TPTP Java Profiler

Mario J Lorenzo (mjlorenz@us.ibm.com), Software Engineer, IBM

This article will demonstrate the Eclipse Test & Performance Tools Platform. This tutorial will include the installation, setup, and usage of TPTP including instructions on remotely profiling an application running in a production or test environment. This tutorial will also provide tips on how to interpret results for various metrics collected.

An Introduction to Eclipse TPTP

The Eclipse TPTP tool is an excellent method to analyzing an application's memory consumption and performance statistics. Although this tool offers many other interesting testing functions, this article will only focus on the profiling tools.

To get started:

(For non-eclipse users)

1. Point your browser to http://eclipse.org/tptp/home/downloads/
2. Click on the 'TPTP all-in-one package'
3. Choose a download mirror and download
4. Extract the zip file
5. Run the 'eclipse.exe'
6. Select the default workspace or create your own.

(For eclipse users)

1. Within your Eclipse workbench file menu, select Software Updates->Find and Install.
2. Choose 'Search for new features to install' option
3. Add a 'Remote site' with the following url: 'http://eclipse.org/tptp/updates/'
4. Click 'finish' and follow the wizard to download the TPTP features.

After installing and running the eclipse workbench you will see a new item on the toolbar:

Playing with an Example

For a simple example to test out your new profiling tool download the following example eclipse project and import into your workbench (File->Import->Existing Eclipse project)

http://www.eclipse.org/articles/Article-TPTP-Profiling-Tool/productCatalogSample.zip

and the corresponding resource files:

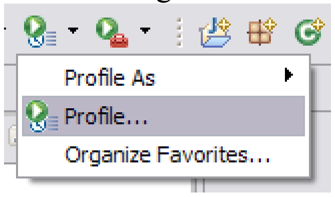
http://www.eclipse.org/articles/Article-TPTP-Profiling-Tool/xmlProductFiles.zip[1]

After installing the sample project and the resource files, run the project as a 'Java Application'. You should see the console view display information about various products. These displayed products are the result of the sample application parsing many XML files and outputting their content.

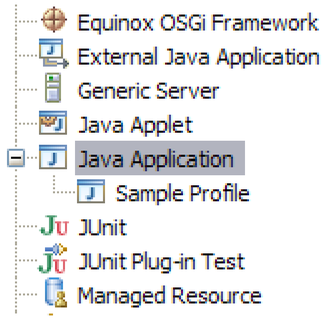
Configuring the Java Profiler

[1] *Java Application Profiling using TPTP* (Valentina Popescu, eclipse.org, February 2006): http://www.eclipse.org/articles/Article-TPTP-Profiling-Tool/tptpProfilingArticle.html

Before profiling this Sample Project you must first configure the Java Profiler. By clicking on the 'down-arrow' on the newly added profile item on the toolbar and selecting 'Profile …'



Once the dialog appears, double-click on 'Java Application' from the list:

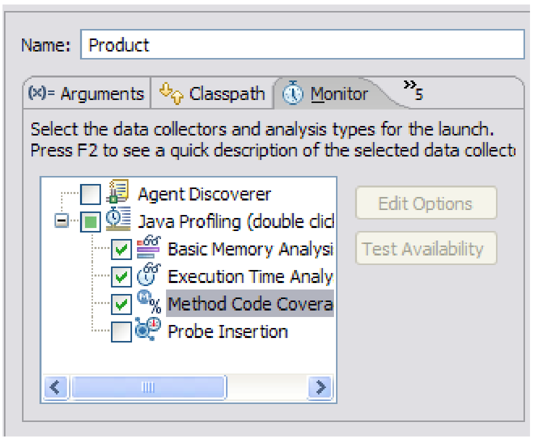


After clicking on 'Java Application' a new item will appear and some configuration tabs will appear on the right-side of the dialog.

First give this new profile a name (the text field atop of the tabs). You will notice that in the 'Main' tab the 'Project' field already selected a Java Project from your Workspace. If this is not the Sample project, then click on 'Browse' and select the appropriate project.

Once you select a Java Project the 'Main class' field will be populated with the main class for the project. If you are working with the sample project this should read: "com.sample.product.Product"

There are many tabs that allow you to configure runtime information. These tabs are standard when running any Java Application and will not be covered in this tutorial. Fortunately you don't need to configure any of those tabs. The entire Profiler configuration can be manipulated through the 'Monitor' tab. This tab allows you to configure all sorts of information regarding the Java Profiler.


Name: Product

Select the data collectors and analysis types for the launch.
Press F2 to see a quick description of the selected data collecti

Here you will be to control the following:

- Profiling Filters
- Basic memory Analysis
- Execution Time Analysis
- Method Code Coverage

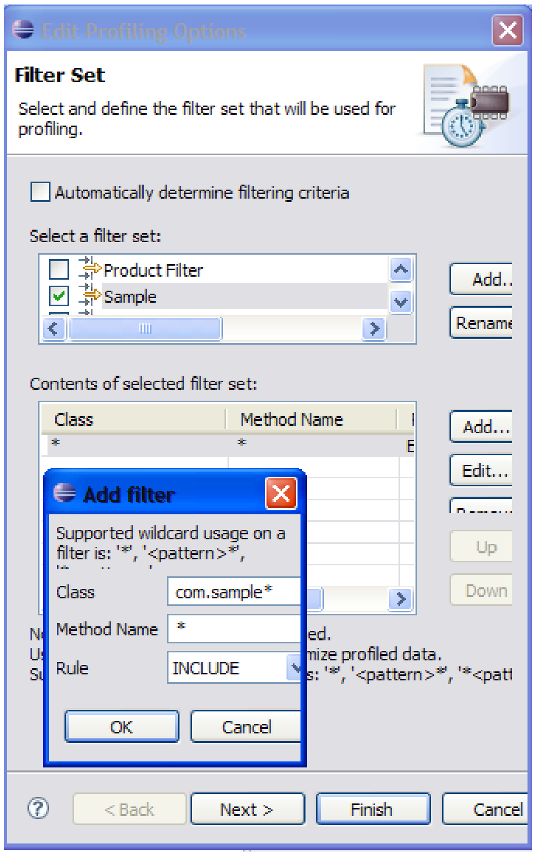
Creating a Profiling Filter

Specifying a filter is important. A filter allows you to exclude or include metrics for any given java packages. Since a simple Java Runtime Environment can contain thousands of classes, it is beneficial to suppress some of this data that only obscure your object. With a well defined filter you will be able to easily identify

the classes that you are interested in analyzing.
Start by double-clicking on the 'Java Profiling' root tree item. A new dialog will appear with several pre-defined filters. Click on 'add' and specify a name for your new filter.
Now you are able to add filter rules. For this sample (and for many of your profiling activities to come) it is best to create two rules:

- Include your packages only
- Exclude all other packages

The syntax for specifying a rule is simply: '*yourpackage*[*]'
The asterisk specifies that you wish to include any and all classes that include this pattern.



Edit Profiling Options

Filter Set

Select and define the filter set that will be used for profiling.

[ ] Automatically determine filtering criteria

Select a filter set:

[ ] Product Filter
[✓] Sample

Contents of selected filter set:

Class	Method Name	
*	*	E

Add filter

Supported wildcard usage on a filter is: '*', '<pattern>*',

Class: com.sample*
Method Name: *
Rule: INCLUDE

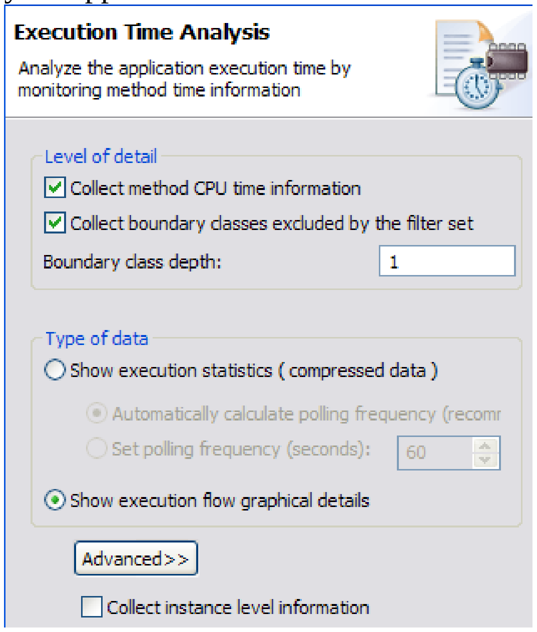
For the Exclude rule just specify a '*' for both class and method.

Basic Memory Analysis

You will notice an item called 'Basic Memory Analysis' within the Monitor tab. If selected, this will provide metrics involving the memory consumption for each class that you are filtering for. Select this option and then click on 'Edit Options'. You will see an option to collect instance level information. **Note:** It is not recommended to collect instance level data unless you have a very restricting filter. Otherwise the profiling process will not be very response and can lock up your workbench.

Execution Time Analysis

Execution time analysis is the most important option to use when profiling an application. This feature provides useful information about performance of the application and various options for metrics that will whet your appetite.



Execution Time Analysis

Analyze the application execution time by monitoring method time information

Level of detail

[✓] Collect method CPU time information
[✓] Collect boundary classes excluded by the filter set
Boundary class depth: 1

Type of data

( ) Show execution statistics (compressed data)
(•) Automatically calculate polling frequency (recomr
( ) Set polling frequency (seconds): 60
(•) Show execution flow graphical details

[ ] Collect instance level information

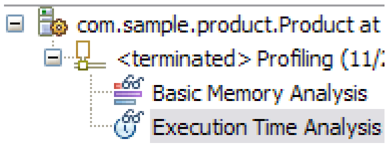
Method CPU Time
Execution Time Analysis dialog offers various options, the first of which is related to method CPU time. If selected, this option will provide cpu time metrics for each method call that matches your filter.
The time information includes base time, average time and cumulative time.
Base time calculates the cpu time for the method excluding any calls made to other methods. So the base time reflects the time actually spent in the method.
Average time of a method is the average base time of the method for all calls to the method. So the avg gives a representation of the base time over the number of calls made to the method.
Cumulative time represents the total time taken by the method including any subsequent calls to other methods.

Boundary Depth
The boundary depth represents how many method calls, starting at some method that meets your filter, should be displayed for methods that don't meet your filter. In essence, how many method calls would you like to be included in the reports for a stack trace of method calls that fall outside your filter?
Including boundary information is useful to provide context to your method calls. It allows you to see what surrounding methods (and their metrics) were called for each of your filter-matched methods. Generally you should set the depth from 1 to 3. Keep in mind that this is an exponential growth in information that must be collected therefore it is best to keep it small.

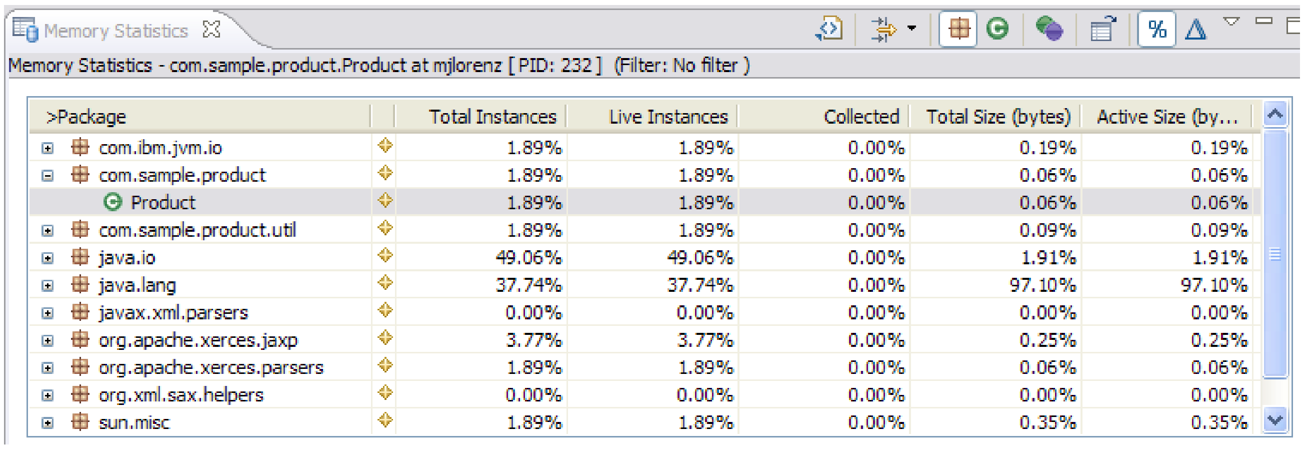
Execution Report Type
There are two options to consider when selecting the format of the execution statistics in your reports.
The first format is tabular data. This means that all the data will be in tables that you can sort, filter, and examine. These include views that allow you to sort by base, average, cum. time. It also allows you to sort by package, class, or method. This will be covered later on, but for now it is important to decide what format you prefer.
The next format provides visual models and representation of the execution flow of the application you are profiling.
This option is called graphical details. It provides UML sequence diagrams that model the dynamic (or interactive) behavior of the application. It also provides great way to view threads and to drill in and find which method calls within a given thread is taking up execution time. This is my preferred option to look at the data. However keep in mind that the responsiveness of this graphical format is slower, and if you are not being exclusive in your filtering, this may not be useful to profile huge applications during real-time.

Method Code Coverage
As you would expect, this option provides details about method coverage of your application. It is a good way to see what percentages of the methods are being called as well as the count. This helps to determine the main execution flow of an application especially when profiling an unfamiliar application. I tend not to collect this information since I am mostly interested in execution time.

Profiling the Example
Once you have selected the options, click on 'Profile' to begin profiling.
The workbench should automatically switch you to an Eclipse Profiling and Logging perspective.
On the navigator view (left side) you will see the application that you are profiling with a report (with timestamp) for each profile instance.


com.sample.product.Product at
<terminated> Profiling (11/
Basic Memory Analysis
Execution Time Analysis

Double-click the 'Basic memory Analysis' report. You should see a tab view called 'Memory Statistics' displayed with all the memory metrics for this application.



Memory Statistics - com.sample.product.Product at mjlorenz [PID: 232] (Filter: No filter)

>Package		Total Instances	Live Instances	Collected	Total Size (bytes)	Active Size (by...
com.ibm.jvm.io		1.89%	1.89%	0.00%	0.19%	0.19%
com.sample.product		1.89%	1.89%	0.00%	0.06%	0.06%
Product		1.89%	1.89%	0.00%	0.06%	0.06%
com.sample.product.util		1.89%	1.89%	0.00%	0.09%	0.09%
java.io		49.06%	49.06%	0.00%	1.91%	1.91%
java.lang		37.74%	37.74%	0.00%	97.10%	97.10%
javax.xml.parsers		0.00%	0.00%	0.00%	0.00%	0.00%
org.apache.xerces.jaxp		3.77%	3.77%	0.00%	0.25%	0.25%
org.apache.xerces.parsers		1.89%	1.89%	0.00%	0.06%	0.06%
org.xml.sax.helpers		0.00%	0.00%	0.00%	0.00%	0.00%
sun.misc		1.89%	1.89%	0.00%	0.35%	0.35%