# Spring:

This spring tutorial provides in-depth concepts of Spring Framework with simplified examples. It was developed by Rod Johnson in 2003. Spring framework makes the easy development of JavaEE application.

It is helpful for beginners and experienced persons.

# Spring Framework:

Spring is a lightweight framework. It can be thought of as a framework of frameworks because it provides support to various frameworks such as Struts, Hibernate, Tapestry, EJB, JSF etc. The framework, in broader sense, can be defined as a structure where we find solution of the various technical problems.

The Spring framework comprises several modules such as IOC, AOP, DAO, Context, ORM, WEB MVC etc. We will learn these modules in next page. Let's understand the IOC and Dependency Injection first.

# Inversion Of Control (IOC) and Dependency Injection

These are the design patterns that are used to remove dependency from the programming code. They make the code easier to test and maintain. Let's understand this with the following code:

class Employee{

Address address;

Employee(){

address=new Address();

}

}

In such case, there is dependency between the Employee and Address (tight coupling). In the Inversion of Control scenario, we do this something like this:

class Employee{

Address address;

Employee(Address address){

this.address=address;

}

}

Thus, IOC makes the code loosely coupled. In such case, there is no need to modify the code if our logic is moved to new environment.

# Spring MVC:

Spring MVC tutorial provides an elegant solution to use MVC in spring framework by the help of DispatcherServlet.

In Spring Web MVC, DispatcherServlet class works as the front controller. It is responsible to manage the flow of the spring mvc application.

The @Controller annotation is used to mark the class as the controller in Spring 3.

The @RequestMapping annotation is used to map the request url. It is applied on the method.

# Understanding the flow of Spring Web MVC

# C:\Users\srinath\Desktop\mvc.jpg

# Spring MVC example

As displayed in the figure, all the incoming request is intercepted by the DispatcherServlet that works as the front controller. The DispatcherServlet gets entry of handler mapping from the xml file and forwards the request to the controller. The controller returns an object of ModelAndView. The DispatcherServlet checks the entry of view resolver in the xml file and invokes the specified view component.

# Spring Web MVC Framework Example

Let's see the simple example of spring web MVC framework. There are given 7 steps for creating the spring MVC application. The steps are as follows:

1.Create the request page (optional)

2.Create the controller class

3.Provide the entry of controller in the web.xml file.

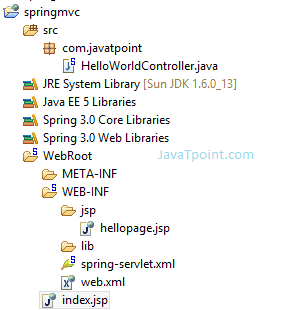
4.Define the bean in the xml file

5.Display the message in the JSP page

6.Load the spring core and mvc jar files

7.Start server and deploy the project

# Directory Structure



# Directory Structure of Spring MVC using Maven

# C:\Users\srinath\Desktop\spring maven.jpg

# Required Jar files or Maven Dependency

To run this example, you need to load:

1.Spring Core jar files

2.Spring Web jar files

Download Link: Download all the jar files for spring including core, web, aop, mvc, j2ee, remoting, oxm, jdbc, orm etc.

If you are using Maven, you don't need to add jar files. Now, you need to add maven dependency in pom.xml file.

# pom.xml

<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4\_0\_0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>com.javatpoint</groupId>

<artifactId>SpringMVC</artifactId>

<packaging>war</packaging>

<version>0.0.1-SNAPSHOT</version>

<name>SpringMVC Maven Webapp</name>

<url>http://maven.apache.org</url>

<dependencies>

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>3.8.1</version>

<scope>test</scope>

</dependency>

<dependency>

<groupId>javax.servlet</groupId>

<artifactId>servlet-api</artifactId>

<version>3.0-alpha-1</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-core</artifactId>

<version>3.1.2.RELEASE</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>3.1.2.RELEASE</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-beans</artifactId>

<version>3.1.2.RELEASE</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-webmvc</artifactId>

<version>3.1.2.RELEASE</version>

</dependency>

</dependencies>

<build>

<finalName>SpringMVC</finalName>

</build>

</project>

# 1) Create the request page (optional)

This is the simple jsp page containing a link. It is optional page. You may direct invoke the action class instead.

index.jsp

<a href="hello.html">click</a>

# 2) Create the controller class

To create the controller class, we are using two annotations @Controller and @RequestMapping.

1.The @Controller annotation marks this class as Controller.

2.The @Requestmapping annotation is used to map the class with the specified name.

This class returns the instance of ModelAndView controller with the mapped name, message name and message value. The message value will be displayed in the jsp page.

# HelloWorldController.java

package com.javatpoint;

import org.springframework.stereotype.Controller;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.servlet.ModelAndView;

@Controller

public class HelloWorldController {

@RequestMapping("/hello")

public ModelAndView helloWorld() {

String message = "HELLO SPRING MVC HOW R U";

return new ModelAndView("hellopage", "message", message);

}

}

# 3) Provide the entry of controller in the web.xml file

In this xml file, we are specifying the servlet class DispatcherServlet that acts as the front controller in Spring Web MVC. All the incoming request for the html file will be forwarded to the DispatcherServlet.

# 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/xml/ns/javaee/web-app\_2\_5.xsd">

<servlet>

<servlet-name>spring</servlet-name>

<servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>

<load-on-startup>1</load-on-startup>

</servlet>

<servlet-mapping>

<servlet-name>spring</servlet-name>

<url-pattern>\*.html</url-pattern>

</servlet-mapping>

</web-app>

# 4) Define the bean in the xml file

This is the important configuration file where we need to specify the ViewResolver and View components.

The context:component-scan element defines the base-package where DispatcherServlet will search the controller class.

Here, the InternalResourceViewResolver class is used for the ViewResolver.

The prefix+string returned by controller+suffix page will be invoked for the view component.

This xml file should be located inside the WEB-INF directory.

# spring-servlet.xml

<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xmlns:p="http://www.springframework.org/schema/p"

xmlns:context="http://www.springframework.org/schema/context"

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.javatpoint" />

<bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">

<property name="prefix" value="/WEB-INF/jsp/" />

<property name="suffix" value=".jsp" />

</bean>

</beans>

# 5) Display the message in the JSP page

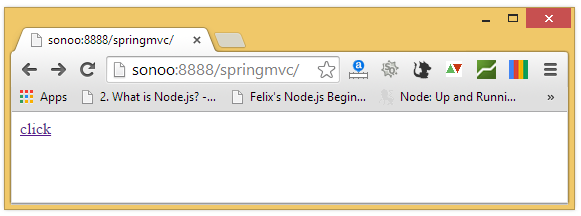
This is the simple JSP page, displaying the message returned by the Controller.

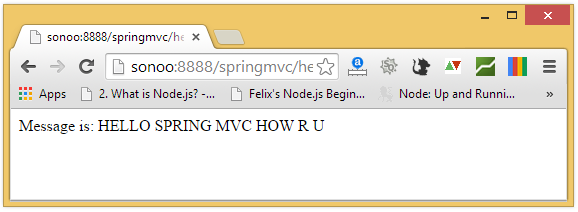
It must be located inside the WEB-INF/jsp directory for this example only.

# hellopage.jsp

Message is: ${message}

Output:





# Spring 3 MVC Multiple Controller Example

We can have a lot of controller classes in Spring Framework. In this example, we are creating two Controller classes HelloWorldController and WelcomeWorldController.

# 1) Controller Classes

# HelloWorldController.java

package com.javatpoint;

import org.springframework.stereotype.Controller;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.servlet.ModelAndView;

@Controller

public class HelloWorldController {

@RequestMapping("/hello")

public ModelAndView helloWorld() {

String message = "HELLO SPRING MVC";

return new ModelAndView("hellopage", "message", message);

}

}

# WelcomeWorldController.java

package com.javatpoint;

import org.springframework.stereotype.Controller;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.servlet.ModelAndView;

@Controller

public class WelcomeWorldController {

@RequestMapping("/welcome")

public ModelAndView helloWorld() {

String message = "WELCOME SPRING MVC";

return new ModelAndView("welcomepage", "message", message);

}

}

# 2) View components

To run this example, It must be located inside the WEB-INF/jsp directory.

# hellopage.jsp

Message is: ${message}

welcomepage.jsp

Message is: ${message}

# 3) Index page

It is the optional welcome page, that provide the links to invoke both controller.

# index.jsp

<a href="hello.html">click</a>|

<a href="welcome.html">click</a>

# Spring MVC Request Response Example

We can simply create login application by following the Spring MVC. We need to pass HttpServletRequest and HttpServletResponse objects in the request processing method of the Controller class. Let's see the example:

# 1) Controller Class

# HelloWorldController.java

package com.javatpoint;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

import org.springframework.stereotype.Controller;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.servlet.ModelAndView;

@Controller

public class HelloWorldController {

@RequestMapping("/hello")

public ModelAndView helloWorld(HttpServletRequest request,HttpServletResponse res) {

String name=request.getParameter("name");

String password=request.getParameter("password");

if(password.equals("admin")){

String message = "HELLO "+name;

return new ModelAndView("hellopage", "message", message);

}

else{

return new ModelAndView("errorpage", "message","Sorry, username or password error");

}

}

}

# 2) View components

To run this example, It must be located inside the WEB-INF/jsp directory.

# hellopage.jsp

Message is: ${message}

# errorpage.jsp

${message}

<jsp:include page="/index.jsp"></jsp:include>

# 3) Index page

It is the login page, that recieve name and password from the user.

# index.jsp

<form action="hello.html" method="post">

Name:<input type="text" name="name"/><br/>

Password:<input type="password" name="password"/><br/>

<input type="submit" value="login"/>

</form>

# Spring MVC Form Example

Here, we will learn how to handle a form data in spring MVC without using database. Here, we will use @Controler, @RequestMapping and @ModelAttribute annotations.

To display the input form, we are going to use <form:form> tag of spring framework. Let's see a simple example to store form data in a model object and display data of a list.

# Required Jar files

To run this example, you need to load:

1.Spring Core jar files

2.Spring Web jar files

# index.jsp

<a href="empform">Add Employee</a>

<a href="viewemp">View Employees</a>

# Emp.java

package com.javatpoint.beans;

public class Emp {

private int id;

private String name;

private float salary;

private String designation;

public Emp() {}

public Emp(int id, String name, float salary, String designation) {

super();

this.id = id;

this.name = name;

this.salary = salary;

this.designation = designation;

}

public int getId() {

return id;

}

public void setId(int id) {

this.id = id;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public float getSalary() {

return salary;

}

public void setSalary(float salary) {

this.salary = salary;

}

public String getDesignation() {

return designation;

}

public void setDesignation(String designation) {

this.designation = designation;

}

}

# EmpController.java

package com.javatpoint.controllers;

import java.util.ArrayList;

import java.util.List;

import org.springframework.stereotype.Controller;

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 com.javatpoint.beans.Emp;

@Controller

public class EmpController {

@RequestMapping("/empform")

public ModelAndView showform(){

//command is a reserved request attribute name, now use <form> tag to show object data

return new ModelAndView("empform","command",new Emp());

}

@RequestMapping(value="/save",method = RequestMethod.POST)

public ModelAndView save(@ModelAttribute("emp") Emp emp){

//write code to save emp object

//here, we are displaying emp object to prove emp has data

System.out.println(emp.getName()+" "+emp.getSalary()+" "+emp.getDesignation());

//return new ModelAndView("empform","command",emp);//will display object data

return new ModelAndView("redirect:/viewemp");//will redirect to viewemp request mapping

}

@RequestMapping("/viewemp")

public ModelAndView viewemp(){

//write the code to get all employees from DAO

//here, we are writing manual code of list for easy understanding

List<Emp> list=new ArrayList<Emp>();

list.add(new Emp(1,"rahul",35000f,"S.Engineer"));

list.add(new Emp(2,"aditya",25000f,"IT Manager"));

list.add(new Emp(3,"sachin",55000f,"Care Taker"));

return new ModelAndView("viewemp","list",list);

}

}

# 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/xml/ns/javaee/web-app\_2\_5.xsd">

<servlet>

<servlet-name>spring</servlet-name>

<servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>

<load-on-startup>1</load-on-startup>

</servlet>

<servlet-mapping>

<servlet-name>spring</servlet-name>

<url-pattern>/</url-pattern>

</servlet-mapping>

</web-app>

# spring-servlet.xml

<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xmlns:p="http://www.springframework.org/schema/p"

xmlns:context="http://www.springframework.org/schema/context"

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.javatpoint.controllers"></context:component-scan>

<bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">

<property name="prefix" value="/WEB-INF/jsp/"></property>

<property name="suffix" value=".jsp"></property>

</bean>

</beans>

# empform.jsp

<%@ taglib uri="http://www.springframework.org/tags/form" prefix="form"%>

<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>

<form:form method="post" action="save">

<table >

<tr>

<td>Name : </td>

<td><form:input path="name" /></td>

</tr>

<tr>

<td>Salary :</td>

<td><form:input path="salary" /></td>

</tr>

<tr>

<td>Designation :</td>

<td><form:input path="designation" /></td>

</tr>

<tr>

<td colspan="2"><input type="submit" value="Save" /></td>

</tr>

</table>

</form:form>

# viewemp.jsp

<%@ taglib uri="http://www.springframework.org/tags/form" prefix="form"%>

<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>

<table border="2" width="70%" cellpadding="2">

<tr><th>Id</th><th>Name</th><th>Salary</th><th>Designation</th></tr>

<c:forEach var="emp" items="${list}">

<tr>

<td>${emp.id}</td>

<td>${emp.name}</td>

<td>${emp.salary}</td>

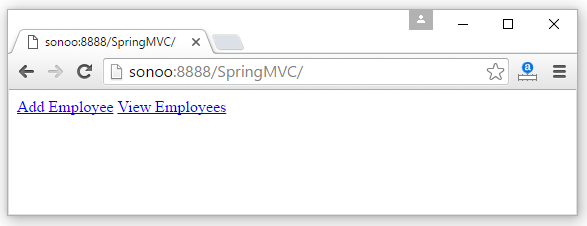
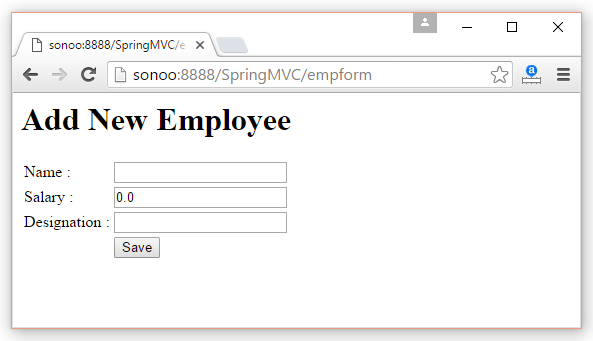
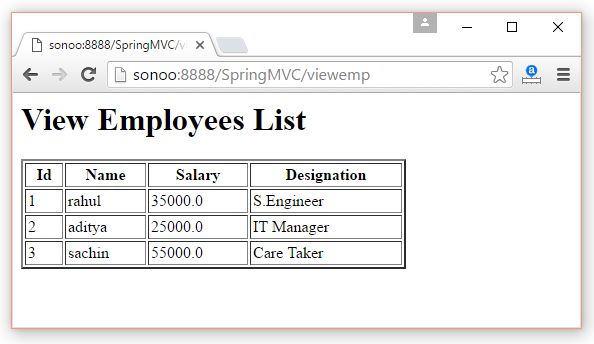
<td>${emp.designation}</td>

</tr>

</c:forEach>

</table>

# Output

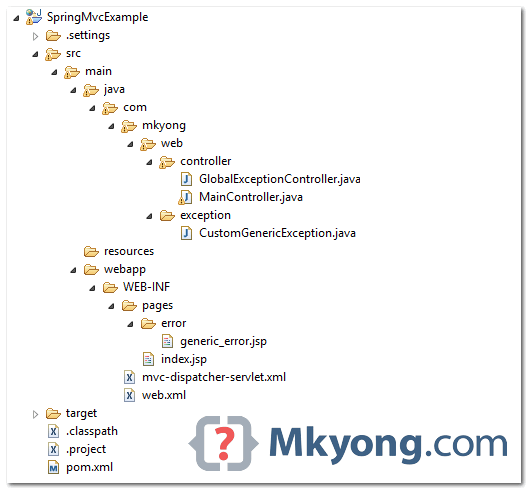
# Spring MVC @ExceptionHandler Example

In this tutorial, we show you how to do exception handling in Spring MVC frameworks. Normally, we use @ExceptionHandler to decide which “view” should be returned back if certain exception is raised.

P.S This @ExceptionHandler class is available since Spring 3.0

# 1. Project Structure

Review the project directory structure, a standard Maven project.



# 2. Custom Exception

A custom exception, with custom error code and error description.

# CustomGenericException.java

package com.mkyong.web.exception;

public class CustomGenericException extends RuntimeException {

private static final long serialVersionUID = 1L;

private String errCode;

private String errMsg;

public String getErrCode() {

return errCode;

}

public void setErrCode(String errCode) {

this.errCode = errCode;

}

public String getErrMsg() {

return errMsg;

}

public void setErrMsg(String errMsg) {

this.errMsg = errMsg;

}

public CustomGenericException(String errCode, String errMsg) {

this.errCode = errCode;

this.errMsg = errMsg;

}

}

# 3. Spring Controller

A Spring controller, review the execution-flows below :

1.If user provide a /error request, it throws “CustomGenericException”, and the handleCustomException() method will be fired.

2.If user provide a /io-error request, it throws “IOException”, and the handleAllException() method will be fired.

# MainController.java

package com.mkyong.web.controller;

import java.io.IOException;

import org.springframework.stereotype.Controller;

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.servlet.ModelAndView;

import com.mkyong.web.exception.CustomGenericException;

@Controller

public class MainController {

@RequestMapping(value = "/{type:.+}", method = RequestMethod.GET)

public ModelAndView getPages(@PathVariable("type") String type)

throws Exception {

if ("error".equals(type)) {

// go handleCustomException

throw new CustomGenericException("E888", "This is custom message");

} else if ("io-error".equals(type)) {

// go handleAllException

throw new IOException();

} else {

return new ModelAndView("index").addObject("msg", type);

}

}

@ExceptionHandler(CustomGenericException.class)

public ModelAndView handleCustomException(CustomGenericException ex) {

ModelAndView model = new ModelAndView("error/generic\_error");

model.addObject("errCode", ex.getErrCode());

model.addObject("errMsg", ex.getErrMsg());

return model;

}

@ExceptionHandler(Exception.class)

public ModelAndView handleAllException(Exception ex) {

ModelAndView model = new ModelAndView("error/generic\_error");

model.addObject("errMsg", "this is Exception.class");

return model;

}

}

# 4. JSP Pages

pages/index.jsp

<%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>

<html>

<body>

<h2>Spring MVC @ExceptionHandler Example</h2>

<c:if test="${not empty msg}">

<h2>${msg}</h2>

</c:if>

</body>

</html>

pages/error/generic\_error.jsp

<%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>

<html>

<body>

<c:if test="${not empty errCode}">

<h1>${errCode} : System Errors</h1>

</c:if>

<c:if test="${empty errCode}">

<h1>System Errors</h1>

</c:if>

<c:if test="${not empty errMsg}">

<h2>${errMsg}</h2>

</c:if>

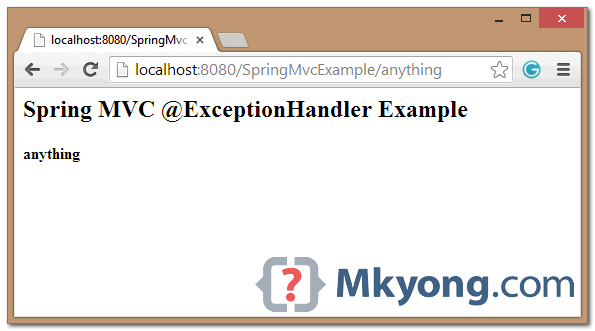
</body>

</html>

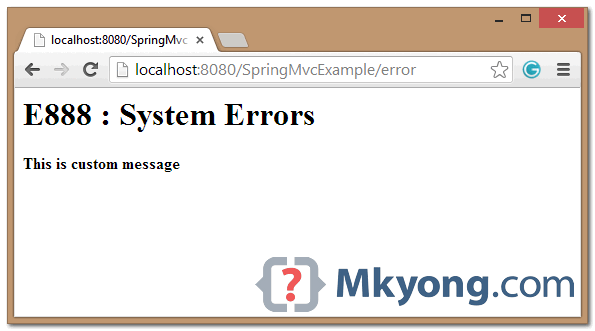
# 5. Testing

Review following 3 test cases :

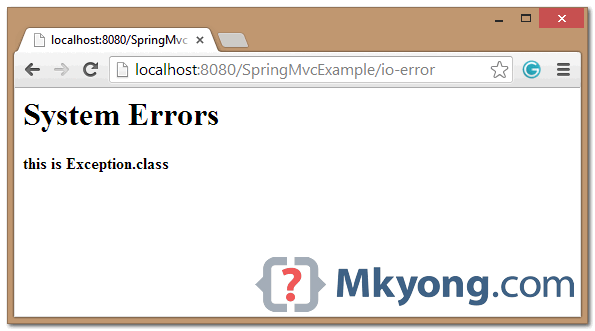
1. http://localhost:8080/SpringMvcExample/anything



2. http://localhost:8080/SpringMvcExample/error



3. http://localhost:8080/SpringMvcExample/io-error



# 6. @ControllerAdvice Example

The above @ExceptionHandler example is only apply to a single controller, to apply it globally (all controllers), annotate a class with @ControllerAdvice.

# GlobalExceptionController.java

package com.mkyong.web.controller;

import org.springframework.web.bind.annotation.ControllerAdvice;

import org.springframework.web.bind.annotation.ExceptionHandler;

import org.springframework.web.servlet.ModelAndView;

import com.mkyong.web.exception.CustomGenericException;

@ControllerAdvice

public class GlobalExceptionController {

@ExceptionHandler(CustomGenericException.class)

public ModelAndView handleCustomException(CustomGenericException ex) {

ModelAndView model = new ModelAndView("error/generic\_error");

model.addObject("errCode", ex.getErrCode());

model.addObject("errMsg", ex.getErrMsg());

return model;

}

@ExceptionHandler(Exception.class)

public ModelAndView handleAllException(Exception ex) {

ModelAndView model = new ModelAndView("error/generic\_error");

model.addObject("errMsg", "this is Exception.class");

return model;

}

}

# 7. Download Source Code

Download – SpringMvc-ExceptionHandler-Example.zip (15 KB)

# References

Spring @ExceptionHandler JavaDoc

Spring @ControllerAdvice JavaDoc

Spring MVC Exception Handling Example

Tags : exception spring mvc

* Your understanding is correct. The ApplicationContext is where your Spring beans live. The purpose of the ContextLoaderListener is two-fold:
* to tie the lifecycle of the ApplicationContext to the lifecycle of the ServletContext and
* to automate the creation of the ApplicationContext, so you don't have to write explicit code to do create it - it's a convenience function.
* Another convenient thing about the ContextLoaderListener is that it creates a WebApplicationContext and WebApplicationContext provides access to the ServletContext via ServletContextAware beans and the getServletContext method.
* asdffasdfasdfasdfasddfasdfasdfasdfasdffasdfasddfasd