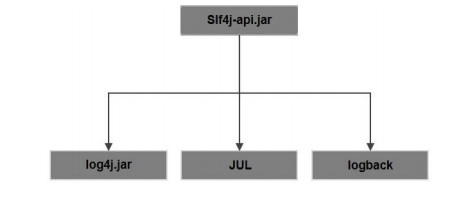
**SLF4J – Краткое руководство**

May 26, 2019

**SLF4J – Обзор**

SLF4J расшифровывается как **S** реализует **F** академию для **J** ava. Он обеспечивает простую абстракцию всех каркасов логирования в Java. Таким образом, он позволяет пользователю работать с любой из сред ведения журналов, таких как Log4j, Logback и JUL (java.util.logging), используя единственную зависимость. Вы можете перейти на необходимую инфраструктуру ведения журналов во время выполнения / развертывания.

Ceki Gülcü создал SLF4J в качестве альтернативы каркасу регистрации в Джакарте.



**Преимущества SLF4J**

Ниже приведены преимущества SLF4J –

* Используя платформу SLF4J, вы можете перейти на желаемую платформу журналирования во время развертывания.
* Slf4J обеспечивает привязки ко всем популярным средам ведения журналов, таким как log4j, JUL, Simple logging и NOP. Поэтому вы можете переключиться на любую из этих популярных платформ во время развертывания.
* SLF4J обеспечивает поддержку параметризованных сообщений регистрации независимо от используемой вами привязки.
* Поскольку SLF4J разделяет каркас приложений и журналов, вы можете легко писать приложения независимо от каркасов журналов. Вам не нужно беспокоиться о каркасе, который используется для написания приложения.
* SLF4J предоставляет простой инструмент Java, известный как мигратор. Используя этот инструмент, вы можете перенести существующие проекты, которые используют каркас ведения журнала, например, Jakarta Commons Logging (JCL) или, log4j или Java.util.logging (JUL) в SLF4J.

**SLF4J – каркасы логгирования**

Вход в программирование, относится к записи действий / событий. Обычно разработчики приложений должны позаботиться о ведении журнала.

Чтобы упростить работу по ведению журналов, Java предоставляет различные среды – log4J, java.util.logging (JUL), крошечный журнал, logback и т. Д.

**Обзор каркаса логирования**

Каркас регистрации обычно содержит три элемента:

**лесоруб**

Захватывает сообщение вместе с метаданными.

**Formatter**

Форматирует сообщения, захваченные регистратором.

**укротитель**

В конце концов, обработчик или приложение отправляет сообщения либо печатая на консоли, либо сохраняя в базе данных, либо отправляя по электронной почте.

Некоторые платформы объединяют элементы logger и appender для ускорения операций.

**Объект Logger**

Для регистрации сообщения приложение отправляет объект регистратора (иногда вместе с исключениями, если таковые имеются) с именем и уровнем безопасности.

**Уровень опасности**

Зарегистрированные сообщения будут иметь различные уровни. В следующей таблице перечислены общие уровни ведения журнала.

|  |  |
| --- | --- |
| **Sr.No** | **Уровень серьезности и описание** |
| 1 | **FATAL** Серьезная проблема, которая приводит к завершению приложения. |
| 2 | **ERROR** Ошибки во время выполнения. |
| 3 | **WARN** В большинстве случаев ошибки связаны с использованием устаревших API. |
| 4 | **INFO** События, которые происходят во время выполнения. |
| 5 | **DEBUG** Информация о потоке системы. |
| 6 | **TRACE** Более подробная информация о работе системы. |

**SLF4J против Log4j**

**Что такое log4j?**

log4j – это надежная, быстрая и гибкая **среда ведения журналов (API), написанная на Java** , которая распространяется по лицензии Apache Software License.

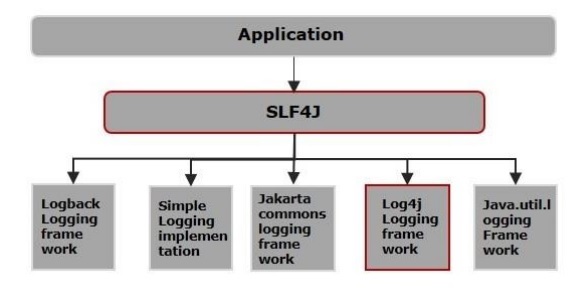
log4j легко настраивается с помощью внешних файлов конфигурации во время выполнения. Он рассматривает процесс ведения журнала с точки зрения уровней приоритетов и предлагает механизмы для направления информации регистрации в самые разные пункты назначения, такие как база данных, файл, консоль, системный журнал UNIX и т. Д. (Более подробную информацию о log4j см. В нашем руководстве).

**Сравнение SLF4J и Log4j**

В отличие от log4j, SLF4J ( **S** реализует язык для **J** ava) не является реализацией каркаса логирования, это **абстракция для всех тех каркасов логирования в Java, аналогичных log4J** . Таким образом, вы не можете сравнить оба. Однако всегда трудно выбрать один из двух.

Если у вас есть выбор, абстракция ведения журнала всегда предпочтительнее, чем структура ведения журнала. Если вы используете абстракцию журналирования, в частности SLF4J, вы можете перейти на любую инфраструктуру журналирования, которая вам нужна во время развертывания, не выбирая одиночную зависимость.

Обратите внимание на следующую диаграмму для лучшего понимания.



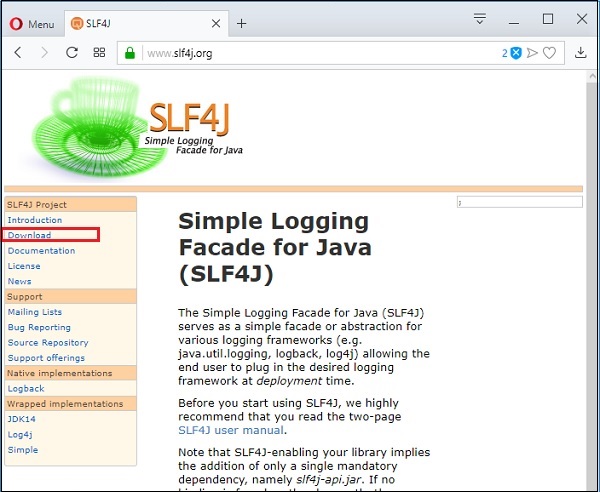
**SLF4J – Настройка среды**

В этой главе мы объясним, как настроить среду SLF4J в Eclipse IDE. Прежде чем продолжить установку, убедитесь, что в вашей системе уже установлен Eclipse. Если нет, скачайте и установите Eclipse.

Для получения дополнительной информации об Eclipse, пожалуйста, обратитесь к нашему [учебному пособию по Eclipse.](https://translate.googleusercontent.com/translate_c?depth=1&rurl=translate.google.ru&sl=en&sp=nmt4&tl=ru&u=https://www.tutorialspoint.com/eclipse/index.htm&xid=17259,15700022,15700186,15700190,15700256,15700259,15700262&usg=ALkJrhhGUkl8qQJt0KsOO0ybWKgFFdofrw)

**Шаг 1: Загрузите JAR-файл зависимости**

Откройте официальную [домашнюю страницу](https://translate.googleusercontent.com/translate_c?depth=1&rurl=translate.google.ru&sl=en&sp=nmt4&tl=ru&u=https://www.slf4j.org/&xid=17259,15700022,15700186,15700190,15700256,15700259,15700262&usg=ALkJrhg5WBFyyDk92zQqiIfVevR_WntG2w) сайта SLF4J и перейдите на страницу загрузки.

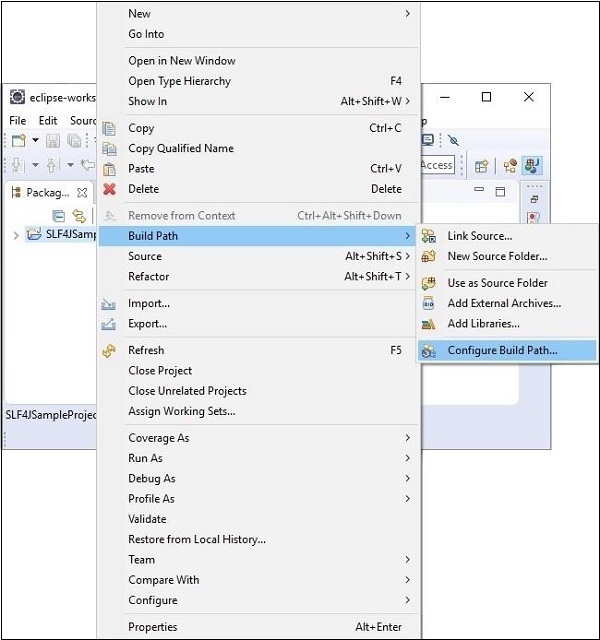


Теперь загрузите последнюю стабильную версию ***slf4j-XXtar.gz*** или *slf4j-XXzip в* соответствии с вашей операционной системой (если это файл .zip для Windows или файл tar.gz для Linux).

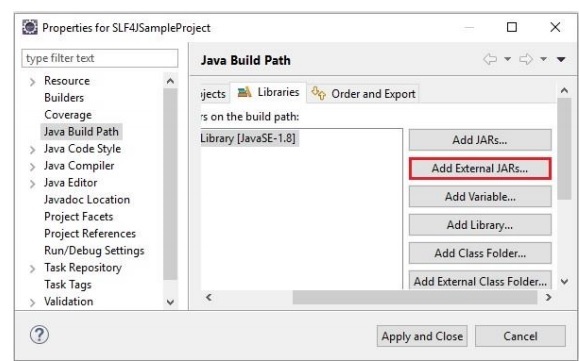
В загруженной папке вы найдете *slf4j-api-XXjar* . Это необходимый файл Jar.

**Шаг 2: Создайте проект и установите путь сборки**

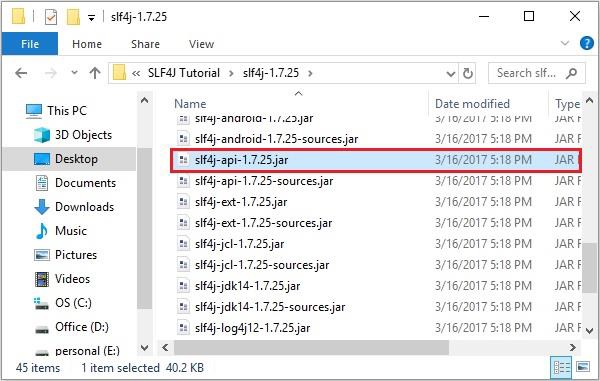
Откройте затмение и создайте пример проекта. Щелкните правой кнопкой мыши по проекту, выберите опцию **Build Path → Configure Build Path…,** как показано ниже.



Во фрейме **пути сборки Java** на вкладке « **Библиотеки** » нажмите « **Добавить внешние JAR-файлы».**

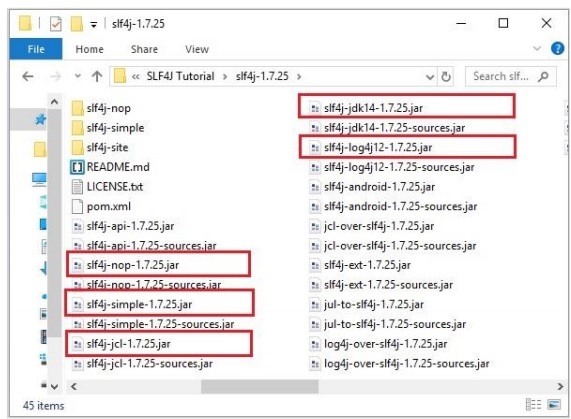


Выберите **загруженный** файл **slf4j-api.xxjar** и нажмите « **Применить и закрыть»** .



**SLF4J Привязки**

В дополнение к **файлу slf4j-api.xxjar** , **SLF4J** предоставляет несколько других файлов Jar, как показано ниже. Это так называемые **привязки SLF4J** .



Где каждая привязка для соответствующей структуры ведения журнала.

В следующей таблице перечислены привязки SLF4J и соответствующие им каркасы.

|  |  |
| --- | --- |
| **Sr.No** | **Jar file & Logging Framework** |
| 1 | **SLF4J-NOP-xxjar** Нет операции, удаляет все записи. |
| 2 | **SLF4J-простой xxjar** Простая реализация, где печатаются сообщения для информации и выше, и все выходы сохраняются в System.err. |
| 3 | **SLF4J-JCL-xxjar** Каркас лесозаготовок Джакарты. |
| 4 | **SLF4J-jdk14-xxjar** Java.util.logging Framework (JUL). |
| 5 | **SLF4J-log4j12-xxjar** Каркас Log4J работает. Кроме того, вам нужно иметь **log4j.jar** . |

Чтобы заставить SLF4J работать вместе с slf4l-api-xxjar, вам нужно добавить соответствующий Jar-файл (привязку) желаемой среды логгера в путь к классам проекта (установить путь сборки).

Для переключения с одного фреймворка на другой необходимо заменить соответствующую привязку. Если границы не найдены, по умолчанию используется режим без операций.

**Pom.xml для SLF4J**

Если вы создаете проект maven, откройте **файл pom.xml,** вставьте в него следующее содержимое и обновите проект.

<project xmlns = "http://maven.apache.org/POM/4.0.0"

xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation = "http://maven.apache.org/POM/4.0.0

http://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>Sample</groupId>

<artifactId>Sample</artifactId>

<version>0.0.1-SNAPSHOT</version>

<build>

<sourceDirectory>src</sourceDirectory>

<plugins>

<plugin>

<artifactId>maven-compiler-plugin</artifactId>

<version>3.7.0</version>

<configuration>

<source>1.8</source>

<target>1.8</target>

</configuration>

</plugin>

</plugins>

</build>

<dependencies>

<dependency>

<groupId>org.slf4j</groupId>

<artifactId>slf4j-api</artifactId>

<version>1.7.25</version>

</dependency>

</dependencies>

</project>

**SLF4J – ссылочный API**

В этой главе мы обсудим классы и методы, которые мы будем использовать в последующих главах этого урока.

**Интерфейс логгера**

Интерфейс регистратора пакета **org.slf4j** является точкой входа в API SLF4J. Ниже перечислены важные методы этого интерфейса.

|  |  |
| --- | --- |
| **Sr.No.** | **Методы и описание** |
| 1 | **void debug (String msg)** Этот метод регистрирует сообщение на уровне DEBUG. |
| 2 | **void error (String msg)** Этот метод регистрирует сообщение на уровне ОШИБКИ. |
| 3 | **void info (Строка сообщения)** Этот метод регистрирует сообщение на уровне INFO. |
| 4 | **пустая трассировка (String msg)** Этот метод регистрирует сообщение на уровне TRACE. |
| 5 | **void warn (String msg)** Этот метод регистрирует сообщение на уровне WARN. |

**LoggerFactory Class**

Класс **LoggerFactory** пакета **org.slf4j** – это служебный класс, который используется для создания регистраторов для различных API-интерфейсов журналирования, таких как log4j, JUL, NOP и простой регистратор.

|  |  |
| --- | --- |
| **Sr.No.** | **Метод и описание** |
| 1 | **Logger getLogger (имя строки)**  Этот метод принимает строковое значение, представляющее имя, и возвращает объект **Logger** с указанным именем. |

**Профилировщик Класс**

Этот класс принадлежит пакету **org.slf4j,** который используется для профилирования, и известен как профилировщик бедняков. Используя это, программист может узнать время, затраченное на выполнение длительных задач.

Ниже приведены важные методы этого класса.

|  |  |
| --- | --- |
| **Sr.No.** | **Методы и описание** |
| 1 | **пустое начало (строковое имя)** Этот метод запускает новый дочерний секундомер (с именем) и останавливает более ранние дочерние секундомеры (или временные инструменты). |
| 2 | **TimeInstrument stop ()** Этот метод остановит недавний дочерний секундомер и глобальный секундомер и вернет текущий инструмент времени. |
| 3 | **void setLogger (Logger logger)** Этот метод принимает объект Logger и связывает указанный регистратор с текущим профилировщиком. |
| 4 | **журнал пустот ()** Регистрирует содержимое инструмента текущего времени, связанного с регистратором. |
| 5 | **void print ()** Печатает содержимое инструмента текущего времени. |

**SLF4J – Привет, мир**

В этой главе мы увидим простую базовую программу-регистратор, использующую SLF4J. Выполните шаги, описанные ниже, чтобы написать простой регистратор.

**Шаг 1 – Создайте объект интерфейса slf4j.Logger**

Поскольку **slf4j.Logger** является точкой входа в API SLF4J, сначала вам нужно получить / создать его объект

Метод **getLogger ()** класса **LoggerFactory** принимает строковое значение, представляющее имя, и возвращает объект **Logger** с указанным именем.

Logger logger = LoggerFactory.getLogger("SampleLogger");

**Шаг 2 – Зарегистрируйте необходимое сообщение**

Метод **info ()** интерфейса **slf4j.Logger** принимает строковое значение, представляющее требуемое сообщение, и записывает его на информационном уровне.

logger.info("Hi This is my first SLF4J program");

**пример**

Ниже приведена программа, которая демонстрирует, как написать образец регистратора на Java с использованием SLF4J.

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

public class SLF4JExample {

public static void main(String[] args) {

//Creating the Logger object

Logger logger = LoggerFactory.getLogger("SampleLogger");

//Logging the information

logger.info("Hi This is my first SLF4J program");

}

}

**Выход**

При первоначальном запуске следующей программы вы получите следующий вывод вместо желаемого сообщения.

SLF4J: Failed to load class "org.slf4j.impl.StaticLoggerBinder".

SLF4J: Defaulting to no-operation (NOP) logger implementation

SLF4J: See http://www.slf4j.org/codes.html#StaticLoggerBinder for further

details.

Поскольку мы не установили classpath для какой-либо привязки, представляющей структуру ведения журнала, как упоминалось ранее в этом руководстве, SLF4J по умолчанию стал реализацией без операций. Итак, чтобы увидеть сообщение, нужно добавить желаемую привязку в classpath проекта. Поскольку мы используем eclipse, задайте **путь сборки** для соответствующего файла JAR или добавьте его зависимость в файл pom.xml.

Например, если нам нужно использовать JUL (инфраструктура Java.util.logging), нам нужно установить путь сборки для файла jar **slf4j-jdk14-xxjar** . И если мы хотим использовать каркас журналирования log4J, нам нужно установить путь сборки или добавить зависимости для jar-файлов **slf4j-log4j12-xxjar** и **log4j.jar** .

После добавления в проект привязки, представляющей любую из каркасов ведения журналов, кроме **slf4j-nopx.x.jar** (classpath), вы получите следующий вывод.

Dec 06, 2018 5:29:44 PM SLF4JExample main

INFO: Hi Welcome to Tutorialspoint

**SLF4J – Сообщения об ошибках**

В этой главе мы обсудим различные сообщения об ошибках или предупреждения, которые мы получаем при работе с SLF4J, и причины / значения этих сообщений.

**Не удалось загрузить класс “org.slf4j.impl.StaticLoggerBinder”.**

Это предупреждение, которое вызывается, если в пути к классам нет привязок SLF4J.

Ниже приводится полное предупреждение –

SLF4J: Failed to load class "org.slf4j.impl.StaticLoggerBinder".

SLF4J: Defaulting to no-operation (NOP) logger implementation

SLF4J: See http://www.slf4j.org/codes.html#StaticLoggerBinder for further

details.

Чтобы решить эту проблему, необходимо добавить одну из привязок каркаса ведения журнала. Это объясняется в главе **Hello World** этого урока.

**Примечание.** Это происходит в версиях SLF4J между 1.6.0 и 1.8.0-бета2.

**Поставщики SLF4J не найдены**

В slf4j-1.8.0-beta2 вышеупомянутое предупреждение более ясно говорит, что **«поставщики SLF4J не были найдены»** .

Ниже приводится полное предупреждение –

SLF4J: No SLF4J providers were found.

SLF4J: Defaulting to no-operation (NOP) logger implementation

SLF4J: See http://www.slf4j.org/codes.html#noProviders for further details.

**Classpath содержит привязки SLF4J для версий slf4j-api до 1.8**

Если вы используете версию SLF4J 1.8 и у вас есть привязки предыдущих версий в classpath, но не привязки 1.8, вы увидите предупреждение, как показано ниже.

SLF4J: No SLF4J providers were found.

SLF4J: Defaulting to no-operation (NOP) logger implementation

SLF4J: See http://www.slf4j.org/codes.html#noProviders for further details.

SLF4J: Class path contains SLF4J bindings targeting slf4j-api versions prior to

1.8.

SLF4J: Ignoring binding found at

[jar:file:/C:/Users/Tutorialspoint/Desktop/Latest%20Tutorials/SLF4J%20Tutorial/

slf4j-1.7.25/slf4j-jdk14-1.7.25.jar!/org/slf4j/impl/StaticLoggerBinder.class]

SLF4J: See http://www.slf4j.org/codes.html#ignoredBindings for an explanation.

**NoClassDefFoundError: org / apache / commons / logging / LogFactory**

Если вы работаете с **slf4j-jcl** и у вас есть только **slf4j-jcl.jar** в вашем пути к классам, вы получите исключение, такое как приведенное ниже.

Exception in thread "main" java.lang.NoClassDefFoundError:

org/apache/commons/logging/LogFactory

at org.slf4j.impl.JCLLoggerFactory.getLogger(JCLLoggerFactory.java:77)

at org.slf4j.LoggerFactory.getLogger(LoggerFactory.java:358)

at SLF4JExample.main(SLF4JExample.java:8)

Caused by: java.lang.ClassNotFoundException:

org.apache.commons.logging.LogFactory

at java.net.URLClassLoader.findClass(Unknown Source)

at java.lang.ClassLoader.loadClass(Unknown Source)

at sun.misc.Launcher$AppClassLoader.loadClass(Unknown Source)

at java.lang.ClassLoader.loadClass(Unknown Source)

... 3 more

Чтобы решить эту проблему, вам нужно добавить **commons-logging.jar** к вашему classpath.

**Обнаружил как jcl-over-slf4j.jar, так и связанный slf4j-jcl.jar на пути к классам.**

Привязка **slf4j-jcl.jar** перенаправляет вызовы регистратора slf4j в JCL, а **jcl-over-slf4j.jar** перенаправляет вызовы регистратора JCL в slf4j. Таким образом, вы не можете иметь оба в classpath вашего проекта. Если вы сделаете это, вы получите исключение, такое как приведенное ниже.

SLF4J: Detected both jcl-over-slf4j.jar AND bound slf4j-jcl.jar on the class

path, preempting StackOverflowError.

SLF4J: See also http://www.slf4j.org/codes.html#jclDelegationLoop for more

details.

Exception in thread "main" java.lang.ExceptionInInitializerError

at org.slf4j.impl.StaticLoggerBinder.<init>(StaticLoggerBinder.java:71)

at org.slf4j.impl.StaticLoggerBinder.<clinit>(StaticLoggerBinder.java:42)

at org.slf4j.LoggerFactory.bind(LoggerFactory.java:150)

at org.slf4j.LoggerFactory.performInitialization(LoggerFactory.java:124)

at org.slf4j.LoggerFactory.getILoggerFactory(LoggerFactory.java:412)

at org.slf4j.LoggerFactory.getLogger(LoggerFactory.java:357)

at SLF4JExample.main(SLF4JExample.java:8)

Caused by: java.lang.IllegalStateException: Detected both jcl-over-slf4j.jar

AND bound slf4j-jcl.jar on the class path, preempting StackOverflowError. See

also http://www.slf4j.org/codes.html#jclDelegationLoop for more details.

at org.slf4j.impl.JCLLoggerFactory.<clinit>(JCLLoggerFactory.java:54)

... 7 more

Чтобы решить эту проблему, удалите любой из файлов JAR.

**Обнаружено несоответствие имени регистратора**

Вы можете создать объект Logger с помощью –

* Передача имени созданного регистратора в качестве аргумента **методу getLogger ()** .
* Передача класса в качестве аргумента этому методу.

Если вы пытаетесь создать объект фабрики регистратора, передавая класс в качестве аргумента, и если вы установили системное свойство **slf4j.detectLoggerNameMismatch** в значение true, тогда имя класса, которое вы передаете в качестве аргумента **методу getLogger ()** и используемый вами класс должен быть таким же, в противном случае вы получите следующее предупреждение –

*«Обнаружено несоответствие имени регистратора.*

Рассмотрим следующий пример.

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

public class SLF4JExample {

public static void main(String[] args) {

System.setProperty("slf4j.detectLoggerNameMismatch", "true");

//Creating the Logger object

Logger logger = LoggerFactory.getLogger(Sample.class);

//Logging the information

logger.info("Hi Welcome to Tutorilspoint");

}

}

Здесь мы установили для свойства *slf4j.detectLoggerNameMismatch* значение true. Имя класса, которое мы использовали, – **SLF4JExample,** а имя класса, которое мы передали методу getLogger (), – **Sample,** так как они оба не равны, мы получим следующее предупреждение.

SLF4J: Detected logger name mismatch. Given name: "Sample"; computed name:

"SLF4JExample".

SLF4J: See http://www.slf4j.org/codes.html#loggerNameMismatch for an

explanation

Dec 10, 2018 12:43:00 PM SLF4JExample main

INFO: Hi Welcome to Tutorilspoint

**Примечание** – это происходит после slf4j 1.7.9

**Classpath содержит несколько привязок SLF4J.**

У вас должна быть только одна привязка в пути к классам. Если у вас есть несколько привязок, вы получите предупреждение с указанием привязок и их местоположения.

Предположим, что если у нас есть привязки **slf4j-jdk14.jar** и **slf4j-nop.jar** в пути к классам, мы получим следующее предупреждение.

SLF4J: Class path contains multiple SLF4J bindings.

SLF4J: Found binding in

[jar:file:/C:/Users/Tutorialspoint/Desktop/Latest%20Tutorials/SLF4J%20Tutorial/

slf4j-1.7.25/slf4j-nop-1.7.25.jar!/org/slf4j/impl/StaticLoggerBinder.class]

SLF4J: Found binding in

[jar:file:/C:/Users/Tutorialspoint/Desktop/Latest%20Tutorials/SLF4J%20Tutorial/

slf4j-1.7.25/slf4j-jdk14-1.7.25.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.helpers.NOPLoggerFactory]

**Обнаружены оба log4j-over-slf4j.jar И привязанные slf4j-log4j12.jar на пути к классам**

Чтобы перенаправить вызовы log4j logger в slf4j, вам нужно использовать **привязку log4j-over-slf4j.jar,** а если вы хотите перенаправить вызовы slf4j в log4j, вам необходимо использовать **привязку slf4j-log4j12.jar** .

Таким образом, вы не можете иметь оба в classpath. Если вы это сделаете, вы получите следующее исключение.

SLF4J: Detected both log4j-over-slf4j.jar AND bound slf4j-log4j12.jar on the

class path, preempting StackOverflowError.

SLF4J: See also http://www.slf4j.org/codes.html#log4jDelegationLoop for more

details.

Exception in thread "main" java.lang.ExceptionInInitializerError

at org.slf4j.impl.StaticLoggerBinder.<init>(StaticLoggerBinder.java:72)

at org.slf4j.impl.StaticLoggerBinder.<clinit>(StaticLoggerBinder.java:45)

at org.slf4j.LoggerFactory.bind(LoggerFactory.java:150)

at org.slf4j.LoggerFactory.performInitialization(LoggerFactory.java:124)

at org.slf4j.LoggerFactory.getILoggerFactory(LoggerFactory.java:412)

at org.slf4j.LoggerFactory.getLogger(LoggerFactory.java:357)

at org.slf4j.LoggerFactory.getLogger(LoggerFactory.java:383)

at SLF4JExample.main(SLF4JExample.java:8)

Caused by: java.lang.IllegalStateException: Detected both log4j-over-slf4j.jar

AND bound slf4j-log4j12.jar on the class path, preempting StackOverflowError.

See also http://www.slf4j.org/codes.html#log4jDelegationLoop for more details.

**SLF4J – параметризованная регистрация**

Как обсуждалось ранее в этом руководстве, SLF4J обеспечивает поддержку параметризованных сообщений журнала.

Вы можете использовать параметры в сообщениях и передавать им значения позже в том же операторе.

**Синтаксис**

Как показано ниже, вам нужно использовать заполнители ({}) в сообщении (String) везде, где вам нужно, и позже вы можете передать значение для заполнителя в форме **объекта** , разделяя сообщение и значение запятой.

Integer age;

Logger.info("At the age of {} ramu got his first job", age);

**пример**

В следующем примере демонстрируется параметризованное ведение журнала (с одним параметром) с использованием SLF4J.

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

public class PlaceHolders {

public static void main(String[] args) {

//Creating the Logger object

Logger logger = LoggerFactory.getLogger(PlaceHolders.class);

Integer age = 23;

//Logging the information

logger.info("At the age of {} ramu got his first job", age);

}

}

**Выход**

После выполнения вышеуказанная программа генерирует следующий вывод:

Dec 10, 2018 3:25:45 PM PlaceHolders main

INFO: At the age of 23 Ramu got his first job

**Преимущество параметризованной регистрации**

В Java, если нам нужно вывести значения в операторе, мы будем использовать оператор конкатенации как –

System.out.println("At the age of "+23+" ramu got his first job");

Это включает в себя преобразование целочисленного значения 23 в строку и конкатенацию этого значения в окружающие его строки.

И если это оператор ведения журнала, и если этот конкретный уровень ведения журнала вашего оператора отключен, то все эти вычисления будут бесполезны.

В таких случаях вы можете использовать параметризованное ведение журнала. В этом формате изначально SLF4J подтверждает, включено ли ведение журнала для определенного уровня. Если так, то он заменяет заполнители в сообщениях соответствующими значениями.

Например, если у нас есть утверждение как

Integer age;

Logger.debug("At the age of {} ramu got his first job", age);

Только если отладка включена, SLF4J преобразует возраст в целое число и в противном случае объединяет его со строками, он ничего не делает. Таким образом, расходы на создание параметров при отключенном уровне ведения журнала.

**Вариант с двумя аргументами**

Вы также можете использовать два параметра в сообщении как –

logger.info("Old weight is {}. new weight is {}.", oldWeight, newWeight);

**пример**

В следующем примере демонстрируется использование двух заполнителей в параметризованном ведении журнала.

import java.util.Scanner;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

public class PlaceHolders {

public static void main(String[] args) {

Integer oldWeight;

Integer newWeight;

Scanner sc = new Scanner(System.in);

System.out.println("Enter old weight:");

oldWeight = sc.nextInt();

System.out.println("Enter new weight:");

newWeight = sc.nextInt();

//Creating the Logger object

Logger logger = LoggerFactory.getLogger(Sample.class);

//Logging the information

logger.info("Old weight is {}. new weight is {}.", oldWeight, newWeight);

//Logging the information

logger.info("After the program weight reduced is: "+(oldWeight-newWeight));

}

}

**Выход**

После выполнения вышеуказанная программа генерирует следующий вывод.

Enter old weight:

85

Enter new weight:

74

Dec 10, 2018 4:12:31 PM PlaceHolders main

INFO: Old weight is 85. new weight is 74.

Dec 10, 2018 4:12:31 PM PlaceHolders main

INFO: After the program weight reduced is: 11

**Вариант с несколькими аргументами**

Вы также можете использовать более двух заполнителей, как показано в следующем примере:

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

public class PlaceHolders {

public static void main(String[] args) {

Integer age = 24;

String designation = "Software Engineer";

String company = "Infosys";

//Creating the Logger object

Logger logger = LoggerFactory.getLogger(Sample.class);

//Logging the information

logger.info("At the age of {} ramu got his first job as a {} at {}", age, designation, company);

}

}

**Выход**

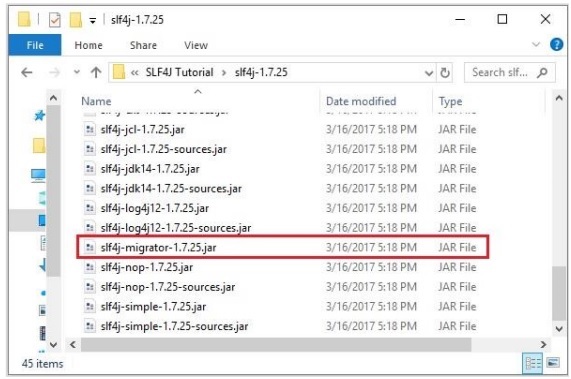
После выполнения вышеуказанная программа генерирует следующий вывод:

Dec 10, 2018 4:23:52 PM PlaceHolders main

INFO: At the age of 24 ramu got his first job as a Software Engineer at Infosys

**SLF4J – Мигратор**

Если у вас есть проект в Jakarta Commons Logging (JCL) или, log4j или java.util.logging (JUL), и вы хотите преобразовать эти проекты в SLF4J, вы можете сделать это с помощью инструмента миграции, предоставленного в дистрибутиве SLF4J.



**Запуск SLF4J Migrator**

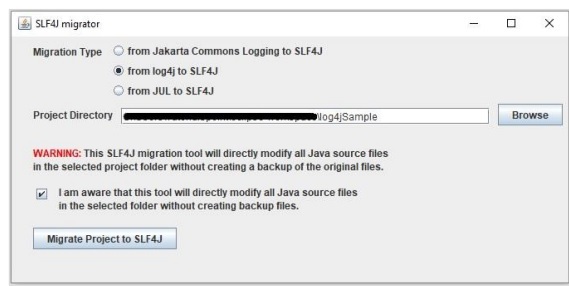
SLF4J – это простой одиночный файл jar (slf4j-migrator.jar), и вы можете запустить его с помощью команды java –jar.

Чтобы запустить его, в командной строке перейдите в каталог, в котором находится этот jar-файл, и выполните следующую команду.

java -jar slf4j-migrator-1.8.0-beta2.jar

Starting SLF4J Migrator

Это запустит миграцию, и вы можете увидеть отдельное Java-приложение как –



Как указано в окне, вам нужно проверить тип миграции, которую вы хотите сделать, выбрать каталог проекта и нажать кнопку «Перенести проект в SLF4J».

Этот инструмент обращается к исходным файлам, которые вы предоставляете, и выполняет простые модификации, такие как изменение строк импорта и объявлений регистратора с текущей среды ведения журнала на SLF4j.

**пример**

Например, предположим, у нас есть пример **проекта log4j (2)** в eclipse с одним файлом следующим образом:

import org.apache.log4j.Logger;

import java.io.\*;

import java.sql.SQLException;

import java.util.\*;

public class Sample {

/\* Get actual class name to be printed on \*/

static Logger log = Logger.getLogger(Sample.class.getName());

public static void main(String[] args)throws IOException,SQLException {

log.debug("Hello this is a debug message");

log.info("Hello this is an info message");

}

}

Чтобы перенести пример **проекта log4j (2)** в slf4j, нам нужно установить переключатель **с log4j на slf4j,** выбрать каталог проекта и нажать « **Выход»** для переноса.



Мигратор изменил приведенный выше код следующим образом. Вот если вы заметили, что операторы import и logger были изменены.

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import java.io.\*;

import java.sql.SQLException;

import java.util.\*;

public class Sample {

static Logger log = LoggerFactory.getLogger(Sample.class.getName());

public static void main(String[] args)throws IOException,SQLException {

log.debug("Hello this is a debug message");

log.info("Hello this is an info message");

}

}

Поскольку у вас уже есть **log4j.jar** в вашем проекте, вам нужно добавить **файлы slf4j-api.jar** и **slf4jlog12.jar** в проект, чтобы выполнить его.

**Ограничения SLF4JMigrator**

Ниже приведены ограничения SLF4J-мигратора.

* Migrator не будет изменять скрипты сборки, такие как ant, maven и ivy, вам нужно сделать это самостоятельно.
* Мигратор не поддерживает сообщения, кроме типа String.
* Мигратор не поддерживает уровень FATAL.
* При работе с log4j migrator не будет переносить вызовы в PropertyConfigurator или DomConfigurator.

**SLF4J – Профилирование**

SLF4J Distribution предоставляет **slf4j-ext.jar, который** содержит API-интерфейсы для таких функций, как профилирование, расширенное ведение журнала, ведение журнала событий и ведение журнала с помощью Java-агента.

**профилирование**

Иногда программист хочет измерить некоторые атрибуты, такие как использование памяти, сложность времени или использование определенных инструкций о программах, чтобы измерить реальные возможности этой программы. Такой вид измерения в программе называется профилированием. Профилирование использует динамический анализ программы, чтобы сделать такое измерение.

SLF4J предоставляет класс с именем **Profiler** в пакете **org.slf4j.profiler** для целей профилирования. Это известно как профиль бедняков. Используя это, программист может узнать время, затраченное на выполнение длительных задач.

**Профилирование с использованием класса Profiler**

Профилировщик содержит секундомеры и дочерние секундомеры, и мы можем запускать и останавливать их, используя методы, предоставляемые классом профилировщика.

Чтобы продолжить профилирование с использованием класса профилировщика, выполните следующие действия.

**Шаг 1 – Создание класса профилировщика**

Создайте экземпляр класса Profiler, передав значение String, представляющее имя профилировщика. Когда мы создаем экземпляр класса Profiler, запускается глобальный секундомер.

//Creating a profiler

Profiler profiler = new Profiler("Sample");

**Шаг 2 – Запустите детский секундомер**

Когда мы вызываем метод **start (),** он запускает новый дочерний секундомер (именованный) и останавливает более ранние дочерние секундомеры (или инструменты времени).

Вызовите метод **start ()** класса **Profiler** , передав значение String, представляющее имя создаваемого дочернего секундомера.

//Starting a child stopwatch and stopping the previous one.

profiler.start("Task 1");

obj.demoMethod1();

После создания этих секундомеров вы можете выполнять свои задачи или вызывать те методы, которые запускают ваши задачи.

**Шаг 3: Запустите другой детский секундомер (если хотите)**

Если вам нужно, создайте еще один секундомер с помощью метода **start ()** и выполните необходимые задачи. Если вы сделаете это, он запустит новый секундомер и остановит предыдущий (т.е. задание 1).

//Starting another child stopwatch and stopping the previous one.

profiler.start("Task 2");

obj.demoMethod2();

**Шаг 4: Остановить часы**

Когда мы вызываем метод **stop ()** , он останавливает недавний дочерний секундомер и глобальный секундомер и возвращает текущий инструмент времени.

// Stopping the current child stopwatch and the global stopwatch.

TimeInstrument tm = profiler.stop();

**Шаг 5: Распечатайте содержимое инструмента времени.**

Распечатайте содержимое инструмента текущего времени, используя метод **print ()** .

//printing the contents of the time instrument

tm.print();

**пример**

В следующем примере демонстрируется профилирование с использованием класса Profiler в SLF4J. Здесь мы взяли две примерные задачи, печатая сумму квадратов чисел от 1 до 10000, печатая сумму чисел от 1 до 10000. Мы пытаемся получить время, затраченное на эти две задачи.

import org.slf4j.profiler.Profiler;

import org.slf4j.profiler.TimeInstrument;

public class ProfilerExample {

public void demoMethod1(){

double sum = 0;

for(int i=0; i< 1000; i++){

sum = sum+(Math.pow(i, 2));

}

System.out.println("Sum of squares of the numbers from 1 to 10000: "+sum);

}

public void demoMethod2(){

int sum = 0;

for(int i=0; i< 10000; i++){

sum = sum+i;

}

System.out.println("Sum of the numbers from 1 to 10000: "+sum);

}

public static void main(String[] args) {

ProfilerExample obj = new ProfilerExample();

//Creating a profiler

Profiler profiler = new Profiler("Sample");

//Starting a child stop watch and stopping the previous one.

profiler.start("Task 1");

obj.demoMethod1();

//Starting another child stop watch and stopping the previous one.

profiler.start("Task 2");

obj.demoMethod2();

//Stopping the current child watch and the global watch.

TimeInstrument tm = profiler.stop();

//printing the contents of the time instrument

tm.print();

}

}

**Выход**

После выполнения вышеуказанная программа генерирует следующий вывод:

Sum of squares of the numbers from 1 to 10000: 3.328335E8

Sum of the numbers from 1 to 10000: 49995000

+ Profiler [BASIC]

|-- elapsed time [Task 1] 2291.827 microseconds.

|-- elapsed time [Task 2] 225.802 microseconds.

|-- Total [BASIC] 3221.598 microseconds.

**Регистрация информации о профилировщике**

Вместо того, чтобы печатать результаты профилировщика для регистрации этой информации, вам нужно –

* Создайте регистратор, используя класс **LoggerFactory** .
* Создайте профилировщик, создав экземпляр класса Profiler.
* **Свяжите** регистратор с профилировщиком, передав созданный объект регистратора **методу setLogger ()** класса **Profiler** .
* Наконец, вместо распечатки журнала информацию профилировщика используют метод **log ()** .

**пример**

В следующем примере, в отличие от предыдущего (вместо печати), мы пытаемся записать содержимое инструмента времени.

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import org.slf4j.profiler.Profiler;

import org.slf4j.profiler.TimeInstrument;

public class ProfilerExample\_logger {

public void demoMethod1(){

double sum = 0;

for(int i=0; i< 1000; i++){

sum = sum+(Math.pow(i, 2));

}

System.out.println("Sum of squares of the numbers from 1 to 10000: "+sum);

}

public void demoMethod2(){

int sum = 0;

for(int i=0; i< 10000; i++){

sum = sum+i;

}

System.out.println("Sum of the numbers from 1 to 10000: "+sum);

}

public static void main(String[] args) {

ProfilerExample\_logger obj = new ProfilerExample\_logger();

//Creating a logger

Logger logger = LoggerFactory.getLogger(ProfilerExample\_logger.class);

//Creating a profiler

Profiler profiler = new Profiler("Sample");

//Adding logger to the profiler

profiler.setLogger(logger);

//Starting a child stop watch and stopping the previous one.

profiler.start("Task 1");

obj.demoMethod1();

//Starting another child stop watch and stopping the previous one.

profiler.start("Task 2");

obj.demoMethod2();

//Stopping the current child watch and the global watch.

TimeInstrument tm = profiler.stop();

//Logging the contents of the time instrument

tm.log();

}

}