JAVA日志体系

引言

正文

日志框架发展史

日志实战

spring4和spring5日志中的不同

1.Spring4日志体系

2.Spring5日志体系

引言

还在为弄不清`commons-logging.jar`、`log4j.jar`、`sl4j-api.jar`等日志框架之间复杂的关系而感到烦恼吗?

还在为如何统一系统的日志输出而感到不知所措嘛?

您是否依然存在这样的烦恼。比如，要更改spring的日志输出为log4j 2，却不知该引哪些jar包，只知道去百度一下所谓的博客，照着人家复制，却无法弄懂其中的原理?

不要急，不要方！本文带你们弄懂其中的原理，只要你静下心看本文，你就能随心所欲更改你系统里的日志框架，统一日志输出!

正文

日志框架发展史

早年，你工作的时候，在日志里使用了log4j框架来输出，于是你代码是这么写的

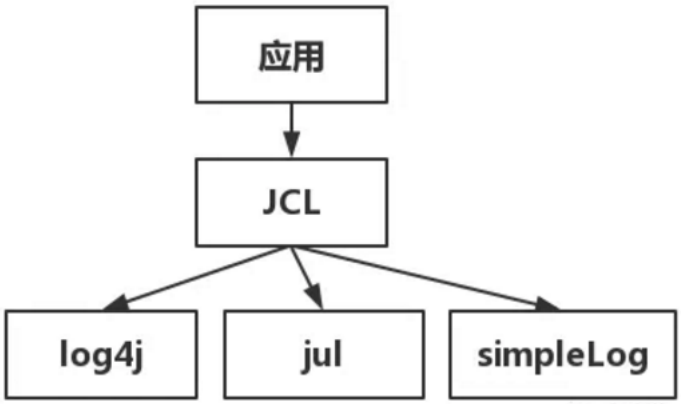
```
import org.apache.log4j.Logger;
\\省略
Logger logger = Logger.getLogger(Test.class);
logger.trace("trace");
\\省略
```

但是，岁月流逝，sun公司对于log4j的出现内心隐隐表示嫉妒。于是在jdk1.4版本后，增加了一个包为java.util.logging，简称为jul，用以对抗log4j。于是，你的领导要你把日志框架改为jul，这时候你只能一行行的将log4j的api改为jul的api，如下所示

```
import java.util.logging.Logger;
\\省略
Logger loggger = Logger.getLogger(Test.class.getName());
logger.finest("finest");
\\省略
```

可以看出，api完全是不同的。那有没有办法，将这些api抽象出接口，这样以后调用的时候，就调用这些接口就好了呢?

这个时候jcl(Jakarta Commons Logging)出现了，说jcl可能大家有点陌生，讲`commons-logging-xx.jar`组件，大家总有印象吧。JCL 只提供 log 接口，具体的实现则在运行时动态寻找。这样一来组件开发者只需要针对 JCL 接口开发，而调用组件的应用程序则可以在运行时搭配自己喜好的日志实践工具。JCL可以实现的集成方案如下图所示



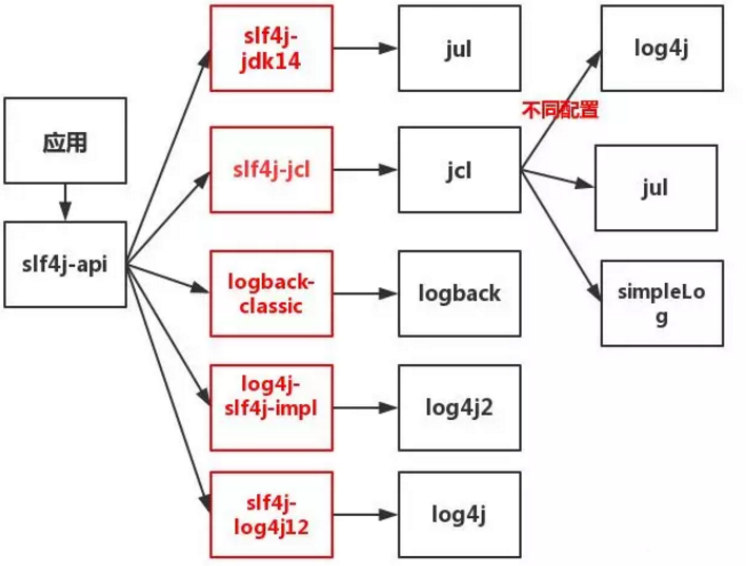
jcl默认的配置：如果能找到Log4j 则默认使用log4j 实现，如果没有则使用jul(jdk自带的) 实现，再没有则使用jcl内部提供的SimpleLog 实现。

于是，你在代码里变成这么写了

```
import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;
\\省略
Log log =LogFactory.getLog(Test.class);
log.trace('trace');
\\省略
```

至于这个Log具体的实现类，JCL会在ClassLoader中进行查找。这么做，有三个缺点，缺点一是效率较低，二是容易引发混乱，三是在使用了自定义ClassLoader的程序中，使用JCL会引发内存泄露。

于是log4j的作者觉得jcl不好用，自己又写了一个新的接口api，那么就是slf4j。关于slf4j的集成图如下所示



如图所示，应用调了sl4j-api，即日志门面接口。日志门面接口本身通常并没有实际的日志输出能力，它底层还是需要去调用具体的日志框架API的，也就是实际上它需要跟具体的日志框架结合使用。由于具体日志框架比较多，而且互相也大都不兼容，日志门面接口要想实现与任意日志框架结合可能需要对应的桥接器，上图红框中的组件即是对应的各种桥接器！

我们在代码中需要写日志，变成下面这么写

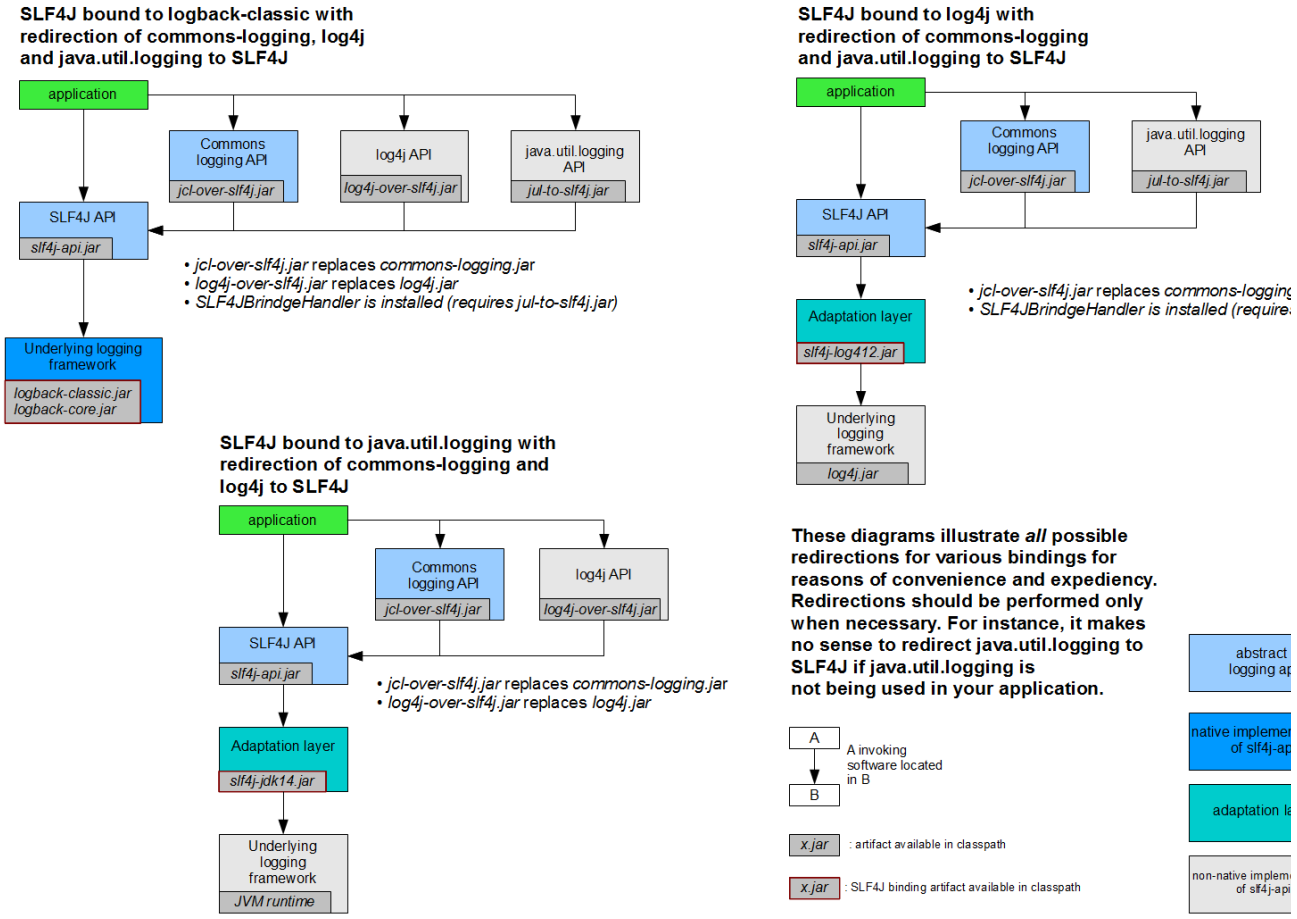
```
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
//省略
Logger logger = LoggerFactory.getLogger(Test.class);
// 省略
logger.info("info");
```

在代码中，并不会出现具体日志框架的api。程序根据classpath中的桥接器类型，和日志框架类型，判断出logger.info应该以什么框架输出！注意了，如果classpath中不小心引了两个桥接器，那会直接报错的！

因此，在阿里的开发手册上才有这么一条

> **强制：应用中不可直接使用日志系统（log4j、logback）中的 API，而应依赖使用日志框架 SLF4J 中的 API。使用门面模式的日志框架，有利于维护和各个类的日志处理方式的统一。**

- **如果要将jcl或jul 转slf4j呢?**



ok，至此，基础知识完毕，下面是实战！

日志实战

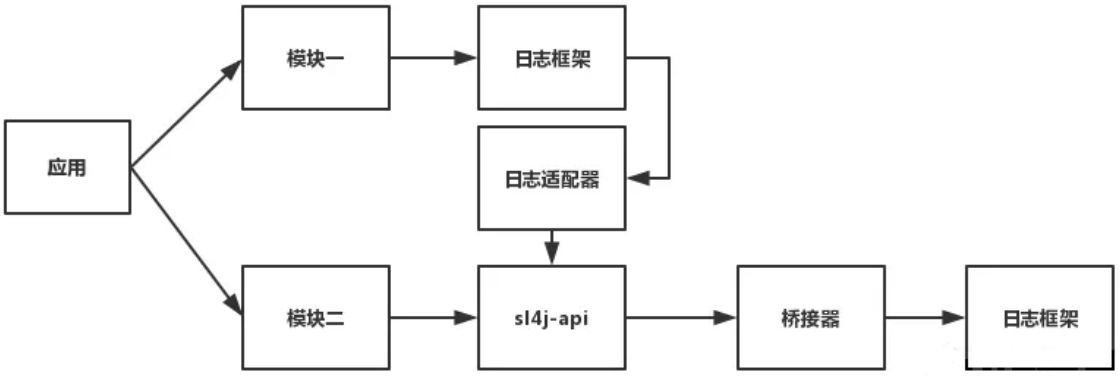
案例一

一个项目，一个模块用log4j，另一个模块用slf4j+log4j2,如何统一输出?

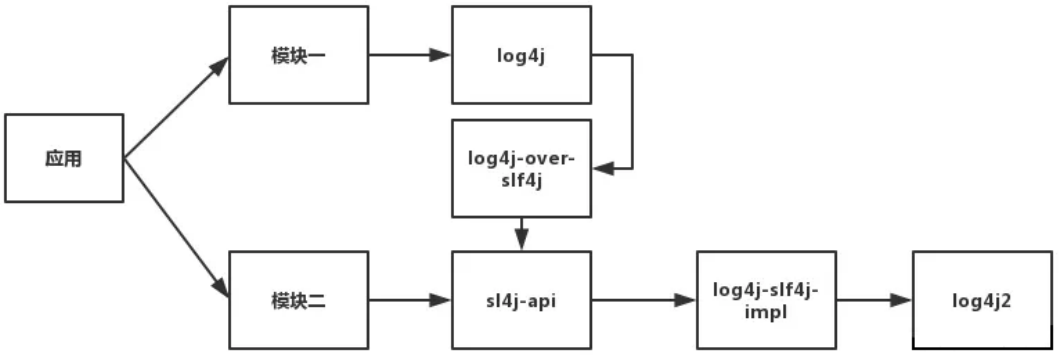
其实在某些中小型公司，这种情况很常见。我曾经见过某公司的项目，因为研发不懂底层的日志原理，日志文件里头既有log4j.properties,又有log4j2.xml，各种API混用，惨不忍睹!

还有人用着jul的API，然后拿着log4j.properties，跑来问我，为什么配置不生效！简直是一言难尽!

OK，回到我们的问题，如何统一输出！OK，这里就要用上slf4j的适配器，slf4j提供了各种各样的适配器，用来将某种日志框架委托给slf4j。其最明显的集成工作方式有如下:



进行选择填空，将我们的案例里的条件填入,根据题意应该选log4j-over-slf4j适配器，于是就变成下面这张图



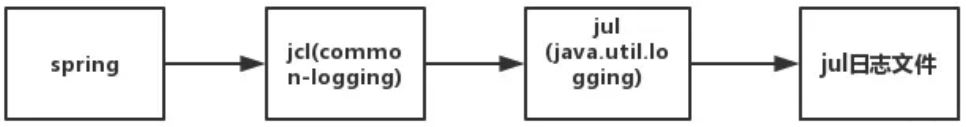
就可以实现日志统一为log4j2来输出!

ps:根据适配器工作原理的不同，被适配的日志框架并不是一定要删除！以上图为例，log4j这个日志框架删不删都可以，你只要能保证log4j的加载顺序在log4j-over-slf4j后即可。因为log4j-over-slf4j这个适配器的工作原理是，内部提供了和log4j一模一样的api接口，因此你在程序中调用log4j的api的时候，你必须想办法让其走适配器的api。如果你删了log4j这个框架，那你程序里肯定是走log4j-over-slf4j这个组件里的api。如果不删log4j，只要保证其在classpth里的顺序比log4j前即可!

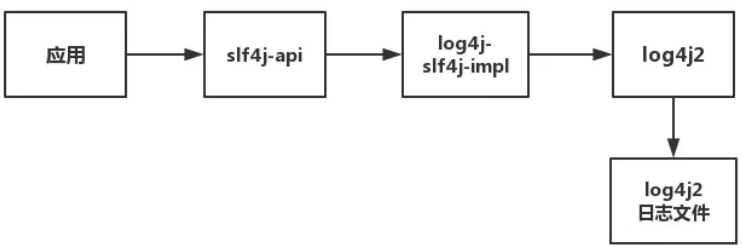
案例二

如何让spring以log4j2的形式输出?

spring默认使用的是jcl输出日志，由于你此时并没有引入Log4j的日志框架，jcl会以jul做为日志框架。此时集成图如下

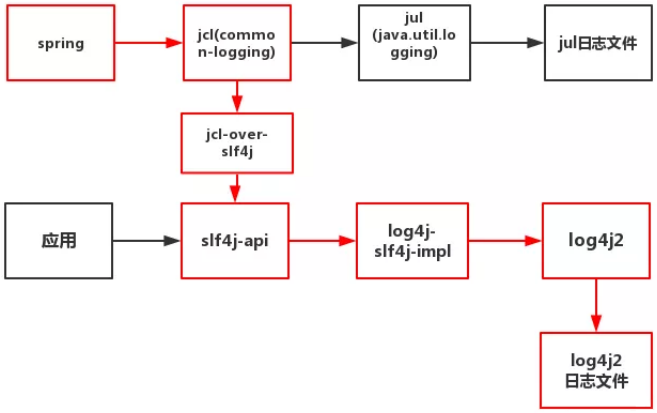


而你的应用中，采用了slf4j+log4j-core，即log4j2进行日志记录，那么此时集成图如下



那我们现在需要让spring以log4j2的形式输出？怎么办?

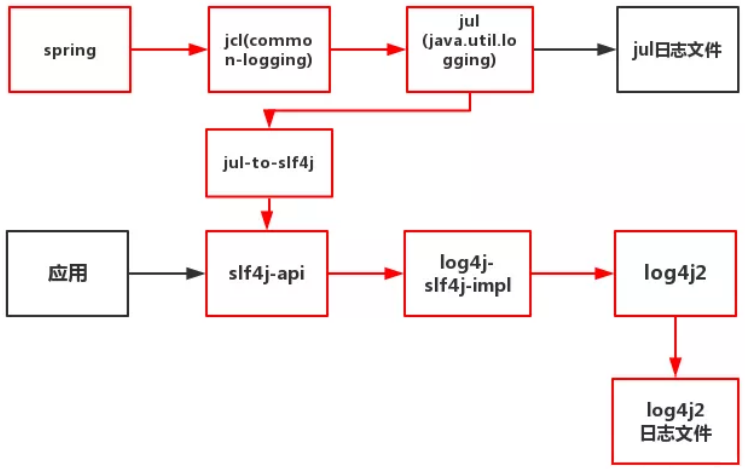
OK,第一种方案，走jcl-over-slf4j适配器，此时集成图就变成下面这样了



在这种方案下，spring框架中遇到日志输出的语句，就会如上图红线流程一样，最终以log4J2的形式输出！

OK，有第二种方案么？

有，走jul-to-slf4j适配器，此时集成图如下



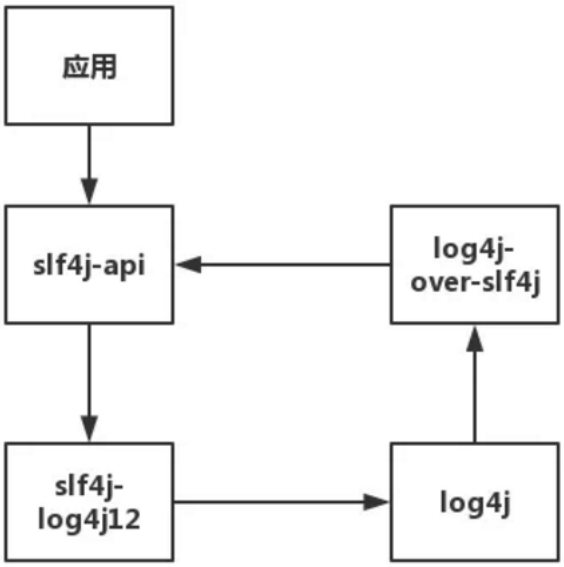
ps:这种情况下，记得在代码中执行

```
SLF4JBridgeHandler.removeHandlersForRootLogger();
SLF4JBridgeHandler.install();
```

这样jul-to-slf4j适配器才能正常工作，详情可以查询该适配器工作原理。

天啦噜！要死循环

假设，我们在应用中调用了sl4j-api，但是呢，你引了四个jar包，slf4j-api-xx.jar,slf4j-log4j12-xx.jar,log4j-xx.jar,log4j-over-slf4j-xx.jar，于是你就会出现如下尴尬的场面



如上图所示，在这种情况下，你调用了slf4j-api，就会陷入死循环中！slf4j-api去调了slf4j-log4j12,slf4j-log4j12又去调用了log4j，log4j去调用了log4j-over-slf4j。最终，log4j-over-slf4j又调了slf4j-api，陷入死循环！

spring4和spring5日志中的不同

1.Spring4日志体系

构建spring4项目

采用java+注解的方式快速构建，pom中只引入spring-context包

```
<dependencies>
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-context</artifactId>
        <version>4.3.14.RELEASE</version>
    </dependency>
</dependencies>
```

运行下面的代码，可以看到有日志输出

```
public static void main(String[] arg){
    AnnotationConfigApplicationContext annotationConfigApplicationContext
            = new AnnotationConfigApplicationContext(AppConfig.class);
}
```

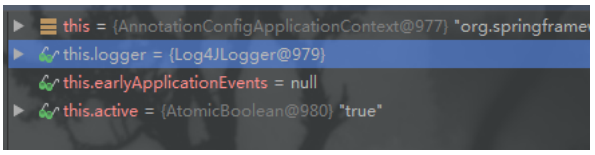
```
va ...
f org.springframework.context.annotation.AnnotationConfigApplicationContext prepareRefresh
ingframework.context.annotation.AnnotationConfigApplicationContext@3f91beef: startup date [Thu Sep 12 14:32:14 CST 2019]; root of context hierarchy
```

找到打印日志的地方，debug模式下，查看输出日志的Log是什么log

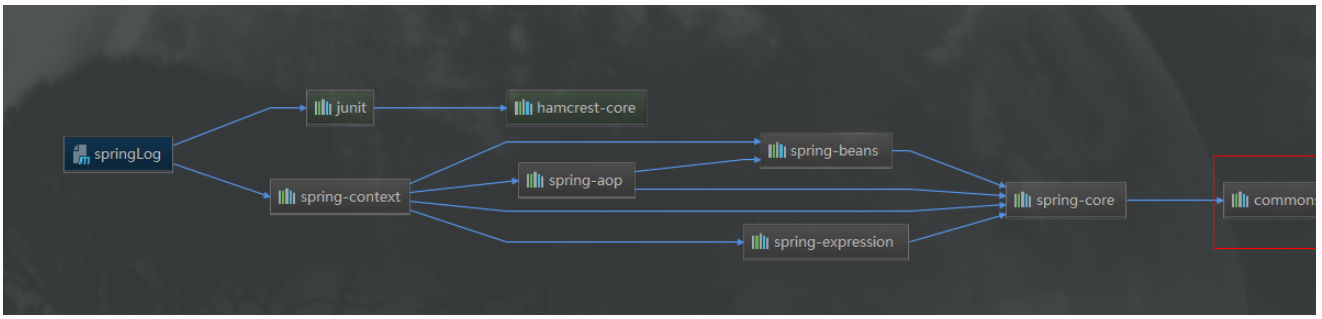
```
this = {AnnotationConfigApplicationContext@809} "org.springframew
this.logger = {Jdk14Logger@811}
this.earlyApplicationEvents = null
this.active = {AtomicBoolean@812} "true"
```

可以看出是jdk14Logger，这个在JCL中说过，这个指的是JUL，也就是说在默认spring日志体系下，采用的是JUL，

接下来，我们按照之前的方法引入log4j，debug运行上面的程序，再次查看日志类型



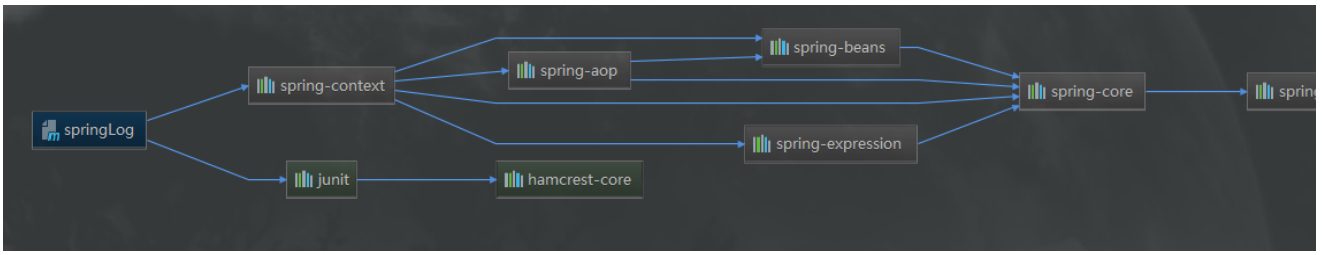
额，这次在增加log4j jar包和配置文件的情况下，spring4有使用了log4j，这么像JCL呢，木错，让我们在idea中打开spring4的日志依赖结构：



common-logging 这不就是JCL使用到的包吗，可以看出，Spring4使用的是原生的JCL，所以在有log4j的时候使用log4j打印日志，没有的时候使用JUL打印日志。

2.Spring5日志体系

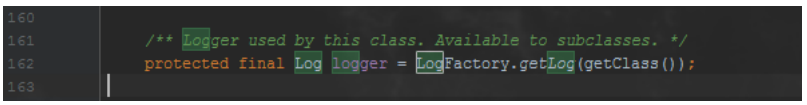
线上依赖结构图：



答题结构没变，只是原来common-logging ，换成了spring-jcl，看名字就知道是spring自造的包，jcl，更是标注了，它使用的是JCL日志体系。

所以还是看源码吧。

按照之前的经验，我们只用debug找到spring内部一个Log，看看他的产生方式和类型。这次我给大家找了AbstractApplicationContext里面找到产生Log的地方



进入这个方法的getLog()中，一直深入，不要怜惜spring，找到LogAdapter中的createLog()方法

```
public static Log createLog(String name) {
    switch (logApi) {
        case LOG4J:
            return Log4jAdapter.createLog(name);
        case SLF4J_LAL:
            return Slf4jAdapter.createLocationAwareLog(name);
        case SLF4J:
            return Slf4jAdapter.createLog(name);
        default:
            // Defensively use lazy-initializing adapter class here as well since the
            // java.logging module is not present by default on JDK 9. We are requiring
            // its presence if neither Log4j nor SLF4J is available; however, in the
            // case of Log4j or SLF4J, we are trying to prevent early initialization
            // of the JavaUtilLog adapter - e.g. by a JVM in debug mode - when eagerly
            // trying to parse the bytecode for all the cases of this switch clause.
            return JavaUtilAdapter.createLog(name);
    }
}
```

可以看出来spring5中对日志的生产，不在像原生JCL中那样使用一个数组，然后进行循环产生，这里用到的是Switch case,这个关键字段LogApi又是在哪一部分赋值的呢？看图

```
static {
    if (isPresent(LOG4J_SPI)) {
        if (isPresent(LOG4J_SLF4J_PROVIDER) && isPresent(SLF4J_SPI)) {
            // log4j-to-slf4j bridge -> we'll rather go with the SLF4J SPI;
            // however, we still prefer Log4j over the plain SLF4J API since
            // the latter does not have location awareness support.
            logApi = LogApi.SLF4J_LAL;
        }
        else {
            // Use Log4j 2.x directly, including location awareness support
            logApi = LogApi.LOG4J;
        }
    }
    else if (isPresent(SLF4J_SPI)) {
        // Full SLF4J SPI including location awareness support
        logApi = LogApi.SLF4J_LAL;
    }
    else if (isPresent(SLF4J_API)) {
        // Minimal SLF4J API without location awareness support
        logApi = LogApi.SLF4J;
    }
    else {
        // java.util.logging as default
        logApi = LogApi.JUL;
    }
}
```

Duang ,没错是在静态代码块中赋的值，为了验证，我们准备用其中提到的log4j2验证（注意：log4j不行，因为这里的switch没有log4j选项），首先我们准备log4j2的配置文件

```
<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="debug">

<AppenderRef ref="Console"/>

</Root>

</Loggers>

</Configuration>
```

然后准备pom

```
<dependencies>
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-context</artifactId>
        <version>5.1.9.RELEASE</version>
    </dependency>
    <dependency>
        <groupId>org.apache.logging.log4j</groupId>
        <artifactId>log4j-core</artifactId>
        <version>2.6.2</version>
    </dependency>
</dependencies>
```

代码还是这一行，直接运行：

```
public class Test {


    public static void main(String[] arg){
        AnnotationConfigApplicationContext annotationConfigApplicationContext
                = new AnnotationConfigApplicationContext(AppConfig.class);
    }
}
```

结果有日志打印出来了

```
16:09:10.986 [main] DEBUG org.springframework.context.annotation.AnnotationConfigApplicationContext - Refreshing c
16:09:11.001 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Creating shared i
16:09:11.161 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Creating shared i
16:09:11.163 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Creating shared i
16:09:11.165 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Creating shared i
16:09:11.166 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Creating shared i
16:09:11.172 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Creating shared i
```

所以，在spring5中，依然使用的是JCL，但是不是原生的，是经过改造的JCL，默认使用的是JUL，而原生JCL中默认使用的是log4j.

文档：架构师必备，带你弄清混乱的JAVA日志体...
链接：http://note.youdao.com/noteshare?id=639aacb1b59f7a4d7c951e021f636661&sub=4A0A990FF7B54AF588567C0F59A75B28