Using Custom Log4J2 Appender

Geode uses log4j2 for logging. In this post we are going to see how to define a custom appender.

Step-by-step guide

The caveats around configuring log4j for Geode are discussed in the docs. Chief among that is the JMX client should continue to get alert messages when they show up in the logs.

1. To ensure that GemFire's alert messaging continues to work, start with the log4j2.xml that ships with Geode. The file is at: `$GEODE_HOME/defaultConfigs/log4j2.xml`
2. Create a custom appender. Use annotations to mark it as a log4j2 plugin.

```
package log4j.extensions;

import org.apache.logging.log4j.core.Filter;
import org.apache.logging.log4j.core.Layout;
import org.apache.logging.log4j.core.LogEvent;
import org.apache.logging.log4j.core.appender.AbstractAppender;
import org.apache.logging.log4j.core.config.plugins.Plugin;
import org.apache.logging.log4j.core.config.plugins.PluginAttribute;
import org.apache.logging.log4j.core.config.plugins.PluginElement;
import org.apache.logging.log4j.core.config.plugins.PluginFactory;
import org.apache.logging.log4j.core.layout.PatternLayout;

import java.io.Serializable;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.concurrent.TimeUnit;



@Plugin(name = "Basic", category = "Core", elementType = "appender", printObject = true)
public class BasicAppender extends AbstractAppender {

  private static volatile BasicAppender instance;

  public BasicAppender(final String name, final Filter filter, final Layout<? extends Serializable> layout)
{
    super(name, filter, layout);
  }

  @PluginFactory
  public static BasicAppender createAppender(@PluginAttribute("name") String name,
  @PluginAttribute("ignoreExceptions") boolean ignoreExceptions,
  @PluginElement("Layout") Layout layout,
  @PluginElement("Filters") Filter filter) {
    if (layout == null) {
      layout = PatternLayout.createDefaultLayout();
  }

    instance = new BasicAppender(name, filter, layout);
 return instance;
 }

 public static BasicAppender getInstance() {
   return instance;
 }

 @Override
 public void append(final LogEvent event) {
    // do something custom
 }
}
```

3. To make sure that log4j picks up this plugin, add the package name of your plugin to the "packages" attribute of the "Configuration" element in log4j2.xml

```
 <Configuration status="FATAL" shutdownHook="disable" packages="com.gemstone.gemfire.internal.logging.log4j,log4j.
extensions">
```

4. Define the appender in the "Appenders" element. (name attribute of the "Plugin" annotation)

```
<Basic name="CUSTOM"/>
```

5. Add a logger that uses this appender. The name here should be the package/class for which you want the custom appender.

```
<Logger name="my.company" level="debug" additivity="true">
<AppenderRef ref="CUSTOM"/>
</Logger>
```

6. In your application, create a logger using that name and start logging to it!

```
static final Logger logger = LogManager.getLogger(Main.class);
```

Related articles

- Using Custom Log4J2 Appender