Spring MVC Exception Handling – @ExceptionHandler, @ControllerAdvice, HandlerExceptionResolver, JSON Response Example

Any web application requires good design for exception handling because we don't want to serve container generated page when any unhandled exception is thrown by our application.

Having a well defined exception handling approach is a huge plus point for any web application framework, that being said Spring MVC framework delivers well when it comes to exception and error handling in our web applications.

Spring MVC Framework provides following ways to help us achieving robust exception handling.

1. **Controller Based** – We can define exception handler methods in our controller classes. All we need is to annotate these methods with `@ExceptionHandler` annotation. This annotation takes Exception class as argument. So if we have defined one of these for Exception class, then all the exceptions thrown by our request handler method will have handled.
   These exception handler methods are just like other request handler methods and we can build error response and respond with different error page. We can also send JSON error response, that we will look later on in our example.

   If there are multiple exception handler methods defined, then handler method that is closest to the Exception class is used. For example, if we have two handler methods defined for IOException and Exception and our request handler method throws IOException, then handler method for IOException will get executed.

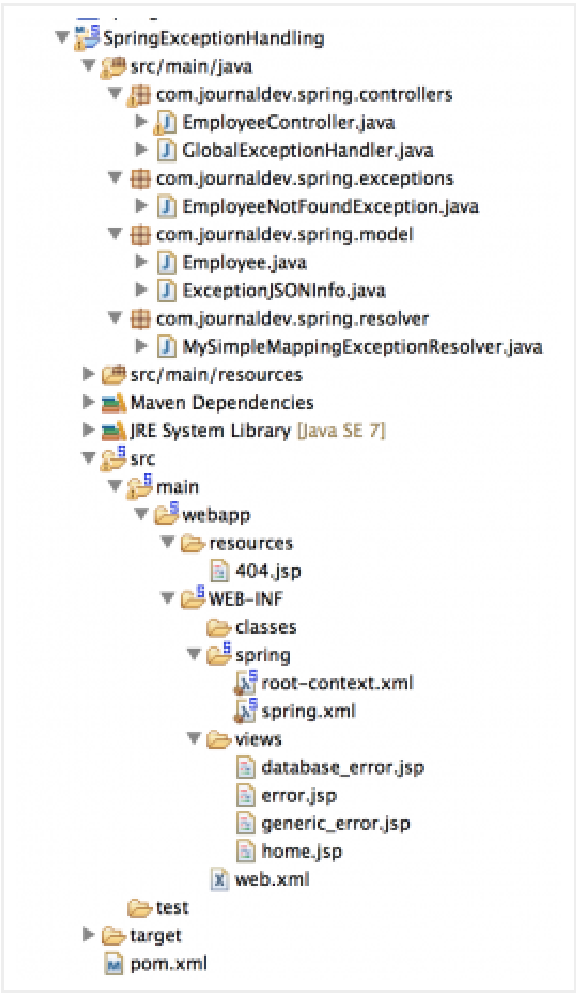
2. **Global Exception Handler** – Exception Handling is a cross-cutting concern, it should be done for all the pointcuts in our application. We have already looked into Spring AOP and that's why Spring provides `@ControllerAdvice` annotation that we can use with any class to define our global exception handler.
   The handler methods in Global Controller Advice is same as Controller based exception handler methods and used when controller class is not able to handle the exception.

3. **HandlerExceptionResolver implementation** – For generic exceptions, most of the times we serve static pages. Spring Framework provides `HandlerExceptionResolver` interface that we can implement to create global exception handler. The reason behind this additional way to define global exception handler is that Spring framework also provides default implementation classes that we can define in our spring bean configuration file to get spring framework

exception handling benefits.
`SimpleMappingExceptionResolver` is the default implementation class, it allows us to configure exceptionMappings where we can specify which resource to use for a particular exception. We can also override it to create our own global handler with our application specific changes, such as logging of exception messages.

Let's create a Spring MVC project where we will look into the implementation of Controller based, AOP Based and Exception Resolver Based exception and error handling approaches. We will also write a exception handler method that will return JSON response, so if you are new to JSON in Spring, read Spring Restful JSON Tutorial.

Our final project will look like below image, we will look at all the components of our application one by one.



```
SpringExceptionHandling
  src/main/java
    com.journaldev.spring.controllers
      EmployeeController.java
      GlobalExceptionHandler.java
    com.journaldev.spring.exceptions
      EmployeeNotFoundException.java
    com.journaldev.spring.model
      Employee.java
      ExceptionJSONInfo.java
    com.journaldev.spring.resolver
      MySimpleMappingExceptionResolver.java
  src/main/resources
  Maven Dependencies
  JRE System Library [Java SE 7]
  src
    main
      webapp
        resources
          404.jsp
        WEB-INF
          classes
          spring
            root-context.xml
            spring.xml
          views
            database_error.jsp
            error.jsp
            generic_error.jsp
            home.jsp
          web.xml
    test
  target
  pom.xml
```

Spring Maven Dependencies

Apart from standard Spring MVC dependencies, we would also need Jackson JSON API dependency for JSON support.

Our final pom.xml file looks like below.

pom.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apach
    <modelVersion>4.0.0</modelVersion>
    <groupId>com.journaldev.spring</groupId>
    <artifactId>SpringExceptionHandling</artifactId>
    <name>SpringExceptionHandling</name>
    <packaging>war</packaging>
    <version>1.0.0-BUILD-SNAPSHOT</version>
    <properties>
        <java-version>1.6</java-version>
        <org.springframework-version>4.0.2.RELEASE</org.springframework-vers
        <org.aspectj-version>1.7.4</org.aspectj-version>
        <org.slf4j-version>1.7.5</org.slf4j-version>
        <jackson.databind-version>2.2.3</jackson.databind-version>
    </properties>
    <dependencies>
        <!-- Jackson -->
        <dependency>
            <groupId>com.fasterxml.jackson.core</groupId>
            <artifactId>jackson-databind</artifactId>
            <version>${jackson.databind-version}</version>
        </dependency>
        <!-- Spring -->
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-context</artifactId>
            <version>${org.springframework-version}</version>
            <exclusions>
                <!-- Exclude Commons Logging in favor of SLF4j -->
                <exclusion>
                    <groupId>commons-logging</groupId>
                    <artifactId>commons-logging</artifactId>
                </exclusion>
            </exclusions>
        </dependency>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-webmvc</artifactId>
            <version>${org.springframework-version}</version>
        </dependency>

        <!-- AspectJ -->
        <dependency>
            <groupId>org.aspectj</groupId>
            <artifactId>aspectjrt</artifactId>
            <version>${org.aspectj-version}</version>
        </dependency>

        <!-- Logging -->
        <dependency>
            <groupId>org.slf4j</groupId>
            <artifactId>slf4j-api</artifactId>
            <version>${org.slf4j-version}</version>
        </dependency>
        <dependency>
            <groupId>org.slf4j</groupId>
            <artifactId>jcl-over-slf4j</artifactId>
            <version>${org.slf4j-version}</version>
            <scope>runtime</scope>
        </dependency>
        <dependency>
            <groupId>org.slf4j</groupId>
            <artifactId>slf4j-log4j12</artifactId>
            <version>${org.slf4j-version}</version>
            <scope>runtime</scope>
        </dependency>
        <dependency>
            <groupId>log4j</groupId>
            <artifactId>log4j</artifactId>
            <version>1.2.15</version>
            <exclusions>
                <exclusion>
                    <groupId>javax.mail</groupId>
                    <artifactId>mail</artifactId>
                </exclusion>
                <exclusion>
                    <groupId>javax.jms</groupId>
                    <artifactId>jms</artifactId>
                </exclusion>
                <exclusion>
                    <groupId>com.sun.jdmk</groupId>
                    <artifactId>jmxtools</artifactId>
                </exclusion>
                <exclusion>
                    <groupId>com.sun.jmx</groupId>
                    <artifactId>jmxri</artifactId>
                </exclusion>
            </exclusions>
            <scope>runtime</scope>
        </dependency>

        <!-- @Inject -->
        <dependency>
            <groupId>javax.inject</groupId>
            <artifactId>javax.inject</artifactId>
            <version>1</version>
        </dependency>

        <!-- Servlet -->
        <dependency>
            <groupId>javax.servlet</groupId>
            <artifactId>servlet-api</artifactId>
            <version>2.5</version>
            <scope>provided</scope>
        </dependency>
        <dependency>
            <groupId>javax.servlet.jsp</groupId>
            <artifactId>jsp-api</artifactId>
            <version>2.1</version>
            <scope>provided</scope>
        </dependency>
        <dependency>
            <groupId>javax.servlet</groupId>
            <artifactId>jstl</artifactId>
            <version>1.2</version>
        </dependency>

        <!-- Test -->
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>4.7</version>
            <scope>test</scope>
        </dependency>
    </dependencies>
    <build>
        <plugins>
            <plugin>
                <artifactId>maven-eclipse-plugin</artifactId>
```

```
                    <version>2.9</version>
                    <configuration>
                        <additionalProjectnatures>
                            <projectnature>org.springframework.ide.eclipse.core.
                        </additionalProjectnatures>
                        <additionalBuildcommands>
                            <buildcommand>org.springframework.ide.eclipse.core.s
                        </additionalBuildcommands>
                        <downloadSources>true</downloadSources>
                        <downloadJavadocs>true</downloadJavadocs>
                    </configuration>
                </plugin>
                <plugin>
                    <groupId>org.apache.maven.plugins</groupId>
                    <artifactId>maven-compiler-plugin</artifactId>
                    <version>2.5.1</version>
                    <configuration>
                        <source>1.6</source>
                        <target>1.6</target>
                        <compilerArgument>-Xlint:all</compilerArgument>
                        <showWarnings>true</showWarnings>
                        <showDeprecation>true</showDeprecation>
                    </configuration>
                </plugin>
                <plugin>
                    <groupId>org.codehaus.mojo</groupId>
                    <artifactId>exec-maven-plugin</artifactId>
                    <version>1.2.1</version>
                    <configuration>
                        <mainClass>org.test.int1.Main</mainClass>
                    </configuration>
                </plugin>
            </plugins>
        </build>
    </project>
```

I have updated Spring Framework, AspectJ, Jackson and slf4j versions to use the latest one.

Deployment Descriptor

Our web.xml file looks like below.

web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.5" xmlns="http://java.sun.com/xml/ns/javaee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com

    <!-- The definition of the Root Spring Container shared by all Servlets ar
    <context-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>/WEB-INF/spring/root-context.xml</param-value>
    </context-param>

    <!-- Creates the Spring Container shared by all Servlets and Filters -->
    <listener>
        <listener-class>org.springframework.web.context.ContextLoaderListener
    </listener>

    <!-- Processes application requests -->
    <servlet>
        <servlet-name>appServlet</servlet-name>
        <servlet-class>org.springframework.web.servlet.DispatcherServlet</ser
        <init-param>
            <param-name>contextConfigLocation</param-name>
            <param-value>/WEB-INF/spring/spring.xml</param-value>
        </init-param>
        <load-on-startup>1</load-on-startup>
    </servlet>

    <servlet-mapping>
        <servlet-name>appServlet</servlet-name>
        <url-pattern>/</url-pattern>
    </servlet-mapping>

    <error-page>
        <error-code>404</error-code>
        <location>/resources/404.jsp</location>
    </error-page>
</web-app>
```

Most of the part is for plugging in Spring Framework for our web application, except the error-page defined for 404 error. So when our application will throw 404 error, this page will be used as response. This configuration is used by container when our spring web application throws 404 error code.

Model Classes

I have defined Employee bean as model class, however we will be using it in our application just to return valid response in specific scenario. We will be deliberately throwing different types of exceptions in most of the cases.

Employee.java

```java
package com.journaldev.spring.model;

public class Employee {

    private String name;
    private int id;

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }
}
```

Since we will be returning JSON response too, let's create a java bean with exception details that will be sent as response.

ExceptionJSONInfo.java

```java
package com.journaldev.spring.model;

public class ExceptionJSONInfo {

    private String url;
    private String message;

    public String getUrl() {
        return url;
    }
    public void setUrl(String url) {
        this.url = url;
    }
    public String getMessage() {
        return message;
    }
    public void setMessage(String message) {
        this.message = message;
    }
}
```

Custom Exception Class

Let's create a custom exception class to be used by our application.

EmployeeNotFoundException.java

```java
package com.journaldev.spring.exceptions;

import org.springframework.http.HttpStatus;
import org.springframework.web.bind.annotation.ResponseStatus;

@ResponseStatus(value=HttpStatus.NOT_FOUND, reason="Employee Not Found") //40
public class EmployeeNotFoundException extends Exception {

    private static final long serialVersionUID = -3332292346834265371L;

    public EmployeeNotFoundException(int id){
        super("EmployeeNotFoundException with id="+id);
    }
}
```

Notice that we can use `@ResponseStatus` annotation with exception classes to define the HTTP code that will be sent by our application when this type of exception is thrown by our application and handled by our exception handling implementations.

As you can see that I am setting HTTP status as 404 and we have an error-page defined for this, so our application should use the error page for this type of exception if we are not returning any view.

We can also override the status code in our exception handler method, think of it as default http status code when our exception handler method is not returning any view page as response.

Controller Class with Exception Handler Methods

Let’s look at our controller class where we will throw different type of exceptions.

EmployeeController.java

```java
package com.journaldev.spring.controllers;

import java.io.IOException;
import java.sql.SQLException;

import javax.servlet.http.HttpServletRequest;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.ExceptionHandler;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.ResponseBody;
import org.springframework.web.servlet.ModelAndView;

import com.journaldev.spring.exceptions.EmployeeNotFoundException;
import com.journaldev.spring.model.Employee;
import com.journaldev.spring.model.ExceptionJSONInfo;

@Controller
public class EmployeeController {

    private static final Logger logger = LoggerFactory.getLogger(EmployeeCont

    @RequestMapping(value="/emp/{id}", method=RequestMethod.GET)
    public String getEmployee(@PathVariable("id") int id, Model model) throws
        //deliberately throwing different types of exception
        if(id==1){
            throw new EmployeeNotFoundException(id);
        }else if(id==2){
            throw new SQLException("SQLException, id="+id);
        }else if(id==3){
            throw new IOException("IOException, id="+id);
        }else if(id==10){
            Employee emp = new Employee();
            emp.setName("Pankaj");
            emp.setId(id);
            model.addAttribute("employee", emp);
            return "home";
        }else {
            throw new Exception("Generic Exception, id="+id);
        }

    }

    @ExceptionHandler(EmployeeNotFoundException.class)
    public ModelAndView handleEmployeeNotFoundException(HttpServletRequest re
        logger.error("Requested URL="+request.getRequestURL());
        logger.error("Exception Raised="+ex);

        ModelAndView modelAndView = new ModelAndView();
        modelAndView.addObject("exception", ex);
        modelAndView.addObject("url", request.getRequestURL());

        modelAndView.setViewName("error");
        return modelAndView;
    }
}
```

Notice that for EmployeeNotFoundException handler, I am returning ModelAndView and hence http status code will be sent as OK (200). If it would have been returning void, then http status code would have been sent as 404. We will look into this type of implementation in our global exception handler implementation.

Since I am handling only EmployeeNotFoundException in controller, all other exceptions thrown by our controller will be handled by global exception handler.

Global Exception Handler using @ControllerAdvice annotation

Here is our global exception handler controller class.

GlobalExceptionHandler.java

```java
package com.journaldev.spring.controllers;

import java.io.IOException;
import java.sql.SQLException;

import javax.servlet.http.HttpServletRequest;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.http.HttpStatus;
import org.springframework.web.bind.annotation.ControllerAdvice;
import org.springframework.web.bind.annotation.ExceptionHandler;
import org.springframework.web.bind.annotation.ResponseStatus;

@ControllerAdvice
public class GlobalExceptionHandler {

    private static final Logger logger = LoggerFactory.getLogger(GlobalExcept

    @ExceptionHandler(SQLException.class)
    public String handleSQLException(HttpServletRequest request, Exception ex
        logger.info("SQLException Occured:: URL="+request.getRequestURL());
        return "database_error";
    }

    @ResponseStatus(value=HttpStatus.NOT_FOUND, reason="IOException occured")
    @ExceptionHandler(IOException.class)
    public void handleIOException(){
        logger.error("IOException handler executed");
        //returning 404 error code
    }
}
```

Notice that for SQLException, I am returning database_error.jsp as response page with http status code as 200.

For IOException, we are returning void with status code as 404, so our error-page will be used in this case.

As you can see that I am not handling any other types of exception here, that part I have left for HandlerExceptionResolver implementation.

HandlerExceptionResolver implementation

We are just extending SimpleMappingExceptionResolver and overriding one of the method, but we can override it's most important method `resolveException` for logging and sending different types of view pages. But that is same as using ControllerAdvice implementation, so I am leaving it. We will be using it to configure view page for all the other exceptions not handled by us by responding with generic error page.

Spring Bean Configuration File

Our spring bean configuration file looks like below.

spring.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<beans:beans xmlns="http://www.springframework.org/schema/mvc"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:beans="http://www.springframework.org/schema/beans"
    xmlns:context="http://www.springframework.org/schema/context"
    xsi:schemaLocation="http://www.springframework.org/schema/mvc http://www.
        http://www.springframework.org/schema/beans http://www.springframewor
        http://www.springframework.org/schema/context http://www.springframew

    <!-- DispatcherServlet Context: defines this servlet's request-processing

    <!-- Enables the Spring MVC @Controller programming model -->
    <annotation-driven />

    <!-- Handles HTTP GET requests for /resources/** by efficiently serving u
    <resources mapping="/resources/**" location="/resources/" />

    <!-- Resolves views selected for rendering by @Controllers to .jsp resour
    <beans:bean class="org.springframework.web.servlet.view.InternalResourceV
        <beans:property name="prefix" value="/WEB-INF/views/" />
        <beans:property name="suffix" value=".jsp" />
    </beans:bean>


    <beans:bean id="simpleMappingExceptionResolver" class="com.journaldev.spr
        <beans:property name="exceptionMappings">
            <beans:map>
                <beans:entry key="Exception" value="generic_error"></beans:en
            </beans:map>
        </beans:property>
```

```
            <beans:property name="defaultErrorView" value="generic_error"/>
        </beans:bean>

        <!-- Configure to plugin JSON as request and response in method handler --
        <beans:bean class="org.springframework.web.servlet.mvc.method.annotation.I
            <beans:property name="messageConverters">
                <beans:list>
                    <beans:ref bean="jsonMessageConverter"/>
                </beans:list>
            </beans:property>
        </beans:bean>

        <!-- Configure bean to convert JSON to POJO and vice versa -->
        <beans:bean id="jsonMessageConverter" class="org.springframework.http.conv
        </beans:bean>

        <context:component-scan base-package="com.journaldev.spring" />

</beans:beans>
```

Notice the beans configured for supporting JSON in our web application. The only part related to exception handling is the simpleMappingExceptionResolver bean definition where we are defining generic_error.jsp as the view page for Exception class. This make sure that any exception not handled by our application will not result in sending server generated error page as the response.

JSP View Pages

It's time to look into the last part of our application, our view pages that will be used in our application.

home.jsp

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<%@ page session="false" %>
<html>
<head>
    <title>Home</title>
</head>
<body>
    <h3>Hello ${employee.name}!</h3><br>
    <h4>Your ID is ${employee.id}</h4>
</body>
</html>
```

home.jsp is used to respond with valid data, i.e when we get id as 10 in the client request.

404.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>404 Error Page</title>
</head>
<body>

```

```
<h2>Resource Not Found Error Occured, please contact support.</h2>

</body>
</html>
```

404.jsp is used for generating view for 404 http status code, for our implementation this should be the response when we get id as 3 in client request.

error.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Error Page</title>
</head>
<body>
<h2>Application Error, please contact support.</h2>

<h3>Debug Information:</h3>

Requested URL= ${url}<br><br>

Exception= ${exception.message}<br><br>

<strong>Exception Stack Trace</strong><br>
<c:forEach items="${exception.stackTrace}" var="ste">
    ${ste}
</c:forEach>

</body>
</html>
```

error.jsp is used when our controller class request handler method is throwing EmployeeNotFoundException. We should get this page in response when id value is 1 in the client request.

database_error.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Database Error Page</title>
</head>
<body>

<h2>Database Error, please contact support.</h2>

</body>
</html>
```

database_error.jsp is used when our application is throwing SQLException, as configured in

GlobalExceptionHandler class. We should get this page as response when id value is 2 in the client request.

generic_error.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Generic Error Page</title>
</head>
<body>

<h2>Unknown Error Occured, please contact support.</h2>

</body>
</html>
```

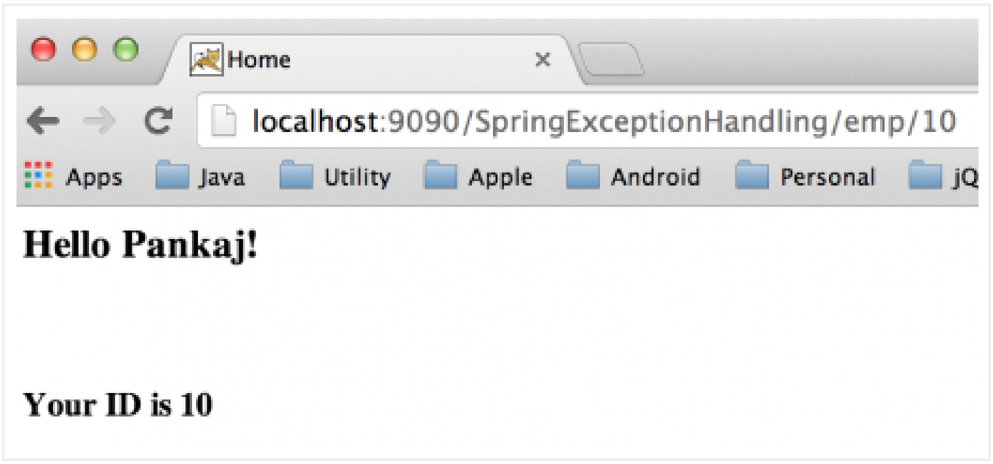
This should be the page as response when any exception occurs not handled by our application code and simpleMappingExceptionResolver bean takes care of that. We should get this page as response when id value in client request is anything other than 1,2,3 or 10.

Running the Spring MVC Application

Just deploy the application in the servlet container you are using, I am using Apache Tomcat 7 for this example.

Below images show the different response pages returned by our application based on the id value.

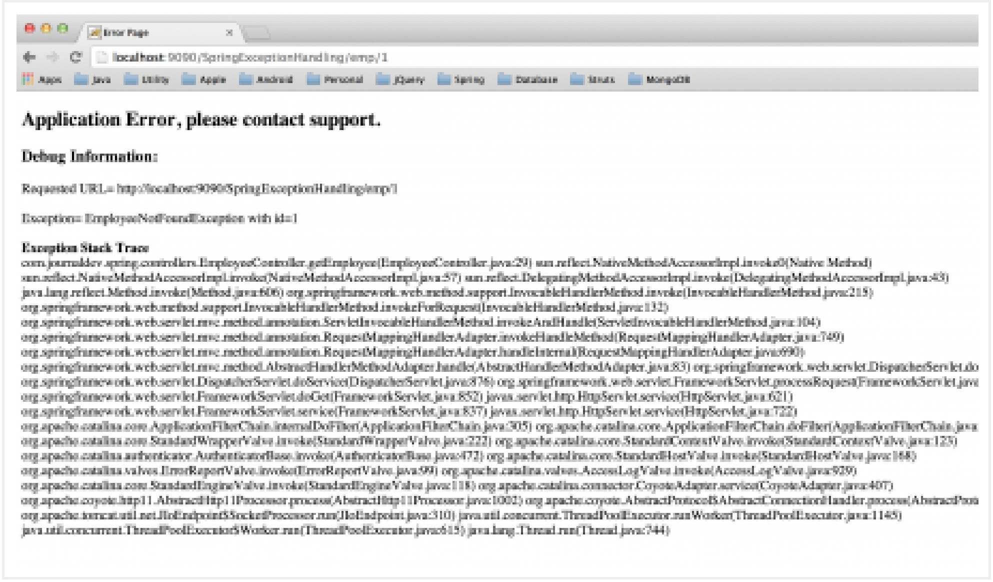
ID=10, valid response.



Hello Pankaj!

Your ID is 10

ID=1, controller based exception handler used


Application Error, please contact support.

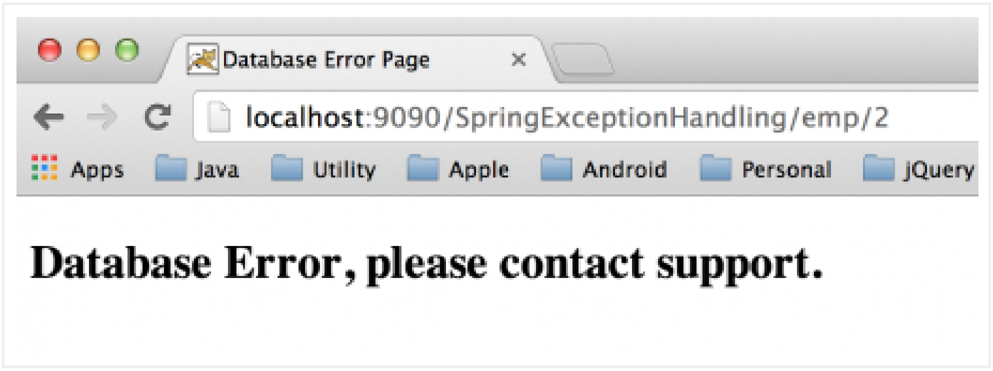
Debug Information:

Requested URL= http://localhost:9090/SpringExceptionHandling/emp/1

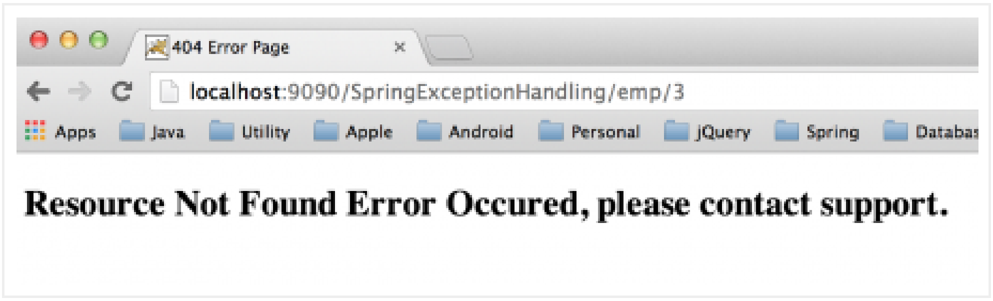
Exception= EmployeeNotFoundException with id=1

Exception Stack Trace
```
com.journaldev.spring.controllers.EmployeeController.getEmployee(EmployeeController.java:29) sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:57) sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
java.lang.reflect.Method.invoke(Method.java:606) org.springframework.web.method.support.InvocableHandlerMethod.invoke(InvocableHandlerMethod.java:215)
org.springframework.web.method.support.InvocableHandlerMethod.invokeForRequest(InvocableHandlerMethod.java:132)
org.springframework.web.servlet.mvc.method.annotation.ServletInvocableHandlerMethod.invokeAndHandle(ServletInvocableHandlerMethod.java:104)
org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerAdapter.invokeHandleMethod(RequestMappingHandlerAdapter.java:749)
org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerAdapter.handleInternal(RequestMappingHandlerAdapter.java:690)
org.springframework.web.servlet.mvc.method.AbstractHandlerMethodAdapter.handle(AbstractHandlerMethodAdapter.java:83) org.springframework.web.servlet.DispatcherServlet.do
org.springframework.web.servlet.DispatcherServlet.doService(DispatcherServlet.java:876) org.springframework.web.servlet.FrameworkServlet.processRequest(FrameworkServlet.java
org.springframework.web.servlet.FrameworkServlet.doGet(FrameworkServlet.java:852) javax.servlet.http.HttpServlet.service(HttpServlet.java:621)
org.springframework.web.servlet.FrameworkServlet.service(FrameworkServlet.java:837) javax.servlet.http.HttpServlet.service(HttpServlet.java:722)
org.apache.catalina.core.ApplicationFilterChain.internalDoFilter(ApplicationFilterChain.java:305) org.apache.catalina.core.ApplicationFilterChain.doFilter(ApplicationFilterChain.java
org.apache.catalina.core.StandardWrapperValve.invoke(StandardWrapperValve.java:222) org.apache.catalina.core.StandardContextValve.invoke(StandardContextValve.java:123)
org.apache.catalina.authenticator.AuthenticatorBase.invoke(AuthenticatorBase.java:472) org.apache.catalina.core.StandardHostValve.invoke(StandardHostValve.java:168)
org.apache.catalina.valves.ErrorReportValve.invoke(ErrorReportValve.java:99) org.apache.catalina.valves.AccessLogValve.invoke(AccessLogValve.java:929)
org.apache.catalina.core.StandardEngineValve.invoke(StandardEngineValve.java:118) org.apache.catalina.connector.CoyoteAdapter.service(CoyoteAdapter.java:407)
org.apache.coyote.http11.AbstractHttp11Processor.process(AbstractHttp11Processor.java:1002) org.apache.coyote.AbstractProtocol$AbstractConnectionHandler.process(AbstractProt
org.apache.tomcat.util.net.JIoEndpoint$SocketProcessor.run(JIoEndpoint.java:310) java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1145)
java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:615) java.lang.Thread.run(Thread.java:744)
```

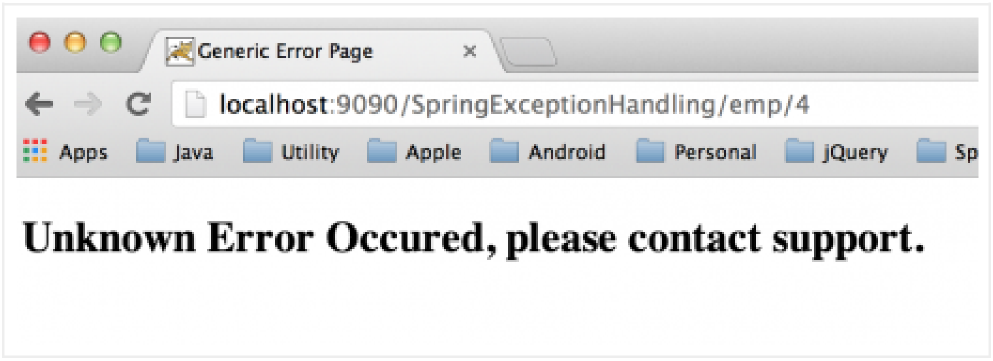
ID=2, global exception handler used with view as response


Database Error, please contact support.

ID=3, 404 error page used


Resource Not Found Error Occured, please contact support.

ID=4, simpleMappingExceptionResolver used for response view


Unknown Error Occured, please contact support.

As you can see that we got the expected response in all the cases.

Spring JSON Response Exception Handler

We are almost done with our tutorial, except the last bit where I will explain how to send JSON response from the exception handler methods.

Our application has all the JSON dependencies and jsonMessageConverter is configured, all we need to implement the exception handler method.

For simplicity, I will rewrite the EmployeeController handleEmployeeNotFoundException() method to return JSON response.

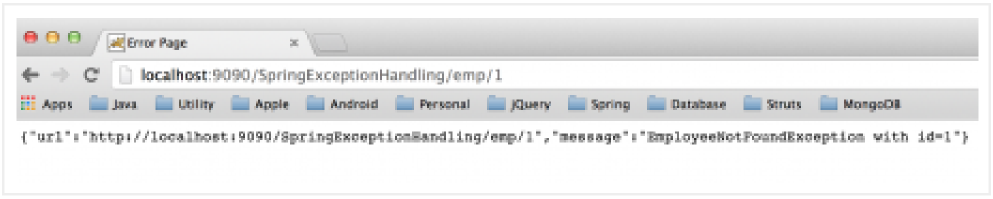
Just update EmployeeController exception handler method with below code and deploy the application again.

```java
@ExceptionHandler(EmployeeNotFoundException.class)
public @ResponseBody ExceptionJSONInfo handleEmployeeNotFoundException(HttpSer

    ExceptionJSONInfo response = new ExceptionJSONInfo();
    response.setUrl(request.getRequestURL().toString());
    response.setMessage(ex.getMessage());

    return response;
}
```

Now when we use id as 1 in client request, we get following JSON response as shown in the below image.


Error Page
localhost:9090/SpringExceptionHandling/emp/1
Apps Java Utility Apple Android Personal jQuery Spring Database Struts MongoDB

```
{"url":"http://localhost:9090/SpringExceptionHandling/emp/1","message":"EmployeeNotFoundException with id=1"}
```

That's all for Exception Handling in Spring MVC Framework, please download the application from below URL and play around with it to learn more.

Download Spring Exception Handling Project
1299 downloads