Customising ProfDog

by

Table of contents

1. ILoggingEventInterpretor

The most important way in which ProfDog can be customised is by providing an implementation of ILoggingEventInterpretor.

An instance of this is referenced by both the LightweightAppender and the ScopingAppender. The former uses it just to determine (as efficiently as possible) if a logging event message has enough information that its ownership (at least) can be established. The latter uses it to interpret events as either the start of a scope, the finish of a scope, or merely logging messages to be nested within a scope.

The default implementation - StandardLoggingEventInterpretor - uses the following rules:

- scoped://start:owner@scopeName@context

  represents the start of a new scope
- scoped://finish:owner@scopeName

  represents the finish point of a scope
- scoped://error:owner@scopeName

  also represents the finish point of a scope, but where the scope finished as a result of an error (eg in a catch block of a try ... catch)
- scoped://owner@msg

  to log as a regular message
- scoped://done:owner

  can be used to force finish the outermost scope. This is not encouraged in normal usage, but is used by the ProfDog Browser to close scopes

Events whose messages that do not follow this format are ignored.

In order to provide a breakdown of timings, all scopes are additionally categorised. As can be seen, the StandardLoggingEventInterpretor does NOT pick up the category from the log message; instead it simply uses the LoggerName property of the log4j event itself. This makes a lot of sense to us: the original name for log4j Loggers was after all Category. Of course, a different implementation of this interface could pick up the category in a different way.

2. ITransportMechanism

While UDP is recommended as the preferred mechanism for transmitting events, the transport mechanism is in fact pluggable. Both the Lightweight appender and the Collator use an instance of transport mechanism. A single implementation -

UDPTransportMechanism is provided by default.

3. IMarshalling

The job of the IMarshalling<V> interface is to abstract out how to pack an event such that it can be transported. Although independent of ITransportMechanism, typically these would be written as a pair. For example, the provided SimpleMarshalling implements IMarshalling<byte[]> so that it can pack events into an array of bytes for transmission as UDP.