An AspectJ Logging Framework for Java Web Applications

by

James J. Curtis

A Project Report Submitted
in
Partial Fulfillment of the
Requirements for the Degree of
Master of Science
in
Computer Science

Supervised by

Jeremy Brown

Department of Computer Science

B. Thomas Golisano College of Computing and Information Sciences
Rochester Institute of Technology
Rochester, New York

May 2016
Abstract

An AspectJ Logging Framework for Java Web Applications

James J. Curtis

Supervising Professor: Jeremy Brown

Web applications are one of the largest areas of current software development. Companies such as Microsoft and Oracle have developed entire frameworks to facilitate the development of small and large scale web applications. Although there are countless tools, libraries, and plugins available to assist in web application development, many developers and companies are left with minimal assistance in application monitoring. Of course some tools exist, such as Log4J, Pingdom, etc. but these generally do not integrate with the application directly, but rather integrate through other interfaces such as the server. Many developers have found these tools insufficient and have built custom, application specific monitoring frameworks. For individual developers or startup companies, this can be extremely cost prohibitive, and even for large corporations the development and maintenance costs can be a significant part of the application upkeep. As such, I propose an AspectJ based logging framework for Spring MVC web applications. This will be a powerful logging framework which requires minimal configuration and integrates directly with the application at a code level in order to provide superior logging ability while significantly reducing development time.
Contents

Abstract ................................................................. ii

1 Introduction ............................................................ 1
  1.1 Background ....................................................... 1
  1.2 Problem Definition ............................................ 2
  1.3 Approach ........................................................ 3

2 Design ..................................................................... 4
  2.1 Framework ......................................................... 4
  2.2 Configuration ...................................................... 5

3 Implementation ......................................................... 6
  3.1 Aspects .............................................................. 6
  3.2 Database ........................................................... 7
  3.3 Website ............................................................. 8

4 Analysis ..................................................................... 10
  4.1 Survey .............................................................. 10

5 Conclusions .............................................................. 12
  5.1 Current Status ..................................................... 12
  5.2 Future Work ........................................................ 12
  5.3 Lessons Learned ................................................ 13
  5.4 Final Remarks ..................................................... 14

Bibliography ............................................................... 15

A SQL Create Schema .................................................... 16

B Properties Specifications .............................................. 18
  B.1 aop.xml ............................................................... 18
Chapter 1

Introduction

1.1 Background

Aspect Oriented Programming (AOP), is a programming paradigm designed to isolate crosscutting concerns in order to reduce repetitive code and increase the overall modularity of a codebase [4]. A common example of a crosscutting concern is application logging, given that it often occurs throughout an application with nearly the same code every time, and yet can rarely be isolated into a single module. Figure 1.1 shows an example of a single concern, socket creation, which is localized to a single module in the codebase. In contrast, Figure 1.2 shows how a crosscutting concern such as application logging is typically spread throughout an application.

Figure 1.1: Socket Creation in Apache Tomcat [5]
The best known implementation of aspect oriented programming is AspectJ [6], an extension of Java which introduces join points as a way for aspects to integrate with other code. A join point is a predefined location in the program execution which a programmer can expose in order to execute code, referred to as advice, in an aspect. Join points in AspectJ can be placed just before or after a function is called or executes, after an exception is thrown, and around the execution of a function. A join point around a executed execution takes control of the program flow, and can choose whether or not the function should be executed. Join points match functions based on regular expressions, so it is simple to match every function in a module or class, functions annotated with a certain annotation, functions that start with ‘get’, or any combination of these and more.

1.2 Problem Definition

Software maintenance and upkeep is a huge problem in modern computing. According to Kaur and Singh, around 60% of effort in software development is spent on maintenance [3]. Web applications must be monitored and maintained in order to achieve certain standards of user satisfaction. However, publicly available tools are often insufficient to monitor large enterprise applications, and can be too time consuming or cost-prohibitive for small
companies or individual developers to implement. Large corporations can remedy this by building internal tools to track errors, usage, and other statistics, however this often requires a significant time investment to set up and maintain. It also, depending on how diverse a company’s suite of applications is, may not be compatible with all the applications they manage. The goal of this project will be to reduce the cost of web application maintenance by reducing the overall time spent on creating, maintaining, and working with internal monitoring tools.

1.3 Approach

In order to reduce the overall cost of web application maintenance, this project will utilize AspectJ to build a web application monitoring framework targeting Spring MVC web applications. It will aim to require minimal configuration and setup time, offer support for a wide variety of web applications, and require little to no time spent by developers to utilize. The framework will consist of both a mechanism to monitor the application, and an application to view and analyze the logged information. It will attempt to provide developers with as much information as possible without significantly impacting the performance or behavior of the application.
Chapter 2

Design

2.1 Framework

This project involves three distinct parts: an aspect library, a database, and a web application. The aspect library is a standalone jar file which can be imported into any Spring web application. Then, after some simple configuration, the aspects integrate directly with the existing codebase and begin logging information to the database. Several aspects respond to annotations added by developers. For example, a timing aspect will record the time taken by a method to execute when the method is annotated as ‘@Timed’. Other aspects require no work from the programmer at all, such as the controller aspect which records any time a controller endpoint executes. These aspects will be discussed further in Chapter 3.

The database component of this framework is very flexible. Since the aspect library and the website will be compatible with a variety of databases, any standard SQL database will suffice.

The third component of the project is a web application designed to display and analyze the data collected by the aspects in an easy to read format. The site offers several filtering and graphing tools depending on the data set, and also requires minimal setup to start running. Though this will be discussed more in Chapter 5, in the future the site and database would ideally be offered as a cloud based solution for smaller businesses which do not wish to build the infrastructure internally.
2.2 Configuration

As discussed previously, this framework is designed to require minimal configuration in order to reduce time spent on setup and maintenance. As such, this project will utilize load-time weaving, a mechanism which delegates weaving in the aspects to the classloader rather than the compiler[6]. This is possible since the default Spring classloader supports load-time weaving, provided that the AspectJ jar files are on the classpath. Load-time weaving allows for significantly easier setup since build scripts and deployment structures will not require any modification.

The framework can be configured in five simple steps.

1. Add an aop.xml file to \texttt{/source/META-INF/aop.xml} an example is provided in Appendix B)

2. Add the line \texttt{<context:load-time-weaver/>} to the spring web.xml file

3. Add a properties file (dbconnect.properties) containing the database connection information (an example is provided in Appendix B)

4. Append the following two lines to the Tomcat VM arguments (replacing the bracketed text as necessary):

   \begin{verbatim}
   -javaagent:${workspace_loc}/.metadata/.plugins/org.eclipse.wst.server.core/tmp0/wtpwebapps/[project_name]/WEB-INF/lib/spring-instrument-3.0.5.RELEASE.jar
   -DapplicationProperties="${workspace_loc}/[project_name]/[path_to_your_properties]/dbconnect.properties;
   \end{verbatim}

5. Ensure that the dependencies listed in Appendix C are on the classpath.

Once this setup is completed, the aspects can be turned on and off by commenting out the line pertaining to each aspect in the AOP.xml file, allowing for greater customization depending on the developer’s needs.
Chapter 3

Implementation

3.1 Aspects

Five aspects were created for this project to facilitate the logging mechanisms. They are as follows: TimingAspect, LoggingAspect, ControllerAspect, RequestAspect, and ErrorAspect. The jar also provides two annotations @Timed and @Loggable for use by developers.

TimingAspect is designed to record the execution time of any function annotated with @Timed. It uses ‘around’ advice to record the time before the function executes, and after it returns. The difference between these two times is then saved to the database along with the fully qualified name of the function.

LoggingAspect will update the database any time a method annotated with @Loggable executes. ‘Before’ advice is used in order to ensure that the method is logged regardless of whether it returns or throws an exception. The aspect will save the fully qualified name of the method to the database, as well as the name and value of each argument passed in to the method (using the generic toString functionality of each argument).

ControllerAspect is similar to LoggingAspect in that it logs the same information about a method, however it will log a request to any Spring Controller endpoint, regardless of annotation. In other words, without any thought or effort from the developer, every controller request will be stored in order to track the usage of all endpoints.

RequestAspect does not actually attach to the developers code base, but instead acts on the ‘execute’ method in the Apache AbstractHttpClient which is used under the hood
for outgoing HTTP requests in Apache Tomcat. The aspect will log the full URI of the outgoing request to the database, as well as each header on the request.

ErrorAspect will capture any uncaught exceptions in the user codebase and save the type of exception, stacktrace, fully qualified name of the method where the exception occurred, and the arguments and values passed into the function. It will not catch the exception or alter the failure behavior in any way so as to not interfere inadvertently with the program design.

3.2 Database

The database used for this framework is a simple SQL database comprised of four tables. Though a MySQL database was used during development, the java.sql library offers a great deal of flexibility to automatically detect what connectors are available in the classpath, so in theory any standard SQL database should be compatible. The SQL used to create the schema is available in Appendix A.

In addition to logging the information from the aspects the database adds a timestamp and unique identifier to each row in order to organize the statements for the website. Method names up to 1000 characters are supported by default (though this can be adjusted), and stacktrace and argument fields are stored as TEXT to ensure they are never truncated. Figure 3.1 shows the full entity relationship diagram of the database.

Figure 3.1: Entity Relationship Diagram
3.3 Website

The website is designed as a lightweight web application which offers access to the logged data in an easy to read format. The homepage displays four tabs: Errors, Logs, Timings, and Requests. Each tab links to information logged by each aspect (Logged methods and Controller methods are combined as their information is identical). Clicking on each tab will display the logged information in a tabular format with options to filter the data by several metrics. An example of the Errors log is shown in figure 3.2.

<table>
<thead>
<tr>
<th>LogCentral</th>
<th>Errors</th>
<th>Logs</th>
<th>Timings</th>
<th>Requests</th>
</tr>
</thead>
<tbody>
<tr>
<td>Errors</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Id</td>
<td>Time</td>
<td>Method</td>
<td>Type</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>5/7/2016 8:01:22 PM</td>
<td>com.trgr.cobalt.trpcarswell.sample.SampleController.testMethod</td>
<td>class java.lang.NullPointerException</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>5/7/2016 8:01:29 PM</td>
<td>com.trgr.cobalt.trpcarswell.sample.SampleController.testMethod</td>
<td>class java.lang.NullPointerException</td>
<td></td>
</tr>
</tbody>
</table>

Figure 3.2: LogCentral: Errors Table

Certain tabs, such as Timings, provide a graph of the information available. For the timing information, the method execution time in milliseconds is graphed over time as shown in Figure 3.3. This is designed to help developers more easily identify possible scale or usage issues.

The website is implemented as an ASP.NET MVC 5 web application. Several JavaScript libraries are included such as Twitter Bootstrap (http://getbootstrap.com) for the front end look and feel, Chart.js (http://www.chartjs.org) to build the graphs, and
jQuery (https://jquery.com) throughout the site. On the back end, Entity Framework 6 is used to connect to the database which allows for databases to be easily interchanged as necessary based on the web.config file.
Chapter 4

Analysis

4.1 Survey

The main analysis of this project consisted of a survey in which enterprise and individual web developers were shown the framework, given a short demonstration, allowed time to experiment with the framework, and then asked a series of questions regarding their experience. The questions were split into three categories: General Information, Application Usefulness, and Aspect Usefulness. The general information questions discussed whether or not the framework would save them time, whether additional features were necessary, and how easy the framework was to configure. The application usefulness section discussed how useful the framework was for small or individual, and large or enterprise web applications. The aspect usefulness section discussed whether the developers would find each aspect useful and in what situations they might use them. In total, nine developers responded to the survey; the results are shown in Figure 4.1

The survey respondents also provided some helpful feedback. One mentioned that enterprises may prefer to integrate this framework with their own existing tools rather than LogCentral and may prefer a plaintext log over a database storage system. Others asked for more configuration options and some additional filtering features in LogCentral. By far the most common response however was that a link between the logs per request would be extremely helpful. For example, if a method is annotated with @Timed, there should be a way in the logs to see not only the information about that methods timing, but also how it relates to an entry from the ControllerAspect and whether any requests were logged by
the RequestAspect further down in the call stack. Nearly all the respondents felt that this would be a useful feature.

Overall, the responses were very positive. All the respondents indicated that they would consider using this framework for personal web applications, and nearly all responded that it would be useful in an enterprise environment. The two that did not cited performance and scalability concerns that they did not feel could be resolved.

Figure 4.1: Survey results
Chapter 5

Conclusions

5.1 Current Status

The project is currently ready for larger testing and preliminary deployment. The aspects function properly with multiple spring applications and do not appear to adversely impact the application during direct testing. The website is fully functional and provides the expected information, and the database seamlessly integrates the aspects with the website.

Based on the survey results, the amount of configuration required by developers is sufficiently small, and the framework offers enough functionality to be useful in their daily work. This project can certainly be considered a successful experiment in improving web application logging functionality with the current implementation.

5.2 Future Work

There are several additional features which would greatly benefit this framework. First, an option for compile-time weaving would make the framework far more efficient for larger web applications. Load-time weaving works very well for development environments and light-weight applications, however it does incur a performance penalty which may be noticed in larger applications. Adding the option for compile time weaving will greatly benefit larger applications and make this framework a much more attractive option for production environments.

Another feature which the survey participants indicated they would want to see is a mechanism which can link the information logged by aspects together per request. For
example, a request would be received by the application, a unique identifier would be generated, and then any information logged by the aspects would be linked to that identifier so that developers can more easily see patterns with issues that may arise. This may begin to incur a larger performance impact on the overall application however, so more investigation is necessary in this area.

In terms of the website, there are two main areas of future work that would benefit the framework. First, the website would benefit from some general polish in the design and interface. Presenting a cleaner and better looking website would add a great deal to the overall project. Second, enterprise developers may prefer to utilize their own existing interface, or an external framework such as ELK (https://www.elastic.co/products) rather than LogCentral to view and analyze the logged data. This project would certainly be more appealing if it offered support for these other frameworks, and could do so by adding an option to write logs to a text or csv file.

5.3 Lessons Learned

The most important lesson learned during this project is that while AspectJ works seamlessly with desktop applications, integrating it with web applications is significantly more complicated. The initial vision for this project was that it would function with any Java web application, however due to variety of ways in which web applications can be compiled, the focus was restricted to just Spring applications.

A second lesson learned is that ease of configuration can easily be confused with a lack of configuration options. This project focused heavily on reducing the amount of configuration necessary for developers to integrate the aspect library with their applications, however the survey respondents seemed to want more flexibility in data output and user interface. It seems that more configuration options should be added, though without sacrificing too much of the simplicity on which this project is built.
5.4 Final Remarks

Logging, while a simple concept at first glance, is truly a complex area of work in which there is no perfect solution. AspectJ goes a long way in facilitating more efficient ways to handle logging in Java, however many other languages have little or no support for aspect oriented programming. This project goes a long way toward improving web application logging, however there are still many opportunities to expand on this project into other languages, other types of applications, and even other programming paradigms.
Bibliography


Appendix A

SQL Create Schema

CREATE DATABASE IF NOT EXISTS 'aspectlogs';
USE 'aspectlogs';

--
-- Table structure for table 'errors'
--

DROP TABLE IF EXISTS 'errors';

CREATE TABLE `errors` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `timestamp` timestamp NOT NULL
    DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
  `type` varchar(100) DEFAULT NULL,
  `method` varchar(1000) DEFAULT NULL,
  `args` text,
  `stacktrace` text,
  PRIMARY KEY (`id`)
);

--
-- Table structure for table 'logs'
--

DROP TABLE IF EXISTS 'logs';

CREATE TABLE `logs` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
```sql
-- Table structure for table `requests`

DROP TABLE IF EXISTS `requests`;

CREATE TABLE `requests` (  
    `id` int(11) NOT NULL AUTO_INCREMENT,  
    `timestamp` timestamp NOT NULL  
        DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,  
    `uri` varchar(1000) DEFAULT NULL,  
    `headers` text,  
    PRIMARY KEY (`id`)  
) ;

-- Table structure for table `timings`

DROP TABLE IF EXISTS `timings`;

CREATE TABLE `timings` (  
    `id` int(11) NOT NULL AUTO_INCREMENT,  
    `timestamp` datetime NOT NULL  
        DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,  
    `method` varchar(1000) DEFAULT NULL,  
    `duration` int(11) DEFAULT NULL,  
    PRIMARY KEY (`id`)  
) ;
```

Appendix B

Properties Specifications

B.1 aop.xml

<!DOCTYPE aspectj PUBLIC
 "-//AspectJ//DTD//EN"
 "http://www.eclipse.org/aspectj/dtd/aspectj.dtd">

<weaver>
  <include within="[path.to.your.project]"/>
  <include within="org.apache.http.impl.client.AbstractHttpClient"/>
</weaver>

<aspects>
  <aspect name="com.jcurtis.msproject.aspects.ControllerAspect" />
  <aspect name="com.jcurtis.msproject.aspects.ErrorAspect" />
  <aspect name="com.jcurtis.msproject.aspects.LoggingAspect" />
  <aspect name="com.jcurtis.msproject.aspects.RequestAspect" />
  <aspect name="com.jcurtis.msproject.aspects.TimingAspect" />
</aspects>

B.2 dbconnect.properties

jcurtis.aspectlogs.connection= jdbc:mysql://localhost:3306/aspectlogs
jcurtis.aspectlogs.username=msproject
jcurtis.aspectlogs.password=msproject123
jcurtis.aspectlogs.dbdriver=com.mysql.jdbc.Driver
Appendix C

Required Dependencies

<!-- DB Connector, mysql is provided as an example -->
<dependency org="mysql" name="mysql-connector-java"
    rev="5.1.38"/>
<dependency org="commons-lang" name="commons-lang" rev="2.6"/>
<dependency org="commons-httpclient" name="commons-httpclient"
    rev="3.1"/>
<dependency org="org.apache.commons" name="commons-lang3" rev="3.1"/>
<dependency org="org.apache.httpcomponents" name="httpcore"
    rev="4.1"/>
<dependency org="org.apache.httpcomponents" name="httpclient"
    rev="4.1.1"/>
<dependency org="org.aspectj" name="aspectjrt" rev="1.8.6"/>
<dependency org="org.aspectj" name="aspectjweaver" rev="1.8.6"/>
<dependency org="org.springframework" name="spring-aop"
    rev="3.0.5.RELEASE"/>
<dependency org="org.springframework" name="spring-aspects"
    rev="3.0.5.RELEASE"/>
<dependency org="org.springframework" name="spring-context"
    rev="3.0.5.RELEASE"/>
<dependency org="org.springframework" name="spring-core"
    rev="3.0.5.RELEASE"/>
<dependency org="org.springframework" name="spring-instrument"
    rev="3.0.5.RELEASE"/>
<dependency org="org.springframework" name="spring-instrument-tomcat"
    rev="3.0.5.RELEASE"/>
<dependency org="org.springframework" name="spring-web"
    rev="3.0.5.RELEASE"/>