# NetBeans Profiler

## Issues of Traditional Application Profiling Schemes

- You might have to add ad-hoc profiling code to the application and rebuild the application
  - > This is not practical if you have only binary bits
- You cannot pick and choose the area of profiling
  - > You have to profile the whole application and underlying system code
  - > Profiling app server code is not something you care about
  - > The interpretation of the collected data is not easy
- Profiling adds significant overhead
  - > Profiling data is tainted by the profiling itself



## NetBeans Application Profiling

- Provides runtime behavior of your application
  - > Heap memory size, GC statistics, thread count, thread state
  - > CPU time used by application methods
- Selectively choose specific part of your app for profiling
  - > No “profiler tainted performance data” problem
  - > Filter out any system level code for profiling
- No need to rebuild the application with ad-hoc profiling code fragments
  - > You can profile running application



## Underlying Technology Used: Dynamic Bytecode Instrumentation

- The insertion of instrumentation bytecodes is done at runtime
- After you have selected a class or method for profiling, the next time it is invoked by the application, the profiler will insert the necessary instrumentation
  - > Using the JVMTI's redefineClass(Class[] classes, byte[][] newBC) method.

## NetBeans Profiling Techniques

- You can change the profiling overhead as the application is running
  - > You can even remove all instrumentation so that your application is no longer being profiled
  - > These changes can be made adaptively as you learn more about the behavior of the application
- The changes can be made without restarting your application



## NetBeans Profiler Features

- Low overhead profiling
- Attaching to running applications
- CPU Performance profiling
- Memory profiling
- Memory leak debugging
- Task-based profiling
- Threads profiling
- Processing collected data offline



## Lab1: Monitor Thread State

- Diagnose performance problem of a Swing application by monitoring thread state
  - > Multiple threads are used by Swing application
  - > AWT-EventQueue-0 thread is the Event Dispatch Thread (EDT) used by Swing to process window events. In a well behaved Swing application, the EDT spends most of its time waiting and very little time running



## Lab2: Monitor a Method

- Diagnose performance problem of a Web application by selectively monitoring CPU time used by a method
  - > Monitor only processRequest(..) method, for example
  - > Filter out system code (even if they are called from the method)

## Lab3: Find Memory Leack

- Find memory leaks by monitoring object creation
  - By “memory leak”, I am not talking about JVM level memory leak but objects that are not garbage collected due to a design problem in your code
  - ex) allocating an objects pair and save them in the HashMap

```
while (!stop) {
  try {
      map.put (new float[1], new double [1]);
      Thread.sleep (100);
      System.gc ();
  } catch (InterruptedException e) {
      return;
  }
}
```



## Resources:

http://www.javapassion.com/netbeans/masterindex.html#Profiler