TP612

Rational. software

Testing J2EE Applications with IBM Rational PurifyPlus

Goran Begik

Table of Contents

Introduction 1
About Jakarta Tomcat 1
About IBM Rational PurifyPlus 1
About Servlets and JavaServerPages 1
Preparing Java Servlets and JavaServerPages for Testing with Rational PurifyPlus 2
Preparing the Windows Environment for Testing JavaServerPages and Java Servlets with Rational PurifyPlus. 2
Preparing the Java Application Server Environment for Running with Rational PurifyPlus 3
Preparing PurifyPlus for Collecting Data from Java Applications Running in Apache Tomcat 3
The Demo Application 6
Run-Time Testing of Java Servlets and JavaServerPages in Rational PurifyPlus 6
Profiling Application Execution Times with Rational Quantify 6
Code Coverage with Rational PureCoverage 8
Memory Profiling with Rational Purify 10
Maximum Yield with Minimum Effort 11
SIDEBAR: Rational PurifyPlus in Action 11
Rational Purify Viewlet: http://www.therationaledge.com/content/sep_01/Viewlets/Purify/Purify_viewlet.html 11
Rational QuantifyViewlet: http://www.therationaledge.com/content/sep_01/Viewlets/Purify/Purify_viewlet.html 11
References 12

Introduction

This whitepaper addresses questions on how to use IBM Rational® PurifyPlus for a server side Java application. Some general workarounds are available in the online Help section for PurifyPlus tools as well as in the Rational Technical Support database http://www.rational.com/support/technotes/index.jsp.

These instructions, however, may not be sufficient for every available deployment environment -- e.g., a Java application server that hosts and executes Java server side components. Plus, there are many Java application servers on the market now. This whitepaper lpresents a proven and efficient way of using PurifyPlus to test JavaServerPages (JSPs) and Java servlet applications running in Apache Jakarta Tomcat, Version 4. Similar workarounds can be applied for the commercial Java application servers; various J2EE applications have been successfully "Purify-ed"while running in BEA WebLogic, Versions 5.1, 6.0 and 6.1, as well as in IBM WebSphere, Versions 3.5 and 4.

About Jakarta Tomcat

Jakarta is an open source project supported by Sun. The final goal of this project is to create a free J2EE server side solution of a quality equal to those of commercial solutions. The Tomcat application is the main part of this project, and in many people's minds is synonymous with the whole Jakarta undertaking. Tomcat is not a full size Java application server, but only the Servlet+JSP Engine, which is more then enough to deploy and test JSPs and Java Servlet applications. It can either be run as a standalone, or integrated into the Apache Web Server. Tomcat was chosen because it is very robust, reliable, and available for free. More information about Tomcat and the Jakarta project can be found in the list of references at the end of this article. The page listed in the references also contains a URL to the Tomcat installation binaries and source code.[1]

About IBM Rational PurifyPlus

PurifyPlus is a complete solution for testing Java applications. It consists of three applications:

- **Rational Purify** -- A memory profiling tool (collects method and object level memory profiling data and pinpoints application memory hot spots). [2]
- **Rational Quantify** -- An application execution time profiling tool (collects method and line level profiling data and pinpoints application performance bottlenecks).
- **Rational PureCoverage** -- A code coverage tool (collects information about the untested parts of the application by highlighting the unexecuted methods and lines of code).

These three tools provide full support not only for Java, but also for Visual C/C++, Visual Basic and .NET applications. In this whitepaper, the term PurifyPlus, refers to Rational PurifyPlus Version 2001A. In order to test Java server side applications with Rational PurifyPlus, you must install it on the server machine.[3]

About Servlets and JavaServerPages

Servlets are Java applications running on the server side; their main purpose is to create content for Web pages upon request from the client. The client side is normally a Web browser and a Web page onto which users can enter data that will be processed by the servlet running on the server side. The results of servlet operations are displayed on a Web page on the client side.

JavaServer Pages (JSPs) are basically HTML pages with special tags that enable them to either embed the Java code in the page or access Java beans and servlets running on the server. Rational PurifyPlus

[1] TIP: I recommend installing Tomcat in the directory without spaces in the names. That makes it easier to include Tomcat directories in the classpath when compiling the servlets, for example.
[2] A full version of Rational PurifyPlus can be downloaded for the evaluation purposes from the Rational web page. The evaluation license expires in fifteen days. Rational Purify, Quantify, and PureCoverage are also available in the following suites of Rational tools: Rational Suite DevelopmentStudio, Rational Suite TestStudio, and Rational Suite Enterprise.
[3] TIP: When installing Rational PurifyPlus select the custom installation option and check the option: "Add Purify (and Quantify and PureCoverage) to the system path". This is not a default setting, and it can be helpful when executing PurifyPlus tools from the command line.

cannot check the HTML syntax, but it can test the Java part of the JSPs by monitoring the events that it collects from the Java Virtual Machine (JVM).[4]

Preparing Java servlets and JavaServer Pages for Testing

PurifyPlus offers two basic levels of data collection: the method level and the line level. To test Java applications on the method level, you don't need to recompile the tested Java application to collect all the relevant data about the methods. If you are interested in the line level information, however, then you need to recompile the Java code with the symbolic debugging information. The switch to get the symbols in the Java class files with the Sun Java compiler is '–g'.

```
>javac -g MyServlet.java
```

All the information about Java applications run in the Sun Java2 compatible virtual machines will be collected through JVMPI (Java Virtual Machine Profiling Interface). It is necessary to use the JVM that is fully compatible with the Sun Java 2 specifications. Rational PurifyPlus also fully supports Microsoft JVM.

You will need to include the servlet library '*servlet.jar*'in the classpath when compiling servlets. This library is installed with Tomcat and can be found in the directory <Tomcat home>\common\lib. Here is an example command line for compiling MyServlet.java with the symbolic debugging information:

```
Javac -g -classpath <<Tomcat home>\common\lib\servlet.jar>
MyServlet.java
```

Preparing the Windows Environment for Testing JavaServerPages and Java Servlets with Rational PurifyPlus.

The environmental variable:

JAVA_HOME

specifies the home directory for the default JVM. It is also the default choice for the Java Virtual Machine that will be used by Rational PurifyPlus. To correctly set up the Java run that allows you to use PurifyPlus, however, you must execute the following command line before you start profiling for the first time after a new Java service setup:

pstart <or qstart, or cstart> -setup

This command will update the Java policy file for the selected Java Run-time Environment (JRE).

Now, let's look at another environmental variable.

JAVA_OPTIONS (or IBM_JAVA_OPTIONS for the IBM JVM)

Rational PurifyPlus collects all the information about Java applications through the Java Virtual Machine Profiling Interface (JVMPI). Since PurifyPlus and JVM run as two different processes, the Java process needs to load a PurifyPlus shared library called PureJVMPI that will listen to the JVM events, collect information about the run through the JVMPI, and send this information to the PurifyPlus tool. This dynamically linked library is loaded through an additional option for the run of the Java executable:'*–Xrun'*. Here is an example of how PurifyPlus can be started from the command line:

```
>java -XrunPureJVMPI:Purify Java_App(or
>java -XrunPureJVMPI:Quantify Java_App or
```

[4] For more information on building Java servlets and JSPs, see Kawar Ahmed and Loïc Julien's article in the February, 2001 issue of *The Rational Edge*. [DESIGNER: LINK TO http://www.therationaledge.com/content/feb_01/t_entrose2_ka.html]

```
>java –XrunPureJVMPI:Coverage Java_App)
```

To profile server side Java applications and Java services, you need to manually create a special system environmental variable with the *'–XRun'* option that will launch the selected PurifyPlus tool every time JVM is engaged. For the Sun JVM. the name of the variable is *_JAVA_OPTIONS* and the value for the variable should be:

```
–XrunPureJVMPI:Purify (or –XrunPureJVMPI:Quantify
or –XrunPureJVMPI:Coverage)
```

If you use the IBM JVM (necessary for running the IBM WebSphere Java application server), then the name of the environmental variable should be IBM_JAVA_OPTIONS.

The above option specified in the *_JAVA_OPTIONS* environment variable will launch Rational Purify (or Quantify, or PureCoverage) every time Tomcat application is started, and Purify (or Quantify, or PureCoverage) will automatically start collecting data for this Java process. You can keep the report free of data collected from parts of the Java process that are irrelevant for testing servlets and JSPs by using both pre-filters and the PurifyPlus Filter Manager.

Preparing the Java Application Server Environment for Running with Rational PurifyPlus

To run Rational PurifyPlus with the Java application server, it is essential to specify the same JAVA_HOME variable for both the server and the PurifyPlus tools. For Apache Tomcat, you can set this up directly by executing the command:

SET JAVA_HOME=<path to JRE installation directory>

For BEA WebLogic Servers and IBM WebSphere, you can modify the batch files you use for setting up the environment to run the server by changing the value of the variable JAVA_HOME as shown above.

Preparing PurifyPlus for Collecting Data from Java Applications Running in Apache Tomcat

To enable Rational PurifyPlus to collect information from your Java servlets and JSPs running in Apache Tomcat, you must create a custom set of pre-filters prior to testing.

PurifyPlus Filters. In PurifyPlus there are two ways of filtering data that is not relevant for testing. The first way is by setting pre-filters for the tool you plan to use (Rational PurifyPlus, Quantify, or PureCoverage). A pre-filter is a list of Java packages from which no data should be collected during the run. This list is defined in the [Prefilter] section of the *Profile.ini* file that can be found in the main directory of each of the PurifyPlus tools.

Each of the Java packages on the list should be in the new line. Let's look at an example of how prefiltering works in PurifyPlus functions.

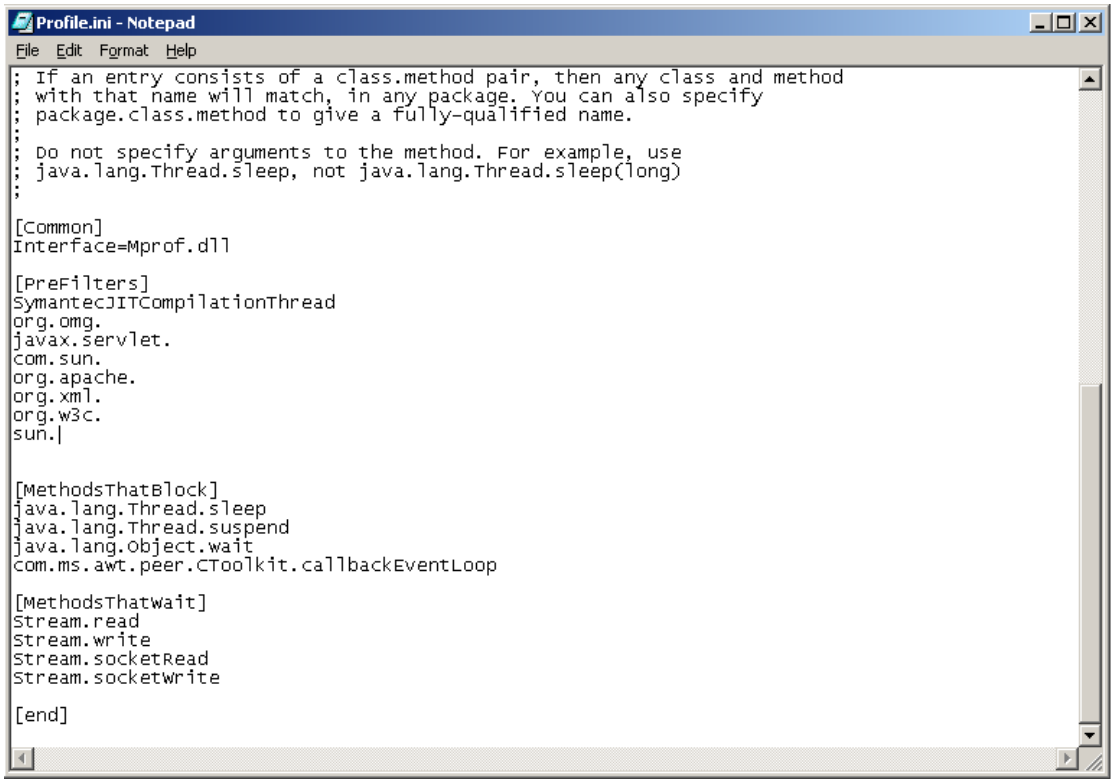


Figure 1: A Profile.ini File for Rational Purify

Let's say we would like to exclude the Java package *com.sun*. If we specify `'com.sun.'` (please note the dot at the end of the string) as a line in the pre-filtering section of *Profile.ini*, then the file and all its sub-packages will be excluded from data collection by the selected PurifyPlus tool during testing (see Figure 1). To prefilter individual classes you can use fully qualified names (package first) without the terminating dot. For example, *com.rational.MyClass* would prefilter *MyClass* in the *com.rational* package. If you specify just the class name (e.g. *MyClass*), then this class will be prefiltered in all packages. PurifyPlus tools do not prefilter individual methods of a class.

The PurifyPlus Filter Manager. The second way to filter data is through the Filter Manager feature available through the Graphical User Interface of each of the tools (see Figures 2A and 2B). The Filter Manager creates a special binary filter file for the tested Java application. It can be used only after the profiling or the coverage data has been collected.

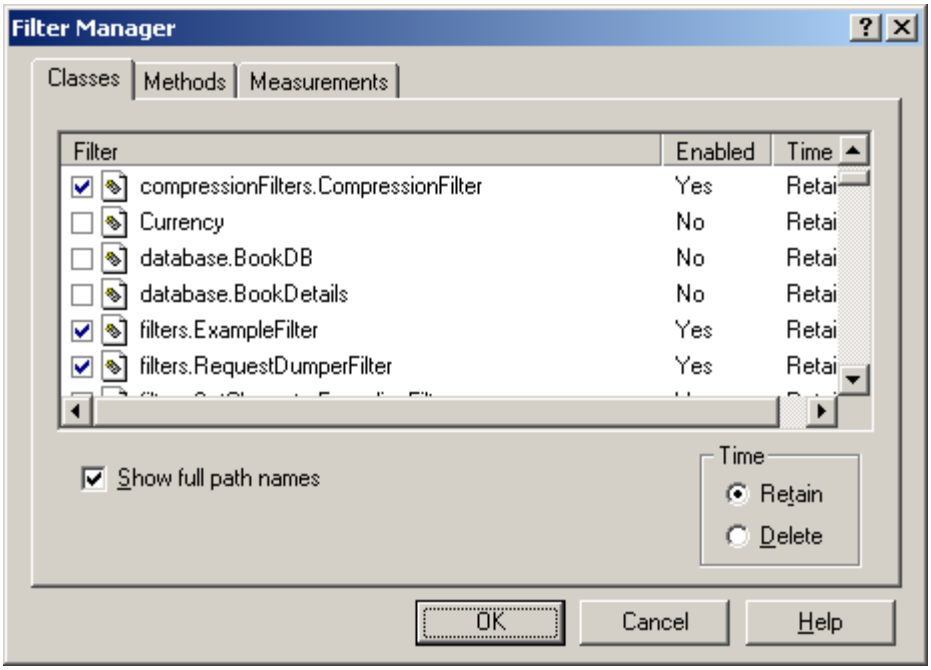


Figure 2A: Main window of the Rational PurifyPlus Filter Manager in Rational Quantify

The class files checked on the list will be excluded from the reports, but the data collected for them will be kept in the overall results (i.e., the "Time" section in the lower right corner of the Filter Manager window shown in Figure 2A).[5]

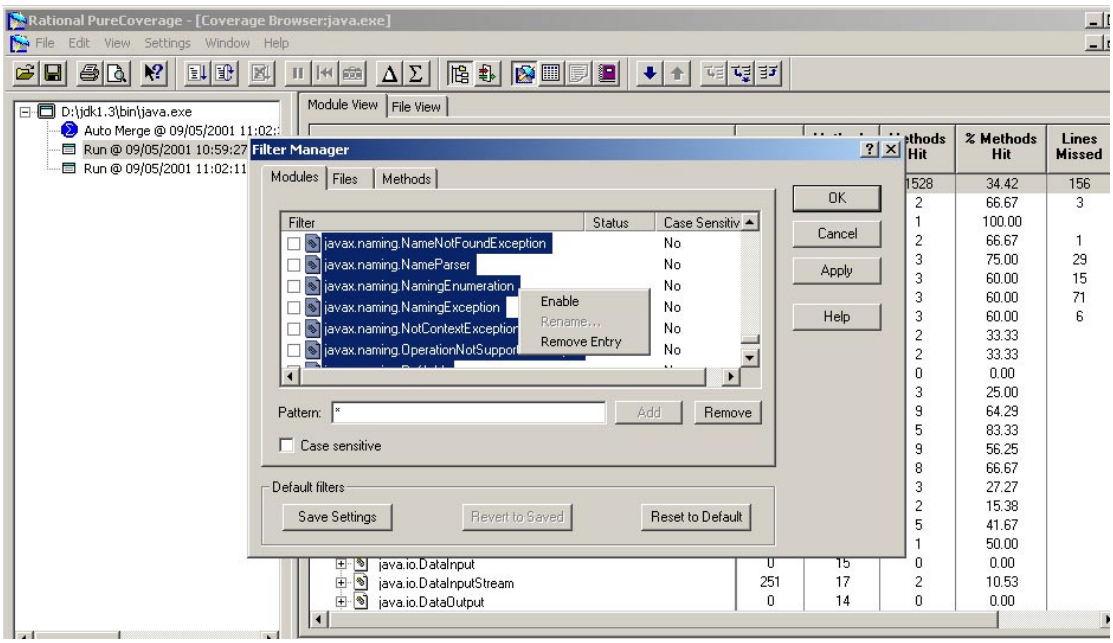


Figure2B: Using the Rational PurifyPlus Filter Manager in Rational PureCoverage

[5]TIP: You can check a large number of classes at once (e.g. all *java.** classes) by marking them on the list of classes with the mouse and holding the CTRL, or SHIFT key on the keyboard. A right-click on the marked set of classes will bring-up the pop-up window, and you can then choose "Enable" to create filters for all marked Java classes at once.

To run Tomcat in PurifyPlus, I recommend the following list of pre-filters to the Profile.ini file for each of the PurifyPlus tools (Purify, Quantify, PureCoverage):

```
org.omg.
javax.servlet.
com.sun.
org.apache.
org.xml.
org.w3c.
sun.
```

For other Java server applications, you will need to create other pre-filters. See my tech tips http://www.therationaledge.com/content/sep_01/t_techTips_gb.html, on how to run Rational PurifyPlus with the BEA WebLogic and IBM WebSphere Java application servers.

The Demo Application

The demo application used in this article is an online bookstore called "Duke's Bookstore." This Web application can be downloaded from Sun Web site. It was originally created as part of a tutorial for creating and running Java servlets. It consists of a series of Web pages and servlets that create the content for these pages. Rational PurifyPlus can be engaged in testing either the whole Web application or only parts of it (individual servlets, for example.)

Run-Time Testing of Java Servlets and JavaServerPages in Rational PurifyPlus

After compiling the Java components with the debugging information and setting up the necessary filters for the tools, you will be ready to collect line level profiling and code coverage information from the servlets and JavaServer Pages.

As a first step, I suggest engaging Rational Quantify and collecting information for an execution time analysis.

Profiling Application Execution Times with Rational Quantify

Assigning the value `'-XrunPureJVMPI:Quantify'` to the environmental variable `_JAVA_OPTIONS` will launch Quantify when you start Tomcat server application. As you're browsing through the demo Web application Quantify will record times spent in executing each method and line of code that was triggered through the Web page. After Tomcat is initialised and started, the test application can be reached by specifying the following URL in the browser:

http://localhost:8080/bookstore/bookstore.html

After you finish the test run, close the browser and stop the Tomcat server application. The results of the profiling will be displayed in several different views in Quantify. The first is the CallGraph view, as shown in Figure 3.

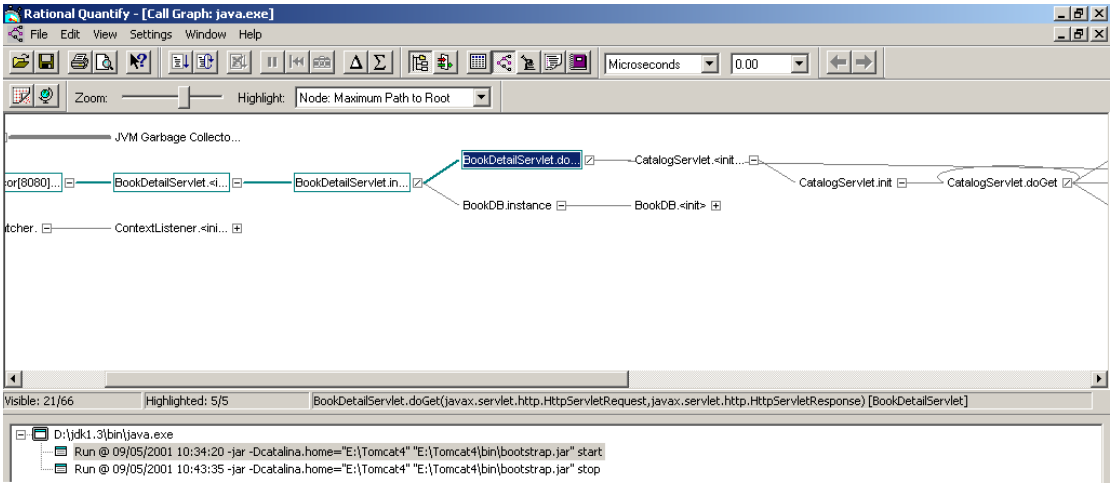


Figure 3: Rational Quantify Call Graph for a Web Application

The Call Graph highlights the chain of calls that consumed most of the execution time. The thicker line highlights the slowest part of the application and represents the percentage of time spent executing the highlighted method, compared to the overall execution time.

The line level information is included in the Annotated Source view for the selected method, as shown in Figure 4:

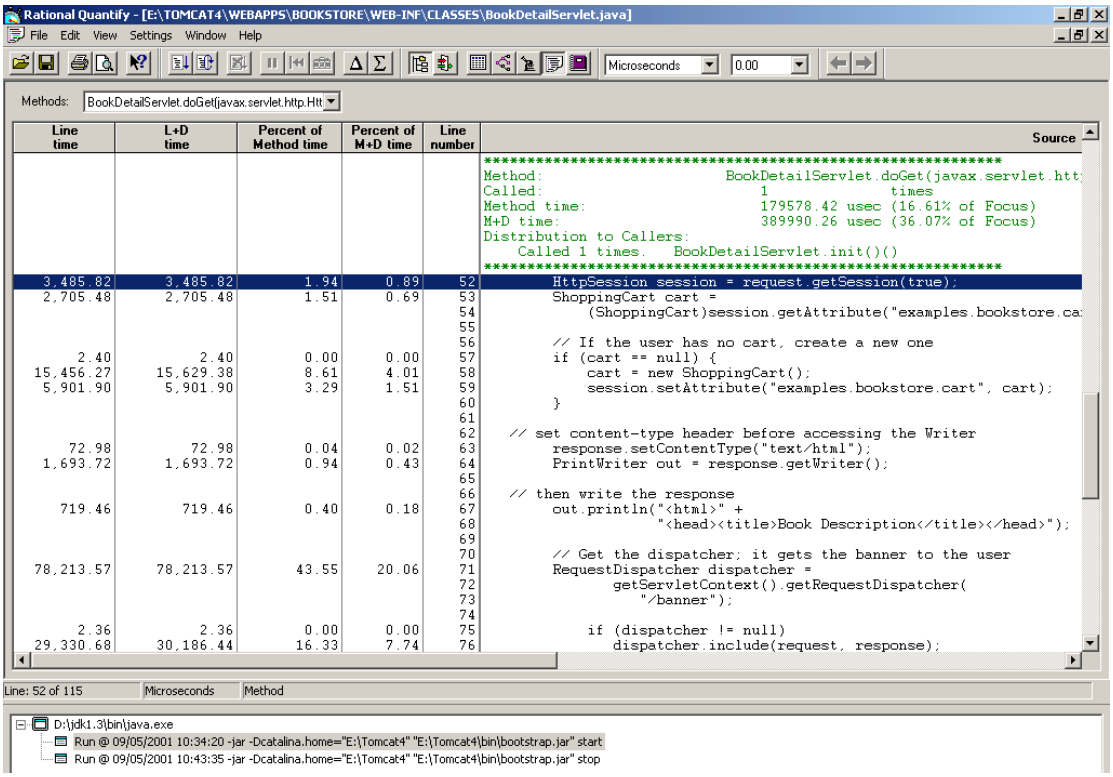


Figure 4: Rational Quantify Annotated Source for One Method in a Web Application.

Further information can be obtained from other views that Quantify creates, including the Function Detail view shown in Figure 5:

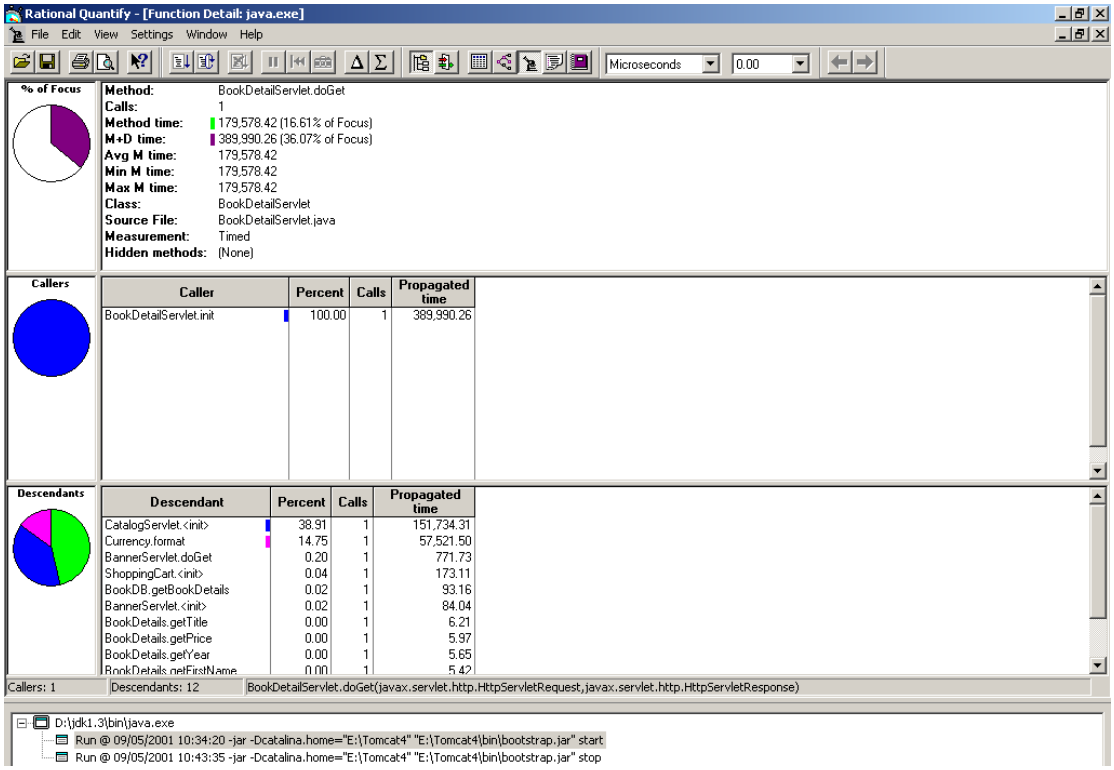


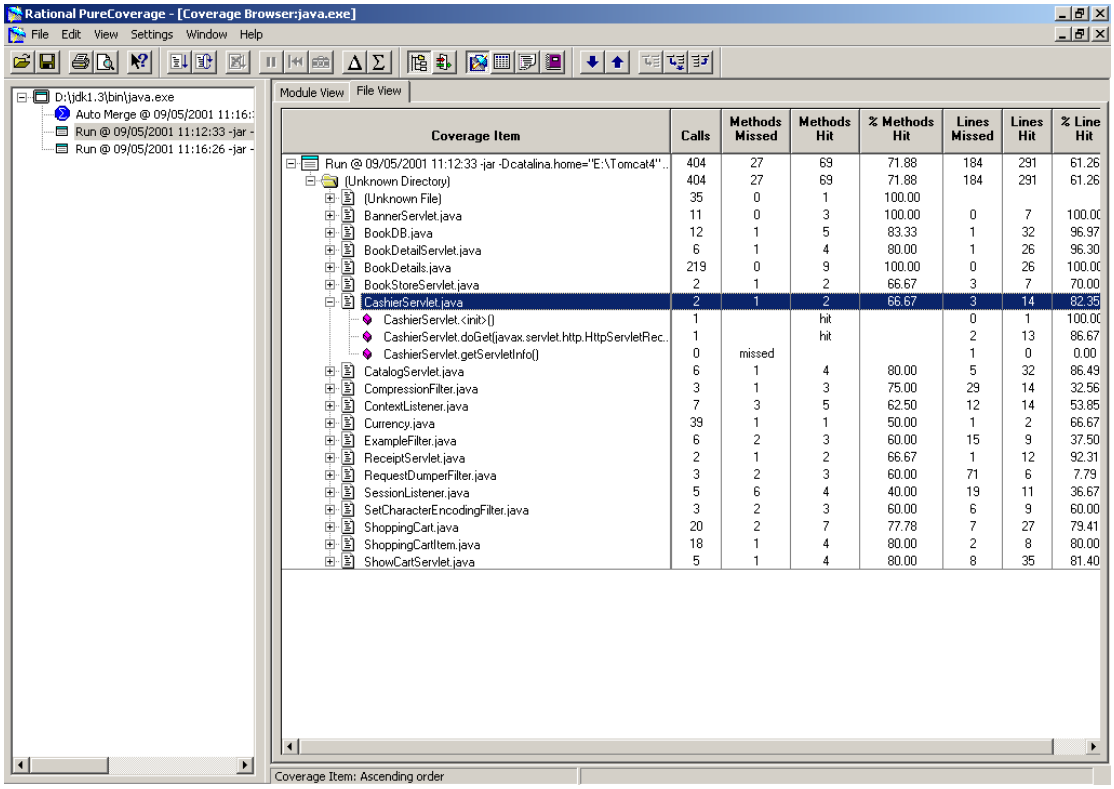
Figure 5: Rational Quantify Function Detail for One Method in a Web Application

The advantage of using Quantify over some conventional solutions is the way the profiling data is represented. Quantify leads you straight to the hart of the performance bottleneck in the tested application.

Code Coverage with Rational PureCoverage

If we now change the value for the *_JAVA_OPTIONS* environmental variable to *'-XrunPureJVMPI:Coverage'* and repeat the test run of the sample Web application, PureCoverage will record the methods and lines of code that were tested and highlight the untested parts of the application:

As Figures 6A and 6B show, PureCoverage provides both method and line level information about coverage of the tested Java servlet application:

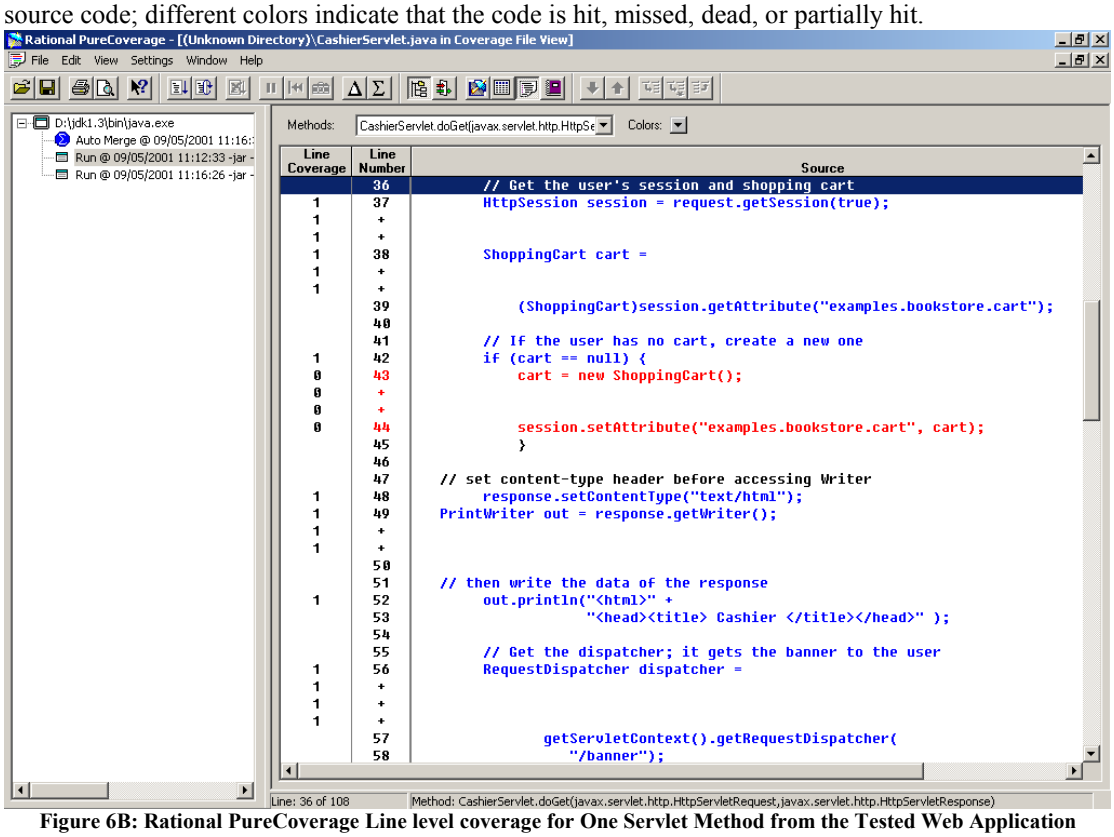


Coverage Item	Calls	Methods Missed	Methods Hit	% Methods Hit	Lines Missed	Lines Hit	% Line Hit
Run @ 09/05/2001 11:12:33 -jar -Dcatalina.home="E:\Tomcat4"...	404	27	69	71.88	184	291	61.26
(Unknown Directory)	404	27	69	71.88	184	291	61.26
(Unknown File)	35	0	1	100.00			
BannerServlet.java	11	0	3	100.00	0	7	100.00
BookDB.java	12	1	5	83.33	1	32	96.97
BookDetailServlet.java	6	1	4	80.00	1	26	96.30
BookDetails.java	219	0	9	100.00	0	26	100.00
BookStoreServlet.java	2	1	2	66.67	3	7	70.00
CashierServlet.java	2	1	2	66.67	3	14	82.35
CashierServlet.<init>()	1		hit		0	1	100.00
CashierServlet.doGet(javax.servlet.http.HttpServletRec...	1		hit		2	13	86.67
CashierServlet.getServletInfo()	0	missed			1	0	0.00
CatalogServlet.java	6	1	4	80.00	5	32	86.49
CompressionFilter.java	3	1	3	75.00	29	14	32.56
ContextListener.java	7	3	5	62.50	12	14	53.85
Currency.java	39	1	1	50.00	1	2	66.67
ExampleFilter.java	6	2	3	60.00	15	9	37.50
ReceiptServlet.java	2	1	2	66.67	1	12	92.31
RequestDumperFilter.java	3	2	3	60.00	71	6	7.79
SessionListener.java	5	6	4	40.00	19	11	36.67
SetCharacterEncodingFilter.java	3	2	3	60.00	6	9	60.00
ShoppingCart.java	20	2	7	77.78	7	27	79.41
ShoppingCartItem.java	18	1	4	80.00	2	8	80.00
ShowCartServlet.java	5	1	4	80.00	8	35	81.40

Figure 6A: Rational PureCoverage Method Coverage for a Tested Web Application

In the method level coverage, PureCoverage provides statistics for the methods of the tested application, sorted by the modules in which the methods reside, or sorted by the source file in which the methods are defined.

The coverage information about the lines of code of the tested application is presented in the annotated source code; different colors indicate that the code is hit, missed, dead, or partially hit.



Line Coverage	Line Number	Source
	36	// Get the user's session and shopping cart
1	37	HttpSession session = request.getSession(true);
1	+	
1	+	
1	38	ShoppingCart cart =
1	+	
1	+	
	39	(ShoppingCart)session.getAttribute("examples.bookstore.cart");
	40	
	41	// If the user has no cart, create a new one
1	42	if (cart == null) {
0	43	cart = new ShoppingCart();
0	+	
0	+	
0	44	session.setAttribute("examples.bookstore.cart", cart);
	45	}
	46	
	47	// set content-type header before accessing Writer
1	48	response.setContentType("text/html");
1	49	PrintWriter out = response.getWriter();
1	+	
1	+	
	50	
	51	// then write the data of the response
1	52	out.println("<html>" +
	53	"<head><title> Cashier </title></head>");
	54	
	55	// Get the dispatcher; it gets the banner to the user
1	56	RequestDispatcher dispatcher =
1	+	
1	+	
1	+	
	57	getServletContext().getRequestDispatcher(
	58	"/banner");

Figure 6B: Rational PureCoverage Line level coverage for One Servlet Method from the Tested Web Application

This type of information can be very useful when determining the steps for run time testing or creating script files that will automate the tests. PureCoverage also allows you to merge the coverage data for different tests on the same application, thereby providing a clear overview about the quality of the tests that have been made on the developed application.

Memory Profiling with Rational Purify

Using Rational Purify to obtain a memory profile of the tested Web application is similar to using PureCoverage and Quantify. Taking "snapshots" of the memory usage for the running application, however, gives you the opportunity to compare the memory footprint of the application at different stages of its execution. It is a very useful method for detecting memory leaks.

Memory leaks in the server applications (which often run 24 X 7) can easily bring both an application and the system down, because the application continuously uses more and more memory as it runs. The impact of memory leaks on application performance is significant as well. The memory footprint that you can record for Java applications by using Rational Purify enables you to analyze memory usage in fine detail.[6]

When applied against Java Servlets and JSPs, Purify produces the same types of reports it produces for Java applications and Java applets. Figure 7 shows a Purify recording for a tested JSP with the invoked JavaBean methods.

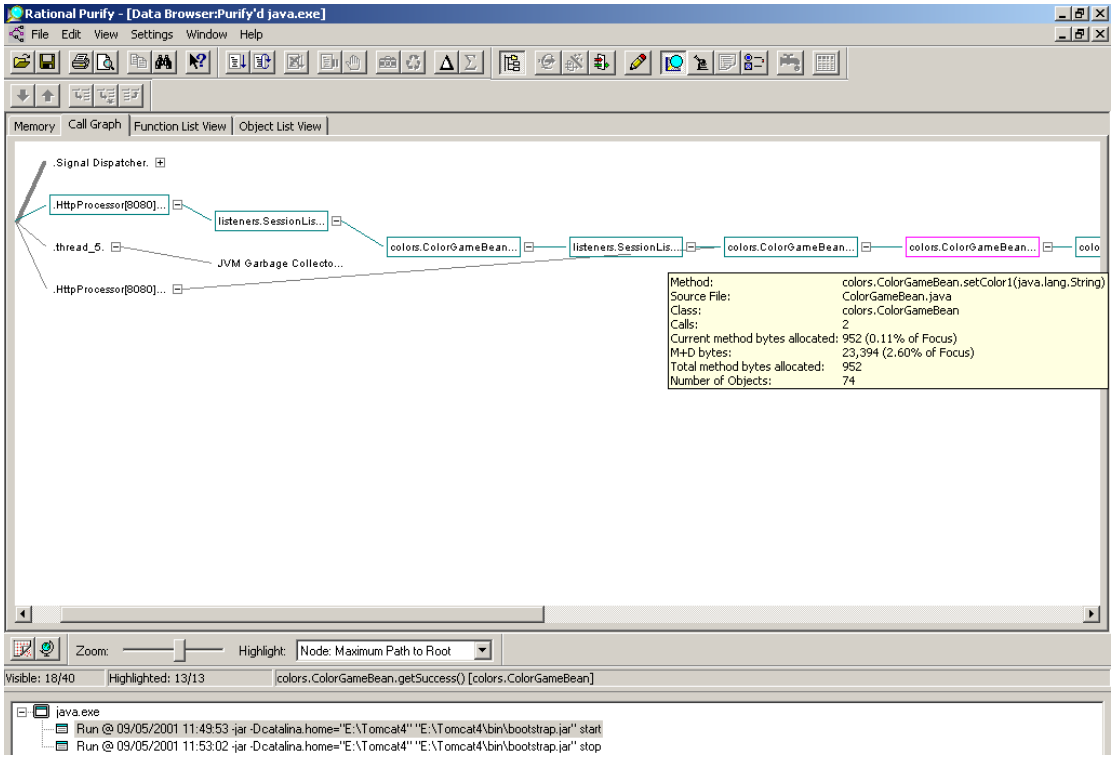
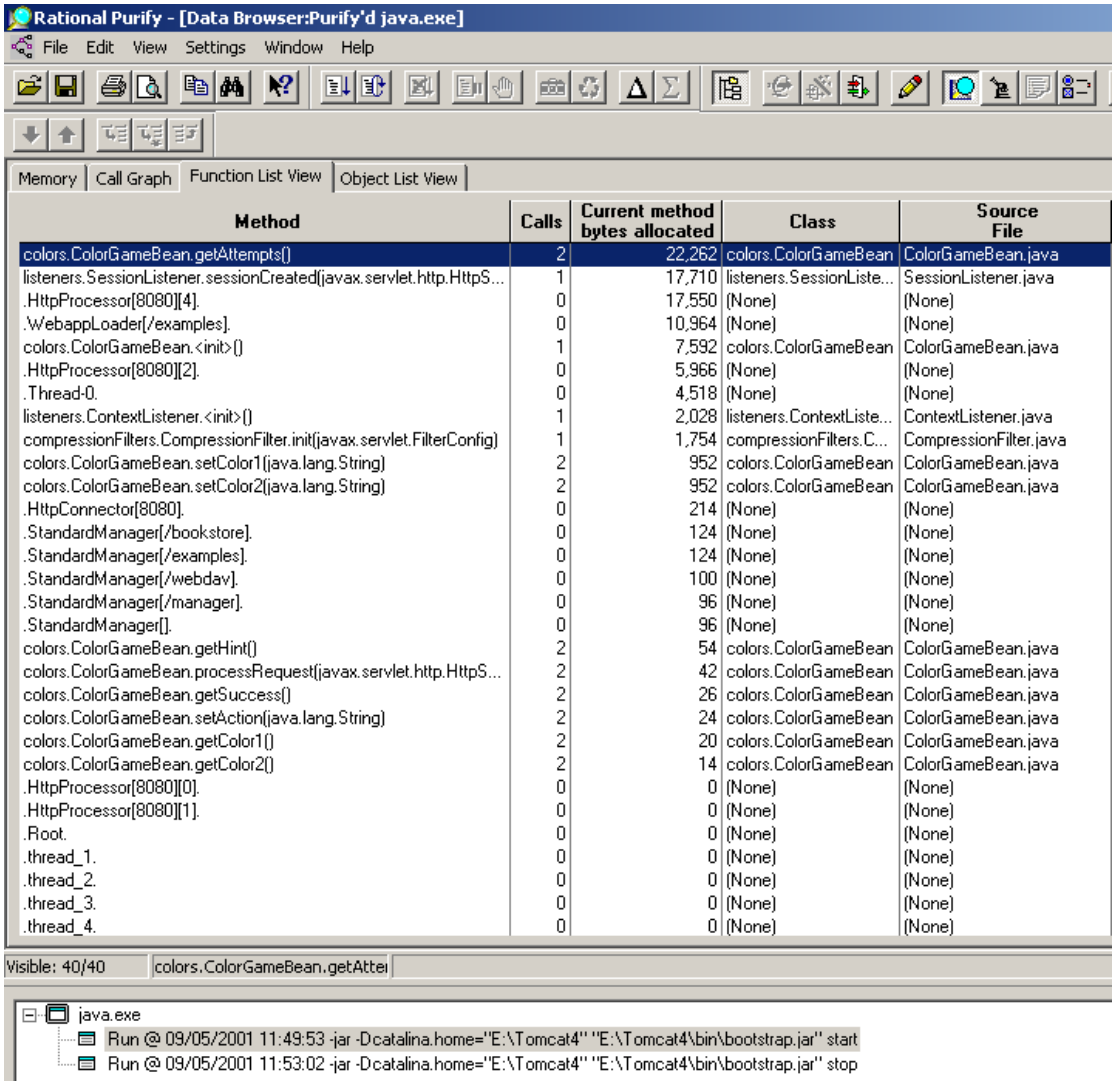


Figure 7: Rational Purify Call Graph for a Demo JSP

The list of methods gives more information about the memory usage for this JSP session, as shown in Figure 8.

[6] For more information on using Rational Purify with Java applications, look for additional whitepapers at www.rational.com or see related articles in theJanuary and June issues of *The Rational Edge,*

http://www.therationaledge.com/content/jan_01/m_memjava_gb.html
http://www.therationaledge.com/content/jun_01/t_profiling_gb.html]



Method	Calls	Current method bytes allocated	Class	Source File
colors.ColorGameBean.getAttempts()	2	22,262	colors.ColorGameBean	ColorGameBean.java
listeners.SessionListener.sessionCreated(javax.servlet.http.HttpS...	1	17,710	listeners.SessionListe...	SessionListener.java
.HttpProcessor[8080][4].	0	17,550	(None)	(None)
.WebappLoader[/examples].	0	10,964	(None)	(None)
colors.ColorGameBean.<init>()	1	7,592	colors.ColorGameBean	ColorGameBean.java
.HttpProcessor[8080][2].	0	5,966	(None)	(None)
.Thread-0.	0	4,518	(None)	(None)
listeners.ContextListener.<init>()	1	2,028	listeners.ContextListe...	ContextListener.java
compressionFilters.CompressionFilter.init(javax.servlet.FilterConfig)	1	1,754	compressionFilters.C...	CompressionFilter.java
colors.ColorGameBean.setColor1(java.lang.String)	2	952	colors.ColorGameBean	ColorGameBean.java
colors.ColorGameBean.setColor2(java.lang.String)	2	952	colors.ColorGameBean	ColorGameBean.java
.HttpConnector[8080].	0	214	(None)	(None)
.StandardManager[/bookstore].	0	124	(None)	(None)
.StandardManager[/examples].	0	124	(None)	(None)
.StandardManager[/webdav].	0	100	(None)	(None)
.StandardManager[/manager].	0	96	(None)	(None)
.StandardManager[].	0	96	(None)	(None)
colors.ColorGameBean.getHint()	2	54	colors.ColorGameBean	ColorGameBean.java
colors.ColorGameBean.processRequest(javax.servlet.http.HttpS...	2	42	colors.ColorGameBean	ColorGameBean.java
colors.ColorGameBean.getSuccess()	2	26	colors.ColorGameBean	ColorGameBean.java
colors.ColorGameBean.setAction(java.lang.String)	2	24	colors.ColorGameBean	ColorGameBean.java
colors.ColorGameBean.getColor1()	2	20	colors.ColorGameBean	ColorGameBean.java
colors.ColorGameBean.getColor2()	2	14	colors.ColorGameBean	ColorGameBean.java
.HttpProcessor[8080][0].	0	0	(None)	(None)
.HttpProcessor[8080][1].	0	0	(None)	(None)
.Root.	0	0	(None)	(None)
.thread_1.	0	0	(None)	(None)
.thread_2.	0	0	(None)	(None)
.thread_3.	0	0	(None)	(None)
.thread_4.	0	0	(None)	(None)

Figure 8: Rational Purify Function List for a JSP

Just as Rational Quantify leads users to the portion of the application that consumes the most execution time, Purify leads the user to the application's memory bottleneck. By sorting the methods in the Function List view, it is easy to locate those with excessive memory usage; Purify also provides numerous other views that can help you accomplish this.

Maximum Yield with Minimum Effort

It is easy to deploy Rational PurifyPlus on the server machine to obtain extensive profiling and code coverage information for Java server side applications. You can use Rational PurifyPlus not only with commercial Java server applications, but also with a lightweight -- and free -- Java servlet and JSP engine such as Apache Tomcat.

SIDEBAR: Rational PurifyPlus in Action

To gain a better understanding of the procedures described in this article, you can download a brief, animated viewlet about each Rational PurifyPlus tool. Just click on the links below.

Rational Purify Viewlet: http://www.therationaledge.com/content/sep_01/Viewlets/Purify/Purify_viewlet.html

Rational QuantifyViewlet: http://www.therationaledge.com/content/sep_01/Viewlets/Purify/Purify_viewlet.html

Rational PureCoverageViewlet: http://www.therationaledge.com/content/sep_01/Viewlets/Coverage/Coverage_viewlet.html

References

1. PurifyPlus Homepage http://www.rational.com/products/pqc/index.jsp

2. Jakarta Project Homepage http://jakarta.apache.org/

3. Java Servlet Specifications http://java.sun.com/products/servlet/

4. JavaServer Pages™ (JSP) Specifications http://java.sun.com/products/jsp/

5. Khawar Ahmed and Loïc Julien, "Enterprise Java and Rational Rose - Part II." *The Rational Edge*, February 2001. http://www.therationaledge.com/content/feb_01/t_entrose2_ka.html

6. Goran Begic, "Memory Profiling in Java." *The Rational Edge*, January 2001.http://www.therationaledge.com/content/jan_01/m_memjava_gb.html

7. Goran Begic, "Monitoring Object Creation in Java Application Profiling with Rational PurifyPlus." *The Rational Edge*, June 2001. http://www.therationaledge.com/content/jun_01/t_profiling_gb.html

8. Programmer's Guide to Servlets in Netscape Enterprise Server 4.0http://developer.netscape.com/docs/manuals/enterprise/40/servlets/1-intro.htm

9. Servlets Bookstore Example http://java.sun.com/docs/books/tutorial/information/download.html

IBM software integrated solutions
IBM Rational supports a wealth of other offerings from IBM software. IBM software solutions can give you the power to achieve your priority business and IT goals.

- *DB2® software helps you leverage information with solutions for data enablement, data management, and data distribution.*
- *Lotus® software helps your staff be productive with solutions for authoring, managing, communicating, and sharing knowledge.*
- *Tivoli® software helps you manage the technology that runs your e-business infrastructure.*
- *WebSphere® software helps you extend your existing business-critical processes to the Web.*
- *Rational® software helps you improve your software development capability with tools, services, and best practices.*

Rational software from IBM
Rational software from IBM helps organizations create business value by improving their software development capability. The Rational software development platform integrates software engineering best practices, tools, and services. With it, organizations thrive in an on demand world by being more responsive, resilient, and focused. Rational's standards-based, cross-platform solution helps software development teams create and extend business applications, embedded systems and software products. Ninety-eight of the Fortune 100 rely on Rational tools to build better software, faster. Additional information is available at www.rational.com and www.therationaledge.com, the monthly e-zine for the Rational community.

IBM Corporation
Software Group
Route 100
Somers, NY 10589
U.S.A.

Printed in the United States of America
01-03 All Rights Reserved.
Made in the U.S.A.

The IBM home page on the Internet can be found at **ibm.com**