Profiling

Profiling software looks for bottlenecks in program execution. In addition to the profiling services provided by IDEs and standalone profilers, the framework provides its own internal support for profiling.

Profiling aspects

{snippet:id=profilingAspect_struts2|javadoc=true|url=com.opensymphony.xwork2.util.profiling.UtilTimerStack}

Using

To enable profiling, first make sure that the `profiling` interceptor is applied to your action, like:

xml <action ... > ... <interceptor-ref name="profiling"> <param name="profilingKey">profiling</param> </interceptor-ref> ... </action>

Then enable profiling using one of the following methods:

Activate Through System property

{snippet:id=activationThroughSystemProperty|lang=xml|javadoc=true|url=com.opensymphony.xwork2.util.profiling.UtilTimerStack} {snippet: id=activationThroughSystemPropertyDescription|javadoc=true|url=com.opensymphony.xwork2.util.profiling.UtilTimerStack}

Activate Through code

{snippet:id=activationThroughCode|lang=xml|javadoc=true|url=com.opensymphony.xwork2.util.profiling.UtilTimerStack} {snippet: id=activationThroughCodeDescription|javadoc=true|url=com.opensymphony.xwork2.util.profiling.UtilTimerStack}

Activate Through parameter

http://host:port/context/namespace/someAction.action?profiling=true

Changing the activation parameter name

Set the `profilingKey` attribute of the `profiling` interceptor to the desired name:

xml <action ... > ... <interceptor-ref name="profiling"> <param name="profilingKey">profiling</param> </interceptor-ref> ... </action>
Profiling activation through a parameter requires `struts.devMode` to be `true`.

Filtering profile information

One could filter out the profile logging by having a System property as follows:

-Dxwork.profile.mintime=10000

With this `xwork.profile.mintime` property, one could only log profile information when its execution time exceed those specified in `xwork.profile.mintime` system property. If no such property is specified, it will be assumed to be 0, hence all profile information will be logged.

Write profiling code

One could extend the profiling feature provided by Struts2 in their web application as well.

Using UtilTimerStack's push and pop

java String logMessage = "Log message"; UtilTimerStack.push(logMessage); try { // do some code } finally { UtilTimerStack.pop(logMessage); // this needs to be the same text as above }

Using a UtilTimerStack's ProfileBlock template

java String result = UtilTimerStack.profile("purchaseItem: ", new UtilTimerStack.ProfilingBlock<String>() { public String doProfiling() { // do some code return "Ok"; } });

Profiling Log files

Profiled result is logged using commons-logging under the logger named `com.opensymphony.xwork2.util.profiling.UtilTimerStack`. Depending on the underlying logging implementation, say if it is Log4j, one could direct the log to appear in a different file, being emailed to someone or have it stored in the db.

Next: Debugging