Servlet Exception and Error Handling Example Tutorial

Sometime back I wrote a post about **Exception Handling in Java** but when it comes to web application, we need more than normal exception handling in java.

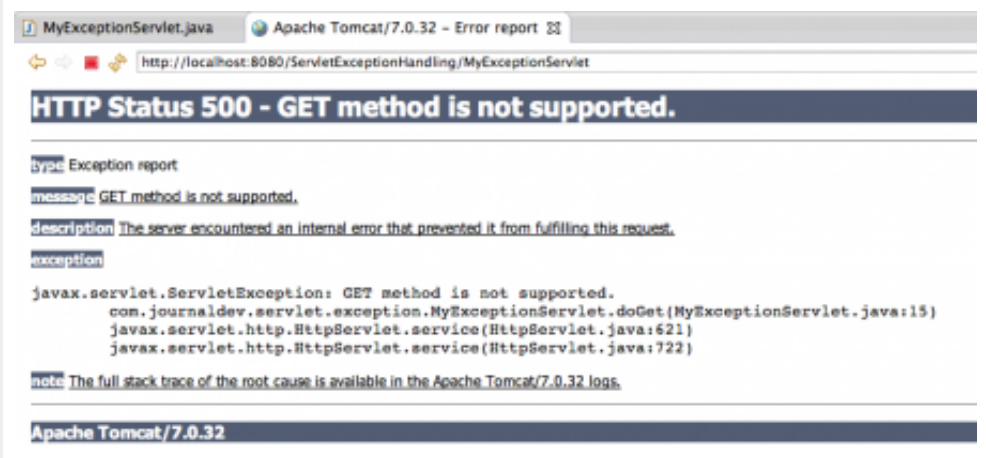
Servlet Exception

If you notice, doGet() and doPost() methods throw `ServletException` and `IOException`, let's see what happens when we throw these exception from our application. I will write a simple servlet that will throw the ServletException.

MyExceptionServlet.java

```
1  package com.journaldev.servlet.exception;
2
3  import java.io.IOException;
4  import javax.servlet.ServletException;
5  import javax.servlet.annotation.WebServlet;
6  import javax.servlet.http.HttpServlet;
7  import javax.servlet.http.HttpServletRequest;
8  import javax.servlet.http.HttpServletResponse;
9
10 @WebServlet("/MyExceptionServlet")
11 public class MyExceptionServlet extends HttpServlet {
12     private static final long serialVersionUID = 1L;
13
14     protected void doGet(HttpServletRequest request, HttpServletResponse resp
15         throw new ServletException("GET method is not supported.");
16     }
17
18 }
```

Now when we invoke this servlet through browser with GET method, we get response like below image.



Since browser understand only HTML, when our application throw exception, servlet container processes the exception and generate a HTML response. This logic is specific to servlet container, I am using tomcat and getting this error page but if you will use some other servers like JBoss or Glassfish, you might get different error HTML response.

The problem with this response is that it's of no value to user. Also it's showing our application classes and server details to user that makes no sense to user and it's not good from security point of view.

Servlet Error

I am sure you must have seen 404 error when you are trying to hit a URL that doesn't exists. Let's see how our servlet container responds to 404 error. If we send request for an invalid URL, we get response HTML like below image.

Again it's a generic HTML generated by server on our behalf and hold little to no value to the user.

Servlet Exception and Error Handling

Servlet API provides support for custom Exception and Error Handler servlets that we can configure in deployment descriptor, the whole purpose of these servlets are to handle the Exception or Error raised by application and send HTML response that is useful for the user. We can provide link to application home page or some details to let user know what went wrong.

So first of all we need to create a custom Exception and Error Handler servlet. We can have multiple exception and error handler servlets for the application but for simplicity I will create a single servlet and use it for both exceptions and errors.

AppExceptionHandler.java

```
1  package com.journaldev.servlet.exception;
2
3  import java.io.IOException;
4  import java.io.PrintWriter;
5
6  import javax.servlet.ServletException;
7  import javax.servlet.annotation.WebServlet;
8  import javax.servlet.http.HttpServlet;
```

```
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

@WebServlet("/AppExceptionHandler")
public class AppExceptionHandler extends HttpServlet {
    private static final long serialVersionUID = 1L;

    protected void doGet(HttpServletRequest request,
            HttpServletResponse response) throws ServletException, IOExceptio
        processError(request, response);
    }

    protected void doPost(HttpServletRequest request,
            HttpServletResponse response) throws ServletException, IOExceptio
        processError(request, response);
    }

    private void processError(HttpServletRequest request,
            HttpServletResponse response) throws IOException {
        // Analyze the servlet exception
        Throwable throwable = (Throwable) request
                .getAttribute("javax.servlet.error.exception");
        Integer statusCode = (Integer) request
                .getAttribute("javax.servlet.error.status_code");
        String servletName = (String) request
                .getAttribute("javax.servlet.error.servlet_name");
        if (servletName == null) {
            servletName = "Unknown";
        }
        String requestUri = (String) request
                .getAttribute("javax.servlet.error.request_uri");
        if (requestUri == null) {
            requestUri = "Unknown";
        }

        // Set response content type
          response.setContentType("text/html");

          PrintWriter out = response.getWriter();
          out.write("<html><head><title>Exception/Error Details</title></head
          if(statusCode != 500){
              out.write("<h3>Error Details</h3>");
              out.write("<strong>Status Code</strong>:"+statusCode+"<br>");
              out.write("<strong>Requested URI</strong>:"+requestUri);
          }else{
              out.write("<h3>Exception Details</h3>");
              out.write("<ul><li>Servlet Name:"+servletName+"</li>");
              out.write("<li>Exception Name:"+throwable.getClass().getName()+'
              out.write("<li>Requested URI:"+requestUri+"</li>");
              out.write("<li>Exception Message:"+throwable.getMessage()+"</li
              out.write("</ul>");
          }

          out.write("<br><br>");
          out.write("<a href=\"index.html\">Home Page</a>");
          out.write("</body></html>");
    }
}
```

Let's see how we can configure it in deployment descriptor and then we will understand it's implementation and how it works.

```
web
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://
  <display-name>ServletExceptionHandling</display-name>
  <welcome-file-list>
    <welcome-file>index.html</welcome-file>
  </welcome-file-list>

  <error-page>
    <error-code>404</error-code>
    <location>/AppExceptionHandler</location>
  </error-page>

  <error-page>
  <exception-type>javax.servlet.ServletException</exception-type>
  <location>/AppExceptionHandler</location>
  </error-page>
</web-app>
```

As you can see, it's very easy to specify Exception handler servlets for the application using **error-page** element. Each error-page element should have either **error-code** or **exception-type** element. We define the exception handler servlet in **location** element.

Based on above configuration, if the application throw 404 error or ServletException, it will be handled by AppExceptionHandler servlet.

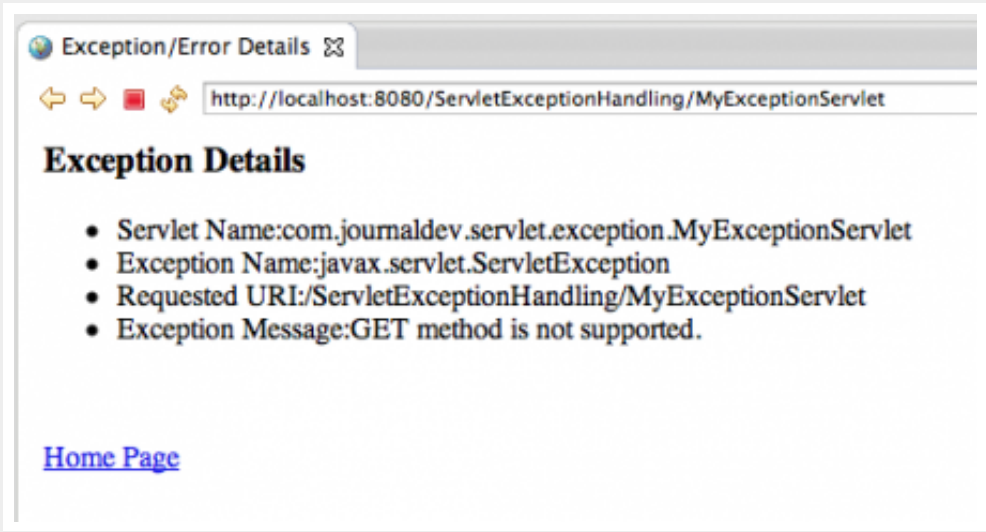
When such exception and error scenario appears, servlet container will invoke the corresponding HTTP method of the Exception Handler servlet and pass the request and response object. Notice that I have provided implementation of both doGet() and doPost() methods so that it can handle GET and POST requests and using a common method to process them.

Before servlet container invokes the servlet to handle the exception, it sets some attributes in the request to get useful information about the exception, some of them are **javax.servlet.error.exception**, **javax.servlet.error.status_code**, **javax.servlet.error.servlet_name** and **javax.servlet.error.request_uri**.

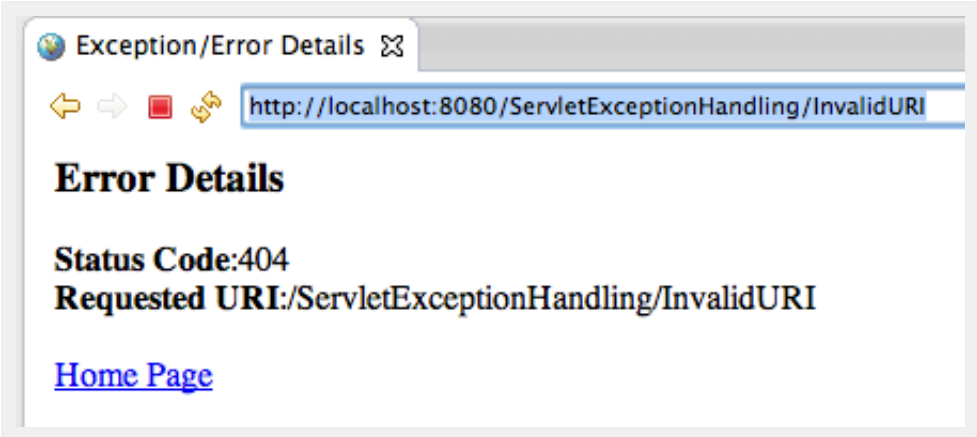
For exception, status code is always 500 that corresponds to the "Internal Server Error", for other types of error we get different error codes such as 404, 403 etc.

Using the status code, our implementation presents different types of HTML response to the user. It also provides a hyperlink to the home page of the application.

Now when we will hit our servlet that is throwing ServletException, we will get a response like below image.



If we try to access an invalid URL that will result in 404 response, we will get response like below image.



Doesn't it look good and helps user to easily understand what happened and provides them a way to go to the correct location. It also avoids sending application sensitive information to the user. We should always have exception handlers in place for our web application.

If you want to handle runtime exceptions and all other exceptions in a single exception handler, you can provide exception-type as Throwable.

```
<error-page>
  <exception-type>java.lang.Throwable</exception-type>
  <location>/AppExceptionHandler</location>
</error-page>
```

If there are multiple error-page entries, let's say one for Throwable and one for IOException and application throws FileNotFoundException then it will be handled by error handler of IOException.

You can also use JSP page as exception handler, just provide the location of jsp file rather than servlet mapping.

That's all for exception handling in web application, I hope you liked it.

Download Servlet Exception Handling Example Project

642 downloads

Check out other articles in this series:

1. Java Web Application
2. Java Servlet Tutorial
3. Servlet Session Management
4. Servlet Filter
5. Servlet Listeners
6. Cookies in Servlets
7. Servlet File Upload and Download Example