Error Handling and Debugging

Syntax Errors

Improperly terminated directive

<%@ page contentType="text/html" >
Missing %

Improperly terminated action

<c:out value="${1 + 2 + 3}" >

it's missing the closing slash required for an empty element). The message reported by Tomcat in this case is: /ch9/error2.jsp(14,0) Unterminated <c:out tag

Mistyped attribute name

<c:out valu="${1 + 2 + 3}" />

Tomcat reports the problem like this:
/ch9/error3.jsp(11,16) According to the TLD attribute value is mandatory for tag out

In this case, the typo is in the name of a mandatory attribute, so Tomcat reports it as missing. If the typo is in the name of an optional attribute, Tomcat reports it as an invalid attribute name.

Missing end quote in attribute value

<c:out value="${1 + 2 + 3} default="Doh!" />

If you look carefully at the <c:out> element, you see that the closing quote for the value attribute is missing. If another attribute is specified for the same element, like the default attribute used here, Tomcat reports the problem like this: /ch9/error4.jsp(11,55) equal symbol expected

Missing both curly braces

<c:out value="$1 + 2 + 3" />

This is an easy mistake to make, but it's not recognized as a syntax error at all. To the container, this means that the value is a plain-text value, not an expression. When used with the <c:out> action, it's easy to figure out what's wrong because the text value is added to the response as is instead of the being evaluated: $1 + 2 + 3.

But if you make this mistake with an attribute value that should provide the action with input to process in some way, the problem may not be so easy to spot. For instance, if you forget the curly braces for the <c:forEach> items attribute, it takes it as a text value and loops once over its body with the text as a single element.

Missing end curly brace

<c:out value="${1 + 2 + 3" />

The error message contains three pieces of information:
• The position:
/ch9/error6.jsp(10,16)

• A generic message that includes the complete EL expression:
"${1 + 2 + 3" contains invalid expression(s)

• A more detailed message about the problem:
Encountered "", expected one of ..."

Misspelled property name

The Current URI: <c:out value="${pageContext.request.requestUri}"

The problem here is that the property name is misspelled: it should be requestURI ("URI" in all caps) instead of requestUri.

Misspelled parameter name

The missing parameter: <c:out value="${param.misspelled}" />

Here it's the name of a request parameter that is misspelled, and it's not reported as an error. Instead the expression evaluates to null, which the <c:out> action converts to an empty string.

https://www.tutorialspoint.com/jsp/jsp_exception_handling.htm

Checked exceptions

A checked exception is an exception that is typically a user error or a problem that cannot be foreseen by the programmer. For example, if a file is to be opened, but the file cannot be found, an exception occurs. These exceptions cannot simply be ignored at the time of compilation.

Runtime exceptions

A runtime exception is an exception that probably could have been avoided by the programmer. As opposed to the checked exceptions, runtime exceptions are ignored at the time of compliation.

Errors

These are not exceptions at all, but problems that arise beyond the control of the user or the programmer. Errors are typically ignored in your code because you can rarely do anything about an error. For example, if a stack overflow occurs, an error will arise. They are also ignored at the time of compilation.

Using Exception Object

The exception object is an instance of a subclass of Throwable (e.g., java.lang. NullPointerException) and is only available in error pages. Following table lists out the important methods available in the Throwable class.

S.No.	Methods & Description
1	public String getMessage() Returns a detailed message about the exception that has occurred. This message is initialized in the Throwable constructor.
2	public Throwable getCause() Returns the cause of the exception as represented by a Throwable object.

3	public String toString() Returns the name of the class concatenated with the result of getMessage().
4	public void printStackTrace() Prints the result of toString() along with the stack trace to System.err, the error output stream.
5	public StackTraceElement [] getStackTrace() Returns an array containing each element on the stack trace. The element at index 0 represents the top of the call stack, and the last element in the array represents the method at the bottom of the call stack.
6	public Throwable fillInStackTrace() Fills the stack trace of this Throwable object with the current stack trace, adding to any previous information in the stack trace.

Using Try...Catch Block

If you want to handle errors within the same page and want to take some action instead of firing an error page, you can make use of the try....catch block.

Following is a simple example which shows how to use the try...catch block. Let us put the following code in main.jsp −

```
<html>
  <head>
    <title>Try...Catch Example</title>
  </head>

  <body>
    <%
      try {
        int i = 1;
        i = i / 0;
        out.println("The answer is " + i);
      }
      catch (Exception e) {
        out.println("An exception occurred: " + e.getMessage());
      }
```

```
    %>
  </body>
</html>
```

The following are a few hints and suggestions that may aid you in your debugging.

Using System.out.println()

System.out.println() is easy to use as a marker to test whether a certain piece of code is being executed or not. We can print out variable values as well. Consider the following additional points −

- Since the System object is part of the core Java objects, it can be used everywhere without the need to install any extra classes. This includes Servlets, JSP, RMI, EJB's, ordinary Beans and classes, and standalone applications.
- Compared to stopping at breakpoints, writing to System.out doesn't interfere much with the normal execution flow of the application, which makes it very valuable when the iming is crucial.

Following is the syntax to use System.out.println() −

```
System.out.println("Debugging message");
```

Following example shows how to use System.out.print() −

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

<html>
  <head><title>System.out.println</title></head>
  <body>
    <c:forEach var = "counter" begin = "1" end = "10" step = "1" >

      <c:out value = "${counter-5}"/></br>
      <% System.out.println( "counter = " + pageContext.findAttribute("counter") ); %>
    </c:forEach>

  </body>
</html>
```

Access the above JSP, the browser will show the following result −

```
-4
-3
-2
-1
```

0
1
2
3
4
5

If you are using Tomcat, you will also find these lines appended to the end of stdout.log in the logs directory.

counter = 1
counter = 2
counter = 3
counter = 4
counter = 5
counter = 6
counter = 7
counter = 8
counter = 9
counter = 10

This way you can bring variables and other information into the system log which can be analyzed to find out the root cause of the problem or for various other reasons.