# 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：


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

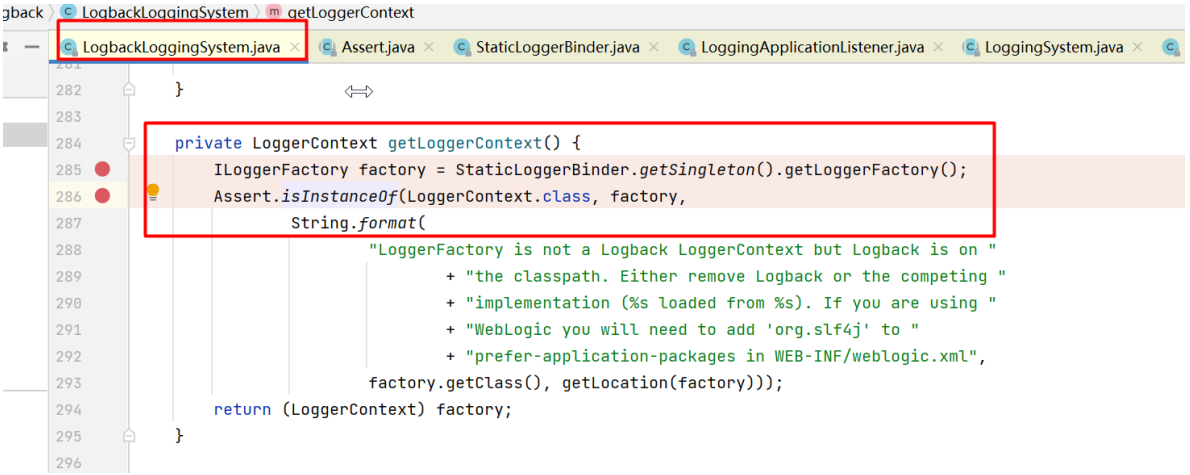
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]

（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



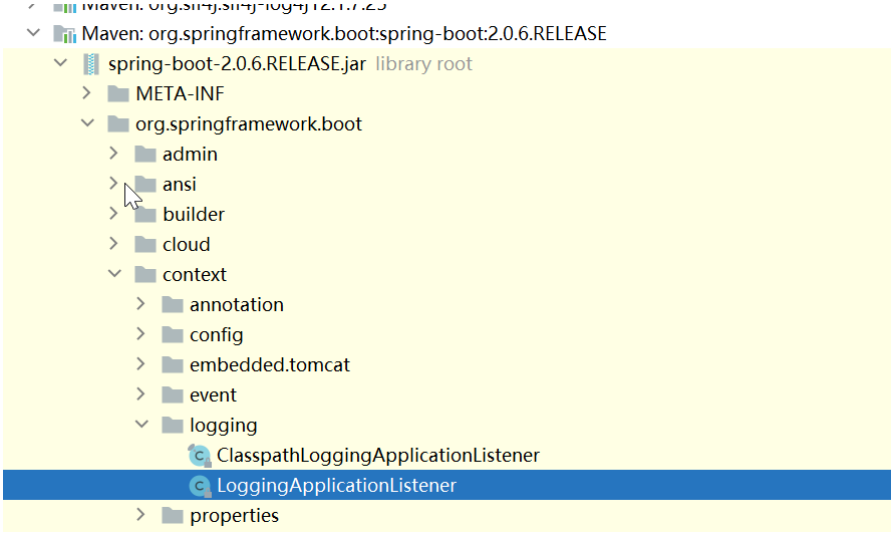

在这段代码中我们看到首先是 获取到ILoggerFactory，这是org.slf4j中的接口，然后判断该factory是否是LoggerContext的实例，这里的LoggerContext是logback中的类。

LogbackLoggingSystem.java | ILoggerFactory.java | Assert.java | St

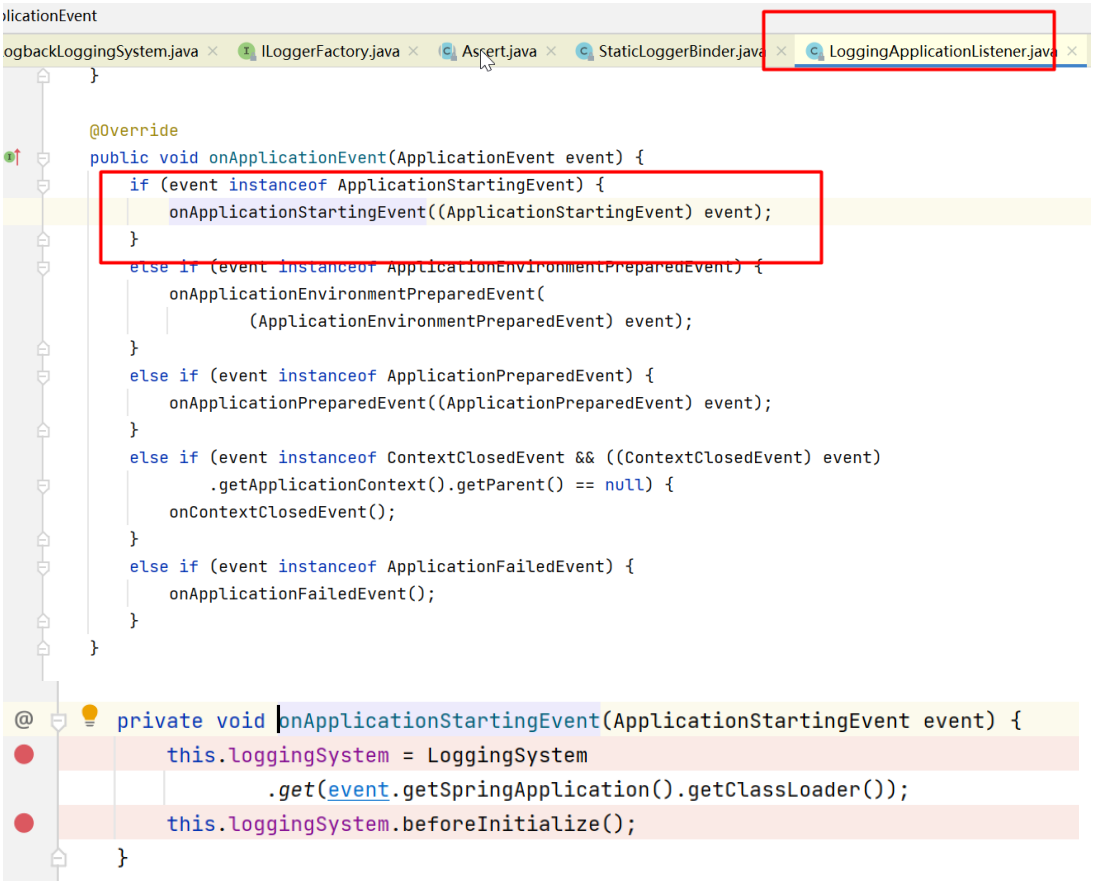
```
1   /.../
25  package org.slf4j;
26
27  /**
28   * <code>ILoggerFactory</code> instances manufacture {@lin
29   * instances by name.
30   *
31   * <p>Most users retrieve {@link Logger} instances through
32   * {@link LoggerFactory#getLogger(String)} method. An inst
33   * interface is bound internally 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：

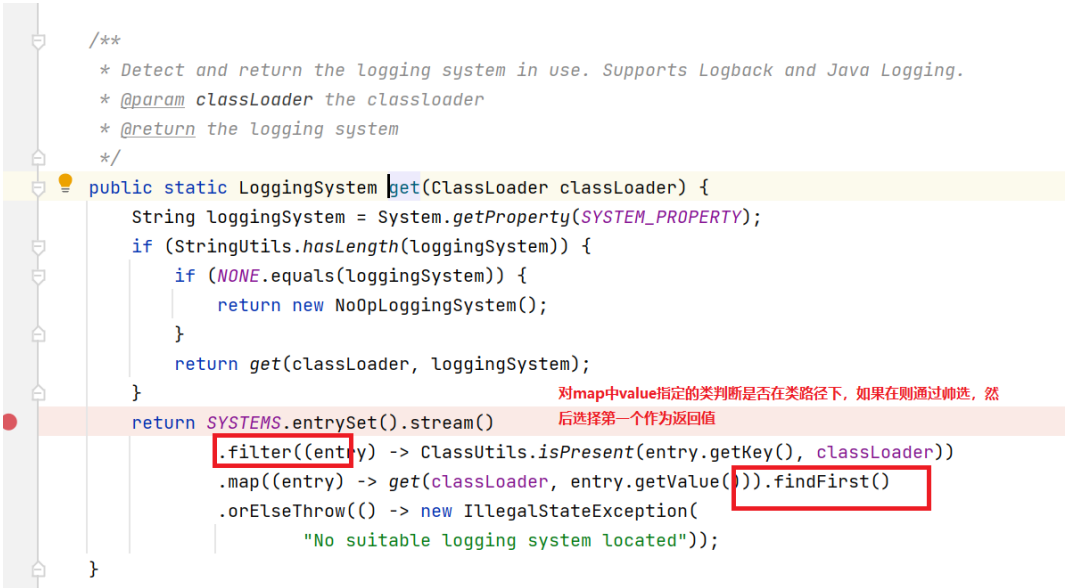
在Spring-boot的jar包中提供了一个LoggingApplicationListener类




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




然后会使用LoggingSystem 类的静态get方法




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。

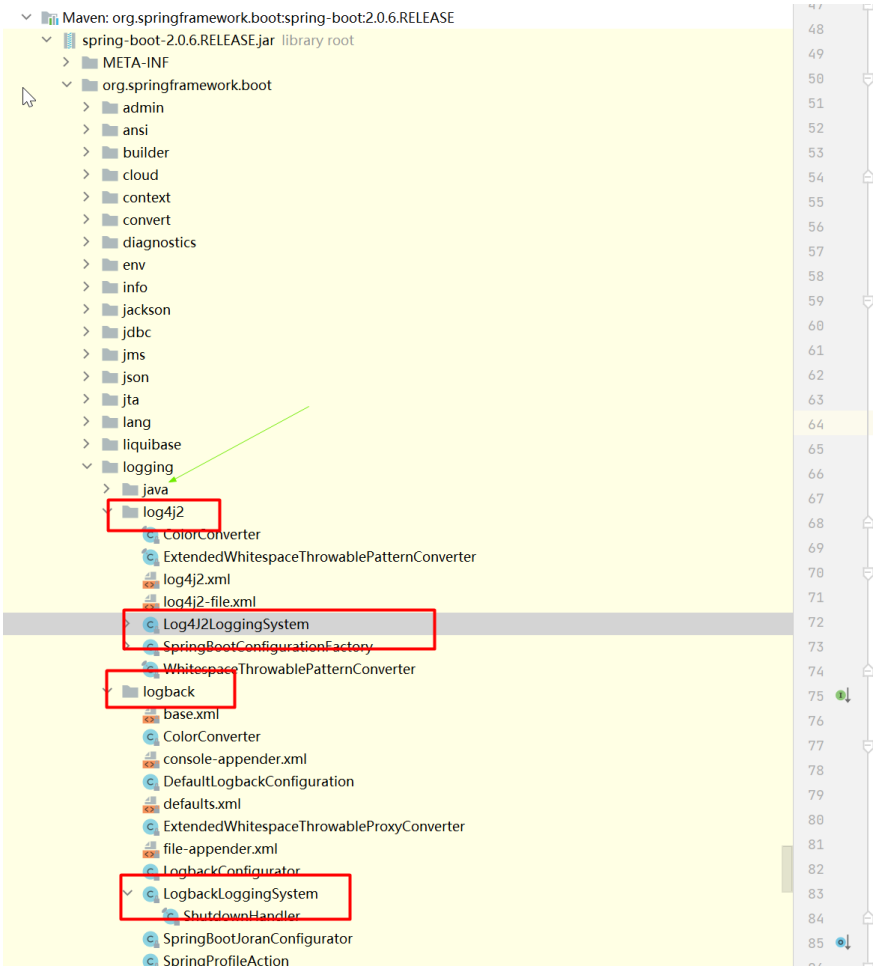
```
    private static final Map<String, String> SYSTEMS;

    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中的元素进行帅选，然后刷选得到第一个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 t
     * @param initializat
     * @param configLocat
     * initialization is
     * @param logFile the
     * console only output
```

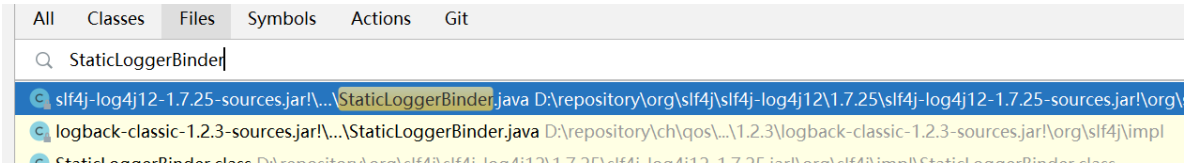
AbstractLoggingSystem (org.springframework.b
JavaLoggingSystem (org.springframework.boot.
Log4J2LoggingSystem (org.springframework.boo
LogbackLoggingSystem (org.springframework.bo
NoOpLoggingSystem in LoggingSystem (org.spri
Slf4JLoggingSystem (org.springframework.boot

在LogbackLoggingSystem的实现中首先getLoggercontext

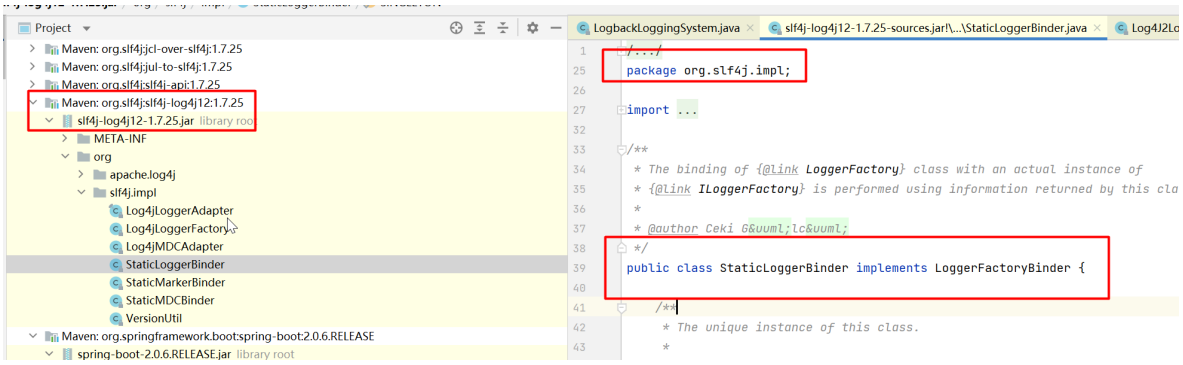
```java
@Override
public void beforeInitialize() {
    LoggerContext loggerContext = getLoggerContext();
    if (isAlreadyInitialized(loggerContext)) {
        return;
    }
    super.beforeInitialize();
    loggerContext.getTurboFilterList().add(FILTER);
}
```

```java
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;
}
```

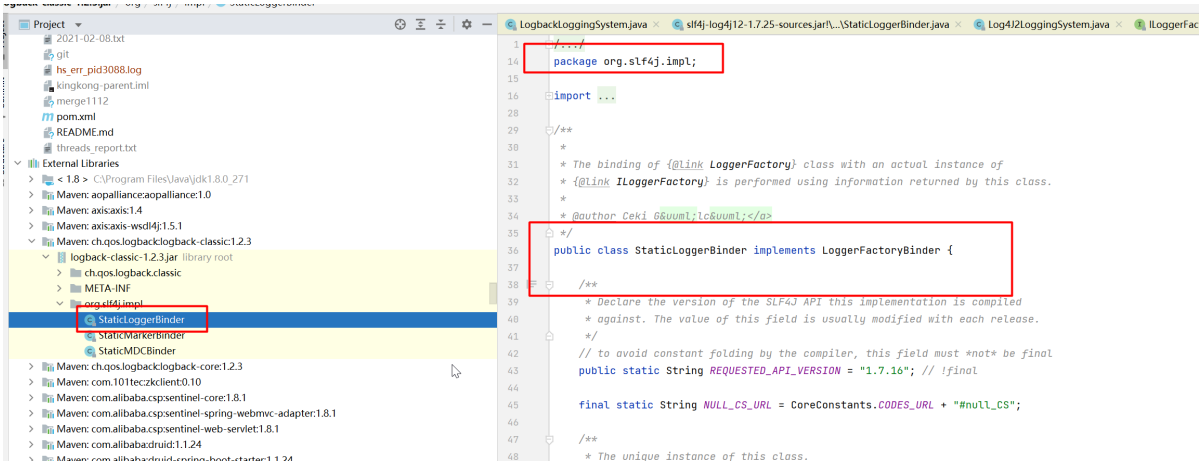
getLoggerContext 中使用了StaticLoggerBinder，经过排查发现 这个类在类路中 不同的jar包中有两个相同包名的同名类，但是其内部实现逻辑完全不同。



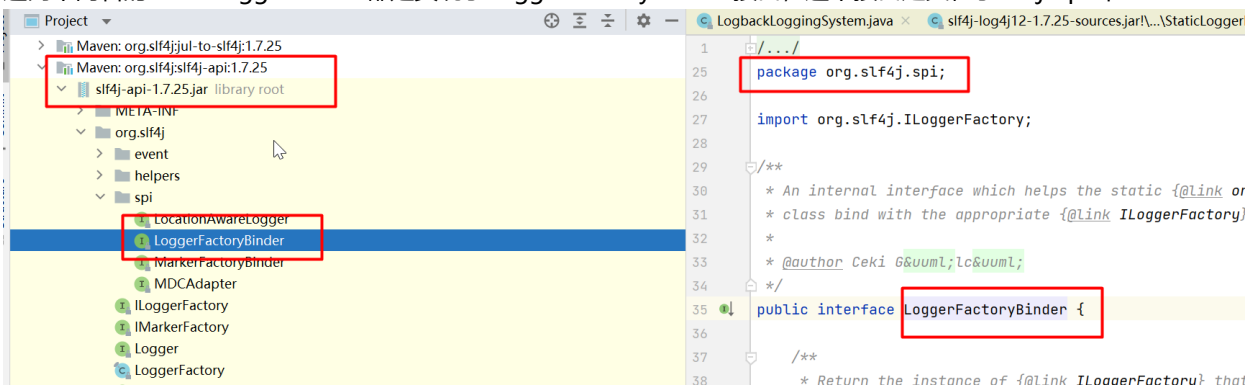

在 slf4j-log4j12 包中存在一个StaticLoggerBinder，如下，其包名叫做 org.slf4j.impl 类名叫做StaticLoggerBinder




在Loggerback-classic jar包中 也有一个StaticLoggerBinder，如下，其包名叫做 org.slf4j.impl 类名叫做StaticLoggerBinder




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

```
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;
}
```

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

```java
LogbackLoggingSystem.java | slf4j-log4j12-1.7.25-sources.jar!\...\StaticLoggerBinder.java | Log4J2LoggingSystem.java | ILogg
 * 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;lc&uuml;
 */
public class StaticLoggerBinder implements LoggerFactoryBinder {

    /**
     * The unique instance of this class.
     *
     */
    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。

```java
LogbackLoggingSystem.java | slf4j-log4j12-1.7.25-sources.jar!\...\StaticLoggerBinder.java | Log4J2LoggingSystem.java
     * The value of this field is modified with each major release.
     */
    // to avoid constant folding by the compiler, this field must *not* be final
    public static String REQUESTED_API_VERSION = "1.6.99"; // !final

    private static final String loggerFactoryClassStr = Log4jLoggerFactory.class.getName();

    /**
     * The ILoggerFactory instance returned by the {@link #getLoggerFactory}
     * method should always be the same object
     */
    private final ILoggerFactory loggerFactory;

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

    public ILoggerFactory getLoggerFactory() { return loggerFactory; }

    public String getLoggerFactoryClassStr() { return loggerFactoryClassStr; }
}
```

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。

```java
    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

```java
/**
 * Logback: the reliable, generic, fast and flexible logging framework.
 * Copyright (C) 1999-2015, QOS.ch. All rights reserved.
 *
 * This program and the accompanying materials are dual-licensed under
 * either the terms of the Eclipse Public License v1.0 as published by
 * the Eclipse Foundation
 *
 *   or (per the licensee's choosing)
 *
 * under the terms of the GNU Lesser General Public License version 2.1
 * as published by the Free Software Foundation.
 */
package org.slf4j.impl;

import ch.qos.logback.core.status.StatusUtil;
import org.slf4j.ILoggerFactory;
import org.slf4j.LoggerFactory;
import org.slf4j.helpers.Util;
import org.slf4j.spi.LoggerFactoryBinder;

import ch.qos.logback.classic.LoggerContext;
import ch.qos.logback.classic.util.ContextInitializer;
import ch.qos.logback.classic.util.ContextSelectorStaticBinder;
import ch.qos.logback.core.CoreConstants;
import ch.qos.logback.core.joran.spi.JoranException;
import ch.qos.logback.core.util.StatusPrinter;

/**
 *
 * 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;lc&uuml;</a>
 */
public class StaticLoggerBinder implements LoggerFactoryBinder {

    /**
     * Declare the version of the SLF4J API this implementation is compiled
     * against. The value of this field is usually modified with each release.
     */
    // to avoid constant folding by the compiler, this field must *not* be final
    public static String REQUESTED_API_VERSION = "1.7.16"; // !final

    final static String NULL_CS_URL = CoreConstants.CODES_URL + "#null_CS";

    /**
     * The unique instance of this class.
     */
    private static StaticLoggerBinder SINGLETON = new StaticLoggerBinder();

    private static Object KEY = new Object();

    static {
        SINGLETON.init();
    }

    private boolean initialized = false;
    private LoggerContext defaultLoggerContext = new LoggerContext();
    private final ContextSelectorStaticBinder contextSelectorBinder = ContextSelectorStaticBinder.getSingleton();

    private StaticLoggerBinder() {
        defaultLoggerContext.setName(CoreConstants.DEFAULT_CONTEXT_NAME);
    }

    public static StaticLoggerBinder getSingleton() {
        return SINGLETON;
    }

    /**
     * Package access for testing purposes.
     */
    static void reset() {
        SINGLETON = new StaticLoggerBinder();
        SINGLETON.init();
    }

    /**
     * Package access for testing purposes.
     */
    void init() {
        try {
            try {
                new ContextInitializer(defaultLoggerContext).autoConfig();
            } catch (JoranException je) {
                Util.report("Failed to auto configure default logger context", je);
            }
            // logback-292
            if (!StatusUtil.contextHasStatusListener(defaultLoggerContext)) {
                StatusPrinter.printInCaseOfErrorsOrWarnings(defaultLoggerContext);
            }
            contextSelectorBinder.init(defaultLoggerContext, KEY);
            initialized = true;
        } catch (Exception t) { // see LOGBACK-1159
            Util.report("Failed to instantiate [" + LoggerContext.class.getName() + "]", t);
        }
    }

    public ILoggerFactory getLoggerFactory() {
        if (!initialized) {
            return defaultLoggerContext;
        }

        if (contextSelectorBinder.getContextSelector() == null) {
            throw new IllegalStateException("contextSelector cannot be null. See also " + NULL_CS_URL);
        }
        return contextSelectorBinder.getContextSelector().getLoggerContext();
    }

    public String getLoggerFactoryClassStr() {
        return contextSelectorBinder.getClass().getName();
    }

}

```

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 <code>LoggerFactory</code> 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/>
 * <code>LoggerFactory</code> is essentially a wrapper around an
 * {@link ILoggerFactory} instance bound with <code>LoggerFactory</code> at
 * compile time.
 * <p/>
 * <p/>
 * Please note that all methods in <code>LoggerFactory</code> are static.
 *
 *
 * @author Alexander Dorokhine
 * @author Robert Elliot
 * @author Ceki G&uuml;lc&uuml;
 *
 */
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。

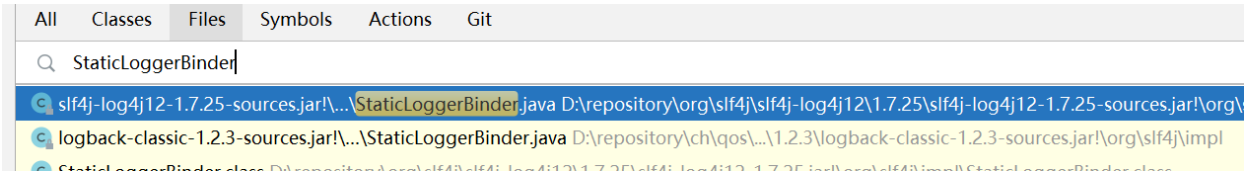
LoggerFactory本质上是一个封装在编译时与LoggerFactory绑定的ILoggerFactory实例的包装器。

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

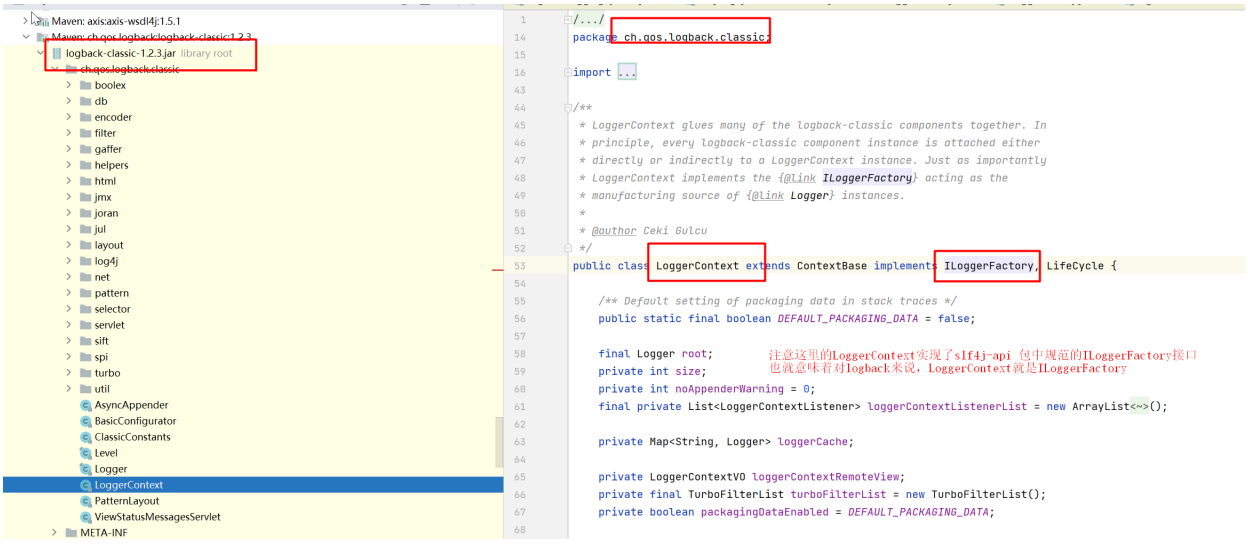
```java
/**
 * Return the {@link ILoggerFactory} instance in use.
 * <p/>
 * <p/>
 * ILoggerFactory instance is bound with this class at compile time.
 *
 * @return the ILoggerFactory instance in use
 */
public static ILoggerFactory getILoggerFactory() {
    if (INITIALIZATION_STATE == UNINITIALIZED)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

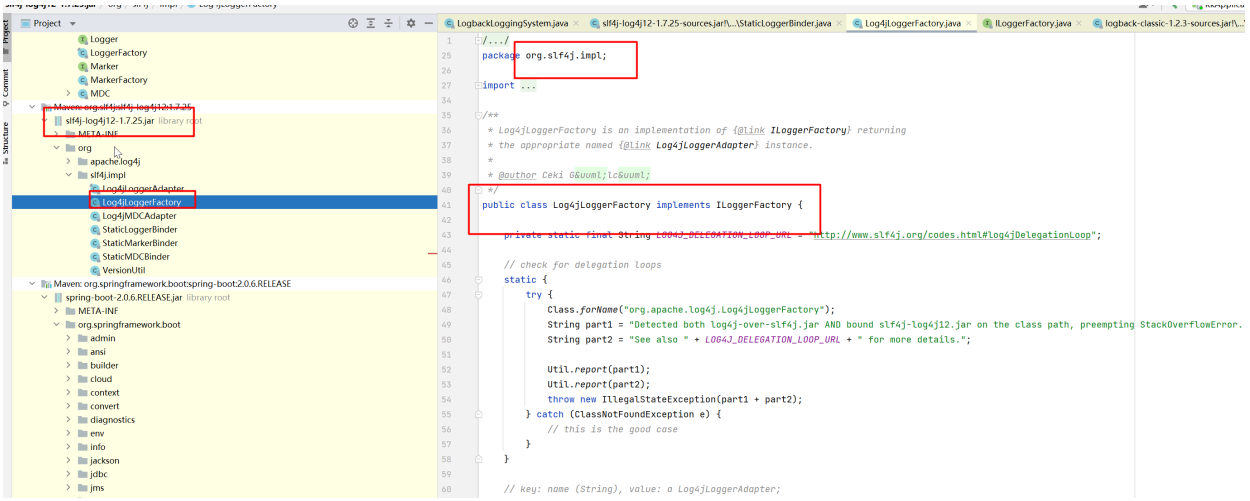
如果没有初始化过则执行初始化，
如果初始化成功了则执行使用StaticLoggerBinder获取LoggerFactory

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




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






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