java 日志框架总结

在项目开发过程中，我们可以通过 debug 查找问题。而在线上环境我们查找问题只能通过打印日志的方式查找问题。因此对于一个项目而言，日志记录是一个非常重要的问题。因此，如何选择一个合适的日志记录框架也非常重要。

在Java开发中，常用的日志记录框架有**JDKLog**、**Log4J**、**LogBack**、**SLF4J**、**SLF4J**。这些日志记录框架各有各的特点，各有各的应用场景。了解这些框架的特点及应用场景，有利于我们做技术选型的时候做出正确的判断。

JDKLog：日志小刀

JDKLog是JDK官方提供的一个记录日志的方式，直接在JDK中就可以使用。

```
import java.util.logging.Logger;

/****
 ** JDKLog Demo
 **/
public class JDKLog
{
    public static void main( String[] args )
    {
        Logger logger = Logger.getLogger("JDKLog");
        logger.info("Hello World.");
    }
}
```

JDKLog 的有点是使用非常简单，直接在 JDK 中就可以使用。但 JDKLog 功能比较太过于简单，不支持占位符显示，拓展性比较差，所以现在用的人也很少。

Log4J：日志大炮

Log4J 是 Apache 的一个日志开源框架，有多个分级（DEBUG/INFO/WARN/ERROR）记录级别，可以很好地将不同日志级别的日志分开记录，极大地方便了日志的查看。

Log4J 有 1.X 版本和 2.X 版本，现在官方推荐使用 2.X 版本，2.X 版本在架构上进行了一些升级，配置文件也发生了一些变化。但好在官方的配置说明文档非常清楚，通过查阅文档能解决大部分的问题。

使用 Log4J 框架首先需要引入依赖的包：

```
<!-- Log4J -->
<dependency>
  <groupId>org.apache.logging.log4j</groupId>
  <artifactId>log4j-api</artifactId>
  <version>2.6.2</version>
</dependency>
<dependency>
  <groupId>org.apache.logging.log4j</groupId>
  <artifactId>log4j-core</artifactId>
  <version>2.6.2</version>
</dependency>
```

增加配置文件 log4j2.xml 放在 resource 目录下：

```
<?xml version="1.0" encoding="UTF-8"?>
<Configuration status="WARN">
    <Appenders>
        <Console name="Console" target="SYSTEM_OUT">
            <PatternLayout pattern="%d{HH:mm:ss.SSS} [%t] %-5level %logger{36} -
%msg%n"/>
        </Console>
    </Appenders>
    <Loggers>
        <Root level="info">
            <AppenderRef ref="Console"/>
        </Root>
    </Loggers>
</Configuration>
```

其中节点的 level 属性表示输出的最低级别。

最后编写一个测试类：

```
import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;

/****
 ** Log4J Demo
 **/
public class Log4jLog {
    public static void main(String args[]) {
        Logger logger = LogManager.getLogger(Log4jLog.class);
        logger.debug("Debug Level");
        logger.info("Info Level");
        logger.warn("Warn Level");
        logger.error("Error Level");
```

```
    }
}
```

运行测试类输出结果：

```
10:16:08.279 [main] INFO  com.chanshuyi.Log4jLog - Info Level
10:16:08.280 [main] WARN  com.chanshuyi.Log4jLog - Warn Level
10:16:08.280 [main] ERROR com.chanshuyi.Log4jLog - Error Level
```

如果没有配置 log4j2.xml 配置文件，那么LOG4J将自动启用类似于下面的的配置文件：

```
<?xml version="1.0" encoding="UTF-8"?>
<Configuration status="WARN">
    <Appenders>
        <Console name="Console" target="SYSTEM_OUT">
            <PatternLayout pattern="%d{HH:mm:ss.SSS} [%t] %-5level %logger{36} -
%msg%n"/>
        </Console>
    </Appenders>
    <Loggers>
        <Root level="error">
            <AppenderRef ref="Console"/>
        </Root>
    </Loggers>
</Configuration>
```

使用默认配置文件的输出结果：

```
ERROR StatusLogger No log4j2 configuration file found. Using default configuration:
logging only errors to the console.
11:40:07.377 [main] ERROR com.chanshuyi.Log4jLog - Error Level
```

从上面的使用步骤可以看出 Log4J 的使用稍微复杂一些，但是条理还是很清晰的。而且因为 Log4J 有多个分级（DEBUG/INFO/WARN/ERROR）记录级别，所以可以很好地记录不同业务问题。因为这些优点，所以在几年前几乎所有人都使用 Log4J 作为日志记录框架，群众基础可谓非常深厚。

但 Log4J 本身也存在一些缺点，比如不支持使用占位符，不利于代码阅读等缺点。但是相比起 JDKLog，Log4J 可以说是非常好的日志记录框架了。

LogBack：日志火箭

LogBack 其实可以说是 Log4J 的进化版，因为它们两个都是同一个人（Ceki Gülcü）设计的开源日志组件。LogBack 除了具备 Log4j 的所有优点之外，还解决了 Log4J 不能使用占位符的问题。

使用 LogBack 需要首先引入依赖：

```
<!-- LogBack -->
<dependency>
  <groupId>ch.qos.logback</groupId>
  <artifactId>logback-classic</artifactId>
  <version>1.1.7</version>
</dependency>
```

配置 logback.xml 配置文件：

```xml
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
    <appender name="STDOUT" class="ch.qos.logback.core.ConsoleAppender">
        <layout class="ch.qos.logback.classic.PatternLayout">
            <Pattern>%d{HH:mm:ss.SSS} [%thread] %-5level %logger{36} - %msg%n</Pattern>
        </layout>
    </appender>
    <logger name="com.chanshuyi" level="TRACE"/>

    <root level="debug">
        <appender-ref ref="STDOUT" />
    </root>
</configuration>
```

LogBack 的日志级别区分可以细分到类或者包，这样就可以使日志记录变得更加灵活。之后在类文件中引入Logger类，并进行日志记录：

```java
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

/****
 ** LogBack Demo
 **/
public class LogBack {
    static final Logger logger = LoggerFactory.getLogger(LogBack.class);
    public static void main(String[] args) {
        logger.trace("Trace Level.");
        logger.debug("Debug Level.");
        logger.info("Info Level.");
        logger.warn("Warn Level.");
        logger.error("Error Level.");
    }
}
```

输出结果：

```
14:34:45.747 [main] TRACE com.chanshuyi.LogBack - Trace Level.
14:34:45.749 [main] DEBUG com.chanshuyi.LogBack - Debug Level.
14:34:45.749 [main] INFO  com.chanshuyi.LogBack - Info Level.
14:34:45.749 [main] WARN  com.chanshuyi.LogBack - Warn Level.
14:34:45.749 [main] ERROR com.chanshuyi.LogBack - Error Level.
```

LogBack 解决了 Log4J 不能使用占位符的问题，这使得阅读日志代码非常方便。除此之外，LogBack 比 Log4J 有更快的运行速度，更好的内部实现。并且 LogBack 内部集成了 SLF4J 可以更原生地实现一些日志记录的实现。

SLF4J：适配器

上面说了 JDKLog、Log4J、LogBack 这几个常用的日志记录框架，它们都有各自的优缺点，适合在不同的场景下使用。可能简单的项目直接用 JDKLog 就可以了，而复杂的项目需要用上 Log4J。

很多时候我们做项目都是从简单到复杂，也就是我们很可能一开始使用的是 JDKLog，之后业务复杂了需要使用 Log4J，这时候我们如何将原来写好的日志用新的日志框架输出呢？

一个最死板的方法就是一行行代码修改，把之前用 JDKLog 的日志代码全部修改成 Log4J 的日志接口。但是这种方式不仅效率低下，而且做的工作都是重复性的工作，这怎么能忍呢。

正式因为在实际的项目应用中，有时候可能会从一个日志框架切换到另外一个日志框架的需求，这时候往往需要在代码上进行很大的改动。为了避免切换日志组件时要改动代码，这时候一个叫做 SLF4J（Simple Logging Facade for Java，即Java简单日志记录接口集）的东西出现了。

SLF4J（Simple Logging Facade for Java，即Java简单日志记录接口集）是一个日志的接口规范，它对用户提供了统一的日志接口，屏蔽了不同日志组件的差异。这样我们在编写代码的时候只需要看 SLF4J 这个接口文档即可，不需要去理会不同日之框架的区别。而当我们需要更换日志组件的时候，我们只需要更换一个具体的日志组件Jar包就可以了。

而整合 SLF4J 和日志框架使用也是一件很简单的事情。

SLF4J+JDKLog

SLF4J + JDKLog 需要在 Maven 中导入以下依赖包：

```
<dependency>
  <groupId>org.slf4j</groupId>
  <artifactId>slf4j-api</artifactId>
  <version>1.7.21</version>
</dependency>
<dependency>
  <groupId>org.slf4j</groupId>
  <artifactId>slf4j-jdk14</artifactId>
  <version>1.7.21</version>
</dependency>
```

编写测试类：

```
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

/****
 ** SLF4J + JDKLog
 **/
public class Slf4jJDKLog {
    final static Logger logger = LoggerFactory.getLogger(Slf4jJDKLog.class);
    public static void main(String[] args) {
        logger.trace("Trace Level.");
        logger.info("Info Level.");
        logger.warn("Warn Level.");
        logger.error("Error Level.");
    }
}
```

输出结果：

```
七月 15, 2016 3:30:02 下午 com.chanshuyi.slf4j.Slf4jJDKLog main
信息: Info Level.
七月 15, 2016 3:30:02 下午 com.chanshuyi.slf4j.Slf4jJDKLog main
警告: Warn Level.
```

```
七月 15, 2016 3:30:02 下午 com.chanshuyi.slf4j.Slf4jJDKLog main
严重: Error Level.
```

SLF4J+LOG4J

需要依赖的 Jar 包：slf4j-api.jar、slf4j-412.jar、log4j.jar，导入Maven依赖：

```
<!-- 2.SLF4J + Log4J -->
<dependency>
  <groupId>org.slf4j</groupId>
  <artifactId>slf4j-api</artifactId>
  <version>1.7.21</version>
</dependency>
<dependency>
  <groupId>org.slf4j</groupId>
  <artifactId>slf4j-log4j12</artifactId>
  <version>1.7.21</version>
</dependency>
<dependency>
  <groupId>log4j</groupId>
  <artifactId>log4j</artifactId>
  <version>1.2.17</version>
</dependency>
```

配置 log4j.xml 文件：

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE log4j:configuration SYSTEM "log4j.dtd">

<log4j:configuration xmlns:log4j='http://jakarta.apache.org/log4j/' >

    <appender name="myConsole" class="org.apache.log4j.ConsoleAppender">
        <layout class="org.apache.log4j.PatternLayout">
            <param name="ConversionPattern"
                   value="[%d{dd HH:mm:ss,SSS\} %-5p] [%t] %c{2\} - %m%n" />
        </layout>
        <!--过滤器设置输出的级别-->
        <filter class="org.apache.log4j.varia.LevelRangeFilter">
            <param name="levelMin" value="debug" />
            <param name="levelMax" value="error" />
            <param name="AcceptOnMatch" value="true" />
        </filter>
    </appender>

    <!-- 根logger的设置-->
    <root>
        <priority value ="debug"/>
        <appender-ref ref="myConsole"/>
    </root>
</log4j:configuration>
```

我们还是用上面的代码，无需做改变，运行结果为：

```
[15 16:04:06,371 DEBUG] [main] slf4j.SLF4JLog - Debug Level.
[15 16:04:06,371 INFO ] [main] slf4j.SLF4JLog - Info Level.
[15 16:04:06,371 WARN ] [main] slf4j.SLF4JLog - Warn Level.
[15 16:04:06,371 ERROR] [main] slf4j.SLF4JLog - Error Level.
```

SLF4J+LogBack

导入依赖：

```
<dependency>
  <groupId>org.slf4j</groupId>
  <artifactId>slf4j-api</artifactId>
  <version>1.7.21</version>
</dependency>
<dependency>
  <groupId>ch.qos.logback</groupId>
  <artifactId>logback-classic</artifactId>
  <version>1.1.7</version>
</dependency>
<dependency>
  <groupId>ch.qos.logback</groupId>
  <artifactId>logback-core</artifactId>
  <version>1.1.7</version>
</dependency>
```

配置 logback.xml 文件：

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
    <appender name="STDOUT" class="ch.qos.logback.core.ConsoleAppender">
        <layout class="ch.qos.logback.classic.PatternLayout">
            <Pattern>%d{HH:mm:ss.SSS} [%thread] %-5level %logger{36} - %msg%n</Pattern>
        </layout>
    </appender>
    <logger name="com.chanshuyi" level="TRACE"/>

    <root level="warn">
        <appender-ref ref="STDOUT" />
    </root>
</configuration>
```

我们还是用上面的代码，无需做改变，运行结果为：

```
16:08:01.040 [main] TRACE com.chanshuyi.slf4j.SLF4JLog - Trace Level.
16:08:01.042 [main] DEBUG com.chanshuyi.slf4j.SLF4JLog - Debug Level.
16:08:01.043 [main] INFO  com.chanshuyi.slf4j.SLF4JLog - Info Level.
16:08:01.043 [main] WARN  com.chanshuyi.slf4j.SLF4JLog - Warn Level.
16:08:01.043 [main] ERROR com.chanshuyi.slf4j.SLF4JLog - Error Level.
```

分类: java

jason.bai
关注 - 6
粉丝 - 42
+加关注

0 0

« 上一篇：分布式事务解决方案
» 下一篇：如何从建表方面展示自己能力--转

posted @ 2017-11-28 17:32 jason.bai 阅读(484) 评论(0) 编辑 收藏