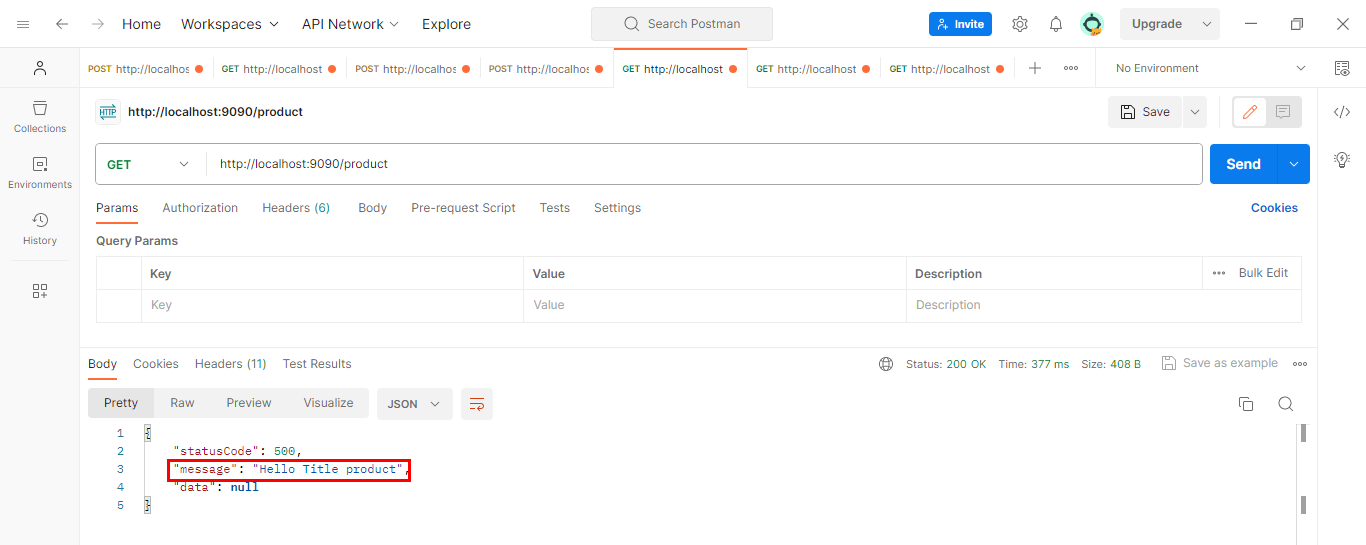


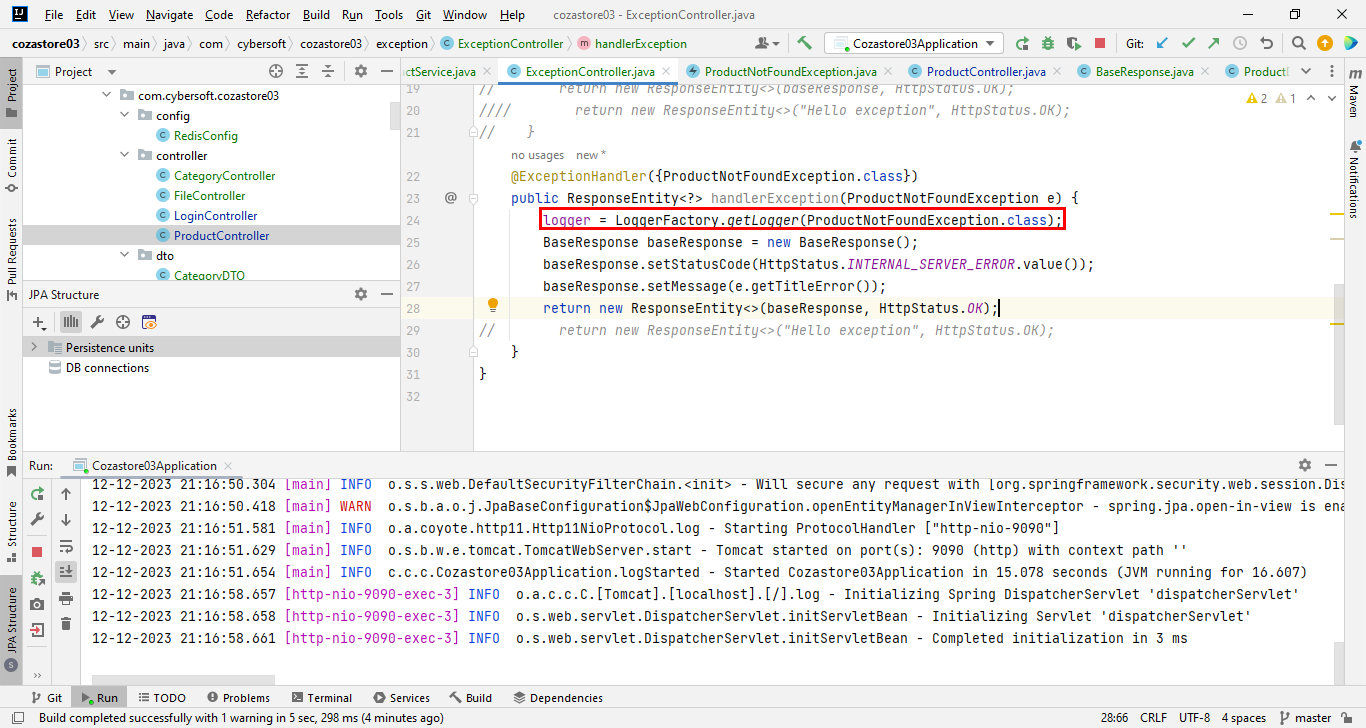
@AllArgsConstructor

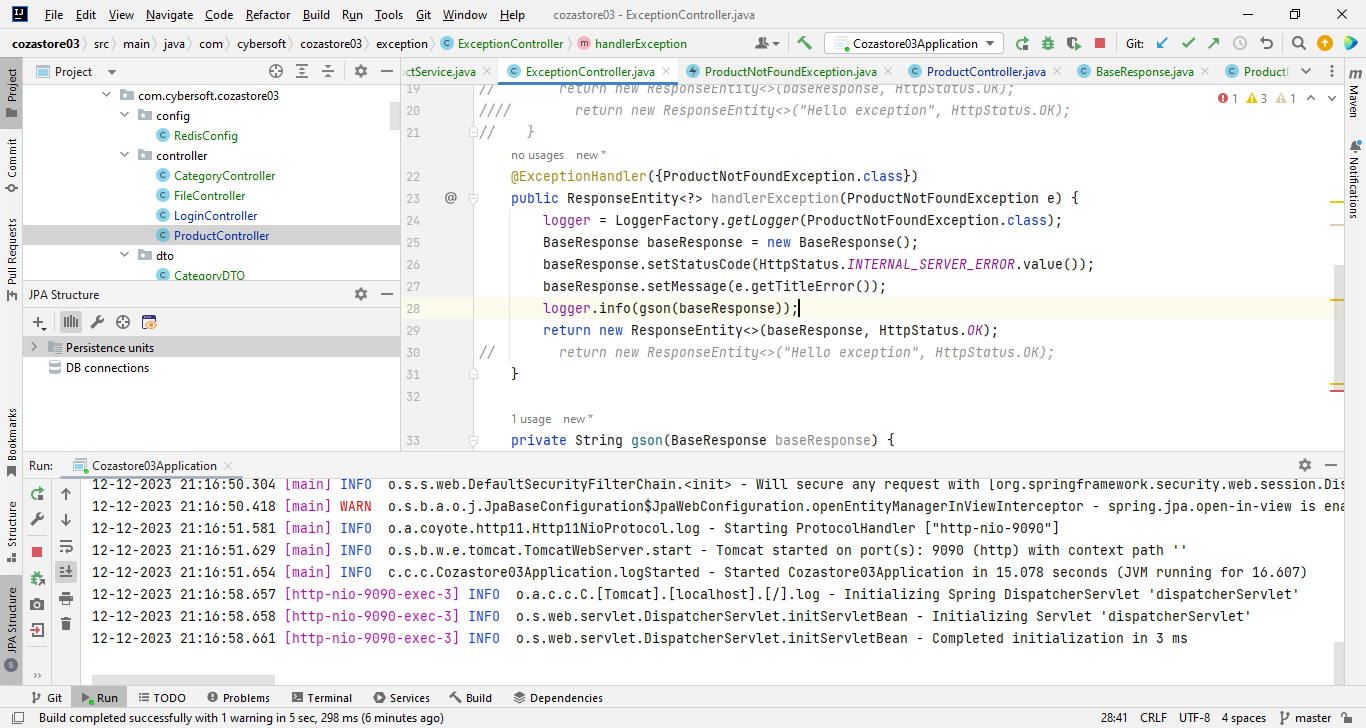
Tạo phương thức khởi tạo constructor bằng Loombook











<https://www.tutorialspoint.com/spring/spring_exception_handling_example.htm>

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Spring - Exception Handling Example**  The following example shows how to write a simple web-based application using Spring MVC Framework, which can handle one or more exceptions raised inside its controllers. To start with, let us have a working Eclipse IDE in place and take the following steps to develop a Dynamic Form-based Web Application using Spring Web Framework −   |  |  | | --- | --- | | **Step** | **Description** | | 1 | Create a *Dynamic Web Project* with a name *HelloWeb* and create a package *com.tutorialspoint* under the *src* folder in the created project. | | 2 | Drag and drop below mentioned Spring and other libraries into the folder *WebContent/WEB-INF/lib*. | | 3 | Create a Java classes *Student*, *StudentController* and *SpringException* under the *com.tutorialspoint* package. | | 4 | Create Spring configuration files *Web.xml* and *HelloWeb-servlet.xml* under the *WebContent/WEB-INF* folder. | | 5 | Create a sub-folder with a name *jsp* under the *WebContent/WEB-INF* folder. Create a view files *student.jsp*, *result.jsp*, *error.jsp*, and *ExceptionPage.jsp* under *jsp* sub-folder. | | 6 | The final step is to create the content of all the source and configuration files and export the application as explained below. |   Following is the content of **Student.java** file  package com.tutorialspoint;  public class Student {  private Integer age;  private String name;  private Integer id;  public void setAge(Integer age) {  this.age = age;  }  public Integer getAge() {  return age;  }  public void setName(String name) {  this.name = name;  }  public String getName() {  return name;  }  public void setId(Integer id) {  this.id = id;  }  public Integer getId() {  return id;  }  }  Following is the content of **SpringException.java** file  package com.tutorialspoint;  public class SpringException extends RuntimeException{  private String exceptionMsg;    public SpringException(String exceptionMsg) {  this.exceptionMsg = exceptionMsg;  }  public String getExceptionMsg(){  return this.exceptionMsg;  }  public void setExceptionMsg(String exceptionMsg) {  this.exceptionMsg = exceptionMsg;  }  }  Following is the content of **StudentController.java** file. Here, you need to annotate a service method using *@ExceptionHandler* where you can specify one or more exceptions to be handled. If you are specifying more than one exception then you can use comma separated values.  package com.tutorialspoint;  import org.springframework.stereotype.Controller;  import org.springframework.web.bind.annotation.ExceptionHandler;  import org.springframework.web.bind.annotation.ModelAttribute;  import org.springframework.web.bind.annotation.RequestMapping;  import org.springframework.web.bind.annotation.RequestMethod;  import org.springframework.web.servlet.ModelAndView;  import org.springframework.ui.ModelMap;  @Controller  public class StudentController {  @RequestMapping(value = "/student", method = RequestMethod.GET)  public ModelAndView student() {  return new ModelAndView("student", "command", new Student());  }  @RequestMapping(value = "/addStudent", method = RequestMethod.POST)  @ExceptionHandler({SpringException.class})  public String addStudent( @ModelAttribute("HelloWeb")Student student,  ModelMap model) {    if(student.getName().length() < 5 ){  throw new SpringException("Given name is too short");  } else {  model.addAttribute("name", student.getName());  }    if( student.getAge() < 10 ){  throw new SpringException("Given age is too low");  } else {  model.addAttribute("age", student.getAge());  }  model.addAttribute("id", student.getId());  return "result";  }  }  Following is the content of Spring Web configuration file **web.xml**  <web-app id = "WebApp\_ID" version = "2.4"  xmlns = "http://java.sun.com/xml/ns/j2ee"  xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance"  xsi:schemaLocation = "http://java.sun.com/xml/ns/j2ee  http://java.sun.com/xml/ns/j2ee/web-app\_2\_4.xsd">    <display-name>Spring Exception Handling</display-name>    <servlet>  <servlet-name>HelloWeb</servlet-name>  <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>  <load-on-startup>1</load-on-startup>  </servlet>    <servlet-mapping>  <servlet-name>HelloWeb</servlet-name>  <url-pattern>/</url-pattern>  </servlet-mapping>    </web-app>  Following is the content of another Spring Web configuration file **HelloWeb-servlet.xml**  <beans xmlns = "http://www.springframework.org/schema/beans"  xmlns:context = "http://www.springframework.org/schema/context"  axmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance"  xsi:schemaLocation = "http://www.springframework.org/schema/beans  http://www.springframework.org/schema/beans/spring-beans-3.0.xsd  http://www.springframework.org/schema/context  http://www.springframework.org/schema/context/spring-context-3.0.xsd">  <context:component-scan base-package = "com.tutorialspoint" />  <bean class = "org.springframework.web.servlet.view.InternalResourceViewResolver">  <property name = "prefix" value = "/WEB-INF/jsp/" />  <property name = "suffix" value = ".jsp" />  </bean>  <bean class = "org.springframework.web.servlet.handler.SimpleMappingExceptionResolver">  <property name = "exceptionMappings">  <props>  <prop key = "com.tutorialspoint.SpringException">  ExceptionPage  </prop>  </props>  </property>  <property name = "defaultErrorView" value = "error"/>  </bean>  </beans>  Here you specified *ExceptionPage* as an exception view in case SpringException occurs, if there is any other type of exception then a generic view *error* will take place.  Following is the content of Spring view file **student.jsp**  <%@taglib uri = "http://www.springframework.org/tags/form" prefix = "form"%>  <html>  <head>  <title>Spring MVC Exception Handling</title>  </head>  <body>  <h2>Student Information</h2>  <form:form method = "POST" action = "/HelloWeb/addStudent">  <table>  <tr>  <td><form:label path = "name">Name</form:label></td>  <td><form:input path = "name" /></td>  </tr>  <tr>  <td><form:label path = "age">Age</form:label></td>  <td><form:input path = "age" /></td>  </tr>  <tr>  <td><form:label path = "id">id</form:label></td>  <td><form:input path = "id" /></td>  </tr>  <tr>  <td colspan = "2"><input type = "submit" value = "Submit"/></td>  </tr>  </table>  </form:form>  </body>    </html>  Following is the content of Spring view file **error.jsp**  <html>  <head>  <title>Spring Error Page</title>  </head>  <body>  <p>An error occured, please contact webmaster.</p>  </body>  </html>  Following is the content of Spring view file **ExceptionPage.jsp**. Here you will access the exception instance via ${exception}.  <%@taglib uri = "http://www.springframework.org/tags/form" prefix = "form"%>  <html>  <head>  <title>Spring MVC Exception Handling</title>  </head>  <body>  <h2>Spring MVC Exception Handling</h2>  <h3>${exception.exceptionMsg}</h3>  </body>  </html>  Following is the content of Spring view file **result.jsp**  <%@taglib uri = "http://www.springframework.org/tags/form" prefix = "form"%>  <html>  <head>  <title>Spring MVC Form Handling</title>  </head>  <body>  <h2>Submitted Student Information</h2>    <table>  <tr>  <td>Name</td>  <td>${name}</td>  </tr>  <tr>  <td>Age</td>  <td>${age}</td>  </tr>  <tr>  <td>ID</td>  <td>${id}</td>  </tr>  </table>  </body>    </html>  Finally, following is the list of Spring and other libraries to be included in your web application. You simply drag these files and drop them in **WebContent/WEB-INF/lib** folder.   * commons-logging-x.y.z.jar * org.springframework.asm-x.y.z.jar * org.springframework.beans-x.y.z.jar * org.springframework.context-x.y.z.jar * org.springframework.core-x.y.z.jar * org.springframework.expression-x.y.z.jar * org.springframework.web.servlet-x.y.z.jar * org.springframework.web-x.y.z.jar * spring-web.jar   Once you are done creating the source and configuration files, export your application. Right-click on your application and use the **Export > WAR File** option and save your**HelloWeb.war** file in Tomcat's *webapps* folder.  Now start your Tomcat server and make sure you are able to access other web pages from webapps folder using a standard browser. Now try to access the URL **http://localhost:8080/HelloWeb/student**. If everything is fine with your Spring Web Applicationand, you should see the following result.  Enter the values as shown above and click the Submit buttom. If everything is fine with your Spring Web Application, you should see the following result.  Spring Exception Result  **Kickstart Your Career**  Get certified by completing the course  [Get Started](https://www.tutorialspoint.com/latest/certifications)  Print Page  [PreviousNext](https://www.tutorialspoint.com/spring/spring_web_mvc_framework.htm) | **Spring - Ví dụ xử lý ngoại lệ**  Ví dụ sau đây cho thấy cách viết một ứng dụng dựa trên web đơn giản bằng cách sử dụng Spring MVC Framework, ứng dụng này có thể xử lý một hoặc nhiều ngoại lệ được đưa ra bên trong bộ điều khiển của nó. Để bắt đầu, chúng ta hãy cài đặt một IDE Eclipse đang hoạt động và thực hiện các bước sau để phát triển Ứng dụng web dựa trên Biểu mẫu động bằng cách sử dụng Spring Web Framework -   |  |  | | --- | --- | | **Bước chân** | **Sự miêu tả** | | 1 | Tạo một *Dự án web động* với tên *HelloWeb* và tạo một gói *com.tutorialspoint* trong thư mục *src* trong thư mục đã tạo dự án. | | 2 | Kéo và thả Spring và các thư viện khác được đề cập bên dưới vào thư mục *WebContent/WEB-INF/lib*. | | 3 | Tạo một lớp Java *Student*, *StudentController* và < /span>.*com.tutorialspoint* trong gói *SpringException* | | 4 | Tạo tệp cấu hình Spring *Web.xml* và *HelloWeb-servlet.xml< a i=4> trong thư mục WebContent/WEB-INF*. | | 5 | Tạo thư mục con có tên *jsp* trong *WebContent/WEB-INF*, thư mục con.*jsp* trong *ExceptionPage.jsp* và *error.jspresult.jsp*, *student.jsp*. Tạo tệp xem | | 6 | Bước cuối cùng là tạo nội dung của tất cả các tệp nguồn và tệp cấu hình rồi xuất ứng dụng như được giải thích bên dưới. |   Sau đây là nội dung của tệp **Student.java**  package com.tutorialspoint;  public class Student {  private Integer age;  private String name;  private Integer id;  public void setAge(Integer age) {  this.age = age;  }  public Integer getAge() {  return age;  }  public void setName(String name) {  this.name = name;  }  public String getName() {  return name;  }  public void setId(Integer id) {  this.id = id;  }  public Integer getId() {  return id;  }  }  Sau đây là nội dung của tệp **SpringException.java**  package com.tutorialspoint;  public class SpringException extends RuntimeException{  private String exceptionMsg;    public SpringException(String exceptionMsg) {  this.exceptionMsg = exceptionMsg;  }  public String getExceptionMsg(){  return this.exceptionMsg;  }  public void setExceptionMsg(String exceptionMsg) {  this.exceptionMsg = exceptionMsg;  }  }  Sau đây là nội dung của tệp **StudentController.java**. Tại đây, bạn cần chú thích một phương thức dịch vụ bằng cách sử dụng *@ExceptionHandler* nơi bạn có thể chỉ định một hoặc nhiều ngoại lệ cần xử lý. Nếu bạn đang chỉ định nhiều ngoại lệ thì bạn có thể sử dụng các giá trị được phân tách bằng dấu phẩy.  package com.tutorialspoint;  import org.springframework.stereotype.Controller;  import org.springframework.web.bind.annotation.ExceptionHandler;  import org.springframework.web.bind.annotation.ModelAttribute;  import org.springframework.web.bind.annotation.RequestMapping;  import org.springframework.web.bind.annotation.RequestMethod;  import org.springframework.web.servlet.ModelAndView;  import org.springframework.ui.ModelMap;  @Controller  public class StudentController {  @RequestMapping(value = "/student", method = RequestMethod.GET)  public ModelAndView student() {  return new ModelAndView("student", "command", new Student());  }  @RequestMapping(value = "/addStudent", method = RequestMethod.POST)  @ExceptionHandler({SpringException.class})  public String addStudent( @ModelAttribute("HelloWeb")Student student,  ModelMap model) {    if(student.getName().length() < 5 ){  throw new SpringException("Given name is too short");  } else {  model.addAttribute("name", student.getName());  }    if( student.getAge() < 10 ){  throw new SpringException("Given age is too low");  } else {  model.addAttribute("age", student.getAge());  }  model.addAttribute("id", student.getId());  return "result";  }  }  Dưới đây là nội dung file cấu hình Spring Web **web.xml**  <web-app id = "WebApp\_ID" version = "2.4"  xmlns = "http://java.sun.com/xml/ns/j2ee"  xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance"  xsi:schemaLocation = "http://java.sun.com/xml/ns/j2ee  http://java.sun.com/xml/ns/j2ee/web-app\_2\_4.xsd">    <display-name>Spring Exception Handling</display-name>    <servlet>  <servlet-name>HelloWeb</servlet-name>  <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>  <load-on-startup>1</load-on-startup>  </servlet>    <servlet-mapping>  <servlet-name>HelloWeb</servlet-name>  <url-pattern>/</url-pattern>  </servlet-mapping>    </web-app>  Sau đây là nội dung của một file cấu hình Spring Web khác **HelloWeb-servlet.xml**  <beans xmlns = "http://www.springframework.org/schema/beans"  xmlns:context = "http://www.springframework.org/schema/context"  axmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance"  xsi:schemaLocation = "http://www.springframework.org/schema/beans  http://www.springframework.org/schema/beans/spring-beans-3.0.xsd  http://www.springframework.org/schema/context  http://www.springframework.org/schema/context/spring-context-3.0.xsd">  <context:component-scan base-package = "com.tutorialspoint" />  <bean class = "org.springframework.web.servlet.view.InternalResourceViewResolver">  <property name = "prefix" value = "/WEB-INF/jsp/" />  <property name = "suffix" value = ".jsp" />  </bean>  <bean class = "org.springframework.web.servlet.handler.SimpleMappingExceptionResolver">  <property name = "exceptionMappings">  <props>  <prop key = "com.tutorialspoint.SpringException">  ExceptionPage  </prop>  </props>  </property>  <property name = "defaultErrorView" value = "error"/>  </bean>  </beans>  Ở đây bạn đã chỉ định *ExceptionPage* làm chế độ xem ngoại lệ trong trường hợp SpringException xảy ra, nếu có bất kỳ loại ngoại lệ nào khác thì chế độ xem chung sẽ xảy ra.*lỗi*  Sau đây là nội dung của file Spring view **student.jsp**  <%@taglib uri = "http://www.springframework.org/tags/form" prefix = "form"%>  <html>  <head>  <title>Spring MVC Exception Handling</title>  </head>  <body>  <h2>Student Information</h2>  <form:form method = "POST" action = "/HelloWeb/addStudent">  <table>  <tr>  <td><form:label path = "name">Name</form:label></td>  <td><form:input path = "name" /></td>  </tr>  <tr>  <td><form:label path = "age">Age</form:label></td>  <td><form:input path = "age" /></td>  </tr>  <tr>  <td><form:label path = "id">id</form:label></td>  <td><form:input path = "id" /></td>  </tr>  <tr>  <td colspan = "2"><input type = "submit" value = "Submit"/></td>  </tr>  </table>  </form:form>  </body>    </html>  Sau đây là nội dung của file Spring view **error.jsp**  <html>  <head>  <title>Spring Error Page</title>  </head>  <body>  <p>An error occured, please contact webmaster.</p>  </body>  </html>  Sau đây là nội dung của tệp Spring view **ExceptionPage.jsp**. Tại đây, bạn sẽ truy cập phiên bản ngoại lệ thông qua ${Exception}.  <%@taglib uri = "http://www.springframework.org/tags/form" prefix = "form"%>  <html>  <head>  <title>Spring MVC Exception Handling</title>  </head>  <body>  <h2>Spring MVC Exception Handling</h2>  <h3>${exception.exceptionMsg}</h3>  </body>  </html>  Sau đây là nội dung của file Spring view **result.jsp**  <%@taglib uri = "http://www.springframework.org/tags/form" prefix = "form"%>  <html>  <head>  <title>Spring MVC Form Handling</title>  </head>  <body>  <h2>Submitted Student Information</h2>    <table>  <tr>  <td>Name</td>  <td>${name}</td>  </tr>  <tr>  <td>Age</td>  <td>${age}</td>  </tr>  <tr>  <td>ID</td>  <td>${id}</td>  </tr>  </table>  </body>    </html>  Cuối cùng, sau đây là danh sách Spring và các thư viện khác sẽ được đưa vào ứng dụng web của bạn. Bạn chỉ cần kéo các tệp này và thả chúng vào thư mục **WebContent/WEB-INF/lib**.   * commons-logging-x.y.z.jar * org.springframework.asm-x.y.z.jar * org.springframework.beans-x.y.z.jar * org.springframework.context-x.y.z.jar * org.springframework.core-x.y.z.jar * org.springframework.biểu thức-x.y.z.jar * org.springframework.web.servlet-x.y.z.jar * org.springframework.web-x.y.z.jar * spring-web.jar   Sau khi bạn tạo xong tệp nguồn và tệp cấu hình, hãy xuất ứng dụng của bạn. Nhấp chuột phải vào ứng dụng của bạn và sử dụng **Export > Tùy chọn WAR File** và lưu tệp**HelloWeb.war** của bạn trong .  Bây giờ hãy khởi động máy chủ Tomcat của bạn và đảm bảo bạn có thể truy cập các trang web khác từ thư mục ứng dụng web bằng trình duyệt chuẩn. Bây giờ hãy thử truy cập URL **http://localhost:8080/HelloWeb/student**. Nếu mọi thứ đều ổn với Ứng dụng Web Spring của bạn và bạn sẽ thấy kết quả sau.  Nhập các giá trị như được hiển thị ở trên và nhấp vào nút Gửi. Nếu mọi thứ đều ổn với Ứng dụng Web Spring của bạn, bạn sẽ thấy kết quả sau.  Kết quả ngoại lệ mùa xuân  **Bắt đầu sự nghiệp của bạn**  Nhận chứng nhận khi hoàn thành khóa học  [Bắt đầu](https://www.tutorialspoint.com/latest/certifications)  Trang in  [TrướcKế tiếp](https://www.tutorialspoint.com/spring/spring_web_mvc_framework.htm) |

<https://spring.io/blog/2013/11/01/exception-handling-in-spring-mvc>

|  |  |
| --- | --- |
| Exception Handling in Spring MVC [**ENGINEERING**](https://spring.io/blog/category/engineering)**| PAUL CHAPMAN | NOVEMBER 01, 2013 |**[**56 COMMENTS**](https://spring.io/blog/2013/11/01/exception-handling-in-spring-mvc/#disqus_thread)  **NOTE:** [*Revised April 2018*](https://spring.io/blog/2013/11/01/exception-handling-in-spring-mvc#recent-updates)  Spring MVC provides several complimentary approaches to exception handling but, when teaching Spring MVC, I often find that my students are confused or not comfortable with them.  Today I'm going to show you the various options available. Our goal is to *not* handle exceptions explicitly in Controller methods where possible. They are a cross-cutting concern better handled separately in dedicated code.  There are three options: per exception, per controller or globally.  A demonstration application that shows the points discussed here can be found at [*http://github.com/paulc4/mvc-exceptions*](http://github.com/paulc4/mvc-exceptions). See [*Sample Application*](https://spring.io/blog/2013/11/01/exception-handling-in-spring-mvc#sample-application) below for details.  **NOTE:** The demo applications has been revamped and updated (April 2018) to use Spring Boot 2.0.1 and is (hopefully) easier to use and understand. I also fixed some broken links (thanks for the feedback, sorry it took a while). Spring Boot [Spring Boot](http://spring.io/spring-boot) allows a Spring project to be setup with minimal configuration and it is likely that you are using it if your application is less than a few years old.  Spring MVC offers no default (fall-back) error page out-of-the-box. The most common way to set a default error page has always been the SimpleMappingExceptionResolver (since Spring V1 in fact). We will discuss this later.  However Spring Boot does provide for a fallback error-handling page.  At start-up, Spring Boot tries to find a mapping for /error. By convention, a URL ending in /error maps to a logical view of the same name: error. In the demo application this view maps in turn to the error.html Thymeleaf template. (If using JSP, it would map to error.jsp according to the setup of your InternalResourceViewResolver). The actual mapping will depend on what ViewResolver (if any) that you or Spring Boot has setup.  If no view-resolver mapping for /error can be found, Spring Boot defines its own fall-back error page - the so-called "Whitelabel Error Page" (a minimal page with just the HTTP status information and any error details, such as the message from an uncaught exception). In the sample applicaiton, if you rename the error.html template to, say, error2.html then restart, you will see it being used.  If you are making a RESTful request (the HTTP request has specified a desired response type other than HTML) Spring Boot returns a JSON representation of the same error information that it puts in the "Whitelabel" error page.  COPY$> curl -H "Accept: application/json" http://localhost:8080/no-such-page  {"timestamp":"2018-04-11T05:56:03.845+0000","status":404,"error":"Not Found","message":"No message available","path":"/no-such-page"}  Spring Boot also sets up a default error-page for the container, equivalent to the <error-page> directive in web.xml (although implemented very differently). Exceptions thrown outside the Spring MVC framework, such as from a servlet Filter, are still reported by the Spring Boot fallback error page. The sample application also shows an example of this.  A more in-depth discussion of Spring Boot error-handling can be found at the end of this article.  The rest of this article applies regardless of whether you are using Spring with or without Spring Boot.  Impatient REST developers may choose to skip directly to the [*section*](https://spring.io/blog/2013/11/01/exception-handling-in-spring-mvc#errors-and-rest) on custom REST error responses. However they should then read the full article as most of it applies equally to all web applications, REST or otherwise. Using HTTP Status Codes Normally any unhandled exception thrown when processing a web-request causes the server to return an HTTP 500 response. However, any exception that you write yourself can be annotated with the @ResponseStatus annotation (which supports all the HTTP status codes defined by the HTTP specification). When an annotated exception is thrown from a controller method, and not handled elsewhere, it will automatically cause the appropriate HTTP response to be returned with the specified status-code.  For example, here is an exception for a missing order.  COPY @ResponseStatus(value=HttpStatus.NOT\_FOUND, reason="No such Order") // 404  public class OrderNotFoundException extends RuntimeException {  // ...  }  And here is a controller method using it:  COPY @RequestMapping(value="/orders/{id}", method=GET)  public String showOrder(@PathVariable("id") long id, Model model) {  Order order = orderRepository.findOrderById(id);  if (order == null) throw new OrderNotFoundException(id);  model.addAttribute(order);  return "orderDetail";  }  A familiar HTTP 404 response will be returned if the URL handled by this method includes an unknown order id. Controller Based Exception HandlingUsing @ExceptionHandler You can add extra (@ExceptionHandler) methods to any controller to specifically handle exceptions thrown by request handling (@RequestMapping) methods in the same controller. Such methods can:   1. Handle exceptions without the @ResponseStatus annotation (typically predefined exceptions that you didn't write) 2. Redirect the user to a dedicated error view 3. Build a totally custom error response   The following controller demonstrates these three options:  COPY@Controller  public class ExceptionHandlingController {  // @RequestHandler methods  ...  // Exception handling methods  // Convert a predefined exception to an HTTP Status code  @ResponseStatus(value=HttpStatus.CONFLICT,  reason="Data integrity violation") // 409  @ExceptionHandler(DataIntegrityViolationException.class)  public void conflict() {  // Nothing to do  }  // Specify name of a specific view that will be used to display the error:  @ExceptionHandler({SQLException.class,DataAccessException.class})  public String databaseError() {  // Nothing to do. Returns the logical view name of an error page, passed  // to the view-resolver(s) in usual way.  // Note that the exception is NOT available to this view (it is not added  // to the model) but see "Extending ExceptionHandlerExceptionResolver"  // below.  return "databaseError";  }  // Total control - setup a model and return the view name yourself. Or  // consider subclassing ExceptionHandlerExceptionResolver (see below).  @ExceptionHandler(Exception.class)  public ModelAndView handleError(HttpServletRequest req, Exception ex) {  logger.error("Request: " + req.getRequestURL() + " raised " + ex);  ModelAndView mav = new ModelAndView();  mav.addObject("exception", ex);  mav.addObject("url", req.getRequestURL());  mav.setViewName("error");  return mav;  }  }  In any of these methods you might choose to do additional processing - the most common example is to log the exception.  Handler methods have flexible signatures so you can pass in obvious servlet-related objects such as HttpServletRequest, HttpServletResponse, HttpSession and/or Principle.  **Important Note:** The Model may **not** be a parameter of any @ExceptionHandler method. Instead, setup a model inside the method using a ModelAndView as shown by handleError() above. Exceptions and Views Be careful when adding exceptions to the model. Your users do not want to see web-pages containing Java exception details and stack-traces. You may have security policies that expressly forbid putting any exception information in the error page. Another reason to make sure you override the Spring Boot white-label error page.  Make sure exceptions are logged usefully so they can be analyzed after the event by your support and development teams.  Please remember the following may be convenient but it is ***not*** best practice in production.  It can be useful to hide exception details in the page source as a comment, to assist testing. If using JSP, you could do something like this to output the exception and the corresponding stack-trace (using a hidden <div> is another option).  COPY <h1>Error Page</h1>  <p>Application has encountered an error. Please contact support on ...</p>  <!--  Failed URL: ${url}  Exception: ${exception.message}  <c:forEach items="${exception.stackTrace}" var="ste"> ${ste}  </c:forEach>  -->  For the Thymeleaf equivalent see [support.html](https://github.com/paulc4/mvc-exceptions/blob/master/src/main/resources/templates/support.html) in the demo application. The result looks like this.  Example of an error page with a hidden exception for support Global Exception HandlingUsing @ControllerAdvice Classes A controller advice allows you to use exactly the same exception handling techniques but apply them across the whole application, not just to an individual controller. You can think of them as an annotation driven interceptor.  Any class annotated with @ControllerAdvice becomes a controller-advice and three types of method are supported:   * Exception handling methods annotated with @ExceptionHandler. * Model enhancement methods (for adding additional data to the model) annotated with   @ModelAttribute. Note that these attributes are not available to the exception handling views.   * Binder initialization methods (used for configuring form-handling) annotated with   @InitBinder.  We are only going to look at exception handling - search the [online manual](https://docs.spring.io/spring/docs/current/spring-framework-reference/web.html#mvc-controller) for more on @ControllerAdvice methods.  Any of the exception handlers you saw above can be defined on a controller-advice class - but now they apply to exceptions thrown from any controller. Here is a simple example:  COPY@ControllerAdvice  class GlobalControllerExceptionHandler {  @ResponseStatus(HttpStatus.CONFLICT) // 409  @ExceptionHandler(DataIntegrityViolationException.class)  public void handleConflict() {  // Nothing to do  }  }  If you want to have a default handler for any exception, there is a slight wrinkle. You need to ensure annotated exceptions are handled by the framework. The code looks like this:  COPY@ControllerAdvice  class GlobalDefaultExceptionHandler {  public static final String DEFAULT\_ERROR\_VIEW = "error";  @ExceptionHandler(value = Exception.class)  public ModelAndView  defaultErrorHandler(HttpServletRequest req, Exception e) throws Exception {  // If the exception is annotated with @ResponseStatus rethrow it and let  // the framework handle it - like the OrderNotFoundException example  // at the start of this post.  // AnnotationUtils is a Spring Framework utility class.  if (AnnotationUtils.findAnnotation  (e.getClass(), ResponseStatus.class) != null)  throw e;  // Otherwise setup and send the user to a default error-view.  ModelAndView mav = new ModelAndView();  mav.addObject("exception", e);  mav.addObject("url", req.getRequestURL());  mav.setViewName(DEFAULT\_ERROR\_VIEW);  return mav;  }  } Going DeeperHandlerExceptionResolver Any Spring bean declared in the DispatcherServlet's application context that implements HandlerExceptionResolver will be used to intercept and process any exception raised in the MVC system and not handled by a Controller. The interface looks like this:  COPYpublic interface HandlerExceptionResolver {  ModelAndView resolveException(HttpServletRequest request,  HttpServletResponse response, Object handler, Exception ex);  }  The handler refers to the controller that generated the exception (remember that @Controller instances are only one type of handler supported by Spring MVC. For example: HttpInvokerExporter and the WebFlow Executor are also types of handler).  Behind the scenes, MVC creates three such resolvers by default. It is these resolvers that implement the behaviours discussed above:   * ExceptionHandlerExceptionResolver matches uncaught exceptions against suitable @ExceptionHandler methods on both the handler (controller) and on any controller-advices. * ResponseStatusExceptionResolver looks for uncaught exceptions annotated by @ResponseStatus (as described in Section 1) * DefaultHandlerExceptionResolver converts standard Spring exceptions and converts them to HTTP Status Codes (I have not mentioned this above as it is internal to Spring MVC).   These are chained and processed in the order listed - internally Spring creates a dedicated bean (the HandlerExceptionResolverComposite) to do this.  Notice that the method signature of resolveException does not include the Model. This is why @ExceptionHandler methods cannot be injected with the model.  You can, if you wish, implement your own HandlerExceptionResolver to setup your own custom exception handling system. Handlers typically implement Spring's Ordered interface so you can define the order that the handlers run in. SimpleMappingExceptionResolver Spring has long provided a simple but convenient implementation of HandlerExceptionResolver that you may well find being used in your appication already - the SimpleMappingExceptionResolver. It provides options to:   * Map exception class names to view names - just specify the classname, no package needed. * Specify a default (fallback) error page for any exception not handled anywhere else * Log a message (this is not enabled by default). * Set the name of the exception attribute to add to the Model so it can be used inside a View   (such as a JSP). By default this attribute is named exception. Set to null to disable. Remember that views returned from @ExceptionHandler methods do not have access to the exception but views defined to SimpleMappingExceptionResolver do.  Here is a typical configuration using Java Configuration:  COPY@Configuration  @EnableWebMvc // Optionally setup Spring MVC defaults (if you aren't using  // Spring Boot & haven't specified @EnableWebMvc elsewhere)  public class MvcConfiguration extends WebMvcConfigurerAdapter {  @Bean(name="simpleMappingExceptionResolver")  public SimpleMappingExceptionResolver  createSimpleMappingExceptionResolver() {  SimpleMappingExceptionResolver r =  new SimpleMappingExceptionResolver();  Properties mappings = new Properties();  mappings.setProperty("DatabaseException", "databaseError");  mappings.setProperty("InvalidCreditCardException", "creditCardError");  r.setExceptionMappings(mappings); // None by default  r.setDefaultErrorView("error"); // No default  r.setExceptionAttribute("ex"); // Default is "exception"  r.setWarnLogCategory("example.MvcLogger"); // No default  return r;  }  ...  }  Or using XML Configuration:  COPY <bean id="simpleMappingExceptionResolver" class=  "org.springframework.web.servlet.handler.SimpleMappingExceptionResolver">  <property name="exceptionMappings">  <map>  <entry key="DatabaseException" value="databaseError"/>  <entry key="InvalidCreditCardException" value="creditCardError"/>  </map>  </property>  <!-- See note below on how this interacts with Spring Boot -->  <property name="defaultErrorView" value="error"/>  <property name="exceptionAttribute" value="ex"/>  <!-- Name of logger to use to log exceptions. Unset by default,  so logging is disabled unless you set a value. -->  <property name="warnLogCategory" value="example.MvcLogger"/>  </bean>  The defaultErrorView property is especially useful as it ensures any uncaught exception generates a suitable application defined error page. (The default for most application servers is to display a Java stack-trace - something your users should never see). Spring Boot provides another way to do the same thing with its "white-label" error page. Extending SimpleMappingExceptionResolver It is quite common to extend SimpleMappingExceptionResolver for several reasons:   * You can use the constructor to set properties directly - for example to enable exception logging and set the logger to use * Override the default log message by overriding buildLogMessage. The default implementation always returns this fixed text:   *Handler execution resulted in exception*   * To make additional information available to the error view by overriding doResolveException   For example:  COPYpublic class MyMappingExceptionResolver extends SimpleMappingExceptionResolver {  public MyMappingExceptionResolver() {  // Enable logging by providing the name of the logger to use  setWarnLogCategory(MyMappingExceptionResolver.class.getName());  }  @Override  public String buildLogMessage(Exception e, HttpServletRequest req) {  return "MVC exception: " + e.getLocalizedMessage();  }  @Override  protected ModelAndView doResolveException(HttpServletRequest req,  HttpServletResponse resp, Object handler, Exception ex) {  // Call super method to get the ModelAndView  ModelAndView mav = super.doResolveException(req, resp, handler, ex);  // Make the full URL available to the view - note ModelAndView uses  // addObject() but Model uses addAttribute(). They work the same.  mav.addObject("url", request.getRequestURL());  return mav;  }  }  This code is in the demo application as [ExampleSimpleMappingExceptionResolver](https://github.com/paulc4/mvc-exceptions/blob/master/src/main/java/demo/example/ExampleSimpleMappingExceptionResolver.java) Extending ExceptionHandlerExceptionResolver It is also possible to extend ExceptionHandlerExceptionResolver and override its doResolveHandlerMethodException method in the same way. It has almost the same signature (it just takes the new HandlerMethod instead of a Handler).  To make sure it gets used, also set the inherited order property (for example in the constructor of your new class) to a value less than MAX\_INT so it runs before the default ExceptionHandlerExceptionResolver instance (it is easier to create your own handler instance than try to modify/replace the one created by Spring). See [ExampleExceptionHandlerExceptionResolver](https://github.com/paulc4/mvc-exceptions/blob/master/src/main/java/demo/example/ExampleExceptionHandlerExceptionResolver.java) in the demo app for more. Errors and REST RESTful GET requests may also generate exceptions and we have already seen how we can return standard HTTP Error response codes. However, what if you want to return information about the error? This is very easy to do. Firstly define an error class:  COPYpublic class ErrorInfo {  public final String url;  public final String ex;  public ErrorInfo(String url, Exception ex) {  this.url = url;  this.ex = ex.getLocalizedMessage();  }  }  Now we can return an instance from a handler as the @ResponseBody like this:  COPY@ResponseStatus(HttpStatus.BAD\_REQUEST)  @ExceptionHandler(MyBadDataException.class)  @ResponseBody ErrorInfo  handleBadRequest(HttpServletRequest req, Exception ex) {  return new ErrorInfo(req.getRequestURL(), ex);  } What to Use When? As usual, Spring likes to offer you choice, so what should you do? Here are some rules of thumb. However if you have a preference for XML configuration or Annotations, that's fine too.   * For exceptions you write, consider adding @ResponseStatus to them. * For all other exceptions implement an @ExceptionHandler method on a @ControllerAdvice class or use an instance of SimpleMappingExceptionResolver. You may well have SimpleMappingExceptionResolver configured for your application already, in which case it may be easier to add new exception classes to it than implement a @ControllerAdvice. * For Controller specific exception handling add @ExceptionHandler methods to your controller. * **Warning:** Be careful mixing too many of these options in the same application. If the same exception can be handed in more than one way, you may not get the behavior you wanted. @ExceptionHandler methods on the Controller are always selected before those on any @ControllerAdvice instance. It is *undefined* what order controller-advices are processed.  Sample Application A demonstration application can be found at [github](http://github.com/paulc4/mvc-exceptions). It uses Spring Boot and Thymeleaf to build a simple web application.  The application has been revised twice (Oct 2014, April 2018) and is (hopefully) better and easier to understand. The fundamentals stay the same. It uses Spring Boot V2.0.1 and Spring V5.0.5 but the code is applicable to Spring 3.x and 4.x also.  The demo is running on Cloud Foundry at <http://mvc-exceptions-v2.cfapps.io/>. About the Demo The application leads the user through 5 demo pages, highlighting different exception handling techniques:   1. A controller with @ExceptionHandler methods to handle its own exceptions 2. A contoller that throws exceptions for a global ControllerAdvice to handle 3. Using a SimpleMappingExceptionResolver to handle exceptions 4. Same as demo 3 but with the SimpleMappingExceptionResolver disabled for comparison 5. Shows how Spring Boot generates its error page   A description of the most important files in the application and how they relate to each demo can be found in the project's [README.md](http://github.com/paulc4/mvc-exceptions/blob/master/README.md).  The home web-page is [index.html](http://github.com/paulc4/mvc-exceptions/blob/master/src/main/resources/templates/index.html) which:   * Links to each demo page * Links (bottom of the page) to Spring Boot endpoints for those interested in Spring Boot.   Each demo page contains several links, all of which deliberately raise exceptions. You will need to use the back-button on your browser each time to return to the demo page.  Thanks to Spring Boot, you can run this demo as a Java application (it runs an embedded Tomcat container). To run the application, you can use one of the following (the second is thanks to the Spring Boot maven plugin):   * mvn exec:java * mvn spring-boot:run   Your choice. The home page URL will be [http://localhost:8080](http://localhost:8080/). Error Page Contents Also in the demo application I show how to create a "support-ready" error page with a stack-trace hidden in the HTML source (as a comment). Ideally support should get this information from the logs, but life isn't always ideal. Regardless, what this page does show is how the underlying error-handling method handleError creates its own ModelAndView to provide extra information in the error page. See:   * ExceptionHandlingController.handleError() on [github](http://github.com/paulc4/mvc-exceptions/blob/master/src/main/java/demo1/web/ExceptionHandlingController.java) * GlobalControllerExceptionHandler.handleError() on [github](https://github.com/paulc4/mvc-exceptions/blob/master/src/main/java/demo2/web/GlobalExceptionHandlingControllerAdvice.java)  Spring Boot and Error Handling [Spring Boot](http://spring.io/spring-boot) allows a Spring project to be setup with minimal configuration. Spring Boot creates sensible defaults automatically when it detects certain key classes and packages on the classpath. For example if it sees that you are using a Servlet environment, it sets up Spring MVC with the most commonly used view-resolvers, hander mappings and so forth. If it sees JSP and/or Thymeleaf, it sets up these view-technologies. Fallback Error Page How does Spring Boot support the default error-handling described at the beginning of this article?   1. In the event of any unhanded error, Spring Boot forwards internally to /error. 2. Boot sets up a BasicErrorController to handle any request to /error. The controller adds error information to the internal Model and returns error as the logical view name. 3. If any view-resolver(s) are configured, they will try to use a corresponding error-view. 4. Otherwise, a default error page is provided using a dedicated View object (making it independent of any view-resolution system you may be using). 5. Spring Boot sets up a BeanNameViewResolver so that /error can be mapped to a View of the same name. 6. If you look in Boot's [ErrorMvcAutoConfiguration](https://github.com/spring-projects/spring-boot/blob/master/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/web/servlet/error/ErrorMvcAutoConfiguration.java) class you will see that the defaultErrorView is returned as a bean called error. This is the View bean found by the BeanNameViewResolver.   The "Whitelabel" error page is deliberately minimal and ugly. You can override it:   1. By defining an error template - in our demo we are using Thymeleaf so the error template is in src/main/resources/templates/error.html (this location is set by the Spring Boot property spring.thymeleaf.prefix - similar properties exist for other supported server-side view technologies such as JSP or Mustache). 2. If you aren't using server-side rendering 2.1 Define your own error View as a bean called error. 2.1 Or disable Spring boot's "Whitelabel" error page by setting the property   server.error.whitelabel.enabled to false. Your container's default error page is used instead.  By convention, Spring Boot properties are normally set in application.properties or application.yml. Integration with SimpleMappingExceptionResolver What if you are already using SimpleMappingExceptionResolver to setup a default error view? Simple, use setDefaultErrorView() to define the same view that Spring Boot uses: error.  Note that in the demo, the defaultErrorView property of the SimpleMappingExceptionResolver is deliberately set not to error but to defaultErrorPage so you can see when the handler is generating the error page and when Spring Boot is responsible. Normally both would be set to error. Container-Wide Exception Handling Exceptions thrown outside the Spring Framework, such as from a servlet Filter, are also reported by Spring Boot's fallback error page.  To do this Spring Boot has to register a default error page for the container. In Servlet 2, there is an <error-page> directive that you can add to your web.xml to do this. Sadly Servlet 3 does not offer a Java API equivalent. Instead Spring Boot does the following:   * For a Jar application, with an embedded container, it registers a default error page using Container specific API. * For a Spring Boot application deployed as a traditional WAR file, a Servlet Filter is used to   catch exceptions raised further down the line and handle it. | Xử lý ngoại lệ trong Spring MVC **[KỸ THUẬT](https://spring.io/blog/category/engineering) | PAUL CHAPMAN | NGÀY 01 THÁNG 11 NĂM 2013 |**[**56 BÌNH LUẬN**](https://spring.io/blog/2013/11/01/exception-handling-in-spring-mvc/#disqus_thread)  **LƯU Ý:** [*Sửa đổi vào tháng 4 năm 2018*](https://spring.io/blog/2013/11/01/exception-handling-in-spring-mvc#recent-updates)  Spring MVC cung cấp một số cách tiếp cận miễn phí để xử lý ngoại lệ, nhưng khi dạy Spring MVC, tôi thường thấy rằng học sinh của mình bối rối hoặc không thoải mái với chúng.  Hôm nay tôi sẽ giới thiệu cho bạn nhiều tùy chọn khác nhau có sẵn. Mục tiêu của chúng tôi là *không* xử lý các ngoại lệ một cách rõ ràng trong các phương thức của Bộ điều khiển nếu có thể. Chúng là mối quan tâm xuyên suốt nên được xử lý riêng biệt bằng mã chuyên dụng.  Có ba tùy chọn: mỗi ngoại lệ, mỗi bộ điều khiển hoặc toàn cầu.  Bạn có thể tìm thấy một ứng dụng trình diễn thể hiện các điểm được thảo luận ở đây tại [*http://github.com/paulc4/mvc-Exceptions*](http://github.com/paulc4/mvc-exceptions). Xem [*Đơn đăng ký mẫu*](https://spring.io/blog/2013/11/01/exception-handling-in-spring-mvc#sample-application) bên dưới để biết chi tiết.  **LƯU Ý:** Các ứng dụng demo đã được cải tiến và cập nhật (tháng 4 năm 2018) để sử dụng Spring Boot 2.0.1 và (hy vọng) dễ sử dụng và dễ hiểu hơn. Tôi cũng đã sửa một số liên kết bị hỏng (cảm ơn bạn đã phản hồi, xin lỗi đã mất một lúc). Spring Boot [Spring Boot](http://spring.io/spring-boot) cho phép thiết lập dự án Spring với cấu hình tối thiểu và có khả năng là bạn đang sử dụng nó nếu ứng dụng của bạn chưa được vài năm tuổi.  Spring MVC không cung cấp trang lỗi mặc định (dự phòng) ngay lập tức. Cách phổ biến nhất để đặt trang lỗi mặc định luôn là SimpleMappingExceptionResolver (thực tế là kể từ Spring V1). Chúng ta sẽ thảo luận vấn đề này sau.  Tuy nhiên, Spring Boot có cung cấp trang xử lý lỗi dự phòng.  Khi khởi động, Spring Boot cố gắng tìm ánh xạ cho /error. Theo quy ước, URL kết thúc bằng /error sẽ ánh xạ tới chế độ xem logic có cùng tên: error. Trong ứng dụng demo, chế độ xem này lần lượt ánh xạ tới mẫu error.html Thymeleaf. (Nếu sử dụng JSP, nó sẽ ánh xạ tới error.jsp theo thiết lập InternalResourceViewResolver của bạn). Việc ánh xạ thực tế sẽ phụ thuộc vào những gì ViewResolver (nếu có) mà bạn hoặc Spring Boot đã thiết lập.  Nếu không tìm thấy ánh xạ trình phân giải chế độ xem cho /error, Spring Boot sẽ xác định trang lỗi dự phòng của chính nó - cái gọi là "Trang lỗi nhãn trắng" (một trang tối thiểu chỉ có thông tin trạng thái HTTP và mọi chi tiết lỗi, chẳng hạn như thông báo từ một ngoại lệ chưa được phát hiện). Trong ứng dụng mẫu, nếu bạn đổi tên mẫu error.html thành error2.html rồi khởi động lại, bạn sẽ thấy nó đang được sử dụng.  Nếu bạn đang thực hiện một yêu cầu RESTful (yêu cầu HTTP đã chỉ định loại phản hồi mong muốn không phải là HTML) thì Spring Boot trả về bản trình bày JSON của cùng thông tin lỗi mà nó đặt trong phần "Whitelabel" trang lỗi.  COPY$> curl -H "Accept: application/json" http://localhost:8080/no-such-page  {"timestamp":"2018-04-11T05:56:03.845+0000","status":404,"error":"Not Found","message":"No message available","path":"/no-such-page"}  Spring Boot cũng thiết lập một trang lỗi mặc định cho vùng chứa, tương đương với <error-page> chỉ thị trong web.xml (mặc dù được triển khai rất khác nhau). Các ngoại lệ được đưa ra bên ngoài khung công tác Spring MVC, chẳng hạn như từ Bộ lọc servlet, vẫn được trang lỗi dự phòng Spring Boot báo cáo. Ứng dụng mẫu cũng cho thấy một ví dụ về điều này.  Bạn có thể tìm thấy cuộc thảo luận sâu hơn về xử lý lỗi Spring Boot ở cuối bài viết này.  Phần còn lại của bài viết này áp dụng bất kể bạn đang sử dụng Spring có hay không có Spring Boot.  Các nhà phát triển REST thiếu kiên nhẫn có thể chọn chuyển thẳng đến [*phần*](https://spring.io/blog/2013/11/01/exception-handling-in-spring-mvc#errors-and-rest) về các phản hồi lỗi REST tùy chỉnh. Tuy nhiên, sau đó họ nên đọc toàn bộ bài viết vì hầu hết bài viết đều áp dụng như nhau cho tất cả các ứng dụng web, REST hoặc các ứng dụng khác. Sử dụng mã trạng thái HTTP Thông thường, bất kỳ ngoại lệ chưa được xử lý nào được đưa ra khi xử lý yêu cầu web đều khiến máy chủ trả về phản hồi HTTP 500. Tuy nhiên, bất kỳ ngoại lệ nào bạn tự viết đều có thể được chú thích bằng chú thích @ResponseStatus (hỗ trợ tất cả các mã trạng thái HTTP được xác định bởi đặc tả HTTP). Khi một ngoại lệ có chú thích được ném ra từ một phương thức điều khiển và không được xử lý ở nơi khác, nó sẽ tự động khiến phản hồi HTTP thích hợp được trả về với status-code đã được chỉ định.  Ví dụ: đây là một ngoại lệ đối với đơn hàng bị thiếu.  COPY @ResponseStatus(value=HttpStatus.NOT\_FOUND, reason="No such Order") // 404  public class OrderNotFoundException extends RuntimeException {  // ...  }  Và đây là một controller method sử dụng nó:  COPY @RequestMapping(value="/orders/{id}", method=GET)  public String showOrder(@PathVariable("id") long id, Model model) {  Order order = orderRepository.findOrderById(id);  if (order == null) throw new OrderNotFoundException(id);  model.addAttribute(order);  return "orderDetail";  }  Phản hồi HTTP 404 quen thuộc sẽ được trả về nếu URL được xử lý bằng phương pháp này bao gồm id đơn hàng không xác định. Xử lý ngoại lệ dựa trên bộ điều khiểnSử dụng @ExceptionHandler Bạn có thể thêm các phương thức (@ExceptionHandler) bổ sung vào bất kỳ bộ điều khiển nào để xử lý cụ thể các ngoại lệ được đưa ra bởi các phương thức xử lý yêu cầu (@RequestMapping) trong cùng một bộ điều khiển. Những phương pháp như vậy có thể:   1. Xử lý các ngoại lệ mà không có chú thích @ResponseStatus (thường là các ngoại lệ được xác định trước mà bạn không viết) 2. Chuyển hướng người dùng đến chế độ xem lỗi chuyên dụng 3. Xây dựng phản hồi lỗi hoàn toàn tùy chỉnh   Bộ điều khiển sau đây thể hiện ba tùy chọn sau:  COPY@Controller  public class ExceptionHandlingController {  // @RequestHandler methods  ...  // Exception handling methods  // Convert a predefined exception to an HTTP Status code  @ResponseStatus(value=HttpStatus.CONFLICT,  reason="Data integrity violation") // 409  @ExceptionHandler(DataIntegrityViolationException.class)  public void conflict() {  // Nothing to do  }  // Specify name of a specific view that will be used to display the error:  @ExceptionHandler({SQLException.class,DataAccessException.class})  public String databaseError() {  // Nothing to do. Returns the logical view name of an error page, passed  // to the view-resolver(s) in usual way.  // Note that the exception is NOT available to this view (it is not added  // to the model) but see "Extending ExceptionHandlerExceptionResolver"  // below.  return "databaseError";  }  // Total control - setup a model and return the view name yourself. Or  // consider subclassing ExceptionHandlerExceptionResolver (see below).  @ExceptionHandler(Exception.class)  public ModelAndView handleError(HttpServletRequest req, Exception ex) {  logger.error("Request: " + req.getRequestURL() + " raised " + ex);  ModelAndView mav = new ModelAndView();  mav.addObject("exception", ex);  mav.addObject("url", req.getRequestURL());  mav.setViewName("error");  return mav;  }  }  Trong bất kỳ phương pháp nào trong số này, bạn có thể chọn thực hiện xử lý bổ sung - ví dụ phổ biến nhất là ghi lại ngoại lệ.  Các phương thức xử lý có chữ ký linh hoạt nên bạn có thể chuyển vào các đối tượng rõ ràng liên quan đến servlet như HttpServletRequest, HttpServletResponse, HttpSession và/hoặc Principle.  **Lưu ý quan trọng:** Model có thể **không** là tham số của bất kỳ phương thức @ExceptionHandler nào. Thay vào đó, hãy thiết lập một mô hình bên trong phương thức bằng cách sử dụng ModelAndView như được hiển thị bởi handleError() ở trên. Ngoại lệ và lượt xem Hãy cẩn thận khi thêm ngoại lệ vào mô hình. Người dùng của bạn không muốn xem các trang web chứa chi tiết ngoại lệ Java và dấu vết ngăn xếp. Bạn có thể có các chính sách bảo mật quy định rõ ràng cấm đưa bất kỳ thông tin ngoại lệ nào vào trang lỗi. Một lý do khác để đảm bảo bạn ghi đè trang lỗi nhãn trắng của Spring Boot.  Đảm bảo các trường hợp ngoại lệ được ghi lại một cách hữu ích để nhóm hỗ trợ và phát triển của bạn có thể phân tích chúng sau sự kiện.  Xin hãy nhớ những điều sau đây có thể thuận tiện nhưng đó không phải là ***không*** cách thực hành tốt nhất trong sản xuất.  Có thể hữu ích khi ẩn các chi tiết ngoại lệ trong trang nguồn dưới dạng nhận xét để hỗ trợ thử nghiệm. Nếu sử dụng JSP, bạn có thể làm điều gì đó như thế này để xuất ngoại lệ và dấu vết ngăn xếp tương ứng (sử dụng <div> ẩn là một tùy chọn khác).  COPY <h1>Error Page</h1>  <p>Application has encountered an error. Please contact support on ...</p>  <!--  Failed URL: ${url}  Exception: ${exception.message}  <c:forEach items="${exception.stackTrace}" var="ste"> ${ste}  </c:forEach>  -->  Để biết thông tin tương đương về Thymeleaf, hãy xem [support.html](https://github.com/paulc4/mvc-exceptions/blob/master/src/main/resources/templates/support.html) trong ứng dụng demo. Kết quả trông như thế này. Xử lý ngoại lệ toàn cầuSử dụng các lớp @ControllerAdvice Lời khuyên về bộ điều khiển cho phép bạn sử dụng chính xác các kỹ thuật xử lý ngoại lệ giống nhau nhưng áp dụng chúng trên toàn bộ ứng dụng chứ không chỉ cho một bộ điều khiển riêng lẻ. Bạn có thể coi chúng như một thiết bị chặn được điều khiển bằng chú thích.  Bất kỳ lớp nào được chú thích bằng @ControllerAdvice đều trở thành lời khuyên về bộ điều khiển và ba loại phương thức được hỗ trợ:   * Phương pháp xử lý ngoại lệ được chú thích bằng @ExceptionHandler. * Các phương pháp nâng cao mô hình (để thêm dữ liệu bổ sung vào mô hình) được chú thích bằng   @ModelAttributekhông có sẵn cho các chế độ xem xử lý ngoại lệ.   * Các phương thức khởi tạo chất kết dính (được sử dụng để định cấu hình xử lý biểu mẫu) được chú thích bằng   @InitBinder.  Chúng ta sẽ chỉ xem xét việc xử lý ngoại lệ - tìm kiếm [hướng dẫn trực tuyến](https://docs.spring.io/spring/docs/current/spring-framework-reference/web.html#mvc-controller) để biết thêm về các phương pháp @ControllerAdvice .  Bất kỳ trình xử lý ngoại lệ nào bạn thấy ở trên đều có thể được xác định trong lớp lời khuyên về bộ điều khiển - nhưng giờ đây chúng áp dụng cho các ngoại lệ được đưa ra từ bất kỳ bộ điều khiển. Đây là một ví dụ đơn giản:  COPY@ControllerAdvice  class GlobalControllerExceptionHandler {  @ResponseStatus(HttpStatus.CONFLICT) // 409  @ExceptionHandler(DataIntegrityViolationException.class)  public void handleConflict() {  // Nothing to do  }  }  Nếu bạn muốn có một trình xử lý mặc định cho bất kỳ ngoại lệ nào, thì có một chút khó khăn. Bạn cần đảm bảo các trường hợp ngoại lệ có chú thích được khung xử lý. Mã trông như thế này:  COPY@ControllerAdvice  class GlobalDefaultExceptionHandler {  public static final String DEFAULT\_ERROR\_VIEW = "error";  @ExceptionHandler(value = Exception.class)  public ModelAndView  defaultErrorHandler(HttpServletRequest req, Exception e) throws Exception {  // If the exception is annotated with @ResponseStatus rethrow it and let  // the framework handle it - like the OrderNotFoundException example  // at the start of this post.  // AnnotationUtils is a Spring Framework utility class.  if (AnnotationUtils.findAnnotation  (e.getClass(), ResponseStatus.class) != null)  throw e;  // Otherwise setup and send the user to a default error-view.  ModelAndView mav = new ModelAndView();  mav.addObject("exception", e);  mav.addObject("url", req.getRequestURL());  mav.setViewName(DEFAULT\_ERROR\_VIEW);  return mav;  }  } Đi sâu hơnTrình xử lý ngoại lệResolver Bất kỳ Spring Bean nào được khai báo trong ngữ cảnh ứng dụng DispatcherServlet' mà triển khai HandlerExceptionResolver sẽ được sử dụng để chặn và xử lý bất kỳ ngoại lệ nào được nêu ra trong Hệ thống MVC và không được Bộ điều khiển xử lý. Giao diện trông như thế này:  COPYpublic interface HandlerExceptionResolver {  ModelAndView resolveException(HttpServletRequest request,  HttpServletResponse response, Object handler, Exception ex);  }  handler đề cập đến bộ điều khiển đã tạo ra ngoại lệ (hãy nhớ rằng các phiên bản @Controller chỉ là một loại trình xử lý được Spring MVC hỗ trợ. Ví dụ: HttpInvokerExporter và WebFlow Executor cũng là các loại trình xử lý).  Đằng sau hậu trường, MVC tạo ra ba trình phân giải như vậy theo mặc định. Chính những trình phân giải này sẽ thực hiện các hành vi được thảo luận ở trên:   * ExceptionHandlerExceptionResolver khớp các ngoại lệ chưa được phát hiện với các phương thức @ExceptionHandler phù hợp trên cả trình xử lý (bộ điều khiển) và trên bất kỳ lời khuyên về bộ điều khiển nào. * ResponseStatusExceptionResolver tìm kiếm các ngoại lệ chưa được phát hiện được chú thích bởi @ResponseStatus (như được mô tả trong Phần 1) * DefaultHandlerExceptionResolverchuyển đổi các ngoại lệ tiêu chuẩn của Spring và chuyển chúng thành Mã trạng thái HTTP (Tôi chưa đề cập đến điều này ở trên vì nó là nội bộ của Spring MVC).   Chúng được xâu chuỗi và xử lý theo thứ tự được liệt kê - bên trong Spring tạo ra một Bean chuyên dụng (HandlerExceptionResolverComposite) để thực hiện việc này.  Lưu ý rằng chữ ký phương thức của resolveException không bao gồm Model. Đây là lý do tại sao không thể chèn các phương thức @ExceptionHandler vào mô hình.  Bạn có thể, nếu muốn, triển khai HandlerExceptionResolver của riêng mình để thiết lập hệ thống xử lý ngoại lệ tùy chỉnh của riêng bạn. Trình xử lý thường triển khai giao diện Ordered của Spring để bạn có thể xác định thứ tự chạy của trình xử lý. SimpleMappingExceptionResolver Spring từ lâu đã cung cấp cách triển khai HandlerExceptionResolver đơn giản nhưng thuận tiện mà bạn có thể thấy đã được sử dụng trong ứng dụng của mình - SimpleMappingExceptionResolver. Nó cung cấp các tùy chọn để:   * Ánh xạ tên lớp ngoại lệ để xem tên - chỉ cần chỉ định tên lớp, không cần gói. * Chỉ định trang lỗi (dự phòng) mặc định cho bất kỳ ngoại lệ nào không được xử lý ở bất kỳ nơi nào khác * Ghi nhật ký tin nhắn (điều này không được bật theo mặc định). * Đặt tên của thuộc tính exception để thêm vào Mô hình để có thể sử dụng nó trong Chế độ xem   (chẳng hạn như một JSP). Theo mặc định, thuộc tính này được đặt tên là exception. Đặt thành null để tắt. Hãy nhớ rằng các chế độ xem được trả về từ các phương thức @ExceptionHandler không có quyền truy cập vào ngoại lệ nhưng các chế độ xem được xác định là SimpleMappingExceptionResolver làm.  Đây là một cấu hình điển hình sử dụng Cấu hình Java:  COPY@Configuration  @EnableWebMvc // Optionally setup Spring MVC defaults (if you aren't using  // Spring Boot & haven't specified @EnableWebMvc elsewhere)  public class MvcConfiguration extends WebMvcConfigurerAdapter {  @Bean(name="simpleMappingExceptionResolver")  public SimpleMappingExceptionResolver  createSimpleMappingExceptionResolver() {  SimpleMappingExceptionResolver r =  new SimpleMappingExceptionResolver();  Properties mappings = new Properties();  mappings.setProperty("DatabaseException", "databaseError");  mappings.setProperty("InvalidCreditCardException", "creditCardError");  r.setExceptionMappings(mappings); // None by default  r.setDefaultErrorView("error"); // No default  r.setExceptionAttribute("ex"); // Default is "exception"  r.setWarnLogCategory("example.MvcLogger"); // No default  return r;  }  ...  }  Hoặc sử dụng Cấu hình XML:  COPY <bean id="simpleMappingExceptionResolver" class=  "org.springframework.web.servlet.handler.SimpleMappingExceptionResolver">  <property name="exceptionMappings">  <map>  <entry key="DatabaseException" value="databaseError"/>  <entry key="InvalidCreditCardException" value="creditCardError"/>  </map>  </property>  <!-- See note below on how this interacts with Spring Boot -->  <property name="defaultErrorView" value="error"/>  <property name="exceptionAttribute" value="ex"/>  <!-- Name of logger to use to log exceptions. Unset by default,  so logging is disabled unless you set a value. -->  <property name="warnLogCategory" value="example.MvcLogger"/>  </bean>  Thuộc tính defaultErrorView đặc biệt hữu ích vì nó đảm bảo mọi ngoại lệ chưa được phát hiện sẽ tạo ra một trang lỗi do ứng dụng xác định phù hợp. (Mặc định cho hầu hết các máy chủ ứng dụng là hiển thị dấu vết ngăn xếp Java - thứ mà người dùng của bạn không bao giờ nhìn thấy). Spring Boot cung cấp một cách khác để thực hiện điều tương tự với "nhãn trắng" trang lỗi. Mở rộng SimpleMappingExceptionResolver Việc mở rộng SimpleMappingExceptionResolver là khá phổ biến vì một số lý do:   * Bạn có thể sử dụng hàm tạo để đặt trực tiếp các thuộc tính - ví dụ: để bật ghi nhật ký ngoại lệ và đặt trình ghi nhật ký để sử dụng * Ghi đè thông điệp tường trình mặc định bằng cách ghi đè buildLogMessage. Việc triển khai mặc định luôn trả về văn bản cố định này:   *Việc thực thi trình xử lý dẫn đến ngoại lệ*   * Để cung cấp thêm thông tin cho chế độ xem lỗi bằng cách ghi đèdoResolveException   Ví dụ:  COPYpublic class MyMappingExceptionResolver extends SimpleMappingExceptionResolver {  public MyMappingExceptionResolver() {  // Enable logging by providing the name of the logger to use  setWarnLogCategory(MyMappingExceptionResolver.class.getName());  }  @Override  public String buildLogMessage(Exception e, HttpServletRequest req) {  return "MVC exception: " + e.getLocalizedMessage();  }  @Override  protected ModelAndView doResolveException(HttpServletRequest req,  HttpServletResponse resp, Object handler, Exception ex) {  // Call super method to get the ModelAndView  ModelAndView mav = super.doResolveException(req, resp, handler, ex);  // Make the full URL available to the view - note ModelAndView uses  // addObject() but Model uses addAttribute(). They work the same.  mav.addObject("url", request.getRequestURL());  return mav;  }  }  Mã này có trong ứng dụng demo dưới dạng [ExampleSimpleMappingExceptionResolver](https://github.com/paulc4/mvc-exceptions/blob/master/src/main/java/demo/example/ExampleSimpleMappingExceptionResolver.java) Mở rộng ExceptionHandlerExceptionResolver Cũng có thể mở rộng ExceptionHandlerExceptionResolver và ghi đè lên nó doResolveHandlerMethodException theo cách tương tự. Nó có chữ ký gần như giống nhau (nó chỉ lấy HandlerMethod mới thay vì Handler).  Để đảm bảo nó được sử dụng, hãy đặt thuộc tính thứ tự được kế thừa (ví dụ: trong hàm tạo của lớp mới của bạn) thành giá trị nhỏ hơn MAX\_INT để nó chạy trước phiên bản ExceptionHandlerExceptionResolver mặc định (việc tạo phiên bản trình xử lý của riêng bạn sẽ dễ dàng hơn là cố gắng sửa đổi/thay thế phiên bản do Spring tạo). Xem [ExampleExceptionHandlerExceptionResolver](https://github.com/paulc4/mvc-exceptions/blob/master/src/main/java/demo/example/ExampleExceptionHandlerExceptionResolver.java) trong ứng dụng demo để biết thêm. Lỗi và REST Các yêu cầu RESTful GET cũng có thể tạo ra ngoại lệ và chúng tôi đã thấy cách có thể trả về mã phản hồi Lỗi HTTP tiêu chuẩn. Tuy nhiên, nếu bạn muốn trả về thông tin về lỗi thì sao? Điều này rất dễ thực hiện. Đầu tiên xác định một lớp lỗi:  COPYpublic class ErrorInfo {  public final String url;  public final String ex;  public ErrorInfo(String url, Exception ex) {  this.url = url;  this.ex = ex.getLocalizedMessage();  }  }  Bây giờ chúng ta có thể trả về một phiên bản từ một trình xử lý dưới dạng @ResponseBody như thế này:  COPY@ResponseStatus(HttpStatus.BAD\_REQUEST)  @ExceptionHandler(MyBadDataException.class)  @ResponseBody ErrorInfo  handleBadRequest(HttpServletRequest req, Exception ex) {  return new ErrorInfo(req.getRequestURL(), ex);  } Sử dụng cái gì khi nào? Như thường lệ, Spring thích đưa ra cho bạn sự lựa chọn, vậy bạn nên làm gì? Dưới đây là một số quy tắc của ngón tay cái. Tuy nhiên, nếu bạn thích cấu hình XML hoặc Chú thích thì cũng không sao.   * Đối với các trường hợp ngoại lệ bạn viết, hãy cân nhắc thêm @ResponseStatus vào chúng. * Đối với tất cả các trường hợp ngoại lệ khác, hãy triển khai phương thức @ExceptionHandler trên lớp @ControllerAdvice hoặc sử dụng một phiên bản của SimpleMappingExceptionResolver. Có thể bạn đã định cấu hình SimpleMappingExceptionResolver cho ứng dụng của mình rồi, trong trường hợp đó, việc thêm các lớp ngoại lệ mới vào ứng dụng có thể dễ dàng hơn việc triển khai @ControllerAdvice. * Để xử lý ngoại lệ dành riêng cho Bộ điều khiển, hãy thêm các phương thức @ExceptionHandler vào bộ điều khiển của bạn. * **Cảnh báo:** Hãy cẩn thận khi kết hợp quá nhiều tùy chọn này trong cùng một ứng dụng. Nếu cùng một ngoại lệ có thể được trao theo nhiều cách, bạn có thể không nhận được hành vi như mong muốn. @ExceptionHandler các phương thức trên Bộ điều khiển luôn được chọn trước các phương thức trên bất kỳ phiên bản @ControllerAdvice nào. Đó là *không xác định* những lời khuyên của người kiểm soát đơn hàng sẽ được xử lý.  Ứng dụng mẫu Bạn có thể tìm thấy ứng dụng trình diễn tại [github](http://github.com/paulc4/mvc-exceptions). Nó sử dụng Spring Boot và Thymeleaf để xây dựng một ứng dụng web đơn giản.  Ứng dụng này đã được sửa đổi hai lần (tháng 10 năm 2014, tháng 4 năm 2018) và (hy vọng) tốt hơn và dễ hiểu hơn. Các nguyên tắc cơ bản vẫn giữ nguyên. Nó sử dụng Spring Boot V2.0.1 và Spring V5.0.5 nhưng mã này cũng có thể áp dụng cho Spring 3.x và 4.x.  Bản demo đang chạy trên Cloud Foundry tại[http://mvc-Exceptions-v2.cfapps.io/](http://mvc-exceptions-v2.cfapps.io/). Giới thiệu về bản demo Ứng dụng dẫn người dùng qua 5 trang demo, nêu bật các kỹ thuật xử lý ngoại lệ khác nhau:   1. Bộ điều khiển có @ExceptionHandler phương thức để xử lý các ngoại lệ của chính nó 2. Bộ điều khiển đưa ra các ngoại lệ để ControllerAdvice toàn cầu xử lý 3. Sử dụng SimpleMappingExceptionResolver để xử lý các ngoại lệ 4. Tương tự như bản demo 3 nhưng SimpleMappingExceptionResolver bị tắt để so sánh 5. Hiển thị cách Spring Boot tạo trang lỗi của nó   Bạn có thể tìm thấy mô tả về các tệp quan trọng nhất trong ứng dụng và cách chúng liên quan đến từng bản demo trong [README.md](http://github.com/paulc4/mvc-exceptions/blob/master/README.md)  Trang chủ là [index.html](http://github.com/paulc4/mvc-exceptions/blob/master/src/main/resources/templates/index.html) trong đó:   * Liên kết đến từng trang demo * Các liên kết (cuối trang) tới điểm cuối Spring Boot dành cho những ai quan tâm đến Spring Boot.   Mỗi trang demo chứa một số liên kết, tất cả đều cố tình đưa ra các ngoại lệ. Bạn sẽ cần sử dụng nút quay lại trên trình duyệt của mình mỗi lần để quay lại trang demo.  Nhờ Spring Boot, bạn có thể chạy bản demo này dưới dạng một ứng dụng Java (nó chạy bộ chứa Tomcat được nhúng). Để chạy ứng dụng, bạn có thể sử dụng một trong các cách sau (thứ hai là nhờ plugin Spring Boot maven):   * mvn exec:java * mvn spring-boot:run   Lựa chọn của bạn. URL trang chủ sẽ là[http://localhost:8080](http://localhost:8080/). Nội dung trang lỗi Ngoài ra, trong ứng dụng demo, tôi cũng trình bày cách tạo "sẵn sàng hỗ trợ" trang lỗi có dấu vết ngăn xếp ẩn trong nguồn HTML (dưới dạng nhận xét). Lý tưởng nhất là bộ phận hỗ trợ nên lấy thông tin này từ nhật ký, nhưng cuộc sống không phải lúc nào cũng lý tưởng. Bất kể điều gì trang này làm thể hiện là cách phương thức xử lý lỗi cơ bản handleError tạo ra < của riêng nó a i=4> để cung cấp thêm thông tin trong trang lỗi. Xem:ModelAndView   * ExceptionHandlingController.handleError() trên [github](http://github.com/paulc4/mvc-exceptions/blob/master/src/main/java/demo1/web/ExceptionHandlingController.java) * GlobalControllerExceptionHandler.handleError() trên [github](https://github.com/paulc4/mvc-exceptions/blob/master/src/main/java/demo2/web/GlobalExceptionHandlingControllerAdvice.java)  Khởi động mùa xuân và xử lý lỗi [Spring Boot](http://spring.io/spring-boot) cho phép thiết lập dự án Spring với cấu hình tối thiểu. Spring Boot tự động tạo các giá trị mặc định hợp lý khi phát hiện các lớp và gói chính nhất định trên đường dẫn lớp. Ví dụ: nếu nó thấy rằng bạn đang sử dụng môi trường Servlet, nó sẽ thiết lập Spring MVC với các trình phân giải chế độ xem, ánh xạ trình xử lý, v.v. được sử dụng phổ biến nhất. Nếu nó thấy JSP và/hoặc Thymeleaf, nó sẽ thiết lập các công nghệ xem này. Trang lỗi dự phòng Spring Boot hỗ trợ xử lý lỗi mặc định được mô tả ở đầu bài viết này như thế nào?   1. Trong trường hợp có bất kỳ lỗi nào xảy ra, Spring Boot sẽ chuyển tiếp nội bộ tới /error. 2. Boot thiết lập BasicErrorController để xử lý mọi yêu cầu tới /error. Bộ điều khiển thêm thông tin lỗi vào Mô hình bên trong và trả về error làm tên chế độ xem logic. 3. Nếu bất kỳ (các) trình phân giải chế độ xem nào được định cấu hình, chúng sẽ cố gắng sử dụng chế độ xem lỗi tương ứng. 4. Nếu không, trang lỗi mặc định sẽ được cung cấp bằng đối tượng View chuyên dụng (làm cho nó độc lập với bất kỳ hệ thống phân giải chế độ xem nào mà bạn có thể đang sử dụng). 5. Spring Boot thiết lập BeanNameViewResolver để /error có thể được ánh xạ tới View có cùng tên.< /span> 6. Nếu bạn nhìn vào lớp [ErrorMvcAutoConfiguration](https://github.com/spring-projects/spring-boot/blob/master/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/web/servlet/error/ErrorMvcAutoConfiguration.java) của Boot's bạn sẽ thấy rằng defaultErrorView được trả về dưới dạng một hạt đậu có tên error. Đây là View Bean được tìm thấy bởi BeanNameViewResolver.   "Nhãn trắng" trang lỗi được cố tình tối thiểu và xấu xí. Bạn có thể ghi đè lên nó:   1. Bằng cách xác định mẫu lỗi - trong bản demo của chúng tôi, chúng tôi đang sử dụng Thymeleaf nên mẫu lỗi nằm ở src/main/resources/templates/error.html (vị trí này được đặt bởi thuộc tính Spring Boot spring.thymeleaf.prefix - các thuộc tính tương tự tồn tại đối với các công nghệ xem phía máy chủ được hỗ trợ khác như JSP hoặc Mustache). 2. Nếu bạn không sử dụng kết xuất phía máy chủ 2.1 Xác định lỗi của riêng bạn Xem dưới dạng đậu có tên error. 2.1 Hoặc tắt "Nhãn trắng" của Spring boot trang lỗi bằng cách đặt thuộc tính   server.error.whitelabel.enabled tới false. Thay vào đó, trang lỗi mặc định của vùng chứa của bạn sẽ được sử dụng.  Theo quy ước, các thuộc tính của Spring Boot thường được đặt trong application.properties hoặc application.yml. Tích hợp với SimpleMappingExceptionResolver Điều gì sẽ xảy ra nếu bạn đang sử dụng SimpleMappingExceptionResolver để thiết lập mặc định xem lỗi? Đơn giản, hãy sử dụng setDefaultErrorView() để xác định cùng một chế độ xem mà Spring Boot sử dụng: error.  Lưu ý rằng trong bản demo, thuộc tính defaultErrorView của SimpleMappingExceptionResolver được cố tình đặt không phải thành error mà là < /span>. sẽ được đặt thành cả haidefaultErrorPage để bạn có thể biết khi nào trình xử lý tạo ra trang lỗi và khi nào Spring Boot chịu trách nhiệm. Thông thường error Xử lý ngoại lệ trên toàn vùng chứa Các ngoại lệ được đưa ra bên ngoài Spring Framework, chẳng hạn như từ Bộ lọc servlet, cũng được báo cáo bởi trang lỗi dự phòng của Spring Boot.  Để thực hiện việc này Spring Boot phải đăng ký một trang lỗi mặc định cho container. Trong Servlet 2, có một lệnh <error-page> mà bạn có thể thêm vào web.xml của mình để thực hiện việc này. Đáng buồn là Servlet 3 không cung cấp API Java tương đương. Thay vào đó Spring Boot thực hiện như sau:   * Đối với ứng dụng Jar, với một vùng chứa được nhúng, nó sẽ đăng ký một trang lỗi mặc định bằng API dành riêng cho Vùng chứa. * Đối với ứng dụng Spring Boot được triển khai dưới dạng tệp WAR truyền thống, Bộ lọc Servlet được sử dụng để   bắt các ngoại lệ được nâng cao hơn nữa và xử lý nó. |