JavaLogging

Cocoon-Style Logging in Non-Cocoon Java Classes

This page explains how to implement Cocoon-style logging, in a Java class that is not inherited from a Cocoon or Avalon component class. The class is however used within a Cocoon application.

A typical use for this might be to implement logging in a Java Bean used in a Cocoon application.

How to Implement Logging

In Cocoon, like in Perl, there's more than one way to do it. In the case of logging the following solutions are possible:

1.Extend `AbstractLogEnabled` - The simplest solution, but it only
works if your class doesn't already extend some other class.
1.Implement `LogEnabled` - A little more work, but more flexible.

Solution One: Extend `AbstractLogEnabled`

In your class you extend the `AbstractLogEnabled` class.

To write log messages you simply call the appropriate log method using the `Logger` provided by the `getLogger()` method, which is available from the parent class `AbstractLogEnabled`.

```
import org.apache.avalon.framework.logger.AbstractLogEnabled;

public class SomeClass extends AbstractLogEnabled {

public void someMethod() {
    ...
    getLogger().debug( "Hello, log. It worked!" );
    getLogger().info(  "Hello, log. Here is info" );
    getLogger().error( "Hello, log. Here is an error" );
    //..etc.
    ...
  }
}
```

This works fine, provided you class doesn't already extend some other class.

Solution Two: Implement `LogEnabled`

Have your class implement the `LogEnabled` interface. A typical class might do the following:

```
import org.apache.avalon.framework.logger.Logger;
import org.apache.avalon.framework.logger.LogEnabled;

class SomeBean extends SomeOtherBean implements LogEnabled {
  ..
  // The LogEnabled interface is one method: enableLogging
  private Logger logger;
  public void enableLogging( Logger logger ) {
    this.logger = logger;
  }

  // Example method that writes to the log
  public void setThing( String thing ) {
    logger.debug( "SomeBean: thing = " + thing );
    ...
  }
}
```

Note that in this case you use the `logger` directly and don't need to use the `getLogger()` accessor method. Note that a *maintainance aware* developer would probably implement their own `getLogger()`.

Enabling Logging

For both of these solutions you must enable logging by calling the `enableLogging()` method on the class. This requires that you have a valid `Logger` object to provide to `enableLogging()`.

Generally you can get the Logger from a Cocoon component class. In my application I called `enableLogging()` from my Cocoon action class, which extends `AbstractXMLFormAction`:

```
   ...
   SomeClass myClass = new SomeClass();
   myClass.enableLogging( getLogger() );
   myClass.someMethod(); // Writes some log messages
   ...
```

Note that many of the Cocoon classes extend Avalon Component classes.

Remember to call `enableLogging()` *before* you call any methods that write log messages. In Cocoon application it is not always obvious when to call `enableLogging()` as the creation and initialization of many of your classes will be handled automatically by Avalon, one of the Cocoon sub-systems.

The `ContainerUtil` class provides a convenient way to enable logging:

```
ContainerUtil.enableLogging(object, logger);
```

This method takes care of the following issues:

- The logger is only passed to the object if it implements `LogEnabled`.
- If the logger is `null`, an exception is thrown.

Links to Avalon Documentation

To be absolutely sure that you are writing solid code, you'll need a basic understanding of the Avalon component life-cycle. This is a big subject and beyond the scope of this page. You can read more at

The Avalon Logkit, which is used by Cocoon:

- http://jakarta.apache.org/avalon/logkit/whitepaper.html

The Avalon Component Lifecycle:

- http://jakarta.apache.org/avalon/framework/reference-the-lifecycle.html

If you're still curious, here is a link to an excellent white paper explaining development using avalon:

- http://jakarta.apache.org/avalon/developing/index.html

..and that's all there is to it.

Many thanks to Marcus Crafter, Judith Andres and KonstantinPiroumian
for their helpful input.

– *AlanHodgkinson*