第九节课:Spring5.0新特性之日志框架

一:**日志框架场景**:

某项目开发人员二蛋,为了了解项目运行情况,在我们代码中加入了，System.out.println("*****")来记录日志,

有一天,项目经理觉得通过这种,System.out.println("*****")的方式很搂,要他把把代码中的,System.out.println

给去掉,但是过了几天之后，项目出问题了,查询很棘手又没有日志,然后经理又要求他把System.out.println加上,。。。。。。。。然后又去掉。。。。。又加上.....

情况一:二蛋是一个脾气暴躁的人，拿起板砖就跟经理干起来,然后牢底坐穿，，，全剧终，，，，，，，没用案例场景，，，，，，**下课。。。。。。**

情况二:

二蛋为了解决System.out.println("*****")比较搂的情况,然后就写了一个记录日志的jar包名称叫 angle-logging.jar用来替代 System.out.println("*****")

二蛋是一个爱专研的人,想出了一写比较牛逼的点子,,,,比如**日志异步记录，日志归档。。。。。。**然后起名叫angle-logging-good.jar,

然后把项目中原来的angle-logging.jar中的卸下来,然后安装新的框架angle-logging-good.jar,但是由于二种方式实现的接口可能不一样，需要修改代码中的

日志打印类。**这种情况 这么办**?

此时二蛋,想到了一个好的点子,我把angle-logging.jar 和angle-logging-good.jar的功能都抽取出来形成一个门面angle-logging-intf.jar（也就是我们的接口）然后在二个日志框架中实现不同的功能.这样，我的业务代码中直接使用的是我们的angle-logging-intf.jar的方法，然后根据需要导入了angle-logging.jar或者angle-logging-good.jar

二:我们Java中常用的日志框架是什么

2.1）我们常常听说的就是如下的日子框架,还不知道这么选?

①:JUL(java.util.logging) ,

②:JCL(Jakarta Commons-Logging) 由apache公司Jakarta 小组开发的,

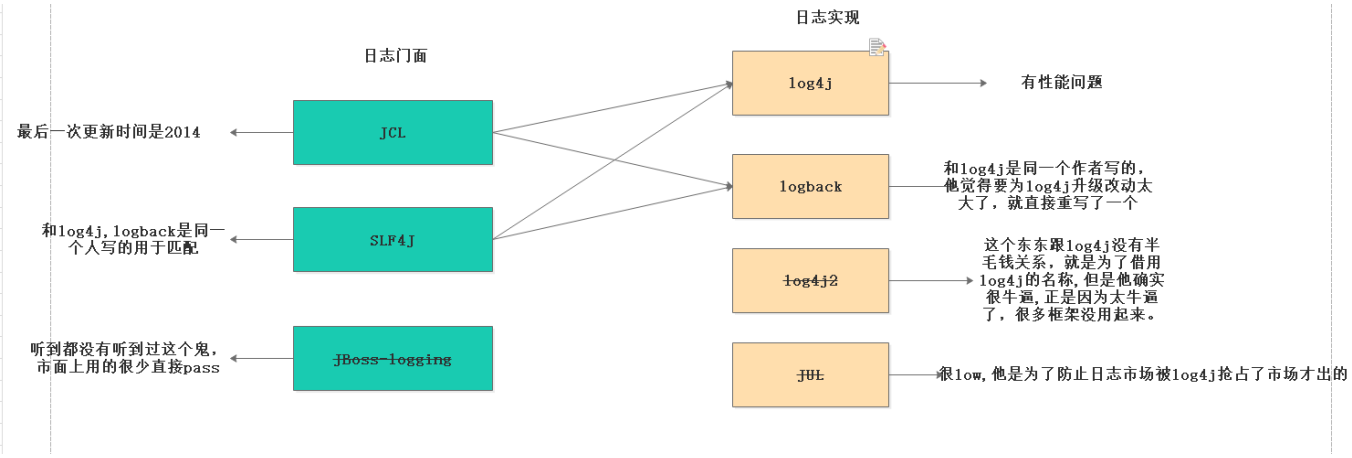
③:JBoss-logging

④:logback

⑤:log4j

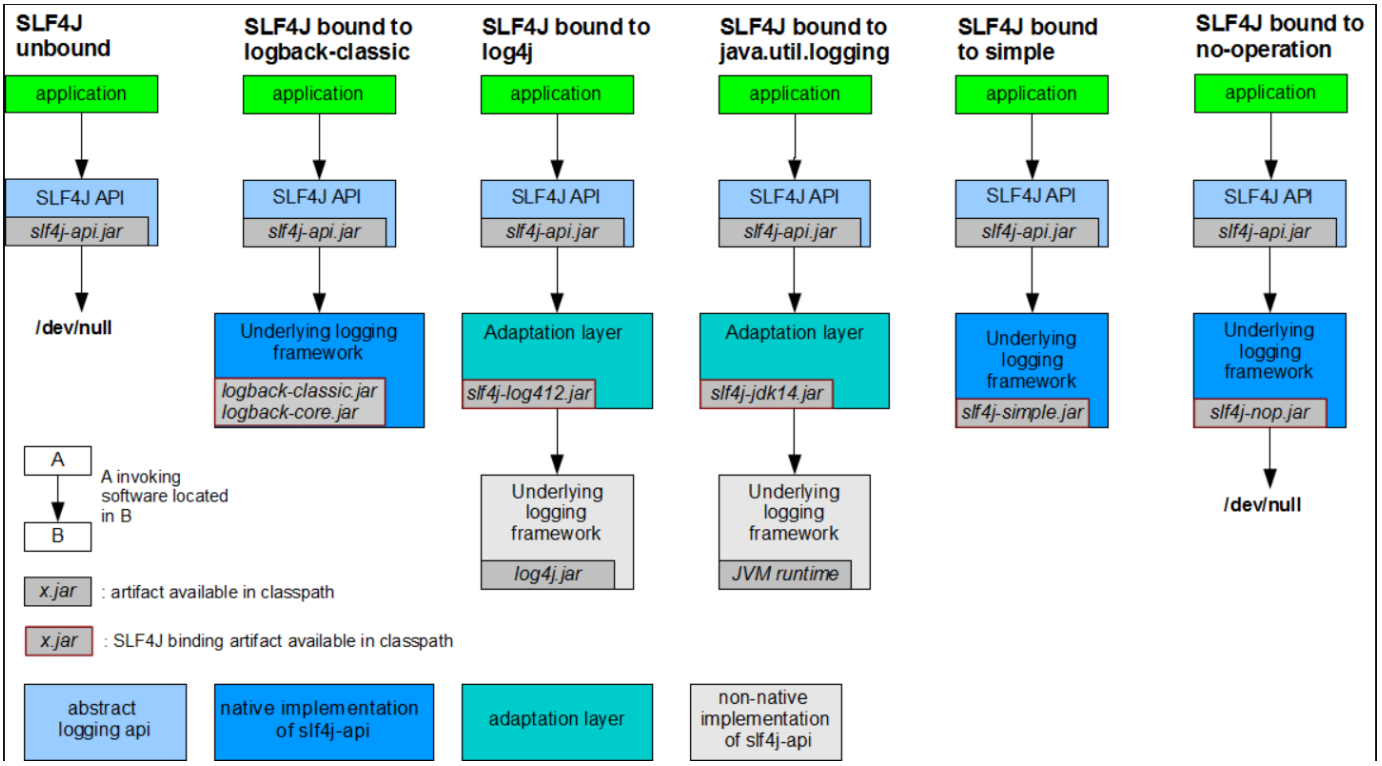
⑥:log4j2

⑦:slf4j（Simple Logging Facade for Java.）



我们Spring底层选择的是我们的这个JCL做为日志门面的

SpringBoot **选择的是** SLF4J**做为我们的日志门面(当时**log4j,**和**logback)**他选择了**logback



2.2)**加入我们系统使用的是**SLF4J**作为日志门面，我们是如何匹配**

①:app(我们的应用系统)+日志门面(Slf4j slf4j-api.jar)+不加我们的日志实现 不会记录日志

②:app(我们的应用系统)+日志门面(Slf4j slf4j-api.jar)+logback(logback-classic.jar,logback-

core.jar)

③:app(我们的应用系统)+日志门面(Slf4j slf4j-api.jar)+适配器(因为早期log4j 压根不知道有一slf4j的日志门面,所以降入适配器slf4j-log412.jar实现我们slf4j的接口,真正实现功能调用我们的log4j.jar)+log4j.jar

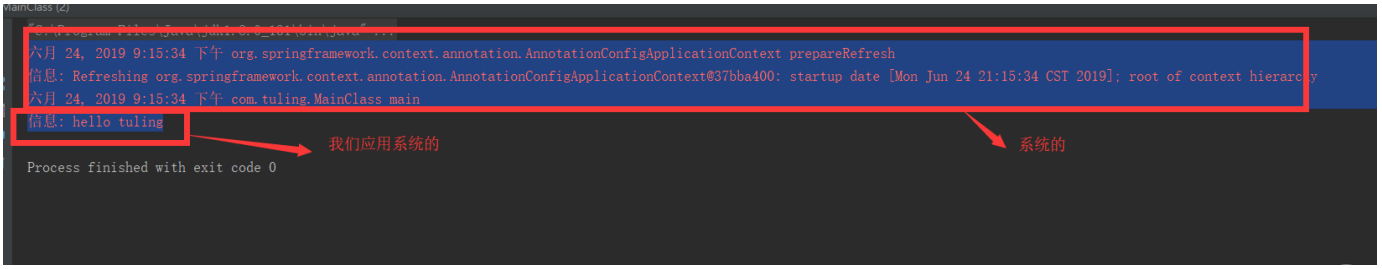
④:app(我们的应用系统)+日志门面(Slf4j slf4j-api.jar)+适配器(因为早期juc 压根不知道有一slf4j的日志门面,所以降入适配器slf4j-jdk14.jar实现我们slf4j的接口,真正实现功能调用我们的juc的jar包的功能)+juc.jar

⑤:app(我们的应用系统)+日志门面(Slf4j slf4j-api.jar)+slf4j(slf4j-simple.jar)

第一中情况(我们业务系统直接使用的是spring底层用的日志框架jcl+jdk的日志实现)

```
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-context</artifactId>
            <version>4.3.20.RELEASE</version>
        </dependency>

public class MainClass {

    private static Logger logger = Logger.getLogger(MainClass.class.getName());

    public static void main(String[] args) {

        AnnotationConfigApplicationContext ctx = new AnnotationConfigApplicationContext(MainConfig.class);

        logger.info("hello tuling");
        ctx.start();
    }
}
```

打印的日志格式是:(我们发现都是红色的)



第二种情况:(我们往pom依赖中加入log4j的包以及加入log4j的配置文件,spring 底层的日志门面使用的是jcl,实现貌似使用了是log4j的格式)

```
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-context</artifactId>
        <version>4.3.20.RELEASE</version>
    </dependency>


    <!--log4j的日志-->
    <dependency>
        <groupId>log4j</groupId>
        <artifactId>log4j</artifactId>
        <version>1.2.17</version>
    </dependency>

import org.apache.log4j.Logger;

private static Logger logger = Logger.getLogger(MainClass.class.getName());

public static void main(String[] args) {

    AnnotationConfigApplicationContext ctx = new AnnotationConfigApplicationContext(MainConfig.class);

    logger.info("hello tuling");
    ctx.start();
}
```

日志打印情况:

```
21:25:22,224 DEBUG DefaultListableBeanFactory:447 - Creating instance of bean 'mainConfig'
21:25:22,225 DEBUG DefaultListableBeanFactory:537 - Eagerly caching bean 'mainConfig' to allow for resolving potential circular references
21:25:22,228 DEBUG DefaultListableBeanFactory:483 - Finished creating instance of bean 'mainConfig'
21:25:22,228 DEBUG DefaultListableBeanFactory:221 - Creating shared instance of singleton bean 'person'
21:25:22,228 DEBUG DefaultListableBeanFactory:447 - Creating instance of bean 'person'
21:25:22,231 DEBUG DefaultListableBeanFactory:251 - Returning cached instance of singleton bean 'mainConfig'
21:25:22,251 DEBUG DefaultListableBeanFactory:537 - Eagerly caching bean 'person' to allow for resolving potential circular references
21:25:22,253 DEBUG DefaultListableBeanFactory:483 - Finished creating instance of bean 'person'
21:25:22,253 DEBUG DefaultListableBeanFactory:251 - Returning cached instance of singleton bean 'org.springframework.context.event.internalEvent
21:25:22,272 DEBUG AnnotationConfigApplicationContext:785 - Unable to locate LifecycleProcessor with name 'lifecycleProcessor': using default [o
21:25:22,272 DEBUG DefaultListableBeanFactory:251 - Returning cached instance of singleton bean 'lifecycleProcessor'
21:25:22,273 DEBUG PropertySourcesPropertyResolver:92 - Could not find key 'spring.liveBeansView.mbeanDomain' in any property source
21:25:22,274  INFO MainClass:24 - hello tuling
21:25:22,275 DEBUG DefaultListableBeanFactory:251 - Returning cached instance of singleton bean 'lifecycleProcessor'
```

系统的和我们自己的都是log4j的格式打印出来的

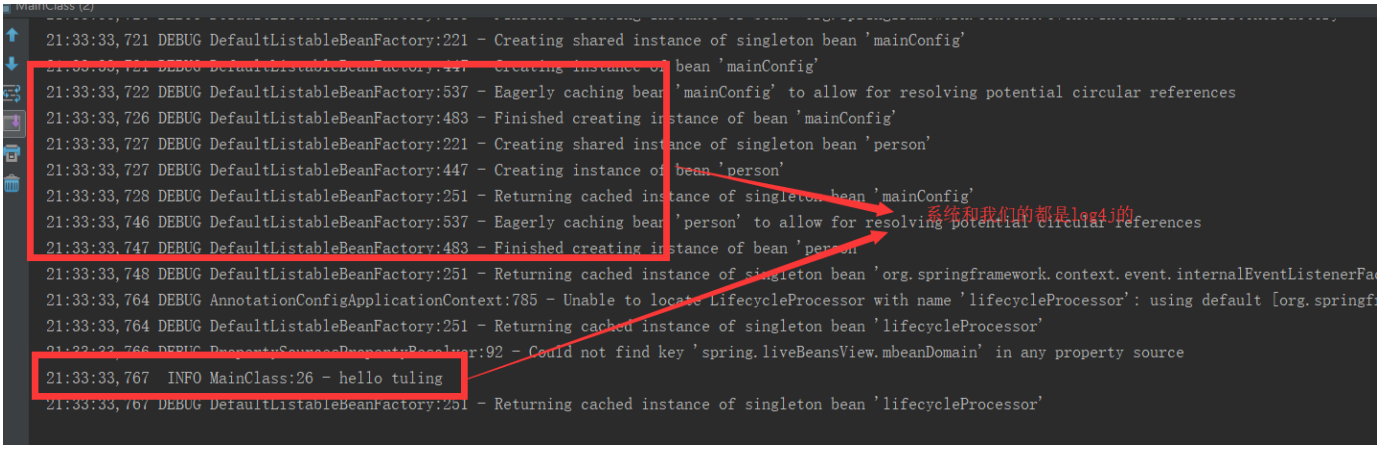
第三种情况(我们往容器中导入了是slf4j的门面，使用log4j的实现)

导入的依赖:

```
    slf4j的门面
    <dependency>
        <groupId>org.slf4j</groupId>
        <artifactId>slf4j-api</artifactId>
```

```
        <version>1.7.10</version>
    </dependency>

    该包是转换包,实现slf4j-api接口，调用log4j的实现
    <dependency>
        <groupId>org.slf4j</groupId>
        <artifactId>slf4j-log4j12</artifactId>
        <version>1.7.10</version>
    </dependency>

    log4j的日志实现
    <dependency>
        <groupId>log4j</groupId>
        <artifactId>log4j</artifactId>
        <version>1.2.17</version>
    </dependency>


import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
public class MainClass {

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

    //private static Logger logger = Logger.getLogger(MainClass.class.getName());

    public static void main(String[] args) {

        AnnotationConfigApplicationContext ctx = new AnnotationConfigApplicationContext(MainConfig.class);

        logger.info("hello tuling");
        ctx.start();
    }
}
```

日志打印情况:



第四种情况:我们往容器中导入了是slf4j的门面，使用logback的实现

加入的依赖:

```
    <dependency>
        <groupId>ch.qos.logback</groupId>
        <artifactId>logback-core</artifactId>
        <version>1.1.2</version>
    </dependency>
    <dependency>
        <groupId>ch.qos.logback</groupId>
        <artifactId>logback-classic</artifactId>
        <version>1.1.2</version>
    </dependency>
    <dependency>
        <groupId>org.slf4j</groupId>
        <artifactId>slf4j-api</artifactId>
        <version>1.7.7</version>
    </dependency>

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
public class MainClass {

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

    //private static Logger logger = Logger.getLogger(MainClass.class.getName());

    public static void main(String[] args) {

        AnnotationConfigApplicationContext ctx = new AnnotationConfigApplicationContext(MainConfig.class);

        logger.info("hello tuling");
        ctx.start();
    }
}
```

```
21:38:01,533 |-INFO in ch.qos.logback.classic.joran.JoranConfigurator@17d99928 - Registering current configuration as safe fallback point

六月 24, 2019 9:38:01 下午 org.springframework.context.annotation.AnnotationConfigApplicationContext prepareRefresh
信息: Refreshing org.springframework.context.annotation.AnnotationConfigApplicationContext@15975490: startup date [Mon Jun 24 21:38:01 CST 2019]; root of context hierarchy
21:38:01.881 [main] INFO  com.tuling.MainClass - hello tuling

Process finished with exit code 0
```

三:Spring4.Xspring底层使用的日志技术

```
for(int i=0; i<classesToDiscover.length && result == null; ++i) {   result: null
    result = createLogFromClass(classesToDiscover[i], logCategory, affectState: true);
}
if
}
re
}
```

classesToDiscover

classesToDiscover = {String[4]@1278}

- 0 = "org.apache.commons.logging.impl.Log4JLogger"
- 1 = "org.apache.commons.logging.impl.Jdk14Logger"
- 2 = "org.apache.commons.logging.impl.Jdk13LumberjackLogger"
- 3 = "org.apache.commons.logging.impl.SimpleLog"

我们从这里可以看出来,spring4.x**获取的日志对象中，**LOGGer**对象是**jCL**的 ，而他底层搭配的技术点就是**

先去找log4j**的日志实现，若没有找到 底层去找**jdk**的日志框架**.**压根不支持** logback，log4j2**的日志技术**.

四:Spring5.x **底层使用的日志技术**

```
private static LogApi logApi = LogApi.JUL;

static {
    ClassLoader cl = LogFactory.class.getClassLoader();
    try {
        //第一步:先尝试去加载 log4j2的日志框架
        cl.loadClass("org.apache.logging.log4j.spi.ExtendedLogger");
        logApi = LogApi.LOG4J;
    }
    catch (ClassNotFoundException ex1) {
        try {
            //第二步:尝试去加载 LogApi.SLF4J_LAL
            cl.loadClass("org.slf4j.spi.LocationAwareLogger");
            logApi = LogApi.SLF4J_LAL;
        }
        catch (ClassNotFoundException ex2) {
            try {
                //尝试去加载slf4j的日志实现
                cl.loadClass("org.slf4j.Logger");
                logApi = LogApi.SLF4J;
            }
            catch (ClassNotFoundException ex3) {
                //使用原生的JUL
            }
        }
    }
}
```

```
    public static Log getLog(String name) {
    switch (logApi) {
        case LOG4J:
            return Log4jDelegate.createLog(name);
        case SLF4J_LAL:
            return Slf4jDelegate.createLocationAwareLog(name);
        case SLF4J:
            return Slf4jDelegate.createLog(name);
        default:
            // Defensively use lazy-initializing delegate 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 JavaUtilDelegate.createLog(name);
    }
}
```