Logging, Tracing, and Timing in Tuscany

Tuscany Logging, Tracing, Timing Information

This page describes the logging, tracing, and timing features of Tuscany. These features can be used to debug problems with the Tuscany runtime, understand code flow, and provide timing metrics for competitive analysis or performance claims.

Java Logging versus Aspect Oriented Programming (AOP) Approach

There are two types of logging embeded in the Tuscany runtime: explicit and implicit. The explicit logging is implemented by the Java 2 SE core logging facilities such as java.util.logging.Logger. This is the logging that many Java programmers are familiar with, and the logging requires calls to the Logger programming interface. Throughout the Tuscany code base you see calls to this facility such as this call in Tuscany org.apache.tuscany.sca.node.launcher.NodeLauncher.

Java Logging in Tuscany

```
        logger.info("Apache Tuscany SCA Node is starting...");
```

Calls to this facility by default go to the standard output of the program. This is the typical result that one sees in the console:

Java Logging Output

```
INFO: Apache Tuscany SCA Node is starting...
```

The Java SE core logging works well as a general purpose logger and gives a general idea of the important phases in the Tuscany runtime. However, one drawback is that this logging system only publishes predetermined messages created by the developers. If you wish to change the logging points, you must change the Tuscany source code, build, and run again with your private code base.

Another logging and tracing facility is available in Tuscany that gives the user a bit of flexibility in what gets logged and requires no source code modifications to run. This is an Aspect Oriented Programming (AOP) approach that runs via a Java agent at runtime. Just like a debugger can start and stop and inspect a Tuscany Java program, so too can an AOP agent inspect and report on a Tuscany Java program. Tuscany employs the AspectJ implementation, and its runtime agent is contained in the aspectjweaver.jar file. The AOP agent is specified from a command line option or runtime options:

Java AOP Agent

```
java -javaagent:"<path to aspectjweaver.jar>" calculator.CalculatorClient
```

The AOP logging and tracing agent can be run with any Tuscany program. No additional calls or log statements need be added to the Tuscany code. The results of running a timing for example, look like this:

Tuscany AOP Output

```
Running org.apache.tuscany.sca.aspectj.TracingTestCase
Timing Around timedSection jp=call(void java.util.logging.Logger.info(String))
Oct 21, 2008 9:26:36 AM org.apache.tuscany.sca.aspectj.TracingTestCase info_aroundBody0
Timing Around timedSection Roundtrip is 32ms for jp.getSignature=void java.util.logging.Logger.info(String)
```

Full documentation on AspectJ is available at the AspectJ web site. Tuscany provides example usage in the module tracing-aspectj in the build tree and the released code.

Brief Apect Oriented Programming Introduction

In order to understand the aspect oriented approach to logging in Tuscany, one must understand a few basic concepts.

A *join point* is well defined point in the program flow. An example join point is the entry to a Java method. Another is after an exception is thrown. These join points contain useful information when running a program such as the argument list, the call stack, and the signature of the point that is executing.

A *point cut* is a subset of all the program join points. For instance, a point cut might be only the join points from exceptions thrown by the Tuscany runtime. Or only the set of calls to a Tuscany API org.apache.tuscany.sca.node.launcher.NodeLauncher.

Advice is the code that is run when a point cut is reached. For instance, your advice might want to print out all the arguments when entering a given method. Or your advice might show the call stack of an exception and print the original cause of the exception.

Finally, an *aspect* is a module that bundles up a particular set of point cuts and advice. Here is an example aspect that is marked with Java 5 annotations. The name of the aspect is *LoggingAspect*, there is a point cut defined at Tuscany calls to org.apache.tuscany.sca.* classes (but not inside of Aspect classes), and when the point cut is run, the advice for this point cut prints out the join point signature.

Example Aspect

```
@Aspect
public class LoggingAspect {
    @Pointcut("call(* org.apache.tuscany.sca..*(..)) && (!within(org.apache.tuscany.sca.aspectj.*Aspect))")
    public void anyMethodCall() {
    }

    @Before("anyMethodCall()")
    public void before(JoinPoint jp) {
        System.out.println("Logging Before anyMethodCall jp.getSignature=" + jp.getSignature());
    }
}
```

Full explanation of these concepts are in the AspectJ documents. The purpose here is to provide enough information to allow a user to run or modify the Tuscany aspects to one's liking.

Logging and Tracing in Tuscany

Following the approach of AspectJ, Tuscany provides two concrete aspects for logging and tracing. org.apache.tuscany.sca.aspectj.LoggingAspect provides logging for methods, constructors, and static initializers in Tuscany. org.apache.tuscany.sca.aspectj.SimpleTracingAspect provide tracing into user methods, constructors, and static initializers. Both provide detailed information when an Exception or Throwable is thrown by the Tuscany runtime or user SCA program.

Both LoggingAspect and SimpleTracingAspect are run by naming the AspectJ runtime agent aspectjweaver.jar at launch. This is shown in the command line example above, and an example is provided in launcher.bat in the tracing-aspectj module.

The runtime configuration file aop.xml has references to these aspects. Uncomment one or both of these in aop.xml to run with this logging enabled. Following is an example logging run with the verbose weaver option specified. Note the verbose output of class names, build times, etc. The logging statements too are quite verbose stating the name of the join point, the point cut, and various signatures and arguments.

Example Logging in Tuscany

```
[INFO] Configured Artifact: org.aspectj:aspectjweaver:1.6.1:jar
[INFO] Copying aspectjweaver-1.6.1.jar to E:\t\java\sca\modules\tracing-aspectj\target\dependency\aspectjweaver-
1.6.1.jar
[AppClassLoader@92e78c] info AspectJ Weaver Version 1.6.1 built on Thursday Jul 3, 2008 at 18:35:41 GMT
[AppClassLoader@92e78c] info register classloader sun.misc.Launcher$AppClassLoader@92e78c
[AppClassLoader@92e78c] info using configuration /E:/t/java/sca/modules/tracing-aspectj/target/classes/META-INF
/aop.xml
[AppClassLoader@92e78c] info register aspect org.apache.tuscany.sca.aspectj.LoggingAspect
[AppClassLoader@92e78c] info define aspect org.apache.tuscany.sca.aspectj.UserTimingAspect

-------------------------------------------------------
 T E S T S
-------------------------------------------------------
Logging Before anyMethodCall jp.getSignature=SCANodeFactory org.apache.tuscany.sca.node.SCANodeFactory.
newInstance()
Logging Before anyConstructor jp.getSignature=org.apache.tuscany.sca.extensibility.ServiceDiscovery()
Logging Before anyConstructor jp.getSignature=org.apache.tuscany.sca.extensibility.ServiceDiscovery.1
(ServiceDiscovery, Class)
Logging Before anyConstructor jp.getArgs=[org.apache.tuscany.sca.extensibility.ServiceDiscovery@ceb6dd,
   class org.apache.tuscany.sca.node.SCANodeFactory]
Logging Before anyMethodCall jp.getSignature=ServiceDiscoverer org.apache.tuscany.sca.extensibility.
ServiceDiscovery.getServiceDiscoverer()
Logging Before anyConstructor jp.getSignature=org.apache.tuscany.sca.extensibility.
ContextClassLoaderServiceDiscoverer()
Logging AfterReturning anyMethodCall jp=call(ServiceDiscoverer org.apache.tuscany.sca.extensibility.
ServiceDiscovery.getServiceDiscoverer()),
   result=org.apache.tuscany.sca.extensibility.ContextClassLoaderServiceDiscoverer@17574b9
Logging Before anyMethodCall jp.getSignature=Set org.apache.tuscany.sca.extensibility.ServiceDiscoverer.discover
(String, boolean)
Logging Before anyMethodCall jp.getArgs=[org.apache.tuscany.sca.node.SCANodeFactory, true]
```

These aspects provide static point cuts and advice. That is, the various methods, constructors, and exception points are constrained by the source code of the aspects. You can change these items via editing LoggingAspect or SimpleTracingAspect and rerunning the application.

Timing

Timing of various calls is also available via the AOP logging and tracing facility in Tuscany. The apsect UserTimingAspect is provided to time certain point cuts. As shown above, this AOP timing feature can be used for timing metrics for competitive analysis of performance gauges. The output of the timing is shown here, as a Logger.info method is called and timed.

Tuscany AOP Timing Output

```
Running org.apache.tuscany.sca.aspectj.TracingTestCase
Timing Around timedSection jp=call(void java.util.logging.Logger.info(String))
Oct 21, 2008 9:26:36 AM org.apache.tuscany.sca.aspectj.TracingTestCase info_aroundBody0
Timing Around timedSection Roundtrip is 32ms for jp.getSignature=void java.util.logging.Logger.info(String)
```

Unlike LoggingAspect and SimpleTracingAspect, the pointcut for UserTimingAspect is settable at runtime via the aop.xml runtime configuration file. Look for the UserTimingAspect name, and note the expression in the pointcut element. You may change the name of expression to any valid AspectJ pointcut expression. Here calls to any Logger.info methods, regardless of arguments or return types are logged. For instance you may change this to "call(* calculator.CalculatorServiceImpl.multiply(..))" to time calls in your SCA application.

Dynamic PointCut for UserTimingAspect

```
        <concrete-aspect name="org.apache.tuscany.sca.aspectj.UserTimingAspect"
           extends="org.apache.tuscany.sca.aspectj.TimingAspect"
           precedence="org.apache.tuscany.sca.aspectj.UserTimingAspect, *">
           <pointcut name="timedCall"
              expression="call(* java.util.logging.Logger.info(..))"/>
        </concrete-aspect>
```

Logging and Tracing in Alternate Containers

So far this article discussed logging and tracing for Tuscany applications. Often Tuscany applications are run standalone in a Java 2 SE environment. You can also turn on logging and tracing in various application servers that support Tuscany SCA applications.

Apache Tomcat Example

Show Apache Tomcat J2EE application server plus Tuscany bundled SCA application.

WebSphere Example

Show WebSphere 7.0.0.1 plus SOA Feature Pack based on Tuscany output.