Before moving forward please go through theory part of Spring MVC Exception Handling from our notes, so that you can understand the concept well.

**What this project does??**

This is ninth project of Spring MVC project series.

**This is Full Spring MVC application with XML configuration which demonstrates how exception handling mechanism works in Spring.**

This is web based maven project.

**Also note that here we are focusing on explanation of Spring Exception Handling mechanism only. Please go through project one to four of Spring MVC project series before this project so that you can understand the project structure and flow of request in the project well.**

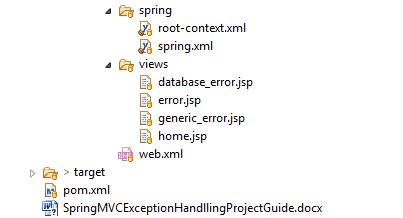
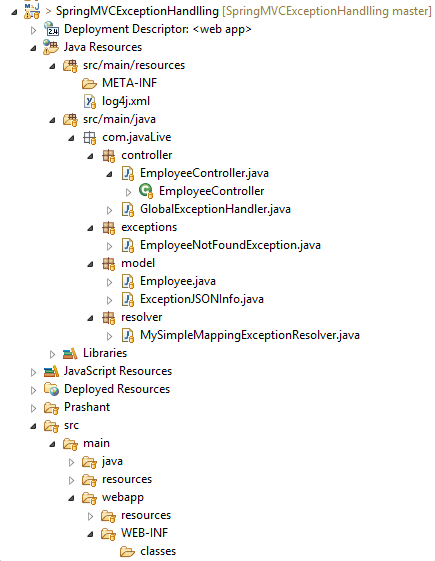
' SpringMVCApplicationFlow': explains functionality of 'spring-servlet.xml' in Spring MVC context

' SpringMVCHelloWorld': explains how controller layer functions in Spring MVC

'SprinArithematicOperation': explains how controller layer and service layer functions in Spring MVC

'SpringMVCHibernateWithXMLFull' : explains full Spring+Hibernate database table CRUD operation **with full XML configuration** which includes all three layers of Spring MVC viz. Controller, Service and DAO.

**Note that this is xml based spring MVC project. We have web config file web.xml and spring configuration file spring-servlet.xml. (Location of these files are shown in screen shot of project structure below.)**

****

**In above project please go through files, packages and folders as follows:**

Package com.javaLive.controller which contains controllers.

Package com.javaLive.exceptions contains user defined exceptions

Package com.javaLive.model contains model classe

Package com.javaLive.resolver contains class extending 'SimpleMappingExceptionResolver' class.

Folder WEB-INF/views which contains various JSP files.

Files WEB-INF/spring-servlet.xml and WEB-INF/web.xml.

**Steps to create project:-**

1. This is web based maven project. Create webapp maven project. Please refer the file **'CreateWebBasedMavenProjectInEclipse.docx'** in this project to know more about creation of web based maven project.

2. Add require dependencies for spring as shown in pom.xml file.

3. Create require packages and add the files.

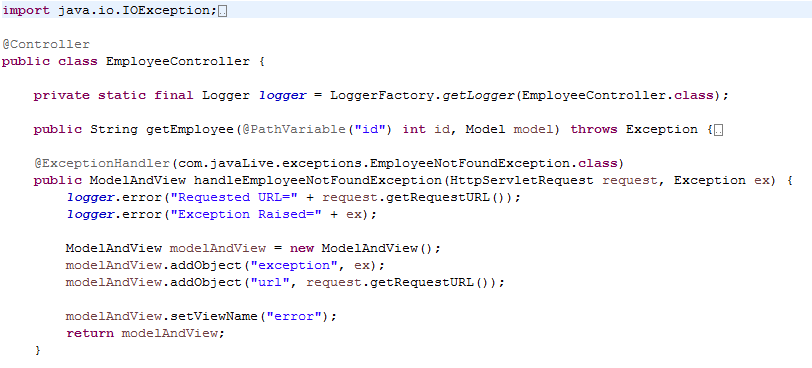
**Functioning of the project**

**Spring MVC Framework provides following three 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.

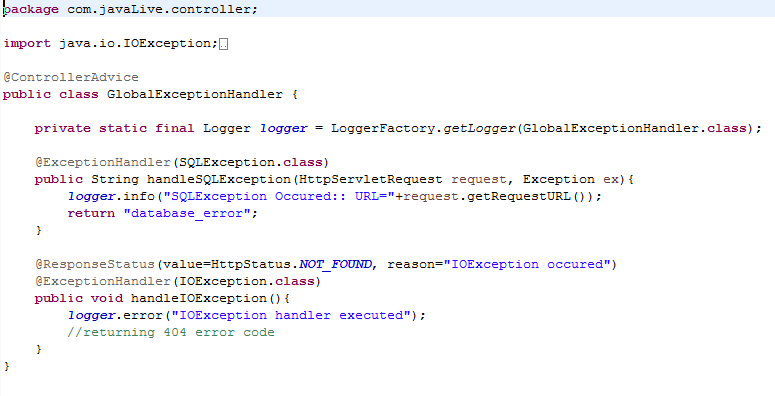
If there are multiple exception handler methods defined, then handler method that is closest to the Exception class is used. For example, if we have two handler methods defined for IOException and Exception and our request handler method throws IOException, then handler method for IOException will get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



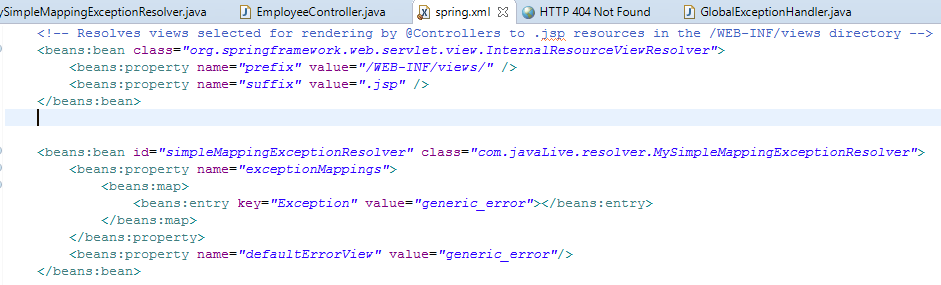
**This approach applies to only the controller in which the handler method is declared.**

**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](https://www.journaldev.com/2583/spring-aop-example-tutorial-aspect-advice-pointcut-joinpoint-annotations) 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. **This approach applies to all controllers in the application.(It uses Spring AOP based approach). And is more preferable than next one as it separates Exception Handlling cross cutting concern.**

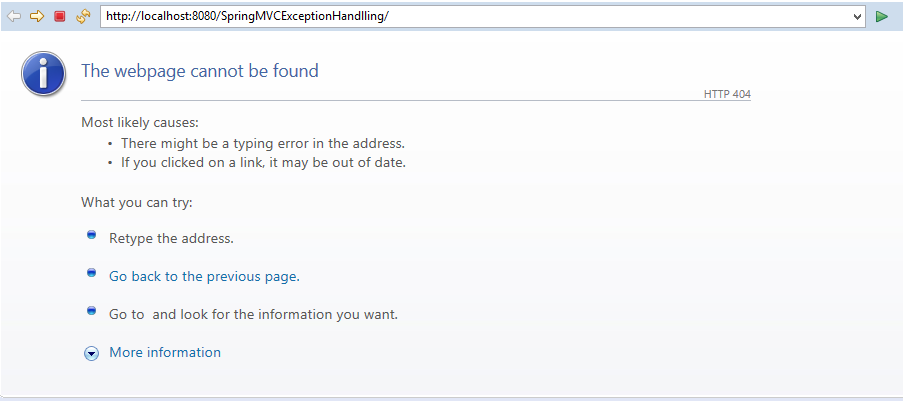
**3. HandlerExceptionResolver –** For generic exceptions, most of the times we serve static pages. Spring Framework provides HandlerExceptionResolver interface that we can implement to create global exception handler. The reason behind this additional way to define global exception handler is that Spring 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.

**This approach applies to all controllers in the application.(It uses Spring MCV based approach).**

****

**Running the Spring MVC Exception Handling Application**

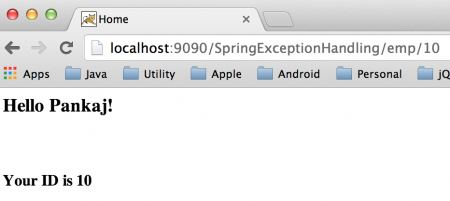
Right click -> Run As -> Run on Server



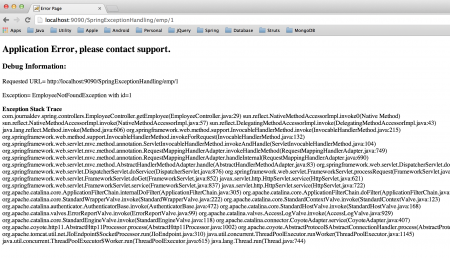
**Just add 'emp/{id}' to address bar**

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

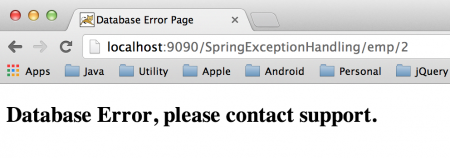
ID=10, valid response.

[](https://cdn.journaldev.com/wp-content/uploads/2014/04/Spring-MVC-Valid-Response.png)

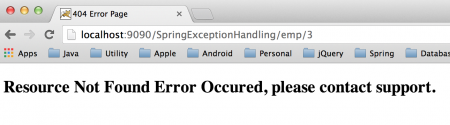
ID=1, controller based exception handler used

[](https://cdn.journaldev.com/wp-content/uploads/2014/04/Spring-MVC-Exception-Handling-Controller.png)

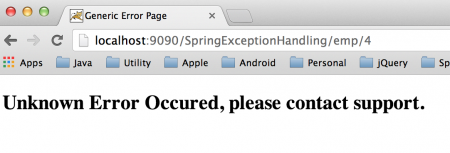
ID=2, global exception handler used with view as response

[](https://cdn.journaldev.com/wp-content/uploads/2014/04/Spring-MVC-Exception-Handling-ControllerAdvice.png)

ID=3, 404 error page used

[](https://cdn.journaldev.com/wp-content/uploads/2014/04/Spring-MVC-Error-Handling.png)

ID=4, simpleMappingExceptionResolver used for response view

[](https://cdn.journaldev.com/wp-content/uploads/2014/04/Spring-MVC-Exception-Resolver.png)

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

**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.