

Spring-boot 是如何整合logger的 LoggerFactory.getLogger()

项目启动报错 显示错误信息如下:

```
2021-08-26 22:27:24 JRebel:
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/D:/repository/org/slf4j/slf4j-log4j12/1.7.25/slf4j-log4j12-1.7.25.jar/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/D:/repository/ch/qos/logback/logback-classic/1.2.3/logback-classic-1.2.3.jar/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
log4j:WARN No appenders could be found for logger [org.springframework.boot.devtools.settings.DevToolsSettings].
log4j:WARN Please initialize the log4j system properly.
log4j:WARN See http://logging.apache.org/log4j/2.x/doc/html/faq.html for more info.
Exception in thread "main" java.lang.AssertionError: LoggerFactory is not a Logback LoggerContext but Logback is on the classpath. Either remove Logback or the competing implementation (class org.slf4j.impl.Log4jLoggerFactory loaded from file:/D:/repository/org/slf4j/slf4j-log4j12/1.7.25/slf4j-log4j12-1.7.25.jar)
    at org.springframework.util.Assert.instanceCheckFailed(Assert.java:637)
    at org.springframework.boot.logging.logback.LogbackLoggingSystem.getLoggerContext(LogbackLoggingSystem.java:286)
    at org.springframework.boot.logging.logback.LogbackLoggingSystem.beforeInitialize(LogbackLoggingSystem.java:302)
    at org.springframework.boot.context.logging.LoggingApplicationListener.onApplicationStartingEvent(LoggingApplicationListener.java:213)
    at org.springframework.boot.context.logging.LoggingApplicationListener.onApplicationEvent(LoggingApplicationListener.java:170)
    at org.springframework.context.event.SimpleApplicationEventMulticaster.doInvokeListener(SimpleApplicationEventMulticaster.java:172)
    at org.springframework.context.event.SimpleApplicationEventMulticaster.invokeListener(SimpleApplicationEventMulticaster.java:169)
    at org.springframework.context.event.SimpleApplicationEventMulticaster.multicastEvent(SimpleApplicationEventMulticaster.java:139)
    at org.springframework.context.event.SimpleApplicationEventMulticaster.multicastEvent(SimpleApplicationEventMulticaster.java:127)
    at org.springframework.boot.context.event.EventPublishingLogListener.starting(EventPublishingLogListener.java:60)
    at org.springframework.boot.SpringApplication.run(SpringApplication.java:293)
    at org.springframework.boot.SpringApplication.run(SpringApplication.java:120)
    at org.springframework.boot.SpringApplication.run(SpringApplication.java:120)
    at com.singun.kingong.XxxApplication.main(XxxApplication.java:28)
Disconnected from the target VM, address: '127.0.0.1:65982', transport: 'socket'
```

错误信息中提示说:

(1) 在ClassPath中发现两个 SLF4J bindings.

SLF4J: Found binding in [jar:file:/D:/repository/org/slf4j/slf4j-log4j12/1.7.25/slf4j-log4j12-1.7.25.jar/org/slf4j/impl/StaticLoggerBinder.class]

SLF4J: Found binding in [jar:file:/D:/repository/ch/qos/logback/logback-classic/1.2.3/logback-classic-1.2.3.jar/org/slf4j/impl/StaticLoggerBinder.class]

(2) 实际使用的是 Log4jLoggerFactory

SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]

(3) 被使用的LoggerFactory 不是 logback的 LoggerContext, 但是logback在类路径中。要么移除logback 要么移除 slf4j-log4j12-1.7.25.jar

LoggerFactory is not a Logback LoggerContext but Logback is on the classpath. Either remove Logback or the competing implementation (class org.slf4j.impl.Log4jLoggerFactory loaded from file:/D:/repository/org/slf4j/slf4j-log4j12/1.7.25/slf4j-log4j12-1.7.25.jar). If you are using WebLogic you will need to add 'org.slf4j' to prefer-application-packages in WEB-INF/weblogic.xml: org.slf4j.impl.Log4jLoggerFactory at org.springframework.util.Assert.instanceCheckFailed(Assert.java:637)

根据错误提示定位到spring-boot的堆栈

```
LogbackLoggingSystem.getLoggerContext
LogbackLoggingSystem.java
Assert.java
StaticLoggerBinder.java
LoggingApplicationListener.java
LoggingSystem.java

282 }
283
284 private LoggerContext getLoggerContext() {
285     ILoggerFactory factory = StaticLoggerBinder.getSingleton().getLoggerFactory();
286     Assert.isInstanceOf(LoggerContext.class, factory,
287         String.format(
288             "LoggerFactory is not a Logback LoggerContext but Logback is on "
289             + "the classpath. Either remove Logback or the competing "
290             + "implementation (%s loaded from %s). If you are using "
291             + "WebLogic you will need to add 'org.slf4j' to "
292             + "prefer-application-packages in WEB-INF/weblogic.xml",
293             factory.getClass(), getLocation(factory));
294     return (LoggerContext) factory;
295 }
296
```

在这段代码中我们看到首先是 获取到ILoggerFactory, 这是org.slf4j中的接口, 然后判断该factory是否是LoggerContext的实例, 这里的LoggerContext是logback中的类。

```

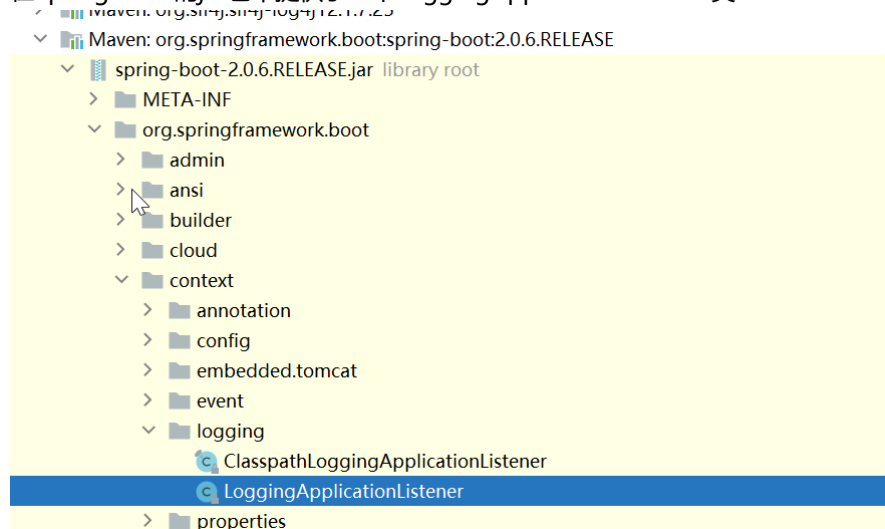
1  .../
25 package org.slf4j;
26
27 /**
28  * ILoggerFactory instances manufacture {@link
29  * instances by name.
30  *
31  * 

Most users retrieve {@link Logger} instances through
32  * {@link LoggerFactory#getLogger(String)} method. An inst
33  * interface is bound internally with {@link LoggerFactory
34  * compile time.
35  *
36  * @author Ceki G&uuml;lc&uuml;
37  */
38 public interface ILoggerFactory {
39


```

从堆栈中我们梳理下Spring-boot启动时整合logger的整个过程：

在Spring-boot的jar包中提供了一个LoggingApplicationListener类



当该Listener监听到spring-boot的ApplicationStarting事件之后将会执行onApplicationStartingEvent

```
ApplicationEvent
logbackLoggingSystem.java x ILoggerFactory.java x Assert.java x StaticLoggerBinder.java x LoggingApplicationListener.java x

@Override
public void onApplicationEvent(ApplicationEvent event) {
    if (event instanceof ApplicationStartingEvent) {
        onApplicationStartingEvent((ApplicationStartingEvent) event);
    }
    else if (event instanceof ApplicationEnvironmentPreparedEvent) {
        onApplicationEnvironmentPreparedEvent(
            (ApplicationEnvironmentPreparedEvent) event);
    }
    else if (event instanceof ApplicationPreparedEvent) {
        onApplicationPreparedEvent((ApplicationPreparedEvent) event);
    }
    else if (event instanceof ContextClosedEvent && ((ContextClosedEvent) event)
        .getApplicationContext().getParent() == null) {
        onContextClosedEvent();
    }
    else if (event instanceof ApplicationFailedEvent) {
        onApplicationFailedEvent();
    }
}

private void onApplicationStartingEvent(ApplicationStartingEvent event) {
    this.loggingSystem = LoggingSystem
        .get(event.getSpringApplication().getClassLoader());
    this.loggingSystem.beforeInitialize();
}
```

然后会使用LoggingSystem 类的静态get方法

```
/**
 * Detect and return the logging system in use. Supports Logback and Java Logging.
 * @param classLoader the classloader
 * @return the logging system
 */
public static LoggingSystem get(ClassLoader classLoader) {
    String loggingSystem = System.getProperty(SYSTEM_PROPERTY);
    if (StringUtils.hasLength(loggingSystem)) {
        if (NONE.equals(loggingSystem)) {
            return new NoOpLoggingSystem();
        }
        return get(classLoader, loggingSystem);
    }
    return SYSTEMS.entrySet().stream()
        .filter((entry) -> ClassUtils.isPresent(entry.getKey(), classLoader))
        .map((entry) -> get(classLoader, entry.getValue()).findFirst())
        .orElseThrow(() -> new IllegalStateException(
            "No suitable logging system located"));
}
```

public static final String SYSTEM_PROPERTY = LoggingSystem.class.getName();

在get方法中首先根据LoggingSystem的name作为key从System属性中获取配置的LoggingSystem，一般为空，因此将会执行最后的return方法。

```

private static final Map<String, String> SYSTEMS;

static {
    Map<String, String> systems = new LinkedHashMap<>();
    systems.put("ch.qos.logback.core.Appender",
        "org.springframework.boot.logging.logback.LogbackLoggingSystem");
    systems.put("org.apache.logging.log4j.core.impl.Log4jContextFactory",
        "org.springframework.boot.logging.log4j2.Log4j2LoggingSystem");
    systems.put("java.util.logging.LogManager",
        "org.springframework.boot.logging.java.JavaLoggingSystem");
    SYSTEMS = Collections.unmodifiableMap(systems);
}

```

变量Systems是一个map,这个map是一个有顺序的linkedHashMap, 该map中的第一个元素时LogbackLoggingSystem, 第二个是Log4jContextFactory, 第三个是JavaLoggingSystem。

在上面的return中 会根据map中的key指定的class是否在类路径下对SystemsMap中的元素进行筛选, 然后筛选得到第一个findFirst作为结果。

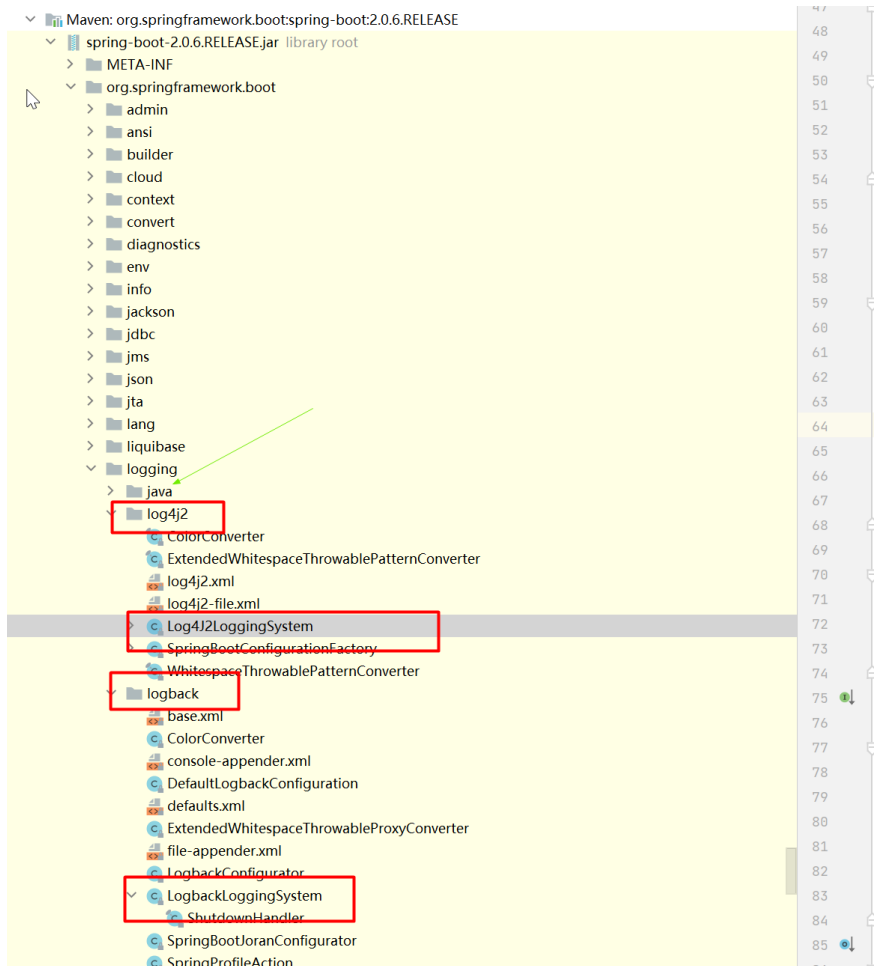
在项目中确认

- (1) ch.qos.logback.core.Appender这个类在项目的类路径中,
- (2) org.apache.logging.log4j.core.impl.Log4jContextFactory 不在项目的类路径中,
- (3) java.util.logging.LogManager 在项目的类路径中,

因此System经过filter过滤得到 Appender 和LogManager, findFirst得到Appender最终得到 LogbackLoggingSystem 作为LoggingSystem

值得注意的是 spring-boot为了支持各个logger, 比如假如说Log4jContextFactory 在类路径中, 则有可能会使用 Log4j2LoggingSystem 作为LoggingSystem

也就是说Log4j2LoggingSystem 是为了支持Log4jContextFactory, LogbackLoggingSystem是为了支持Appender。JavaLoggingSystem是为了支持LogManager, 下面就是spring-boot 的实现, 这些LoggingSystem放置在了spring-boot jar包中



继续上面的讨论

```
@
private void onApplicationStartingEvent(ApplicationStartingEvent event) { event:
    this.LoggingSystem = LoggingSystem loggingSystem: null
        .get(event.getSpringApplication().getClassLoader());
    this.loggingSystem.beforeInitialize();
}
```

当我们得到LoggingSystem是LogbackLoggingSystem后会执行其beforeInitialize方法，不同的LoggingSystem有不同的实现

*/

```
public abstract void beforeInitialize();
```

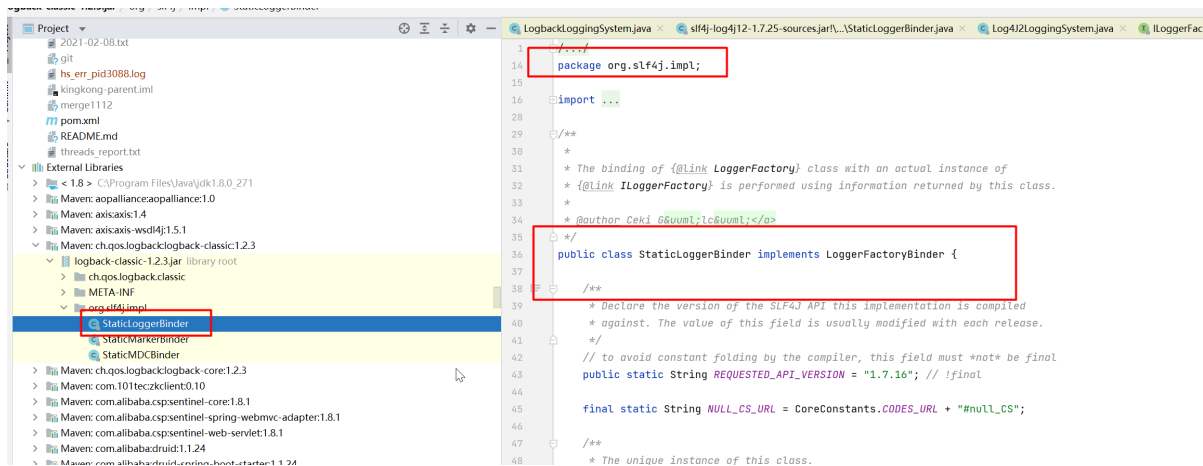
/**

```
* Fully initialize the logging system
* @param initialization the initialization
* @param configLocation the configuration location
* initialization is complete
* @param logFile the log file
* console only output
```

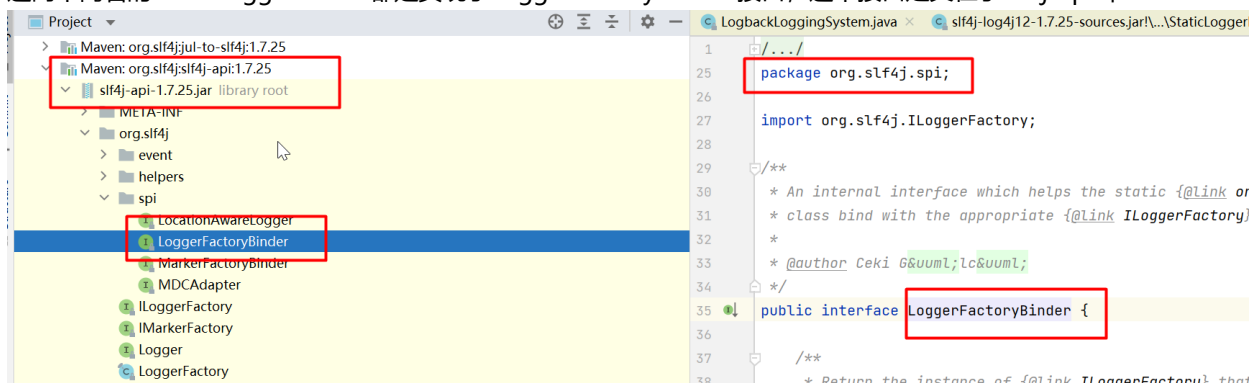
```
AbstractLoggingSystem (org.springframework.boot:
JavaLoggingSystem (org.springframework.boot:
Log4J2LoggingSystem (org.springframework.boot:
LogbackLoggingSystem (org.springframework.boot:
NoOpLoggingSystem in LoggingSystem (org.springframework.boot:
Slf4JLoggingSystem (org.springframework.boot:

```

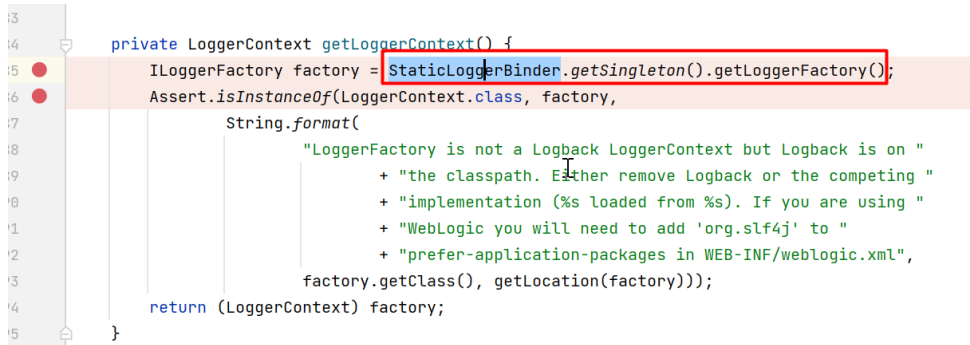
在LogbackLoggingSystem的实现中首先getLoggercontext



这两个同名的StaticLoggerBinder都是实现了LoggerFactoryBinder 接口，这个接口定义在了slf4j-api 中



程序出现错误的原因是 StaticLoggerBinder使用了第一个



这导致StaticLoggerBinder的getSingleton 返回一个slf4j-log4j12包中的StaticLoggerBinder

```

LogbackLoggingSystem.java x slf4j-log4j12-1.7.25-sources.jar!\...StaticLoggerBinder.java x Log4j2LoggingSystem.java x ILogger
* The binding of {@link LoggerFactory} class with an actual instance of
* {@link ILoggerFactory} is performed using information returned by this class.
*
* @author Ceki G&uuml;l;lc&uuml;l;
*/
public class StaticLoggerBinder implements LoggerFactoryBinder {

    /**
     * The unique instance of this class.
     *
     */
    /**
     * Return the singleton of this class.
     *
     * @return the StaticLoggerBinder singleton
     */
    private static final StaticLoggerBinder SINGLETON = new StaticLoggerBinder();

    /**
     * Return the singleton of this class.
     *
     * @return the StaticLoggerBinder singleton
     */
    public static final StaticLoggerBinder getSingleton() {
        return SINGLETON;
    }
}

```

slf4j-log4j12包中的StaticLoggerBinder的的getLoggerFactory 返回的是 Log4jLoggerFactory对象。

```

LogbackLoggingSystem.java x slf4j-log4j12-1.7.25-sources.jar!\...StaticLoggerBinder.java x Log4j2LoggingSystem.java x ILogger
3      * The value of this field is modified with each major release.
7      */
3      // to avoid constant folding by the compiler, this field must *not* be final
1      public static String REQUESTED_API_VERSION = "1.6.99"; // !final
2
5      private static final String loggerFactoryClassStr = Log4jLoggerFactory.class.getName();
4
5      /**
5      * The ILoggerFactory instance returned by the {@link #getLoggerFactory}
7      * method should always be the same object
3      */
7      private final ILoggerFactory loggerFactory;

1      private StaticLoggerBinder() {
2          loggerFactory = new Log4jLoggerFactory();
5          try {
4              /unused/
5              Level level = Level.TRACE;
5              } catch (NoSuchFieldError nsfe) {
7                  Util.report(msg: "This version of SLF4J requires log4j version 1.2.12 or later. See .
3              }
7          }
3      }

1      public ILoggerFactory getLoggerFactory() { return loggerFactory; }
5
5      public String getLoggerFactoryClassStr() { return loggerFactoryClassStr; }
3      }
}

```

这个Log4jLoggerFactory执行下面的判断是否是logback中的LoggerContext 对象的时候就报错了，因为 Log4jLoggerFactory 并不是Logback的LoggerContext对象。

```

private LoggerContext getLoggerContext() {
    ILoggerFactory factory = StaticLoggerBinder.getSingleton().getLoggerFactory();
    Assert.isInstanceOf(LoggerContext.class, factory,
        String.format(
            "LoggerFactory is not a Logback LoggerContext but Logback is on "
            + "the classpath. Either remove Logback or the competing "
            + "implementation (%s loaded from %s). If you are using "
            + "WebLogic you will need to add 'org.slf4j' to "
            + "prefer-application-packages in WEB-INF/weblogic.xml",
            factory.getClass(), getLocation(factory)));
    return (LoggerContext) factory;
}

private Object getLocation(ILoggerFactory factory) {
}

```


我们来看下 Loggerback-classic jar包中 也有一个StaticLoggerBinder 是如何实现getLoggerFactory的

File : D:\repository\ch\qos\logback\logback-classic\1.2.3\logback-classic-1.2.3-sources.jar\org\slf4j\impl\StaticLoggerBinder.java

```
1 /**
2  * Logback: the reliable, generic, fast and flexible logging framework.
3  * Copyright (C) 1999-2015, QOS.ch. All rights reserved.
4  *
5  * This program and the accompanying materials are dual-licensed under
6  * either the terms of the Eclipse Public License v1.0 as published by
7  * the Eclipse Foundation
8  *
9  * or (per the licensee's choosing)
10 *
11 * under the terms of the GNU Lesser General Public License version 2.1
12 * as published by the Free Software Foundation.
13 */
14 package org.slf4j.impl;
15
16 import ch.qos.logback.core.status.StatusUtil;
17 import org.slf4j.ILoggerFactory;
18 import org.slf4j.LoggerFactory;
19 import org.slf4j.helpers.Util;
20 import org.slf4j.spi.LoggerFactoryBinder;
21
22 import ch.qos.logback.classic.LoggerContext;
23 import ch.qos.logback.classic.util.ContextInitializer;
24 import ch.qos.logback.classic.util.ContextSelectorStaticBinder;
25 import ch.qos.logback.core.CoreConstants;
26 import ch.qos.logback.core.joran.spi.JoranException;
27 import ch.qos.logback.core.util.StatusPrinter;
28
29 /**
30 *
31 * The binding of {@link ILoggerFactory} class with an actual instance of
32 * {@link ILoggerFactory} is performed using information returned by this class.
33 *
34 * @author Ceki G?uvm?l;lc&uuml;l;</a>
35 */
36 public class StaticLoggerBinder implements LoggerFactoryBinder {
37
38     /**
39      * Declare the version of the SLF4J API this implementation is compiled
40      * against. The value of this field is usually modified with each release.
41      */
42     // to avoid constant folding by the compiler, this field must *not* be final
43     public static String REQUESTED_API_VERSION = "1.7.16"; // !final
44
45     final static String NULL_CS_URL = CoreConstants.CODES_URL + "#null_CS";
46
47     /**
48      * The unique instance of this class.
49      */
50     private static StaticLoggerBinder SINGLETON = new StaticLoggerBinder();
51
52     private static Object KEY = new Object();
53
54     static {
55         SINGLETON.init();
56     }
57
58     private boolean initialized = false;
59     private LoggerContext defaultLoggerContext = new LoggerContext();
60     private final ContextSelectorStaticBinder contextSelectorBinder = ContextSelectorStaticBinder.getSingleton();
61
62     private StaticLoggerBinder() {
63         defaultLoggerContext.setName(CoreConstants.DEFAULT_CONTEXT_NAME);
64     }
65
66     public static StaticLoggerBinder getSingleton() {
67         return SINGLETON;
68     }
69
70     /**
71      * Package access for testing purposes.
72      */
73     static void reset() {
74         SINGLETON = new StaticLoggerBinder();
75         SINGLETON.init();
76     }
77
78     /**
79      * Package access for testing purposes.
80      */
81     void init() {
82         try {
83             try {
84                 new ContextInitializer(defaultLoggerContext).autoConfig();
85             } catch (JoranException je) {
86                 Util.report("Failed to auto configure default logger context", je);
87             }
88             // logback-292
89             if (!StatusUtil.contextHasStatusListener(defaultLoggerContext)) {
90                 StatusPrinter.printInCaseOfErrorsOrWarnings(defaultLoggerContext);
91             }
92             contextSelectorBinder.init(defaultLoggerContext, KEY);
93             initialized = true;
94         } catch (Exception t) { // see LOGBACK-1159
95             Util.report("Failed to instantiate [" + LoggerContext.class.getName() + "]", t);
96         }
97     }
98
99     public ILoggerFactory getLoggerFactory() {
100         if (!initialized) {
101             return defaultLoggerContext;
102         }
103
104         if (contextSelectorBinder.getContextSelector() == null) {
105             throw new IllegalStateException("contextSelector cannot be null. See also " + NULL_CS_URL);
106         }
107         return contextSelectorBinder.getContextSelector().getLoggerContext();
108     }
109
110     public String getLoggerFactoryClassStr() {
111         return contextSelectorBinder.getClass().getName();
112     }
113 }
114 }
115 }
```

这里有一个静态init方法

init方法中主要是ContextInitializer对象的 autoConfig方法
这个方法会解析查找logback.xml文件
final public static String AUTOCONFIG_FILE = "logback.xml";
configureByResource(URL url)

getLoggerFactory的返回值就是LoggerContext。因此我们可以在LogbackLoggingSystem的getLoggerContext方法中 进行如下判断

ILoggerFactory factory =
StaticLoggerBinder.getSingleton().getLoggerFactory();
Assert.isInstanceOf(LoggerContext.class, factory,

LoggerFactory.getLogger()

```
package org.slf4j;

import ...

/**
 * The LoggerFactory is a utility class producing Loggers for
 * various logging APIs, most notably for log4j, logback and JDK 1.4 logging.
 * Other implementations such as {@link org.slf4j.impl.NOPLogger NOPLogger} and
 * {@link org.slf4j.impl.SimpleLogger SimpleLogger} are also supported.
 * <p/>
 * <p/>
 * LoggerFactory is essentially a wrapper around an I
 * {@link ILoggerFactory} instance bound with LoggerFactory at
 * compile time.
 * <p/>
 * <p/>
 * Please note that all methods in LoggerFactory are static.
 *
 *
 * @author Alexander Dorokhine
 * @author Robert Elliot
 * @author Ceki G&uuml;lc&uuml;m;
 *
 */
public final class LoggerFactory {
```

LoggerFactory是一个为各种日志api生成日志记录器的实用程序类，主要用于log4j、logback和JDK 1.4日志记录。也支持NOPLogger和SimpleLogger等其他实现。

LoggerFactory本质上是一个封装在编译时与LoggerFactory绑定的ILoggerFactory实例的包装器。

```
public static Logger getLogger(String name) {
    ILoggerFactory iLoggerFactory = getILoggerFactory();
    return iLoggerFactory.getLogger(name);
}
```

```

/**
 * Return the {@link ILoggerFactory} instance in use.
 * <p/>
 * <p/>
 * ILoggerFactory instance is bound with this class at compile time.
 *
 *
 * @return the ILoggerFactory instance in use
 */
public static ILoggerFactory getILoggerFactory() {
    if (INITIALIZATION_STATE == UNINITIALIZED) {
        synchronized (LoggerFactory.class) {
            if (INITIALIZATION_STATE == UNINITIALIZED) {
                INITIALIZATION_STATE = ONGOING_INITIALIZATION;
                performInitialization();
            }
        }
    }

    switch (INITIALIZATION_STATE) {
        case SUCCESSFUL_INITIALIZATION:
            return StaticLoggerBinder.getSingleton().getLoggerFactory();
        case NOP_FALLBACK_INITIALIZATION:
            return NOP_FALLBACK_FACTORY;
        case FAILED_INITIALIZATION:
            throw new IllegalStateException(UNSUCCESSFUL_INIT_MSG);
        case ONGOING_INITIALIZATION:
            // support re-entrant behavior.
            // See also http://jira.qos.ch/browse/SLF4J-97
            return SUBST_FACTORY;
    }

    throw new IllegalStateException("Unreachable code");
}

```

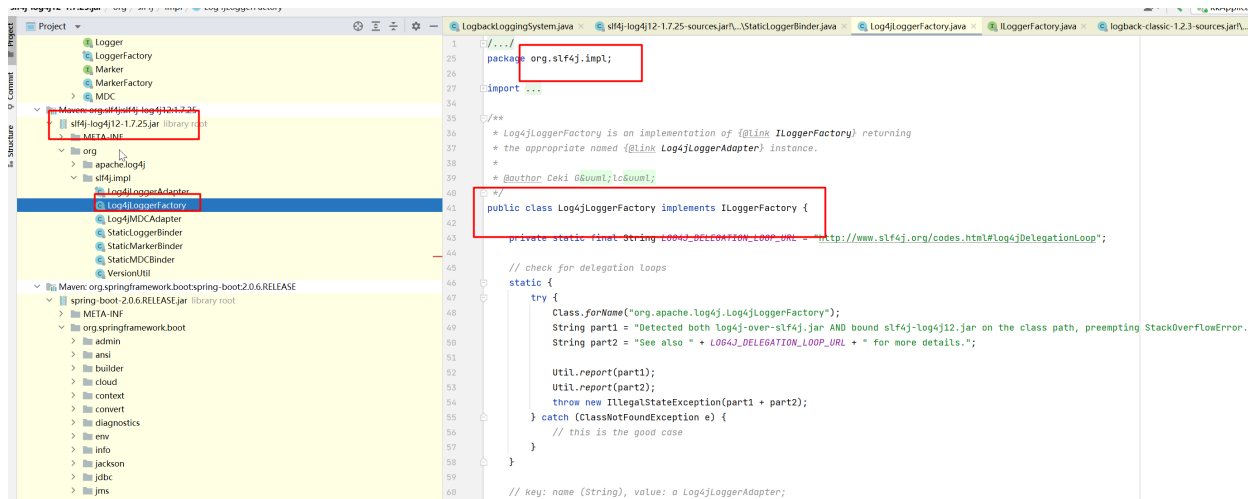
如果没有初始化过则执行初始化，
如果初始化成功了则执行使用
StaticLoggerBinder获取
LoggerFactory

slf4j 为了统一 Logback log4j 分别提供了如下包，这两个包中都有StaticLoggerBinder类。

All	Classes	Files	Symbols	Actions	Git
Q StaticLoggerBinder					
slf4j-log4j12-1.7.25-sources.jar!\...StaticLoggerBinder.java D:\repository\org\slf4j\slf4j-log4j12\1.7.25\slf4j-log4j12-1.7.25-sources.jar\org\					
logback-classic-1.2.3-sources.jar!\...StaticLoggerBinder.java D:\repository\ch\qos\1.2.3\logback-classic-1.2.3-sources.jar\org\slf4j\impl					
StaticLoggerBinder.class D:\repository\slf4j\slf4j-log4j12\1.7.25\slf4j-log4j12-1.7.25-sources.jar\impl\StaticLoggerBinder.class					

同时 这两个包中提供了 Slf4j-api中ILoggerFactory接口规范的实现

Maven: axisaxis-wsd4j:1.5.1	1
logback-classic-1.2.3 library root	package ch.qos.logback.classic;
ch.qos.logback.classic	import ...;
boolean	/**
db	* LoggerContext glues many of the logback-classic components together. In
encoder	* principle, every logback-classic component instance is attached either
filter	* directly or indirectly to a LoggerContext instance. Just as importantly
gaffer	* LoggerContext implements the {@link ILoggerFactory} acting as the
helpers	* manufacturing source of {@link Logger} instances.
html	* @author Ceki Gulcu
jmx	*/
joran	public class LoggerContext extends ContextBase implements ILoggerFactory, Lifecycle {
jul	/** Default setting of packaging data in stack traces */
layout	public static final boolean DEFAULT_PACKAGING_DATA = false;
log4j	final Logger root;
net	private int size;
pattern	private int noAppenderWarning = 0;
selector	final private List<LoggerContextListener> loggerContextListenerList = new ArrayList<>();
servlet	private Map<String, Logger> loggerCache;
sift	private LoggerContextVO loggerContextRemoteView;
spi	private final TurboFilterList turboFilterList = new TurboFilterList();
turbo	private boolean packagingDataEnabled = DEFAULT_PACKAGING_DATA;
util	
AsyncAppender	
BasicConfigurator	
ClassicConstants	
Level	
Logger	
LoggerContext	
PatternLayout	
ViewStatusMessagesServlet	
META-INF	



最终使用slf4j的LoggerFactory.getLogger获取Logger的时候slf4j实现类根据类路径存在的jar来确定返回是logback的logger还是Log4j的logger