In-situ Visualization of Profiling Data

Sebastian Baltes
University of Trier, Germany
@s_baltes
research@sbaltes.com

Oliver Moseler
University of Trier, Germany
olivermoseler@gmx.de
Visual Performance Analysis Tools

- **Profiling tools** record program runs and assign measured performance values to code entities (e.g. runtime or memory consumption)
- We focus on **runtime consumption** and **Java** programs
- Standard user interface: **Lists**

![Profiler Tool Images]

- **VisualVM**
- **YourKit**

*View and compare profiling results*
Our Tool

In-Situ Visualization of Performance Data

- Run time
- Method time
- Callees
- Callers
- Self time
- Color scale: 0% - 100%

```java
private class Values extends AbstractCollection {
    public Iterator iterator() {
        final IteratorChain chain = new IteratorChain();
    }
}
```
In-Situ Visualization of Performance Data

Navigation

```java
package performanceTests;
import ...

public class PerformanceTest {
    public static void main(String[] args) {
        int size = 20000; // Number of elements to store in the multi value map
        MultiValueMap multi = new MultiValueMap();
        for (int i = 0; i < size; i++) // Insert values
            multi.put(1, i); // 0.2% ...

        List<Integer> toContains = new ArrayList<>(); // A list of elements to check
        for (int i = size - 1; i > -1; i--)
            toContains.add(i); // 8.02% ...

        // Get all values of the multi value map
        Collections<Collection> values = multi.values();
        Collections.addAll(values, toContains);

        long start = System.currentTimeMillis(); // Start time measuring
        // Call containsAll on the values
        values.containsAll(toContains); // 99.73%
        long stop = System.currentTimeMillis(); // 99.73% ...
        System.out.println("Time is " + (stop - start) + " ms"); // Print elapsed time
```

Universität Trier
Software Engineering Group
Stack Sampling

• **Profiler:**
  • Analysis tool
  • Measure runtime consumption or memory usage of a program
  • Identify performance bugs
  • Optimize programs

• **Sampling approach:**
  • Heuristic methodology
  • Estimate runtime consumption
  • Stop target program periodically
  • Record a sample of the current state of the stack traces from all threads
  • Target program runs slower
Stack Sampling

• Post mortem analysis:
  • Method time: Method was found within a stack
  • Self time: Method was found on top of a stack
  • Caller and callee runtime: Time spent in called methods

• The approach doesn’t track every single stack trace:
  • Results can vary
  • Run multiple samplings to get more reliable propositions
More Information

debugging.sbaltes.com

Sebastian Baltes
University of Trier, Germany

@s_baltes
research@sbaltes.com

Oliver Moseler
University of Trier, Germany

olivermoseler@gmx.de