Debugging and Profiling J2EE™ /Java™ EE 5 Platform-Based Applications

Ludovic Champenois, Nazrul Islam, Tomáš Hůrka
Sun Microsystems
Project GlassFish, NetBeans

TS-1549

Copyright © 2006, Sun Microsystems Inc., All rights reserved.

Debugging and Profiling Java EE

Understand the set of tools and features that can help you with debugging, monitoring and profiling a Java EE 5 Application.

Learn how NetBeans™ and Java EE 5 Project GlassFish^{SM} can help you.

Agenda

Debugging and Verifying Java EE 5 Apps

Monitoring

Profiling

Agenda

Debugging and Verifying Java EE 5 Apps

Monitoring

Profiling

Debugging and Verifying Java EE 5 Apps

“The Focus of Java EE 5 Is Ease-Of-Development”

- NetBeans 5.5
  - Java EE 5 annotations support
  - Java EE 5 Wizards
  - Project GlassFish, JBoss support
  - Inline verifier, via the Java editor hints
    - JSR 220: The Java editor is the JPA ORM tool
      - Ex: “Missing Entity ID”
    - JAX-WS2.0 hints:
      - Ex: “the serviceName attribute not allowed on interfaces”

Debugging and Verifying Java EE 5 Apps

- NetBeans
    - Static verifier from NetBeans
        - NetBeans uses the EE SDK verifier tool
    - Dynamic verifier from NetBeans (AVK)
        - Optional module, for runtime verification

Debugging

- NetBeans Debugger
  - One click debug session for all project types
    - J2EE/Java EE Apps debugging
  - Call stack filtering
    - Hidden source calls
  - JSP source level debugging
  - Variables evaluation
  - Conditional breakpoints
  - Watches

Debugging

- HTTP Monitor Tool
    - Gather data about HTTP requests
        - Incoming request, Cookies, Sessions, Context, Headers
    - View data, store data, replay and edit previous requests
    - Help isolate problems with data flow from JSP and servlet execution on a Web server

DEMO

Debugging Java EE 5 Apps with NetBeans

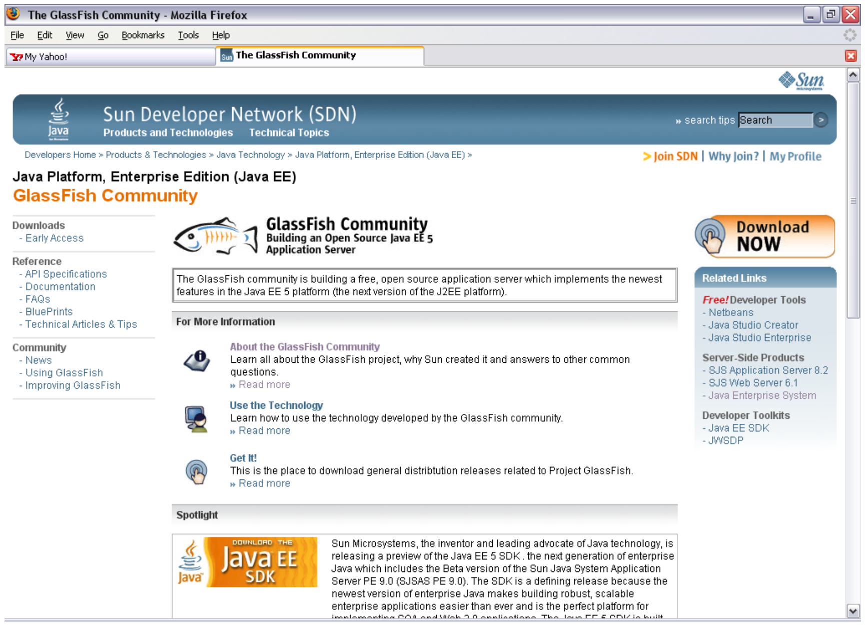
Agenda

Debugging and Verifying Java EE 5 Apps

Monitoring

Profiling

Project GlassFish


The GlassFish Community - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
My Yahoo!
The GlassFish Community
Sun Developer Network (SDN)
Products and Technologies
Technical Topics
» search tips
Search
Developers Home > Products & Technologies > Java Technology > Java Platform, Enterprise Edition (Java EE) >
> Join SDN | Why Join? | My Profile
Download NOW

Java Platform, Enterprise Edition (Java EE)
GlassFish Community

Downloads
- Early Access

Reference
- API Specifications
- Documentation
- FAQs
- BluePrints
- Technical Articles & Tips

Community
- News
- Using GlassFish
- Improving GlassFish

GlassFish Community
Building an Open Source Java EE 5 Application Server

The GlassFish community is building a free, open source application server which implements the newest features in the Java EE 5 platform (the next version of the J2EE platform).

For More Information

About the GlassFish Community
Learn all about the GlassFish project, why Sun created it and answers to other common questions.
» Read more

Use the Technology
Learn how to use the technology developed by the GlassFish community.
» Read more

Get It!
This is the place to download general distribtution releases related to Project GlassFish.
» Read more

Spotlight

DOWNLOAD THE Java EE SDK

Sun Microsystems, the inventor and leading advocate of Java technology, is releasing a preview of the Java EE 5 SDK . the next generation of enterprise Java which includes the Beta version of the Sun Java System Application Server PE 9.0 (SJSAS PE 9.0). The SDK is a defining release because the newest version of enterprise Java makes building robust, scalable enterprise applications easier than ever and is the perfect platform for

Related Links

Free! Developer Tools
- Netbeans
- Java Studio Creator
- Java Studio Enterprise

Server-Side Products
- SJS Application Server 8.2
- SJS Web Server 6.1
- Java Enterprise System

Developer Toolkits
- Java EE SDK
- JWSDP

Building a Java EE 5 open source application server

java.sun.com/javaee/GlassFish

Simplifying Java application development with **Java EE 5 technologies**

Includes JWSDP, EJB 3.0, JSF 1.2, JAX-WS and JAX-B 2.0

Supports > **20** frameworks and apps

Open source CDDL license

Basis for the **Sun Java System Application Server PE 9**

Free to download and **free** to deploy

Over **1200** members & **200,000** downloads

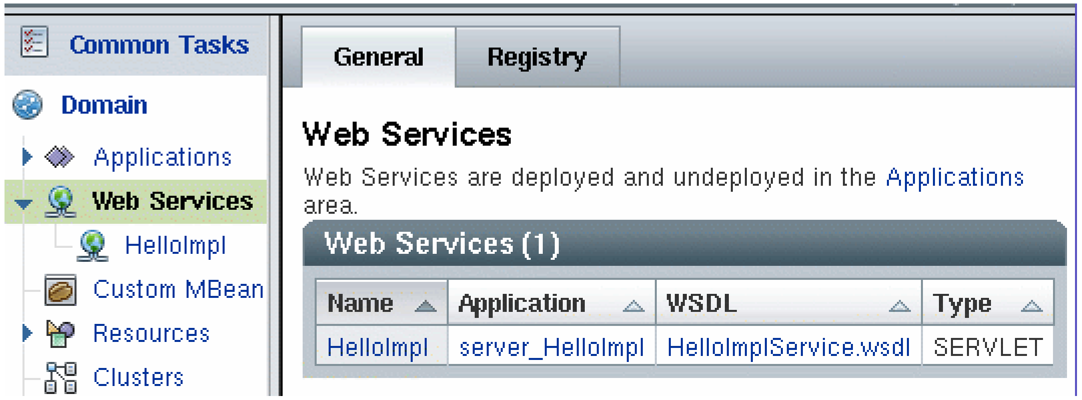
Web Services as Fist-Class Objects

Listing of Web Services

- Auto-discovery of Web Services
  - No need to browse through deployment descriptors

asadmin list-components --type webservice

Server_HelloImpl#HelloImpl <webservice>



Web Services

Web Services are deployed and undeployed in the Applications area.

Web Services (1)

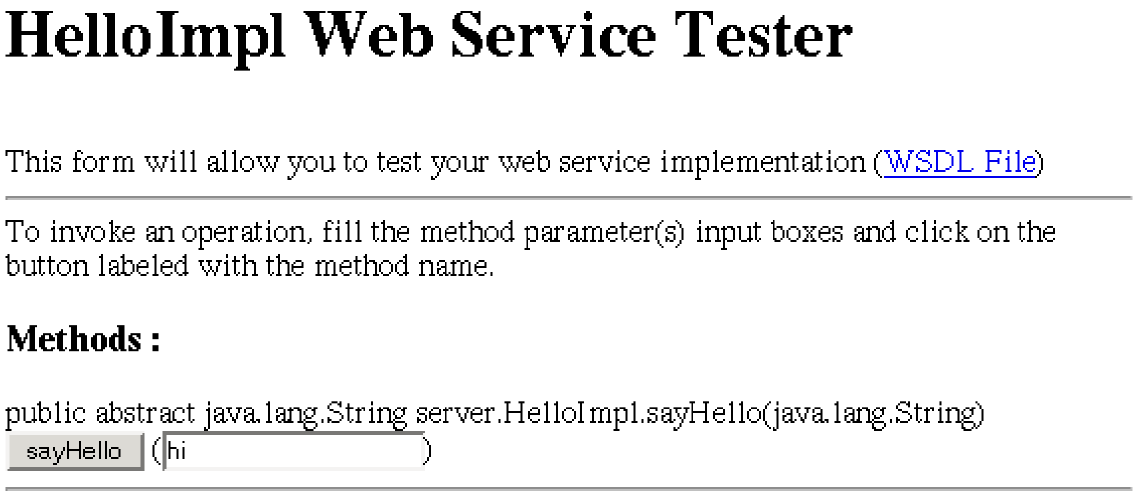
Name	Application	WSDL	Type
HelloImpl	server_HelloImpl	HelloImplService.wsdl	SERVLET

Web Service Client

Auto Generated Test Forms

- Auto generated Test Forms—ping
  - Shows operations and parameter values
  - Supports JAX-WS standard

```
package server;
import javax.jws.WebService;
@WebService
public class HelloImpl {
  public String sayHello(String name) {
        return "Hello, " + name + "!";
  }
}
```



HelloImpl Web Service Tester

This form will allow you to test your web service implementation (WSDL File)

To invoke an operation, fill the method parameter(s) input boxes and click on the button labeled with the method name.

Methods :

public abstract java.lang.String server.HelloImpl.sayHello(java.lang.String)

sayHello (hi)

Monitoring

Statistics

- Operational Statistics
  - Number of requests
  - Throughput
  - Response time (average, min, max) per web service
  - Number of SOAP faults
- Three levels
  - OFF—No data is collected (default)
  - LOW—Operational statistics are collected
  - HIGH—Messages trace is also collected

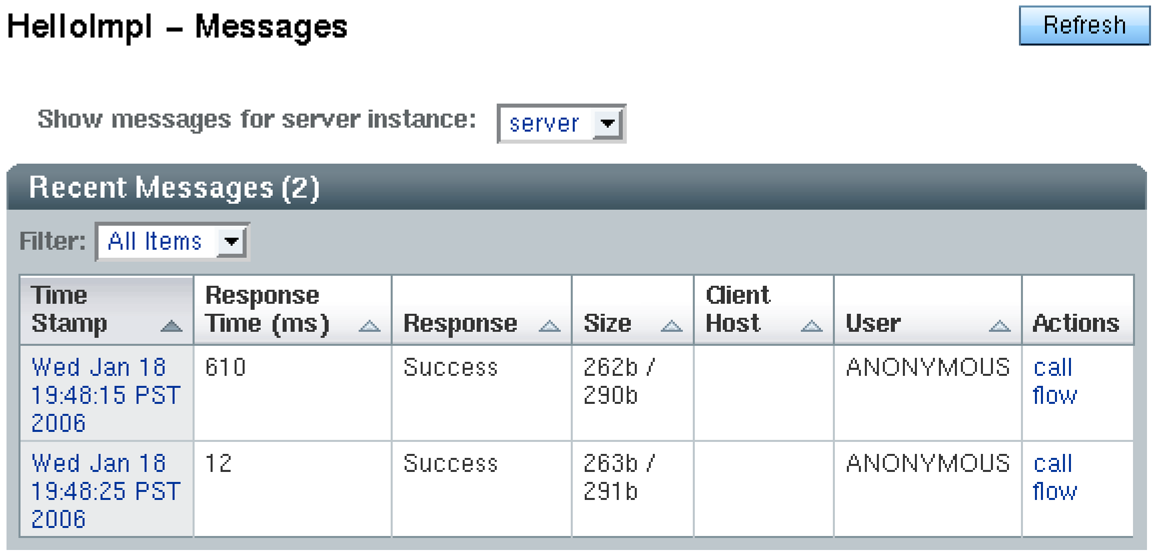
Content Visualization

Message Trace

- SOAP message details are shown
  - SOAP request, response
  - HTTP headers
  - Response time
  - Size of request, response
  - SOAP fault
  - Client IP address and User principle
- Configuration of the number of messages kept in memory, (25 by default)

Monitoring and Content Visualization

Message Trace



HelloImpl – Messages

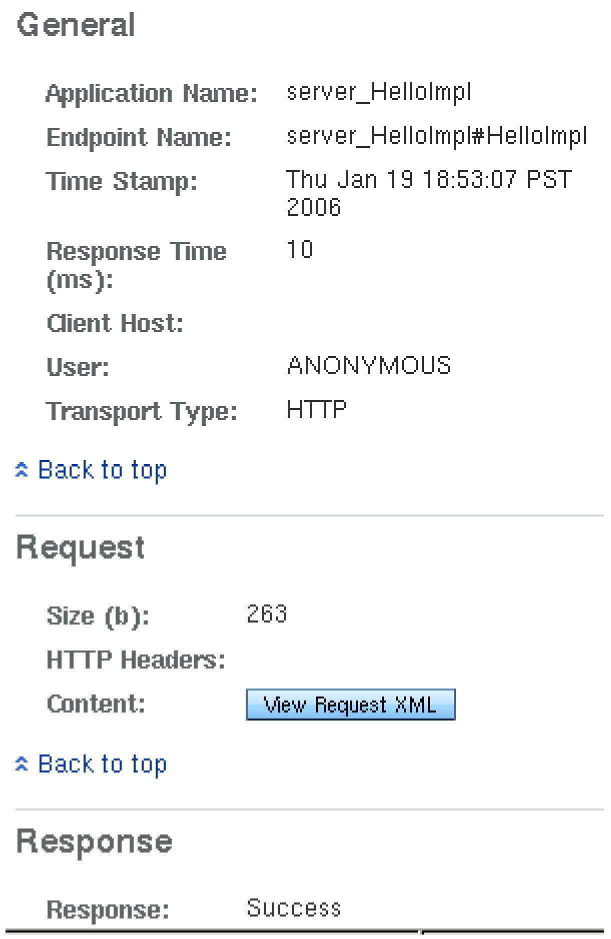
Refresh

Show messages for server instance: server

Recent Messages (2)

Filter: All Items

Time Stamp	Response Time (ms)	Response	Size	Client Host	User	Actions
Wed Jan 18 19:48:15 PST 2006	610	Success	262b / 290b		ANONYMOUS	call flow
Wed Jan 18 19:48:25 PST 2006	12	Success	263b / 291b		ANONYMOUS	call flow



General

Application Name:	server_HelloImpl
Endpoint Name:	server_HelloImpl#HelloImpl
Time Stamp:	Thu Jan 19 18:53:07 PST 2006
Response Time (ms):	10
Client Host:	
User:	ANONYMOUS
Transport Type:	HTTP

Back to top

Request

Size (b):	263
HTTP Headers:	
Content:	View Request XML

Back to top

Response

Response:	Success

Call Flow/Root Cause Analysis

What Is Call Flow?

- Monitor request processing activity in containers (Web, EJB™, JDBC™, ORB) and collect data
- Analyze the collected data and project call flow graphs and statistical information
- Two Phases:
    - Data Collection
    - Data Mining

Call Flow/Root Cause Analysis

Call Flow Phases

- Data Collection
  - Gather Application Name, Component Name, Method Name, Transaction Id, Security Principal, Caller IP Address, and more
- Data Mining
  - Project call flow and statistical information in the form of graphs and pie charts

Call Flow/Root Cause Analysis

How to Use Call Flow?

- Admin GUI provides the following support:
  - Turn ON/OFF Call Flow
  - Located in the Monitor tab area
  - Analyze the Call Flow data using flow graphs (tree view), pie charts
- Admin CLI provides support to turn ON/OFF Call Flow
  - $asadmin start-callflow-monitoring
  - $asadmin stop-callflow-monitoring

Call Flow/Root Cause Analysis

Benefits of Call Flow

- Identify performance bottlenecks
  - Reveals if a request is spending most of its time in a particular container or the backend database
- Measure performance in production environment
  - Using filters, you can inject a request into a running system as a die and watch that flow through the system to evaluate the performance

DEMO

Call Flow and Web Services Monitoring

Agenda

Debugging and Verifying Java EE 5 Apps

Monitoring

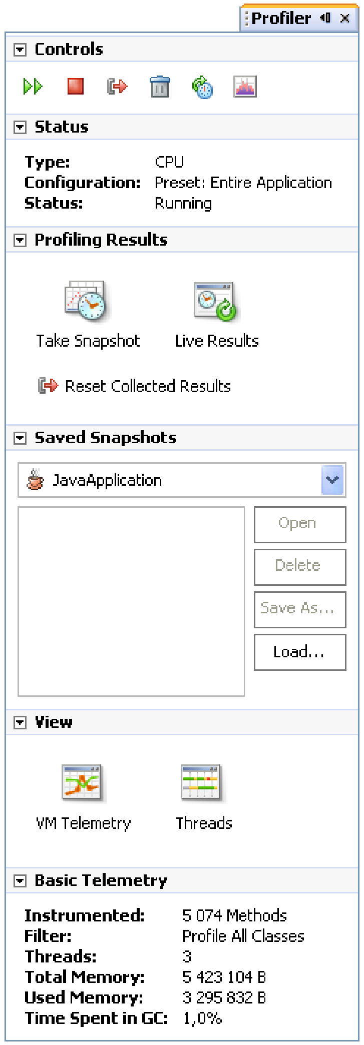
Profiling

Profiler: Agenda

- History
  - 2001-2004 JFluid, SunLabs
  - 08/2004 Profiler Milestone 1, NetBeans 3.6
  - 01/2006 FCS along with NetBeans 5.0
- NetBeans Profiler
- Demo

Profiler: Definitions

- Full featured Java profiler
  - Application monitoring (threads, VM telemetry)
  - Performance profiling (CPU)
  - Memory profiling, memory leaks detection
- Advanced features
  - Ability to fine tune various profiling settings
  - Profiling "big" applications
- Easy to use for beginners
  - Benefit from IDE integration
  - Predefined profiling tasks



Type:	CPU
Configuration:	Preset: Entire Application
Status:	Running

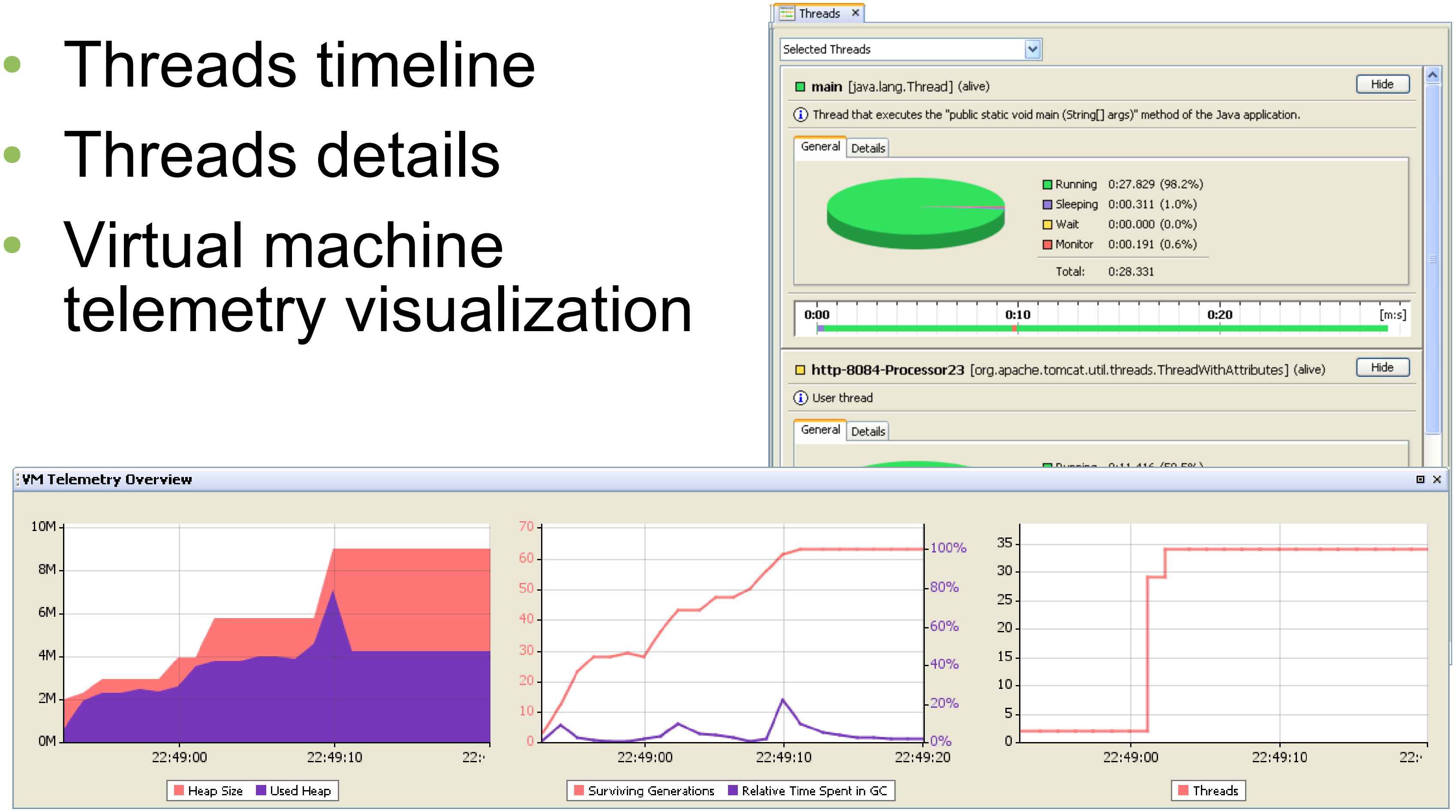
Instrumented:	5 074 Methods
Filter:	Profile All Classes
Threads:	3
Total Memory:	5 423 104 B
Used Memory:	3 295 832 B
Time Spent in GC:	1,0%

Profiling Methods

- General (commonly used) methods:
  - Sampling
  - Instrumentation
- NetBeans Profiler:
  - Dynamic bytecode instrumentation
  - Hybrid sampling/instrumentation approach
  - Instrumentation schemes

Application Monitoring

- Threads timeline
- Threads details
- Virtual machine telemetry visualization



main [java.lang.Thread] (alive)

Thread that executes the "public static void main (String[] args)" method of the Java application.

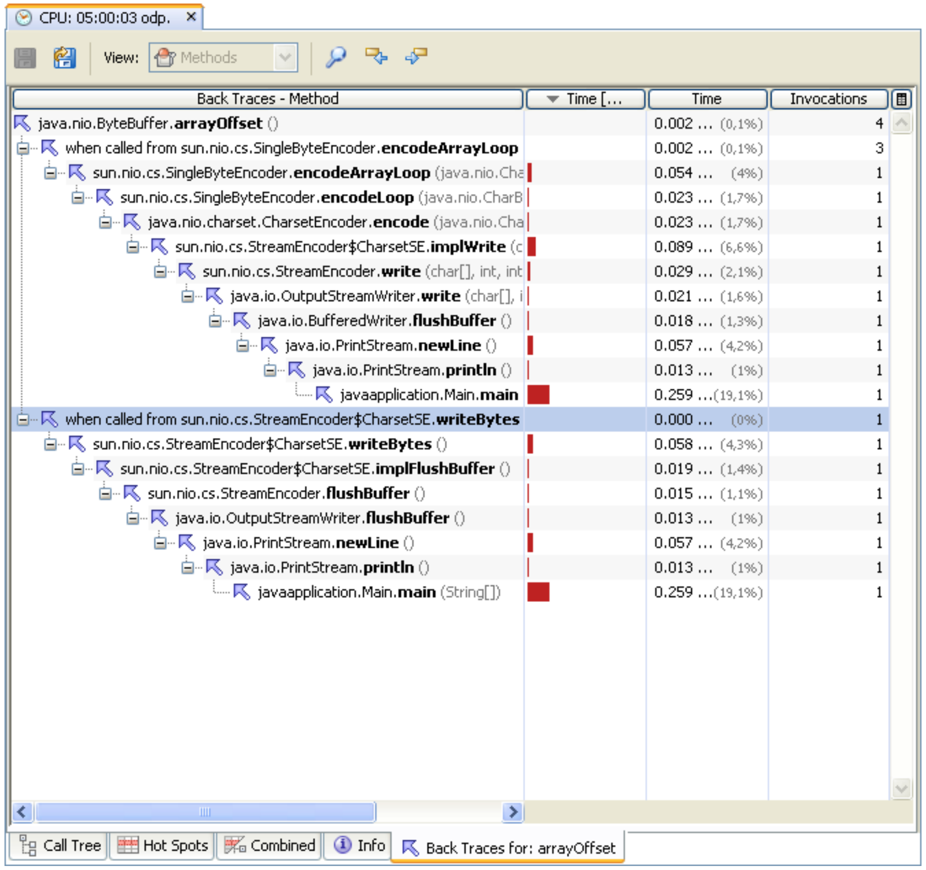
🟩 Running	0:27.829	(98.2%)
🟪 Sleeping	0:00.311	(1.0%)
🟨 Wait	0:00.000	(0.0%)
🟥 Monitor	0:00.191	(0.6%)
Total:	0:28.331	

http-8084-Processor23 [org.apache.tomcat.util.threads.ThreadWithAttributes] (alive)

User thread

CPU Profiling

- Entire application
- Part of application
- Instrumentation filter
- Instr./sampling
- Instr. schemes
- Special methods
- CPU snapshots
- Back traces

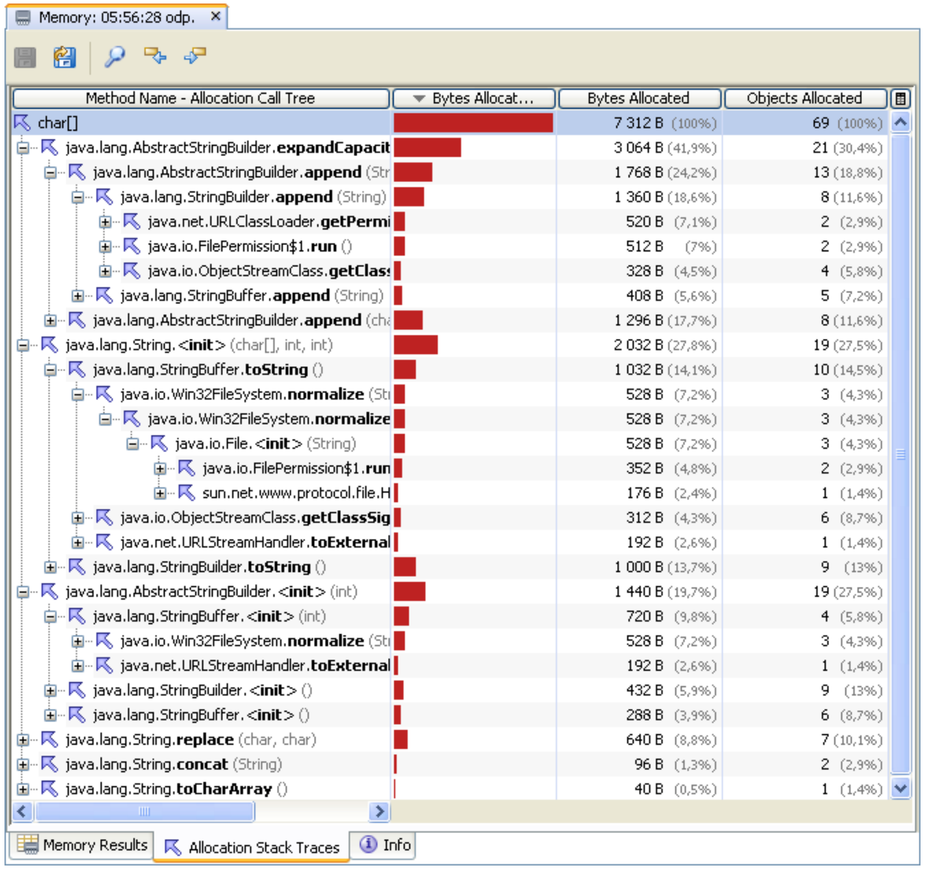

CPU: 05:00:03 odp.

View: Methods

```
Back Traces - Method                                                    Time [...   Time               Invocations
java.nio.ByteBuffer.arrayOffset ()                                                  0.002 ... (0,1%)   4
  when called from sun.nio.cs.SingleByteEncoder.encodeArrayLoop                     0.002 ... (0,1%)   3
    sun.nio.cs.SingleByteEncoder.encodeArrayLoop (java.nio.Cha                      0.054 ...   (4%)   1
      sun.nio.cs.SingleByteEncoder.encodeLoop (java.nio.CharB                       0.023 ... (1,7%)   1
        java.nio.charset.CharsetEncoder.encode (java.nio.Cha                        0.023 ... (1,7%)   1
          sun.nio.cs.StreamEncoder$CharsetSE.implWrite (c                           0.089 ... (6,6%)   1
            sun.nio.cs.StreamEncoder.write (char[], int, int                        0.029 ... (2,1%)   1
              java.io.OutputStreamWriter.write (char[], i                           0.021 ... (1,6%)   1
                java.io.BufferedWriter.flushBuffer ()                               0.018 ... (1,3%)   1
                  java.io.PrintStream.newLine ()                                    0.057 ... (4,2%)   1
                    java.io.PrintStream.println ()                                  0.013 ...   (1%)   1
                      javaapplication.Main.main                                     0.259 ...(19,1%)   1
  when called from sun.nio.cs.StreamEncoder$CharsetSE.writeBytes                    0.000 ...   (0%)   1
    sun.nio.cs.StreamEncoder$CharsetSE.writeBytes ()                                0.058 ... (4,3%)   1
      sun.nio.cs.StreamEncoder$CharsetSE.implFlushBuffer ()                         0.019 ... (1,4%)   1
        sun.nio.cs.StreamEncoder.flushBuffer ()                                     0.015 ... (1,1%)   1
          java.io.OutputStreamWriter.flushBuffer ()                                 0.013 ...   (1%)   1
            java.io.PrintStream.newLine ()                                          0.057 ... (4,2%)   1
              java.io.PrintStream.println ()                                        0.013 ...   (1%)   1
                javaapplication.Main.main (String[])                                0.259 ...(19,1%)   1
```

Memory Profiling

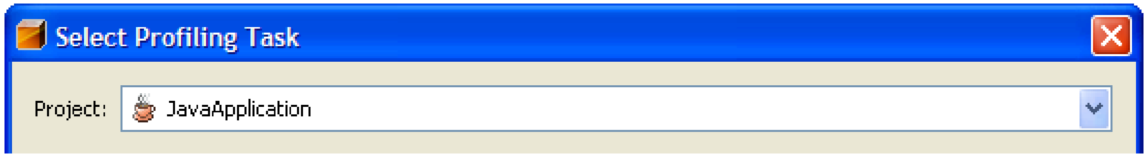
- Tracking allocations or object liveness
- Tracking every nth obj.
- Recording stack traces
- Limited stack depth
- Surviving generations



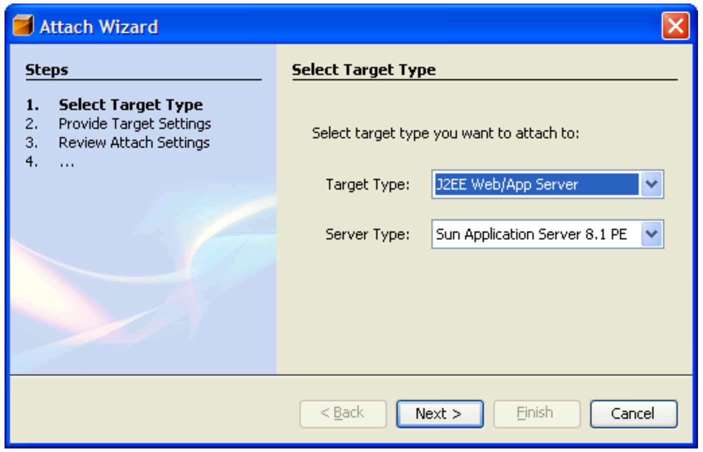
```
Method Name - Allocation Call Tree            Bytes Allocat...   Bytes Allocated    Objects Allocated
char[]                                                            7 312 B (100%)      69 (100%)
java.lang.AbstractStringBuilder.expandCapacit                     3 064 B (41,9%)     21 (30,4%)
  java.lang.AbstractStringBuilder.append (Str                     1 768 B (24,2%)     13 (18,8%)
    java.lang.StringBuilder.append (String)                       1 360 B (18,6%)      8 (11,6%)
      java.net.URLClassLoader.getPermi                              520 B (7,1%)       2 (2,9%)
      java.io.FilePermission$1.run ()                               512 B   (7%)       2 (2,9%)
      java.io.ObjectStreamClass.getClas                             328 B (4,5%)       4 (5,8%)
    java.lang.StringBuffer.append (String)                          408 B (5,6%)       5 (7,2%)
  java.lang.AbstractStringBuilder.append (cha                     1 296 B (17,7%)      8 (11,6%)
java.lang.String.<init> (char[], int, int)                        2 032 B (27,8%)     19 (27,5%)
  java.lang.StringBuffer.toString ()                              1 032 B (14,1%)     10 (14,5%)
    java.io.Win32FileSystem.normalize (Str                          528 B (7,2%)       3 (4,3%)
      java.io.Win32FileSystem.normalize                             528 B (7,2%)       3 (4,3%)
        java.io.File.<init> (String)                                528 B (7,2%)       3 (4,3%)
          java.io.FilePermission$1.run                              352 B (4,8%)       2 (2,9%)
          sun.net.www.protocol.file.H                               176 B (2,4%)       1 (1,4%)
    java.io.ObjectStreamClass.getClassSig                           312 B (4,3%)       6 (8,7%)
    java.net.URLStreamHandler.toExternal                            192 B (2,6%)       1 (1,4%)
  java.lang.StringBuilder.toString ()                             1 000 B (13,7%)      9  (13%)
java.lang.AbstractStringBuilder.<init> (int)                      1 440 B (19,7%)     19 (27,5%)
  java.lang.StringBuffer.<init> (int)                               720 B (9,8%)       4 (5,8%)
    java.io.Win32FileSystem.normalize (Str                          528 B (7,2%)       3 (4,3%)
    java.net.URLStreamHandler.toExternal                            192 B (2,6%)       1 (1,4%)
  java.lang.StringBuilder.<init> ()                                 432 B (5,9%)       9  (13%)
  java.lang.StringBuffer.<init> ()                                  288 B (3,9%)       6 (8,7%)
java.lang.String.replace (char, char)                               640 B (8,8%)       7 (10,1%)
java.lang.String.concat (String)                                     96 B (1,3%)       2 (2,9%)
java.lang.String.toCharArray ()                                      40 B (0,5%)       1 (1,4%)
```

Profiling Capabilities

- Integration with the IDE
  - Profile Main Project, Profile Project
  - Profile File: main, applet, servlet, jsp
  - Profile Test
- Local attach
  - Direct attach
  - Dynamic attach (JFluid 1.4.2)
- Remote attach
  - Direct attach



Project: JavaApplication



Steps

1. Select Target Type
2. Provide Target Settings
3. Review Attach Settings
4. ...

Select Target Type

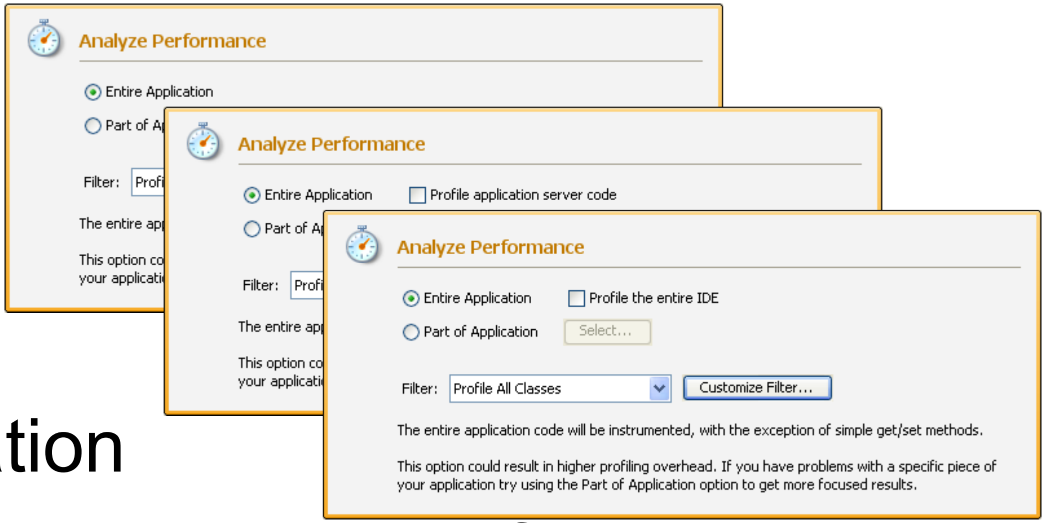
Select target type you want to attach to:

Target Type: J2EE Web/App Server

Server Type: Sun Application Server 8.1 PE

Supported IDE Project Types

- Java Application
- Web Application
- NetBeans Module
- Enterprise Application
- Any project type using Attach Profiler


Analyze Performance

[✓ Entire Application] Part of Application

Profile application server code

Analyze Performance

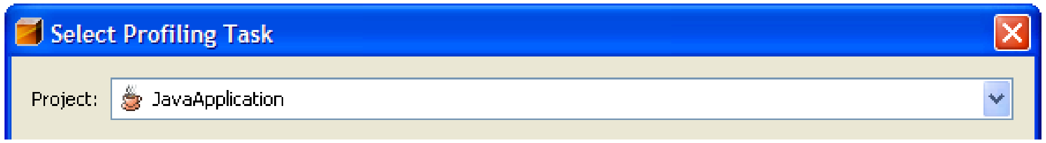
[✓ Entire Application] ☐ Profile the entire IDE

Part of Application Select...

Filter: Profile All Classes Customize Filter...

The entire application code will be instrumented, with the exception of simple get/set methods.

This option could result in higher profiling overhead. If you have problems with a specific piece of your application try using the Part of Application option to get more focused results.


Select Profiling Task

Project: JavaApplication

DEMO

Profiling Java EE 5 Apps with NetBeans

Summary

- With Project GlassFish and NetBeans, you can:
    - Create
    - Develop, Debug
    - Test
    - Verify
    - Monitor
    - Profile
- Java EE 5 today

For More Information

- TS-1969 Blueprints for Using the Simplified Java™EE 5 Programming Model
- TS-1194 Java™ API for XML Web Services (JAX-WS) 2.0
- BOF-2953 Implementing High-Performance Web Services With Next-Generation Java™ Technology APIs
- Java API for XML web services—http://java.sun.com/webservices/jaxws/
- Web Services Management in GlassFish—https://glassfish.dev.java.net/javaee5/ws-mgmt/wsmgmthome.html
- NetBeans Profiler—http://profiler.netbeans.org/

Q&A

Ludovic Champenois, Nazrul Islam, Tomáš Hůrka

http://www.netbeans.org/
https://glassfish.dev.java.net/