

Debugging from JSP and Java Applications

This tutorial gives you an overview of how to use the debugging feature of eclipse to debug your web or Java projects.

General

Author:

Sascha Wolski

Sebastian Hennebrueder

<http://www.laliluna.de/tutorial.html> – Tutorials for Struts, JSF, EJB, Hibernate, xdoclet, eclipse and more.

Datum:

January, 25 2005

Development Tools

Eclipse 3.x

MyEclipse plugin 3.8

(A cheap and quite powerful Extension to Eclipse to develop Web Applications and EJB (J2EE) Applications. I think that there is a test version available at MyEclipse.)

PDF Version des Tutorials

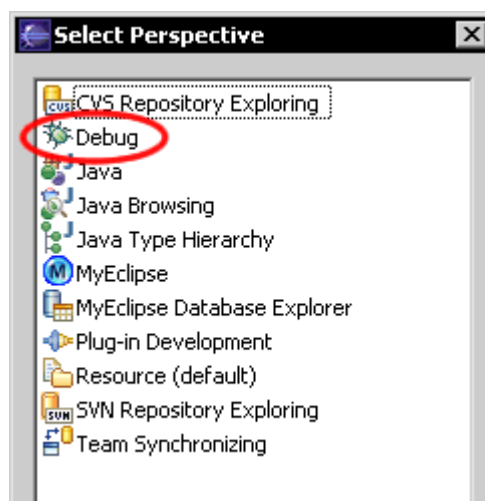
<http://www.laliluna.de/assets/tutorials/debugging-jsp-java-tutorial-en.pdf>

Introduction

The Eclipse Platform features a Java debugger that provides all standard debugging functionality, including the ability to perform step execution, setting of breakpoints and values, inspecting variables and values, and to suspend and resume threads. With the extension MyEclipse you are able to debug JSP and included files too.

The debug perspective

First we will explain the debug perspective. You can activate the perspective under **Windows > Open Perspective > Other...**

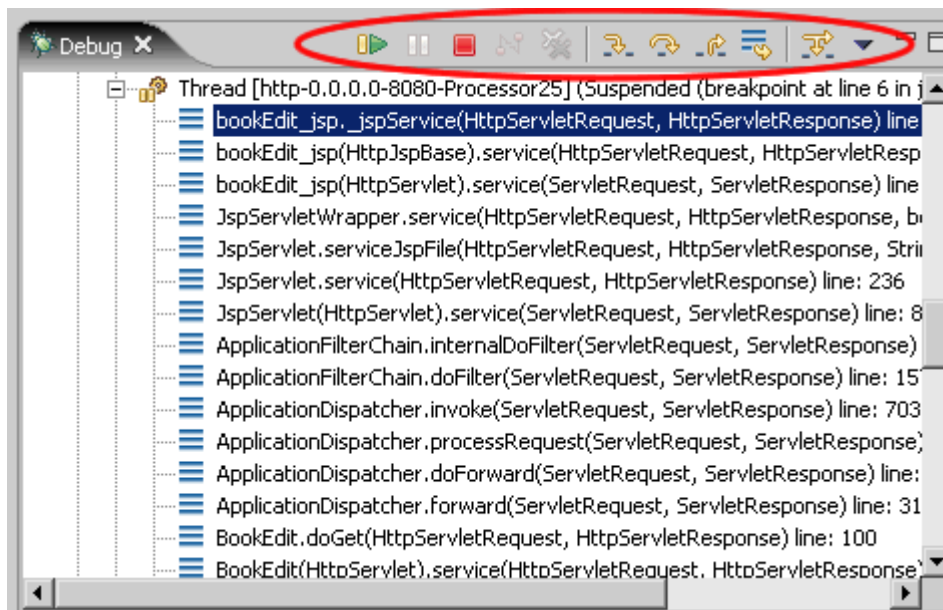


Alternatively, you can click on the debug icon of the perspective tool bar.



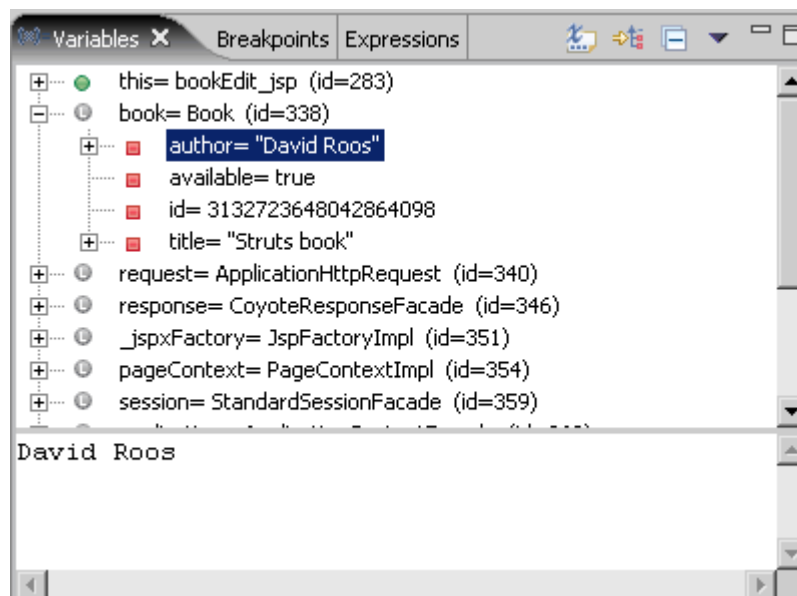
Debug view

The Debug view displays the stack trace for the suspended threads for each target you are debugging. Each entry is the variable state of a method right when it called the next method. The view allows you to manage the debugging of a program in the workbench.



Variables view

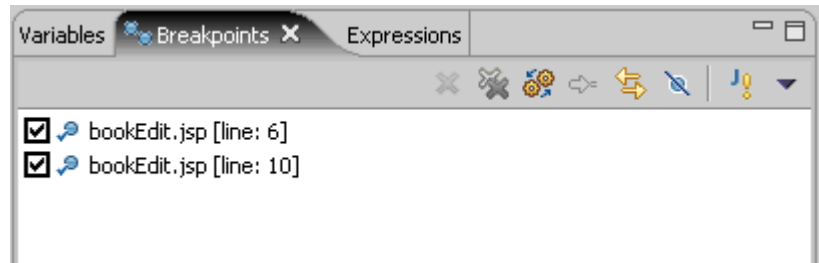
The view displays information about the variables in the selected class. You can get information about the variable like value, size and more from this view.



Breakpoints view

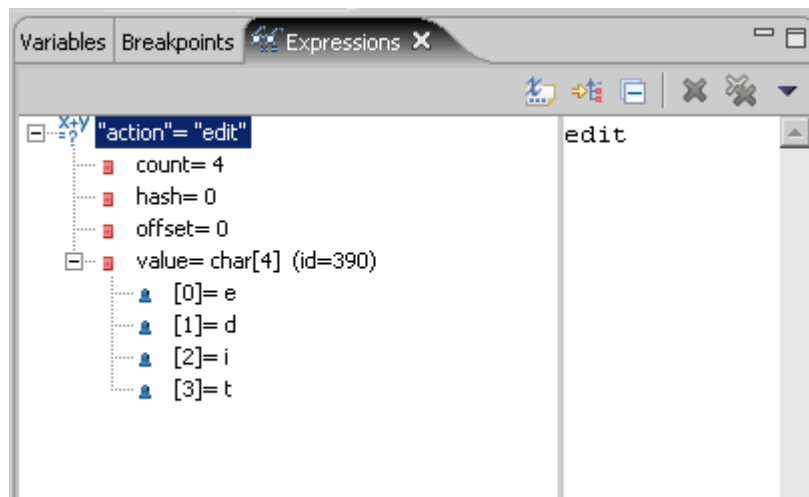
The view lists all breakpoints you have set in the workbench project. In this view, you can enable or disable breakpoints, remove them, or add a new ones. You can also double-click a breakpoint to display its location in the editor.

This view also lists Java exception breakpoints, which suspend execution at the point where the exception is thrown. You can add or remove exceptions.



Expressions view

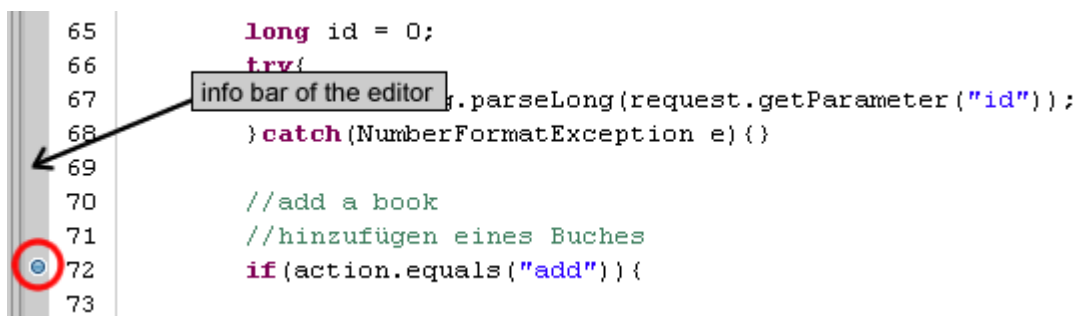
You can inspect data from each class of a suspended thread, and other places in this view. It opens automatically when an item is added to the view.



Debug a web project

In the next steps I will explain how you can debug a project. I will refer to the project JSP + Servlet you can download on our site. <http://www.laliluna.de/assets/tutorials/java-servlets-jsp-tutorial.zip>

Open the class **BookEdit.java** and set a breakpoint in the method **doGet(..)** at line 72. You can set a breakpoint by double-click on the info bar on the left edge of the editor, or right click **Toggle Breakpoint**.



Set the second breakpoint in the **bookEdit.jsp** file on the HTML start tag at line 14.

Note: If you don't have installed MyEclipse, you can't set a breakpoint in jsp and included files.

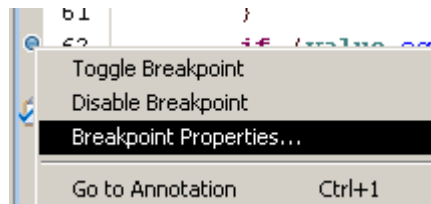
```
8 <%
9 String path = request.getContextPath();
10 String basePath = request.getScheme()+ "://" + request.getS
11 %>
12
13 <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitions
14 <html>
15 <head>
16 <base href="<%=basePath%>">
17
18 <title>Book edit page</title>
19 </head>
```

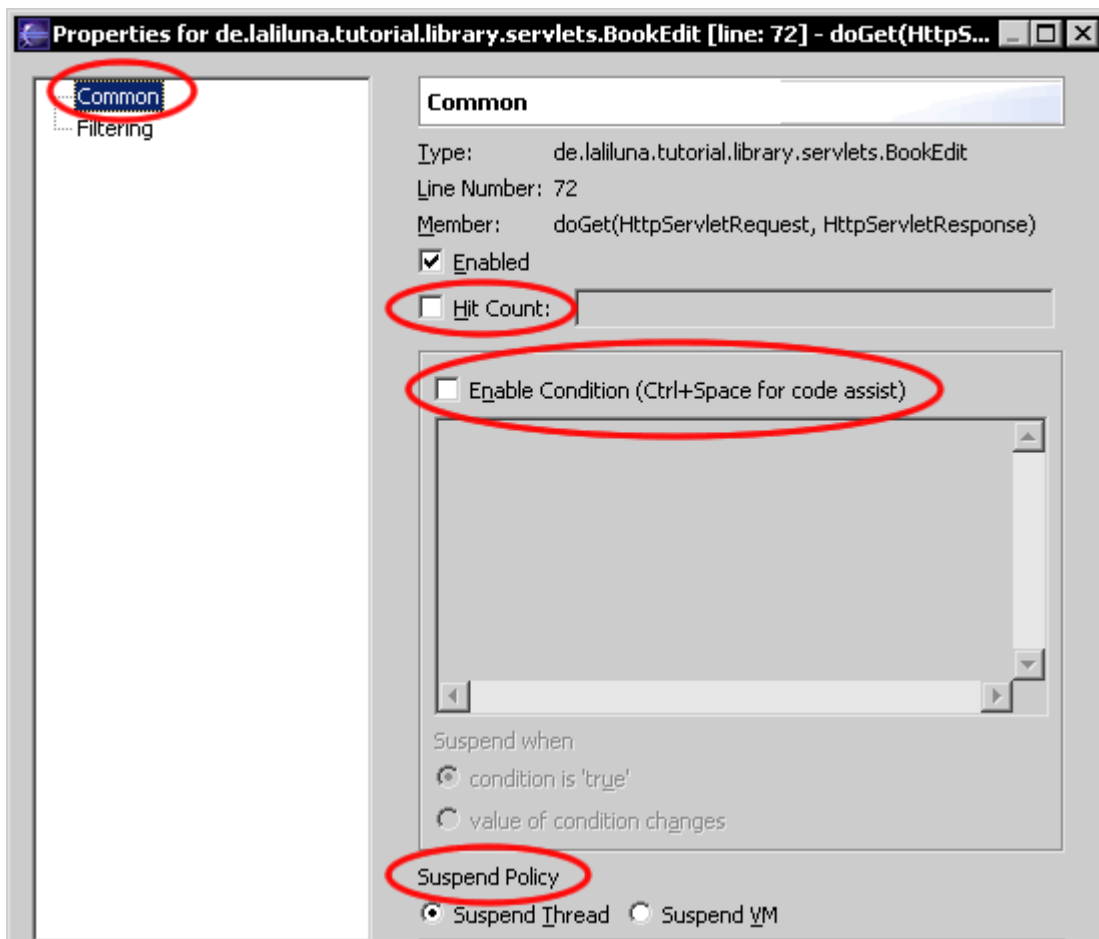
To see all breakpoints you can use the Breakpoints view in the debug perspective.

Breakpoint Properties

You can apply some properties to each breakpoint, for example how many times a breakpoint can hit before it suspends the thread or on which condition it suspends the thread.

Select the breakpoint right mouse button and choose the option **Breakpoint Properties**.





Hit Count

The hit count sets a number of times the breakpoint can be executed before the thread suspends. Very helpful in a loop expression or if you want to know the value of a expression after some hits.

Enable Condition

There are two options to suspend a thread by using a condition.

- if the enabled condition is true
- if the enabled condition changes

If you want that the condition suspends the thread when it is true select the option **condition is true** on **Suspend when**.

Example:

```
action.equals("edit");
```

If you want that the condition suspends the thread when the value of the condition changes select the option **value of the condition changes** on **Suspend when**.

Suspend Policy

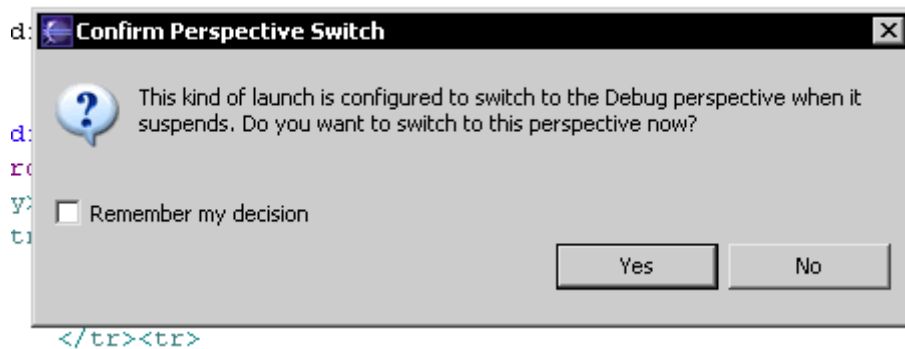
You can define if the breakpoint suspends only the thread or the complete virtual machine.

Deploy and Debug

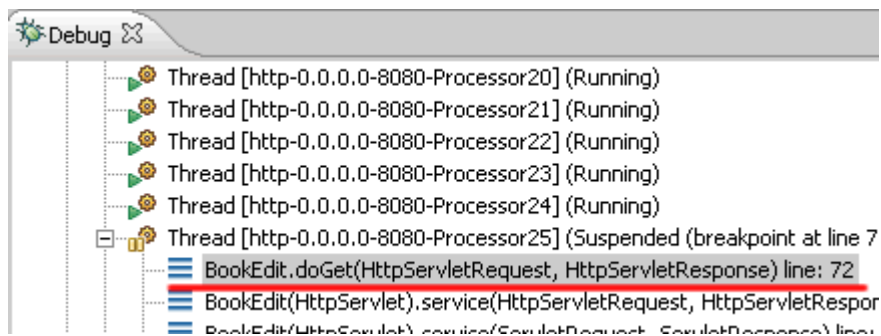
Now we want to debug the project, so deploy it and call it in your web browser.
Choose **Edit** on the list of books.

Author	Book name	Available		
David Roos	Struts book	<input checked="" type="checkbox"/>	Edit	Delete
Micheal Jackson	Java book	<input checked="" type="checkbox"/>	Edit	Delete
Bruce Lee	Java2 book	<input type="checkbox"/>	Edit	Delete
Tom Jones	EJB book	<input checked="" type="checkbox"/>	Edit	Delete
Mc Donald	Jboss for beginners	<input type="checkbox"/>	Edit	Delete
Lars Mars	Using Myeclipse for cooking	<input checked="" type="checkbox"/>	Edit	Delete
Mary Jane	EJB or spending your weekends	<input checked="" type="checkbox"/>	Edit	Delete

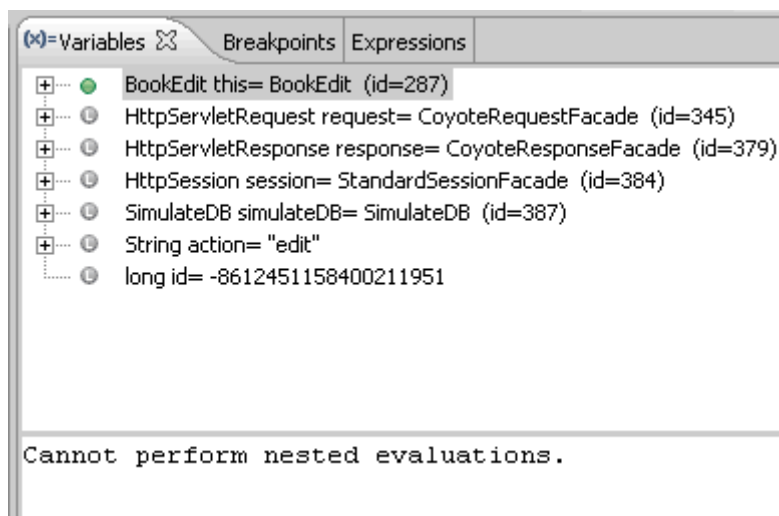
After you clicked the **Edit** link, eclipse shows you the following confirm window. You can choose if you want to switch to the debug perspective. Yes, you want it ;-)



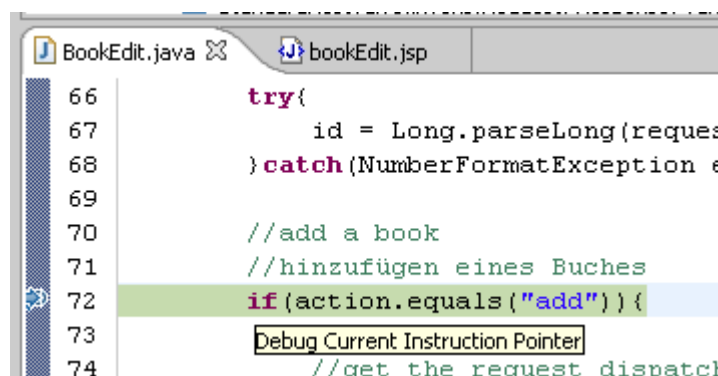
The first entry in the debug view represents the state of the method where the breakpoint was set. You can preview the state of an entry simply by clicking on it, also the file will be open where the method is placed.



You can see the values of all variables of the selected entry in the Variables view.



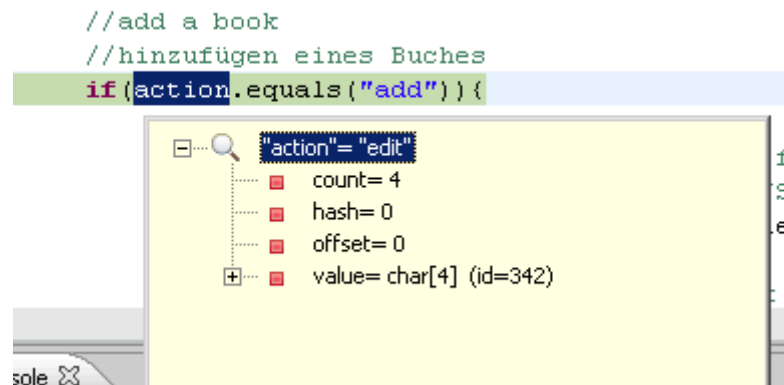
A marked line and an arrow at the breakpoint shows that the debugger is suspended on this line.



If you've decided that you missed an important place in the execution process, perhaps the breakpoint was in the wrong place, or maybe you accidentally stepped over some code you wanted to inspect earlier in the process. Eclipse has a feature called **Drop to frame**, that essentially lets you 'rewind' the execution to the beginning of any method in the stack. This is especially useful when you perform variable modification or code hot swapping. Right click on the frame and choose the option **Drop to Frame**.

Inspect expressions

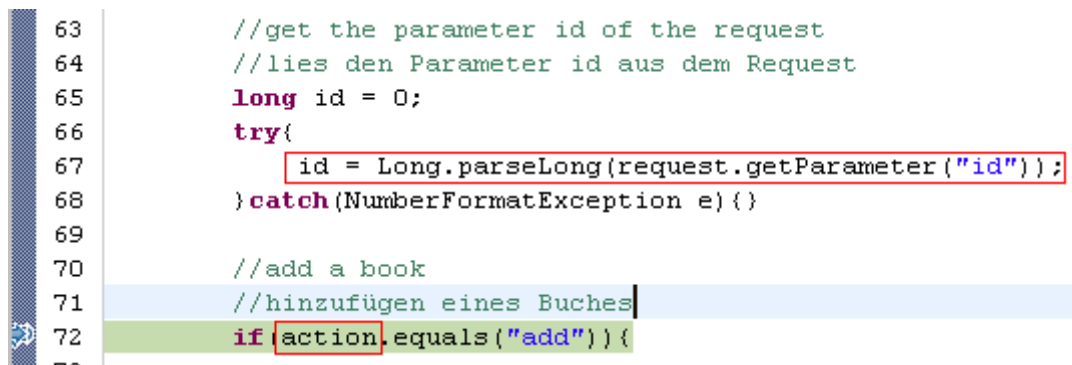
To get informations about expression, you can inspect a expression. Right click on the marked expression, you want to inspect and choose **Inspect** or press **Ctrl + Shift + I**. A pop-up window appears that holds the informations about the expression.



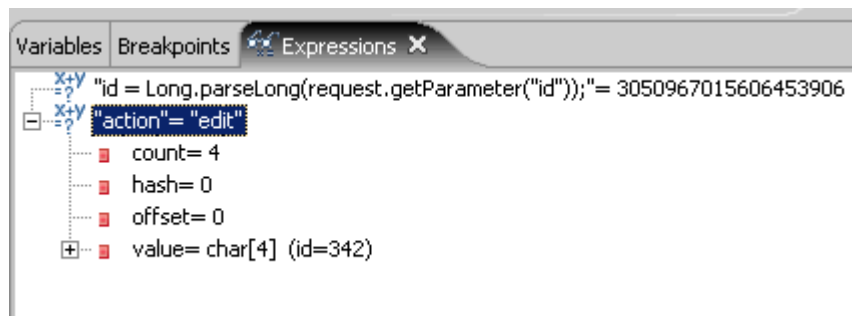
Watch expressions

Watch is similar to inspect an expression. Watch means that the expression will be added to the Expressions view, where you can watch the informations about the expression. Add two expressions to the Expressions view.

Mark the expressions, right click and choose **Watch**.



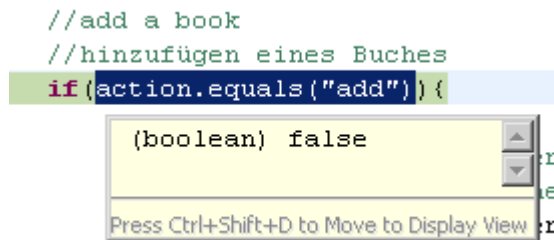
Now the two expressions are in the Expressions view.



Display expressions

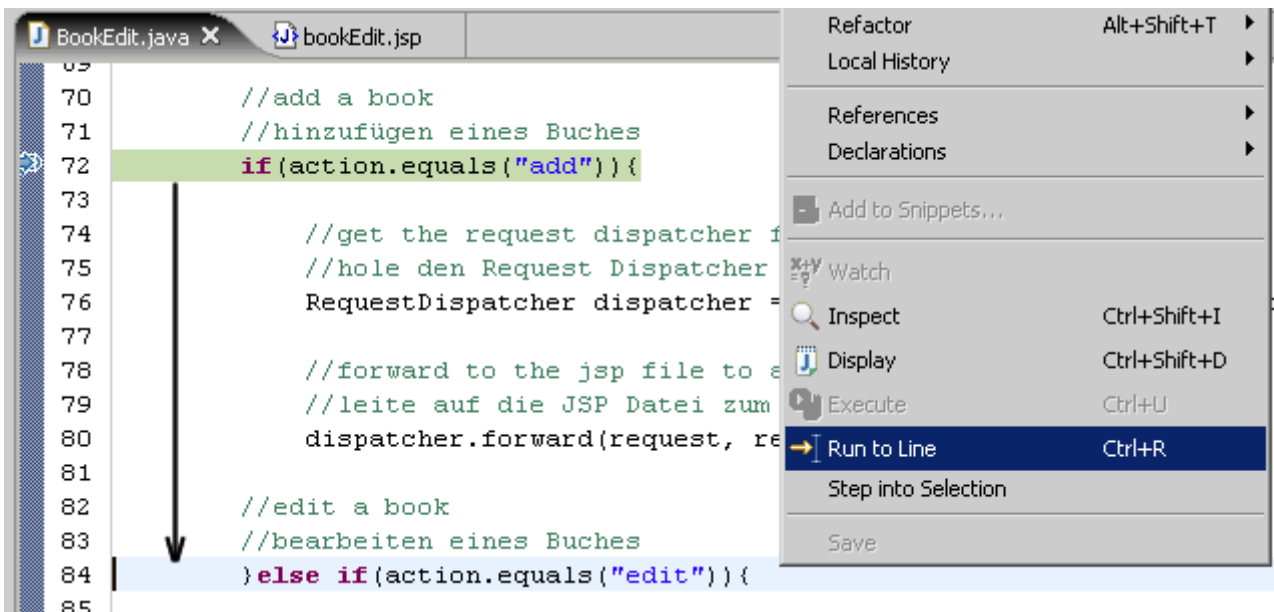
If you want to display the type and the value of an expression, mark the expression and choose the **Display** option from the context menu (right mouse button) or press **Ctrl + Shift + D**.

```
//add a book
//hinzufügen eines Buches
if(action.equals("add")){
```



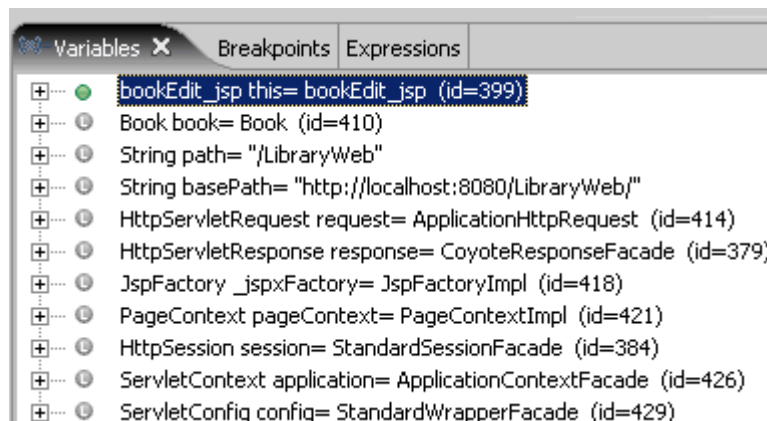
Run to Line

If you set a breakpoint and somewhere under the breakpoint is a line you want to go, you can use the option **Run to Line**. The code between the breakpoint and the selected line will be executed. Select the line you want to go, press the right mouse button and choose the option **Run to Line** or use the key binding **Ctrl + R**.



Debugging JSP files (supported by MyEclipse)

Debugging a JSP file is the same like debugging a Java class, but the most features (Watch, Inspect, etc) are not implemented. The only way to get the values of variables is the Variables view.



That's it. You will only need to debug, when you make mistakes. Avoid them and forget the tutorial.