DebuggingCocoon

This page describes how to use a **Java-level debugger** to debug **Cocoon itself**. For notes on debugging Cocoon-based **applications**, see also: DebuggingWithViews, Cocoon Profiler], and [http://cocoon.apache.org/2.1/userdocs/concepts/profiler.html. However, Java-level debugging can be useful for tracing Cocoon-based applications as well.

On the Cocoon users mailing list *Leszek Gawron* wrote the original step by step guide to debug Cocoon using Eclipse or Swat. Now here's the wikified version.

Debugging Cocoon requires two steps:

1. Run Cocoon (in a servlet container) in a Java virtual machine (JVM) that has remote debugging enabled.
2. Tell your Java debugger/IDE, such as JSwat or Eclipse, to connect to the JVM process.

1. Run Cocoon with remote debugging enabled

The easiest way to run Cocoon set up for remote debugging is to run
`cocoon.bat servlet-debug`
or
`cocoon.sh servlet-debug`

Note: If you prefer to use tomcat for remote debugging, run: `$TOMCAT_HOME/bin/catalina.sh jpda start`

The **default debug port** set in the `cocoon.bat` or `cocoon.sh` is port **`8000`**.

The simplest way to test that your system is set up for remote debugging is to use jdb as described in step 2.

2. Connect your debugger/IDE to Cocoon JVM

jdb

jdb is not generally recommended as a debugging tool because its user interface is very primitive. However, you can use it as a simple test case to make sure that your Cocoon JVM is set up to accept remote debugging connections. The jdb executable is typically found in the same directory in your path as the java executable. Type

```
jdb -attach localhost:8000
```

(for un*x/linux) or

```
jdb -connect com.sun.jdi.SocketAttach:hostname=localhost,port=8000
```

(for Windows). A successful connection is indicated by

```
Set uncaught java.lang.Throwable
Set deferred uncaught java.lang.Throwable
Initializing jdb ...
```

The error `java.io.IOException: shmemBase_attach failed: The system cannot find the file specified` will occur if you are on Windows and try to use `jdb -attach localhost:8000`; this will try to use shmem by default instead of sockets. The error `Connection refused: connect` on Windows may indicate that the Windows Firewall is blocking the connection (it may not notify you of the blockage, even if configured to do so). Go to Control Panel / Windows Firewall / Exceptions / Add Port, and set Name = remote debugging, Port = 8000, TCP.

JSwat 2.3

- Attach to Cocoon VM.
- Set source path to:
  - {cocoon-cvs-home}\src\java
  - {your-win-profile}\Local Settings\Temp\Jetty_8888_\cocoon-files (2 underscores before and after the port).
- If you want to debug Cocoon core:
  - Set breakpoint providing:
    - class name,
    - line number/method name.
- If you want to debug XSP:
  - Run it at least once to have `.java` file generated.
  - Browse class hierarchy (there is a special view for that in jSwat), select the appropriate class and place a breakpoint.
- Hit the page you want to debug in a browser - the execution will stop at the breakpoint.
- Now you can run your code by steps. Source browser will show you the current position.

Everything works quite fine but jSwat functionality is not enough for us. So lets switch to:

Eclipse

- LoadInEclipse. (You do not need to use SVN / Subclipse unless you want to stay on the bleeding edge of Cocoon internals, e.g. if you are a Cocoon committer.)
- Prepare new remote debug configuration (in your current project, so Eclipse will be able to resolve sources).
    - Run / Debug... / Remote Java Application / New
    - Fill in Name (of debug configuration); Host = localhost, Port = 8000. You may want to uncheck "Launch in background" on Common tab.
    - Click "Apply".
- Attach to Cocoon VM
    - On the Run / Debug... dialog, make sure your debug configuration is selected and click the Debug button.
    - If you get "Connection refused", check your Windows Firewall (see above under **jdb**).
- Browse your project for appropriate class, open the source file and set a breakpoint in a place you want to.
- Hit the page - wait for breakpoint to be triggered.
- Eclipse will open a source file and from now on you can step through the code.

This procedure works in IntelliJ IDEA as well. Verified Nov 2004 by MicahDubinko

For XSP debugging you need additional steps:

- Edit

  ```
  {cocoon-cvs-home}\build\webapp\WEB-INF\web.xml
  ```

  to change the location of Cocoon temporary files. All you have to do is to uncomment this section:

```
<init-param>
    <param-name>work-directory</param-name>
    <param-value>WEB-INF/work</param-value>
</init-param>
```

You have to have all source files to be placed relatively from your project home.

This change needs a servlet container restart.

- Add additional source path in Eclipse: {{ {cocoon-cvs-home}\build\webapp\WEB-INF\work}}
- Everything would be fine but Eclipse does not want to work with XSP generated java files properly. When first run the `.java` file is being generated from XSP, so from now on it is visible from Eclipse IDE. You can compile it (if you add all Cocoon libraries to your project), you can also set a breakpoint. The execution will be stopped at breakpoint, but you will get a message that Eclipse cannot find source for this class. I've tried any combination and still Eclipse is not aware of the source. It works for any other classes though (not generated).