How to handle exceptions in Spring MVC

Last Updated on 20 January 2016 | Print Email

Download Aspose for all your file format manipulation needs

In Spring MVC, unexpected exceptions might be thrown during execution of its controllers. Spring provides two approaches for handling these exceptions:

- **Using XML configuration**: this is similar to exception handling in Servlet/JSP, by declaring a `SimpleMappingExceptionResolver` bean in Spring's application context file and map exception types with view names. This approach applies to all controllers in the application.
- **Using exception handler method**: Spring provides the `@ExceptionHandler` annotation type which is used to annotate a method to handle exceptions raised by the controller's methods. This approach applies to only the controller in which the handler method is declared.

Let's look at each approach in details.

1. Using XML configuration

This approach uses XML to configure exceptions handling declaratively. Consider the following bean declaration in Spring's application context file:

```
<bean
        class="org.springframework.web.servlet.handler.SimpleMappingExceptionResolver">

        <property name="exceptionMappings">
                <props>
                        <prop key="java.lang.ArithmeticException">MathError</prop>
                </props>
        </property>

</bean>
```

That will map any exceptions of type `java.lang.ArithmeticException` (or its sub types) to the view named `MathError`. During execution of a Spring controller, if such an exception is thrown, the client will be redirected to the mapped view. For example, if we configure a view resolver likes this:

```
<bean id="viewResolver"
        class="org.springframework.web.servlet.view.InternalResourceViewResolver">
        <property name="prefix" value="/WEB-INF/views/" />
        <property name="suffix" value=".jsp" />
</bean>
```

Then Spring will redirect the client to the page `/WEB-INF/views/MathError.jsp` in case a `java.lang.ArithmeticException` exception is thrown. Accessing the exception in the `MathError.jsp` page as follows:

```
Arithmetic Error: ${exception}
```

Assuming we have the following controller class:

```
@Controller
@RequestMapping("/doMath")
public class MathController {
	@RequestMapping(method = RequestMethod.GET)
	public ModelAndView calculateSum(@RequestParam int a, @RequestParam int b) {
		ModelAndView model = new ModelAndView("MathResult");

		model.addObject("sum", (a + b));
		model.addObject("subtract", (a - b));
		model.addObject("multiply", (a * b));
		model.addObject("divide", (a / b));

		return model;
	}
}
```

The method `calculateSum()` will handle the request */doMath*, it takes two numbers a and b from the request and calculates sum, subtract, multiply and divide of them. The results are added to the model of the view called "MathResult". In this code, there are two possible exceptions:

- Either a or b is not a number.
- b is zero, so the operation `a / b` will throw a `java.lang.ArithmeticException` exception.

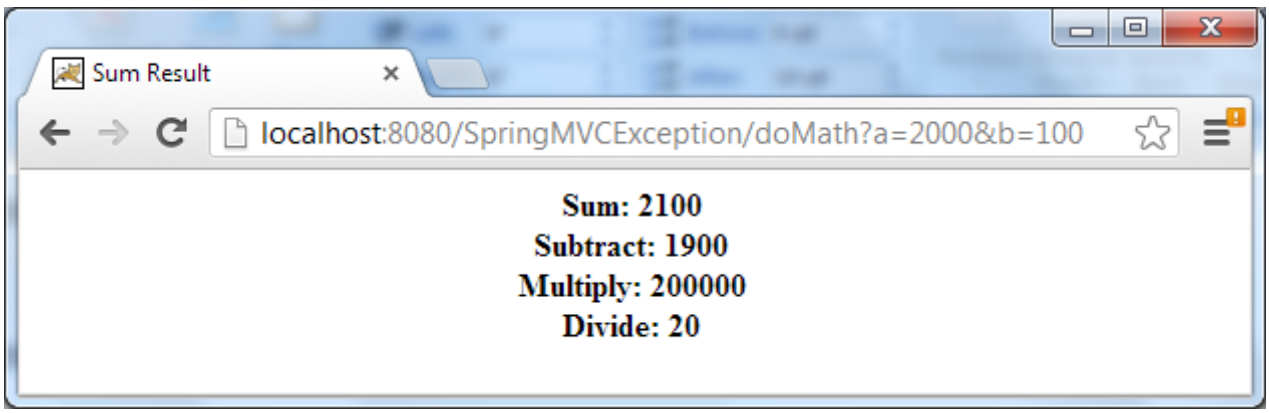
Following is code of the `MathResult.jsp` page:

```
<%@ 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.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Sum Result</title>
</head>
<body>
	<center>
		<b>Sum: ${sum} </b><br/>
		<b>Subtract: ${subtract} </b><br/>
		<b>Multiply: ${multiply} </b><br/>
		<b>Divide: ${divide} </b><br/>
	</center>
</body>
</html>
```

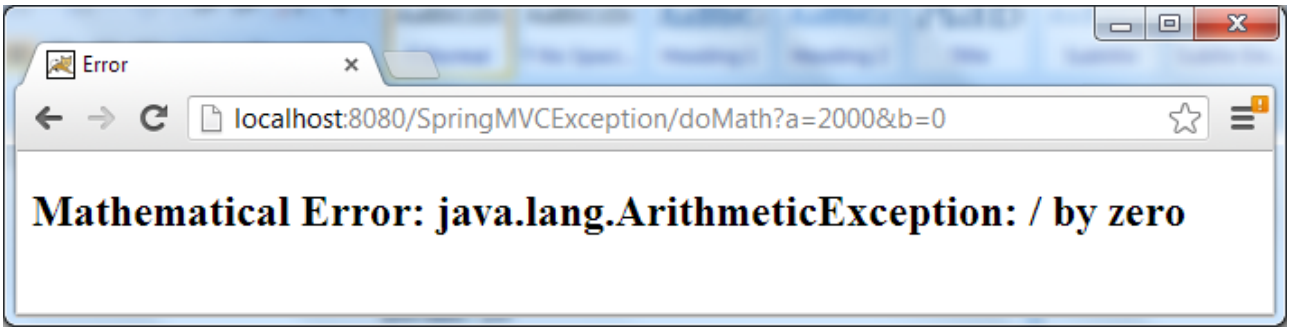
Code of the `MathError.jsp` page:

```
<%@ 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.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Error</title>
</head>
<body>
        <h2>
                Mathematical Error: ${exception} <br/>
        </h2>
</body>
</html>
```

Output when testing the application with two numbers a = 2000 and b = 100:



If we pass b = 0, then the `MathError.jsp` page will be displayed:



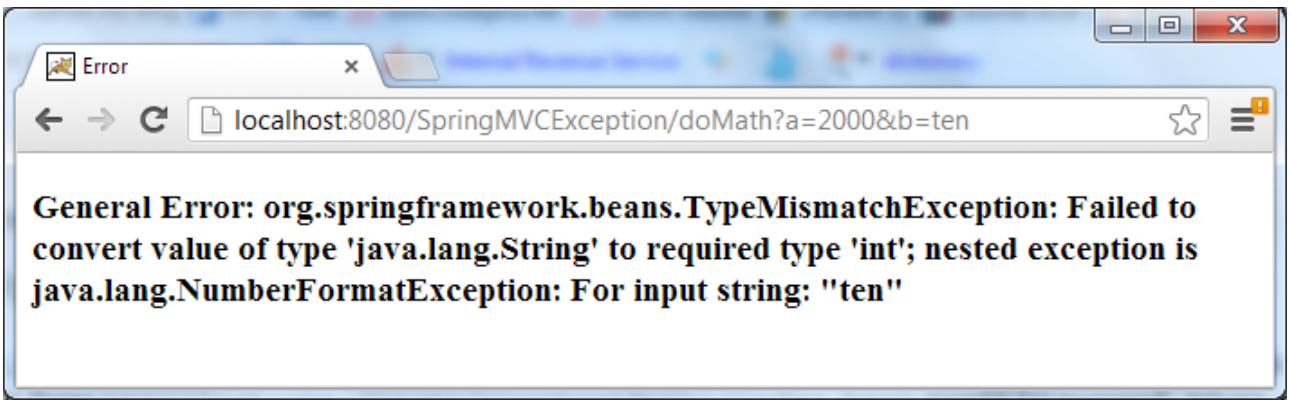
We can configure a default error view for all exceptions which are not specified in the `exceptionMappings` property, by specifying a view name for the `defaultErrorView` property. For example:

```
<bean
        class="org.springframework.web.servlet.handler.SimpleMappingExceptionResolver">
        <property name="exceptionMappings">
                <props>
                        <prop key="java.lang.ArithmeticException">MathError</prop>
                </props>
        </property>

        <property name="defaultErrorView" value="Error" />

</bean>
```

That tells Spring to redirect the client to the `Error.jsp` page if any exceptions other than the `java.lang.ArithmeticException` is thrown. Code of the `Error.jsp` page is similar to the `MathError.jsp` page:

```
<%@ 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.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Error</title>
</head>
<body>
        <h3>
                General Error: ${exception}
        </h3>
</body>
</html>
```

Output when testing the application with a = 2000 and b = ten (not a number):



As we can see, the `Error.jsp` page is displayed because the exception thrown is of type `TypeMismatchException`, not `ArithmeticException`.

We also can specify multiple mappings for exception types - view names as follows:

```
<bean
        class="org.springframework.web.servlet.handler.SimpleMappingExceptionResolver">

        <property name="exceptionMappings">
                <props>
                        <prop key="java.lang.ArithmeticException">MathError</prop>
                        <prop key="java.io.IOException">IOError</prop>
                </props>
        </property>

</bean>
```


2. Using exception handler method

This approach uses the `@ExceptionHandler` annotation to annotate a method in controller class to handle exceptions raised during execution of the controller's methods. Consider the following controller class:

```
@Controller
public class FileUploadController {

        @RequestMapping(value = "/uploadFile", method = RequestMethod.GET)
        public String doFileUpload(@RequestParam int a) throws IOException, SQLException {

                // handles file upload stuff...
                if (a == 1) {
                        throw new IOException("Could not read upload file.");
                } else if (a == 2) {
                        throw new SQLException("Database exception!!!");
                }

                return "done";
        }

        @ExceptionHandler({IOException.class, java.sql.SQLException.class})
        public ModelAndView handleIOException(Exception ex) {
                ModelAndView model = new ModelAndView("IOError");

                model.addObject("exception", ex.getMessage());

                return model;
        }
}
```

The method `doFileUpload()` may throw an `IOException`, and the handler method is declared as follows:

```
@ExceptionHandler(IOException.class)
public ModelAndView handleIOException(IOException ex) {
        ModelAndView model = new ModelAndView("IOError");

        model.addObject("exception", ex.getMessage());

        return model;
}
```

This `handleIOException()` method will be invoked whenever an exception of type `java.io.IOException` (or its sub types) is raised within the controller class. Spring will pass the exception object into the method's argument.

Using exception handler method is very flexible, as it allows us doing some processing before returning to the error view. We can also pass additional information to the view's model when necessary, for example:

```
model.addObject("info", "extra info for the exception");
```

It's also possible to specify a list of exception classes in the `@ExceptionHanlder` annotation, for example:

```
@ExceptionHandler({IOException.class, java.sql.SQLException.class})
public ModelAndView handleIOException(Exception ex) {
        ModelAndView model = new ModelAndView("IOError");

        model.addObject("exception", ex.getMessage());

        return model;
}
```

If the exception being thrown is one of the types (or sub types) in the list, the annotated method will be invoked. In the code above, if the exception is either of type `IOException` or `SQLException`, then the `handleIOException()` method will be invoked.

3. Conclusion

Using XML configuration (declaration of `SimpleMappingExceptionResolver` bean) will apply exception handling rules for the entire application, whereas using handler method (annotated by `@ExceptionHandler` annotation) will have effect only within the enclosing controller class. The XML declaration approach takes precedence over the annotation approach, so if we declared a `SimpleMappingExceptionResolver` bean in Spring's application context file, then any `@ExceptionHandler` annotations will be ignored.

You may be also interested in:

- Understanding the core of Spring framework
- Spring MVC beginner tutorial with Spring Tool Suite IDE
- How to handle exceptions in Struts2
- How to handle exceptions in JSP
- Bootstrapping a Spring Web MVC application programmatically

Share this article:

Attachments:

SpringMVCException1.zip	[exception handling using XML declaration]	3342 kB
SpringMVCException2.zip	[exception handling using annotation]	3341 kB