Springboot日志整合大集合

Preface

日志文件的重要性不言而喻，这里就不赘述。此篇文档主要针对如下几个问题，提供主流的解决方案说明及详细实现教程：

- Springboot主流日志框架及对比
- Springboot主流日志框架之一的使用
- AOP实现全局日志记录及全局捕获异常
- 日志储存方案及日志分文件储存
- 日志可视化分析解决方案-ELK Stack
- ELK Stack的简单实现
- Filebeat + ELK实现多文件日志
- 企业级日志规范

在实战中，日志部分三步走。（1）引入日志框架的依赖并实现基本的日志功能；（2）全局异常及日志，实现日志功能的高内聚低耦合；（3）ELK技术栈，实现日志的存储，搜索，管理，可视化，分析等功能。

该文档在持续优化与更新中.....

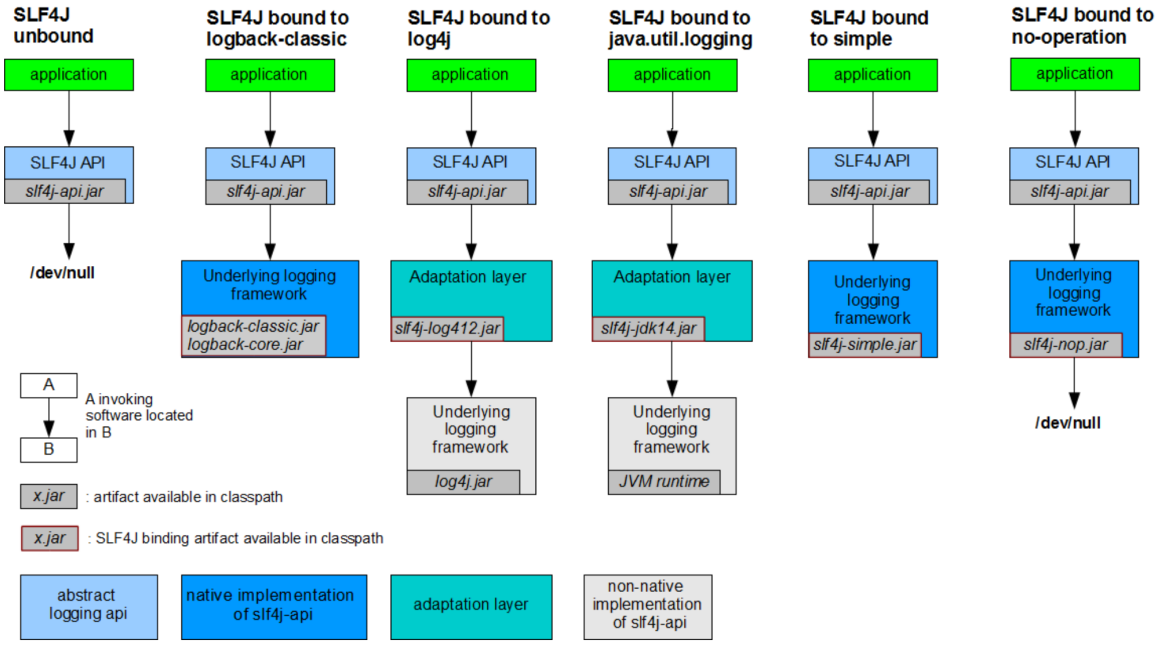
Springboot主流日志框架

1. SLF4J

http://www.slf4j.org/

> The Simple Logging Facade for Java (SLF4J) serves as a simple facade or abstraction for various logging frameworks, such as java.util.logging, logback and log4j. SLF4J allows the end-user to plug in the desired logging framework at *deployment* time. Note that SLF4J-enabling your library/application implies the addition of only a single mandatory dependency, namely slf4j-api-2.0.0-alpha2-SNAPSHOT.jar*.

以上是官方的介绍，可以知道，SLF4J并没有封装实现Log的功能，而是对主流Log框架进行了适配与抽象，其负责与开发者交互，而具体的功能由其后端的各个Log框架实现。由此，用户可以在不修改代码的情况下，切换使用的Log框架。SLF4J的架构图如下：



如果需要更换SLF4J使用的后端Log实现框架，只需要引入不同的jar包即可。例如：如果要从java.util.logging切换到log4j，只需要将slf4j-jdk14-1.7.28.jar替换为slf4j-log4j12-1.7.28.jar。

其使用手册：http://www.slf4j.org/manual.html

SLF4J怎么选择backbone?

实战：SLF4J + Log4J 实现Log功能

```
 1: import org.slf4j.Logger;
 2: import org.slf4j.LoggerFactory;
 3:
 4: public class Wombat {
 5:
 6:   final Logger logger = LoggerFactory.getLogger(Wombat.class);
 7:   Integer t;
 8:   Integer oldT;
 9:
10:   public void setTemperature(Integer temperature) {
11:
12:     oldT = t;
13:     t = temperature;
14:
15:     logger.debug("Temperature set to {}. Old temperature was {}.", t, oldT);
16:
```

```
17:     if(temperature.intValue() > 50) {
18:       logger.info("Temperature has risen above 50 degrees.");
19:     }
20:   }
21: }
```

Springboot默认使用Logback作为日志输出的后端框架，当你想要使用Log4j作为SLF4J的后端日志框架时，可能会有`multiple binding`的报警提示，该报警提示可能不会影响程序启动，但是可能引起Log只能输出到控制台，而无法根据log4j.properties文件的配置输出到文件。解决方法如下：

方法1：在pom.xml文件中引入SLF4J，并且将Logback设置为**Optional**。添加代码如下：

```
<dependency>
    <groupId>org.slf4j</groupId>
    <artifactId>slf4j-api</artifactId>
    <version>${slf4j.version}</version>
</dependency>
<dependency>
    <groupId>ch.qos.logback</groupId>
    <artifactId>logback-classic</artifactId>
    <version>${logback.version}</version>
    <scope>runtime</scope>
    <optional>true</optional>
</dependency>
```

方法2：进入project structure，定位到当前模组, 选择Dependencies选项卡，在其中找出logback的依赖项，并移除这些依赖。

方法3：在**pom.xml**文件中 ，找到需要引用Logback的Dependencies，然后添加Exclusions，如下：

```
<dependency>
    <groupId>your.company</groupId>
    <artifactId>libraryname</artifactId>
    <version>${theirlibrary.version}</version>
    <exclusions>
        <exclusion>
            <groupId>ch.qos.logback</groupId>
            <artifactId>logback-classic</artifactId>
        </exclusion>
    </exclusions>
</dependency>
```

SLF4J + Log4j实现日志功能：

AOP实现全局捕获异常及日志

面向切面的编程思想

ELK Stack 快速搭建

> https://www.elastic.co/cn/downloads/

并不需要编写代码，只需要进行简单的配置即可。

终止运行：`CTRL + C`

注意，路径中不能有空格，否则出现"无法找到或加载主类"的错误。

可能的问题1： 路径中不能有空格

可能问题2： classpath的引号问题

在开发B/S系统时，对于LOG，需要关注：

1. 日志信息的集中采集、存储、信息检索：在WEB集群节点越来越多的情况下，让开发及系统维护人员能很方便的查看日志信息
2. 日志信息的输出策略：日志信息输出全而不乱，便于跟踪和分析问题
3. 关键业务的日志输出：基于度量数据采集、数据核查、系统安全等方面的考虑，关键业务系统对输出的日志信息有特殊的要求，需要做针对性的设计

本文主要从这3个方面进行说明，重点说明日志输出的使用

日志的采集和存储
对于目前存储日志，主要存在2种方式：

1. 本地日志：直接存放在本地磁盘上
2. 远程日志：发往日志平台，用作数据分析和日志处理并展现。如用户访问行为的记录，异常日志，性能统计日志等。

日志工具的选择

推荐使用SLF4J（Simple Logging Facade for Java）作为日志的api，SLF4J是一个用于日志系统的简单Facade，允许最终用户在部署其应用时使用其所希望的日志系统。与使用apache commons-logging和直接使用log4j相比，SLF4J提供了一个名为参数化日志的高级特性，可以显著提高在配置为关闭日志的情况下的日志语句性能

可以看出SLF4J的方式一方面更简略易读，另一方面少了字符串拼接的开销,并且在日志级别达不到时（这里例子即为设置级别为debug以上），不会调用对象的toString方法。

日志输出级别（由高到低）

- **ERROR**：系统中发生了非常严重的问题，必须马上有人进行处理。没有系统可以忍受这个级别的问题的存在。比如：NPEs（空指针异常），数据库不可用，关键业务流程中断等等
- **WARN**：发生这个级别问题时，处理过程可以继续，但必须对这个问题给予额外关注。这个问题又可以细分成两种情况：一种是存在严重的问题但有应急措施（比如数据库不可用，使用Cache）；第二种是潜在问题及建议（ATTENTION），比如生产环境的应用运行在Development模式下、管理控制台没有密码保护等。系统可以允许这种错误的存在，但必须及时做跟踪检查
- **INFO**：重要的业务处理已经结束。在实际环境中，系统管理员或者高级用户要能理解INFO输出的信息并能很快的了解应用正在做什么。比如，一个和处理机票预订的系统，对每一张票要有且只有一条INFO信息描述 "[Who] booked ticket from [Where] to [Where]"。另外一种对INFO信息的定义是：记录显著改变应用状态的每一个action，比如：数据库更新、外部系统请求
- **DEBUG**：用于开发人员使用。将在TRACE章节中一起说明这个级别该输出什么信息
- **TRACE**：非常具体的信息，只能用于开发调试使用。部署到生产环境后，这个级别的信息只能保持很短的时间。这些信息只能临时存在，并将最终被关闭。要区分DEBUG和TRACE会比较困难，对一个在开发及测试完成后将被删除的LOG输出，可能会比较适合定义为TRACE级别

推荐使用debug，info，warn，error级别即可，对于不同的级别可以设置不同的输出路径,如debug，info输出到一个文件，warn，error输出到一个带error后缀的文件

Log对象的声明和初始化，仅以下代码是符合规范的

```
// （推荐）
private static final Logger logger = LoggerFactory.getLogger(Xxx.class);

private final Logger logger = LoggerFactory.getLogger(getClass());

private  static  final Logger logger =  LoggerFactory.getLogger("loggerName");

private static Logger logger = LoggerFactory.getLogger(Xxx.class);

protected final Logger logger = LoggerFactory.getLogger(getClass());

private Logger logger = LoggerFactory.getLogger(getClass());

protected Logger logger = LoggerFactory.getLogger(getClass());
```

不得使用System.out, System.err进行日志记录，请改使用logger.debug、logger.error

debug/info级别的信息，信息本身需要计算或合并的，必须加 isXxxEnabled() 判断在前，这样可以大大提高高并发下的效率。如：

```
if (logger.isDebugEnabled()) {
    logger.debug(test());
}

private String test(){
    int i = 0;
    while (i < 1000000) {
        i++;
    }

    return "";
}
```

如果不加 isXxxEnabled() 判断，test()在info级别下也会执行。

注意error和warn级别的区别，导致业务不正常服务的，用error级别；错误是预期会发生的，并且已经有了其他的处理流程，使用warn级别

正确的记录异常信息
记录异常信息是“记录所有信息”中的一个重要组成，但很多开发人员只是把logging当做处理异常的一种方式。他们通常返回缺省值，然后当做什么都没发生。 其他时候，他们先log异常信息，然后再抛出包装过的异常。如：

注意：捕获异常后不处理也不输出log是一种非常不负责任的行为，这会造成问题很难被定位，极大地提高debug成本！

重要方法入口，业务流程前后及处理的结果等，推荐记录log，并使用debug级别，如：

```
public String printDocument(Document doc, Mode mode) {

    log.debug("Entering printDocument(doc={}, mode={})", doc, mode);

    //Lengthy printing operation
    String id = "Id";

    log.debug("Leaving printDocument(): {}", id);

    return id;

}
```

因为对于非开发人员掌控的环境（无法做DEBUG），记录方法调用、入参、返回值的方式对于排查问题会有很大帮助。

- Log的内容一定要确保不会因为Log语句的问题而抛出异常造成中断
- 避免拖慢应用系统
  输出太多日志信息：通常每小时输出到disk的数据量达到几百MiB就已经是上限了不正确使用toString() 或字符串拼接方法。
- 日志信息中尽量包含数据和描述：`easy to read, easy to parse`

关键业务系统日志的要求

用户浏览日志
使用WEB服务器或应用服务器实现日志输出，关键信息包括：`访问时间`、`用户IP`、`访问的URL`、`用户浏览器信息`、`HTTP状态码`、`请求处理时间`

用户登录日志
用于记录用户的Login、Logout、CheckLogin请求情况，关键信息如下：
`Login`：`请求时间`、`用户IP`、`用户名`、`渠道信息`、`用户浏览器信息`、`登录处理结果`、`请求花费时间`、`tokenId`、`sessionid`
`Logout`：`请求时间`、`用户IP`、`用户名`、`渠道信息`、`用户浏览器信息`、`登出结果`、`请求花费时间`、`tokenid`、`sessionid`
`CheckLogin`：`请求时间`、`用户IP`、`用户名`、`渠道信息`、`用户浏览器信息`、`检查结果`、`检查花费时间`、`tokenid`、`sessionId`

服务接口调用日志
所有外部接口的调用需要记录接口访问信息，关键信息如下：
请求时间、接口URL、接口方法、调用结果、执行时间

配置规范

- 统一使用log4j.xml、log4j2.xml、logback配置。
- 所有的jar包中不推荐包含log4j.xml、log4j.properties、logback.xml文件，避免干扰实际的业务系统。
- 注意Logger间的继承关系，如：

> - log4j的继承是通过命名来实现的。
> - 子logger会默认继承父logger的appender，将它们加入到自己的Appender中；除非加上了additivity="false"，则不再继承父logger的appender。
> - 子logger只在自己未定义输出级别的情况下，才会继承父logger的输出级别。

- Log文件位置和命名，目前Log文件的位置统一放在相同目录下面，Log名字通常以业务名开头，如xxx.log.2015-11-19等。
- 日志格式：必选打印数据项: 发生时间、日志级别、日志内容,可选文件和行号。
- 远程日志的输出需要注意host和port，区分cagegory。

日志文件

文件命名

[强制] 以 " {日期} {文件名分隔符} [{级别}] . log " 格式命名

[强制] {日期}格式可选范围：yyyymmdd (年月日) , yyyymmddhh (年月日时)

[推荐] {文件名分隔符}使用：点

[参考] {文件名分隔符} 可选范围：中划线，下划线，点

[正例] /data/logs/tsb/user/20170913.INFO.log

[正例] /data/logs/tsb/user/2017091314.ERROR.log

日志输出

[强制] 不同级别日志通过配置分开输出。

[推荐] 对于不能通过配置作出分级别输出的工具组件，应将ERROR以上级别单独输出。

内容格式

日志内容

[强制] 简明扼要，无冗余

[强制] 关键业务必须可通过日志回溯请求，并定义明确的日志级别

[强制] 异常与错误必须记录日志，并定义明确的日志级别

[强制] 不允许将已捕获的异常栈随意丢进日志，应给出明确的级别和语义描述

[强制] 每一条日志内容必须包括且不限于以下内容：时间、进程ID、日志级别、日志内容。

日志格式

[强制] 每一条日志记录为一行

[推荐] 对于日志内容中有换行操作的，应计划处理为一行，否则日志收集之后将出现不可查看或分析问题

[强制] 单条日志内容格式为

{时间点} {日志分隔符} {级别} {日志分隔符} {进程ID} {日志分隔符} [{线程名}] {日志分隔符} {日志内容}

[强制] {时间点} 格式为：yyyy-MM-dd hhмmss[.SSS] (年-月-日 小时:分钟:秒[.毫秒])

[推荐] {日志分隔符} 使用：竖线

[参考] {日志分隔符} 可选范围：竖线，空格

[正例] 2017-09-13 19:35:54 | ERROR | 26922 | api error /api/getuserinfo 404

[正例] 2017-09-13 19:35:54 ERROR 26922 api error /api/getuserinfo 404