**SLF4J инструкция пользователя**

Простой фасад ведения журнала для Java (SLF4J) служит простым фасадом или абстракцией для различных сред ведения журналов, таких как java.util.logging, logback и log4j.

SLF4J позволяет конечному пользователю подключить желаемую структуру ведения журналов во время *развертывания* . Обратите внимание, что SLF4J-включение вашей библиотеки / приложения подразумевает добавление только одной обязательной зависимости, а именно *slf4j-api-2.0.0-alpha2-SNAPSHOT.jar* .

**НАЧИНАЯ С 1.6.0.** Если привязка к пути к классу не найдена, то по умолчанию SLF4J будет реализован без операций.

**НАЧИНАЯ С 1.7.0.** Методы печати в [Logger](http://www.slf4j.org/apidocs/org/slf4j/Logger.html) интерфейсе теперь предлагают варианты, принимающие [переменные](http://docs.oracle.com/javase/1.5.0/docs/guide/language/varargs.html) вместо Object[]. Это изменение подразумевает, что SLF4J требует JDK 1.5 или более поздней версии. Под капотом Java-компилятор превращает часть varargs в методы в Object[]. Таким образом, интерфейс Logger, сгенерированный компилятором, неотличим в 1.7.x от его аналога 1.6.x. Из этого следует, что SLF4J версии 1.7.x полностью на 100% совместим с SLF4J версии 1.6.x.

**С 1.7.5** Значительное улучшение времени поиска loggerа. Учитывая степень улучшения, пользователям настоятельно рекомендуется перейти на SLF4J 1.7.5 или более поздней версии.

**НАЧИНАЯ С 1.7.9.** Если для slf4j.detectLoggerNameMismatch свойства системы установлено значение true, SLF4J может автоматически [определять loggerы с неправильными именами](http://www.slf4j.org/codes.html#loggerNameMismatch) .

**НАЧИНАЯ С 2.0.0** SLF4J API версии 2.0.0 требует Java 8 и представляет обратно-совместимый API-интерфейс регистрации данных. Под обратной совместимостью мы подразумеваем, что существующие структуры ведения журналов не должны изменяться, чтобы пользователь мог воспользоваться [API-интерфейсом свободной регистрации](http://www.slf4j.org/manual.html#fluent) .

Привет, мир

Как это принято в традициях программирования, здесь приведен пример, иллюстрирующий простейший способ вывода «Hello world» с использованием SLF4J. Он начинается с получения логгера с именем «HelloWorld». Этот logger, в свою очередь, используется для регистрации сообщения «Hello World».

import org.slf4j.Logger;  
import org.slf4j.LoggerFactory;  
  
public class HelloWorld {  
  public static void main(String[] args) {  
    Logger logger = LoggerFactory.getLogger(HelloWorld.class);  
    logger.info("Hello World");  
  }  
}

Чтобы запустить этот пример, сначала нужно [скачать дистрибутив slf4j](http://www.slf4j.org/download.html) , а затем распаковать его. Как только это будет сделано, добавьте файл *slf4j-api-1.7.28.jar* в путь к классам.

Компиляция и запуск *HelloWorld* приведут к выводу следующего вывода на консоль.

SLF4J: Не удалось загрузить класс "org.slf4j.impl.StaticLoggerBinder".

SLF4J: по умолчанию реализация логгера без операций (NOP)

SLF4J: см. Http://www.slf4j.org/codes.html#StaticLoggerBinder для получения дополнительной информации.

Это предупреждение выводится на печать, поскольку в пути к классам не найдена привязка slf4j.

Предупреждение исчезнет, ​​как только вы добавите [привязку](http://www.slf4j.org/manual.html#swapping) к пути вашего класса. Предполагая, что вы добавили *slf4j-simple-1.7.28.jar,* чтобы ваш путь к классу содержал:

* SLF4J-апи-1.7.28.jar
* SLF4J-простой 1.7.28.jar

Компиляция и запуск *HelloWorld* теперь приведут к следующему выводу на консоль.

0 [главная] ИНФОРМАЦИЯ HelloWorld - Hello World

Типичная модель использования

Пример кода ниже иллюстрирует типичную схему использования SLF4J. Обратите внимание на использование {} -местителей в строке 15. См. Вопрос [«Какой самый быстрый способ регистрации?»](http://www.slf4j.org/faq.html#logging_performance)в FAQ для более подробной информации.

1: **import org.slf4j.Logger;**  
 2: **import org.slf4j.LoggerFactory;**  
 3:   
 4: public class Wombat {  
 5:    
 6:   **final Logger logger = LoggerFactory.getLogger(Wombat.class);**  
 7:   Integer t;  
 8:   Integer oldT;  
 9:  
10:   public void setTemperature(Integer temperature) {  
11:      
12:     oldT = t;          
13:     t = temperature;  
14:  
15:     **logger.debug("Temperature set to {}. Old temperature was {}.", t, oldT);**  
16:  
17:     if(temperature.intValue() > 50) {  
18:       **logger.info("Temperature has risen above 50 degrees.");**  
19:     }  
20:   }  
21: }

Fluent Logging API

**НАЧИНАЯ С 2.0.0** SLF4J API версии 2.0.0 требует Java 8 и представляет обратно-совместимый API-интерфейс регистрации данных. Под обратной совместимостью мы подразумеваем, что существующие структуры ведения журналов не должны изменяться, чтобы пользователь мог воспользоваться [API-интерфейсом свободной регистрации](http://www.slf4j.org/manual.html#fluent) .

Идея состоит в том, чтобы построить событие регистрации по частям с помощью [LoggingEventBuilder](http://www.slf4j.org/apidocs/org/slf4j/spi/LoggingEventBuilder.html) и вести журнал, когда событие будет полностью построено. atTrace(), atDebug(), atInfo(), atWarn()И atError()методы, все новое в org.slf4j.Loggerинтерфейсе, возвращал экземпляр [LoggingEventBuilder](http://www.slf4j.org/apidocs/org/slf4j/spi/LoggingEventBuilder.html). Для отключенных уровней журнала возвращаемый LoggingEventBuilderэкземпляр ничего не делает, тем самым сохраняя производительность наносекундного уровня традиционного интерфейса ведения журнала.

Вот несколько примеров использования:

Заявление

logger . atInfo (). журнал ( «Привет, мир» );

эквивалентно:

logger . info ( «Привет, мир.» );

Следующие операторы журнала эквивалентны в своем выводе (для реализации по умолчанию):

       int newT = 15;  
      int oldT = 16;  
  
        // using traditional API SLF4J  
        logger.debug("Temperature set to {}. Old temperature was {}.", newT, oldT);  
  
        // используя существующее (не SLF4J) API, добавить аргументы один за другим, а затем log сообщение

logger.**atDebug()**.addArgument(newT).addArgument(oldT).log("Temperature set to {}. Old temperature was {}.");  
  
        // using fluent API, log message with arguments  
logger.**atDebug()**.log("Temperature set to {}. Old temperature was {}.", newT, oldT);  
  
        // using fluent API, add one argument and then log message providing one more argument  
logger.**atDebug()**.addArgument(newT).log("Temperature set to {}. Old temperature was {}.", oldT);  
  
        // using fluent API, add one argument with a Supplier and then log message with one more argument.  
        // Предположим, что метод t16 () возвращает 16.  
logger.**atDebug()**.addArgument(() -> t16()).log(msg, "Temperature set to {}. Old temperature was {}.", oldT);

Беглый API-интерфейс регистрации позволяет задавать множество различных типов данных org.slf4j.Logger без комбинаторного взрыва числа методов в Loggerинтерфейсе.

Теперь можно передавать несколько [маркеров](http://www.slf4j.org/apidocs/org/slf4j/Marker.html) , передавать аргументы [поставщику](https://docs.oracle.com/javase/8/docs/api/java/util/function/Supplier.html) или передавать несколько пар ключ-значение. Пары ключ-значение особенно полезны в сочетании с анализаторами данных журнала, которые могут интерпретировать их автоматически.

Следующие операторы журнала эквивалентны:

       int newT = 15;  
       int oldT = 16;  
  
        // using classical API  
logger.debug("oldT={} newT={} Temperature changed.", newT, oldT);  
  
        // using fluent API  
logger.atDebug().**addKeyValue("oldT", oldT)**.addKeyValue("newT", newT). log("Temperature changed.");

Вариант пары ключ-значение API хранит пары ключ-значение как отдельные объекты. Реализация по умолчанию как отправка в org.slf4j.Logger классе *префиксирует* пары ключ-значение к сообщению. Бэкэнды могут свободно настраивать вывод по своему усмотрению.

Связывание с каркасом ведения журнала во время развертывания

Как упоминалось ранее, SLF4J поддерживает различные каркасы журналирования. Дистрибутив SLF4J поставляется с несколькими jar-файлами, называемыми «привязками SLF4J», причем каждая привязка соответствует поддерживаемой среде.

***SLF4J-log4j12-1.7.28.jar***

Привязка для [log4j версии 1.2](http://logging.apache.org/log4j/1.2/index.html) , широко используемой среды ведения журналов. Вы также должны поместить *log4j.jar* в ваш путь к классу.

***SLF4J-jdk14-1.7.28.jar***

Связывание для java.util.logging, также называемое журналированием JDK 1.4

***SLF4J-NOP-1.7.28.jar***

Привязка для [NOP](http://www.slf4j.org/api/org/slf4j/helpers/NOPLogger.html) , молча отбрасывая все записи.

***SLF4J-простой 1.7.28.jar***

Привязка для [простой](http://www.slf4j.org/apidocs/org/slf4j/impl/SimpleLogger.html)реализации, которая выводит все события в System.err. Печатаются только сообщения уровня INFO и выше. Эта привязка может быть полезна в контексте небольших приложений.

***SLF4J-JCL-1.7.28.jar***

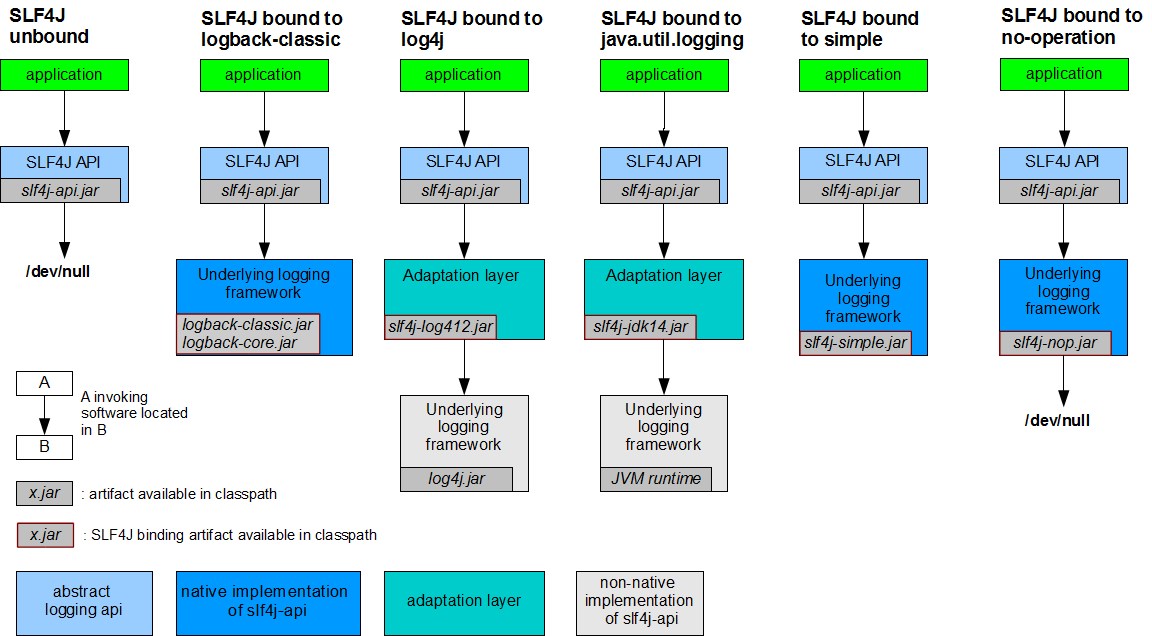
Переплет для [Джакарта Commons Logging](http://commons.apache.org/logging/) . Эта привязка делегирует всю запись SLF4J в JCL.

***logback-classic-1.2.3.jar (требуется logback-core-1.2.3.jar)***

**СОБСТВЕННАЯ РЕАЛИЗАЦИЯ** Существуют также привязки SLF4J, внешние по отношению к проекту SLF4J, например, [logback,](http://logback.qos.ch/) который изначально реализует SLF4J. Класс Logback является прямой реализацией интерфейса SLF4J . Таким образом, использование SLF4J в сочетании с logback требует строго нулевой памяти и вычислительных затрат. [ch.qos.logback.classic.Logger](http://logback.qos.ch/apidocs/ch/qos/logback/classic/Logger.html)[org.slf4j.Logger](http://www.slf4j.org/apidocs/org/slf4j/Logger.html)

Для переключения каркасов журналов просто замените привязки slf4j на вашем пути к классам. Например, чтобы переключиться с java.util.logging на log4j, просто замените slf4j-jdk14-1.7.28.jar на slf4j-log4j12-1.7.28.jar.

SLF4J не использует какой-либо спецтехники. Фактически, каждая привязка SLF4J жестко привязана *во время компиляции,* чтобы использовать одну и только одну специальную структуру ведения журнала. Например, привязка slf4j-log4j12-1.7.28.jar связана во время компиляции для использования log4j. В вашем коде, в дополнение к *slf4j-api-1.7.28.jar* , вы просто перетаскиваете **одну и только одну** привязку по вашему выбору в соответствующее местоположение пути класса. Не размещайте более одной привязки на вашем пути к классам. Вот графическая иллюстрация общей идеи.

[](http://www.slf4j.org/images/concrete-bindings.png)

Интерфейсы SLF4J и их различные адаптеры чрезвычайно просты. Большинство разработчиков, знакомых с языком Java, смогут прочитать и полностью понять код менее чем за час. Знание загрузчиков классов не требуется, поскольку SLF4J не использует и не имеет прямого доступа к загрузчикам классов. Как следствие, SLF4J не страдает ни от каких проблем загрузчика классов или утечек памяти, наблюдаемых при Jakarta Commons Logging (JCL).

Учитывая простоту интерфейсов SLF4J и модель его развертывания, разработчики новых каркасов журналирования должны находить, что связывание SLF4J очень легко.

Библиотеки

Авторы широко распространяемых компонентов и библиотек могут кодировать интерфейс SLF4J, чтобы избежать навязывания структуры ведения журнала их конечному пользователю. Таким образом, конечный пользователь может выбрать желаемую структуру ведения журналов во время развертывания, вставив соответствующую привязку slf4j в путь к классу, которая может быть изменена позже путем замены существующей привязки другой на пути к классам и перезапуска приложения. Этот подход оказался простым и очень надежным.

**Начиная с SLF4J версии 1.6.0** , если на пути к классам привязка не найдена, то по умолчанию slf4j-api будет реализован без операций, отбрасывая все запросы журнала. Таким образом, вместо того, чтобы выбрасывать a NoClassDefFoundErrorиз-за org.slf4j.impl.StaticLoggerBinderотсутствия класса, SLF4J версии 1.6.0 и выше выдает одно предупреждение об отсутствии привязки и продолжает отбрасывать все запросы журнала без дальнейшего протеста. Например, пусть Wombat будет некоторой биологической платформой, зависящей от SLF4J для ведения журнала. Во избежание навязывания инфраструктуры ведения журнала конечному пользователю дистрибутив Wombat включает в себя *slf4j-api.jar* но без привязки. Даже при отсутствии какой-либо привязки SLF4J на пути к классам, дистрибутив Wombat будет работать без каких-либо дополнительных настроек, и при этом конечному пользователю не потребуется загружать привязку с веб-сайта SLF4J. Только когда конечный пользователь решит включить ведение журнала, он должен будет установить привязку SLF4J, соответствующую выбранной им структуре ведения журнала.

**ОСНОВНОЕ ПРАВИЛО Встроенные компоненты, такие как библиотеки или платформы, не должны объявлять зависимость от какой-либо привязки SLF4J, а зависят только от slf4j-api** . Когда библиотека объявляет транзитивную зависимость от конкретной привязки, эта привязка накладывается на конечного пользователя, отрицая назначение SLF4J. Обратите внимание, что объявление нетранзитивной зависимости от привязки, например, для тестирования, не влияет на конечного пользователя.

Использование SLF4J во встроенных компонентах также обсуждается в разделе часто задаваемых вопросов в отношении [конфигурации ведения журнала](http://www.slf4j.org/faq.html#configure_logging) , [уменьшения зависимости](http://www.slf4j.org/faq.html#optional_dependency) и [тестирования](http://www.slf4j.org/faq.html#optional_dependency) .

Объявление зависимостей проекта для ведения журнала Maven

Учитывая правила транзитивной зависимости Maven, для «обычных» проектов (не библиотек или фреймворков) объявление зависимостей может быть выполнено с помощью одного объявления зависимостей.

**LOGBACK-CLASSIC** Если вы хотите использовать logback-classic в качестве базового каркаса ведения журналов, все, что вам нужно сделать, это объявить "ch.qos.logback: logback-classic" в качестве зависимости в вашем файле *pom.xml,* как показано ниже. В дополнение к *logback-classic-1.2.3.jar* , это *подтянет slf4j-api-1.7.28.jar,* а также *logback-core-1.2.3.jar* в ваш проект. Обратите внимание, что явное объявление зависимости от *logback-core-1.2.3* или *slf4j-api-1.7.28.jar* не является неправильным и может быть необходимо для наложения правильной версии указанных артефактов в соответствии с правилом передачи зависимостей «ближайшего определения» Maven. ,

<dependency>   
  <groupId>ch.qos.logback</groupId>  
  <artifactId>logback-classic</artifactId>  
  <version>1.2.3</version>  
</dependency>

**LOG4J** Если вы хотите использовать log4j в качестве базовой структуры ведения журналов, все, что вам нужно сделать, это объявить "org.slf4j: slf4j-log4j12" как зависимость в вашем файле *pom.xml,* как показано ниже. В дополнение к *slf4j-log4j12-1.7.28.jar* , это *втянет* в ваш проект как *slf4j-api-1.7.28.jar,* так и *log4j-1.2.17.jar* . Обратите внимание, что явное объявление зависимости от *log4j-1.2.17.jar* или *slf4j-api-1.7.28.jar* не является неправильным и может быть необходимо для наложения правильной версии указанных артефактов в соответствии с правилом передачи зависимостей "ближайшего определения" Maven. ,

<dependency>   
  <groupId>org.slf4j</groupId>  
  <artifactId>slf4j-log4j12</artifactId>  
  <version>1.7.28</version>  
</dependency>

**JAVA.UTIL.LOGGING** Если вы хотите использовать java.util.logging в качестве базовой структуры ведения журнала, все, что вам нужно сделать, это объявить "org.slf4j: slf4j-jdk14" как зависимость в вашем файле *pom.xml,* как показано ниже , В дополнение к *slf4j-jdk14-1.7.28.jar* это *добавит slf4j-api-1.7.28.jar* в ваш проект. Обратите внимание, что явное объявление зависимости от *slf4j-api-1.7.28.jar* не является неправильным и может быть необходимо для наложения правильной версии указанного артефакта в соответствии с правилом передачи зависимостей «ближайшего определения» Maven.

<dependency>   
  <groupId>org.slf4j</groupId>  
  <artifactId>slf4j-jdk14</artifactId>  
  <version>1.7.28</version>  
</dependency>

Бинарная совместимость

Привязка SLF4J обозначает артефакт, такой как *slf4j-jdk14.jar* или *slf4j-log4j12.jar,* используемый для *привязки* slf4j к базовой структуре ведения журнала, скажем, java.util.logging и, соответственно, log4j.

С точки зрения клиента все версии slf4j-api совместимы. Клиентский код, скомпилированный с slf4j-api-N.jar, будет отлично работать с slf4j-api-M.jar для любых N и M. Вам нужно только убедиться, что версия вашей привязки совпадает с версией slf4j-api.jar. Вам не нужно беспокоиться о версии slf4j-api.jar, используемой данной зависимостью в вашем проекте.

**Смешивание разных версий *slf4j-api.jar* и SLF4J может вызвать проблемы. Например, если вы используете slf4j-api-1.7.28.jar, вам также следует использовать slf4j-simple-1.7.28.jar, использование slf4j-simple-1.5.5.jar не будет работать.**

Однако, с точки зрения клиента, все версии slf4j-api совместимы. Клиентский код, скомпилированный с *slf4j-api-N.jar,* будет отлично работать с *slf4j-api-M.jar* для любых N и M. Вам нужно только убедиться, что версия вашей привязки совпадает с версией slf4j-api.jar. Вам не нужно беспокоиться о версии slf4j-api.jar, используемой данной зависимостью в вашем проекте. Вы всегда можете использовать любую версию *slf4j-api.jar* , и если версия *slf4j-api.jar* и его привязка совпадают, у вас все будет в порядке.

Во время инициализации, если SLF4J подозревает, что может быть проблема несовпадения версий slf4j-api и обязательной версии, он выдаст предупреждение о предполагаемом несовпадении.

Консолидация регистрации через SLF4J

Часто конкретный проект зависит от различных компонентов, которые используют API журналирования, отличные от SLF4J. Обычно проекты находят в зависимости от комбинации JCL, java.util.logging, log4j и SLF4J. Затем становится желательным объединить ведение журнала по одному каналу. SLF4J обслуживает этот общий вариант использования, предоставляя модули моста для JCL, java.util.logging и log4j. Для более подробной информации, пожалуйста, обратитесь к странице **[Bridging legacy APIs](http://www.slf4j.org/legacy.html)** .

Поддержка сопоставленного диагностического контекста (MDC)

«Сопоставленный диагностический контекст» - это, по сути, карта, поддерживаемая каркасом ведения журнала, где код приложения предоставляет пары ключ-значение, которые затем могут быть вставлены каркасом ведения журнала в сообщения журнала. Данные MDC также могут быть очень полезны для фильтрации сообщений или запуска определенных действий.

SLF4J поддерживает MDC или сопоставленный диагностический контекст. Если базовая структура ведения журнала предлагает функциональность MDC, то SLF4J делегирует MDC базовой структуры. Обратите внимание, что в настоящее время только log4j и logback предлагают функциональность MDC. Если базовая структура не предлагает MDC, например java.util.logging, то SLF4J будет по-прежнему хранить данные MDC, но информацию в них необходимо будет получить с помощью пользовательского кода пользователя.

Таким образом, как пользователь SLF4J, вы можете воспользоваться информацией MDC при наличии log4j или logback, но без навязывания этих рамок журналирования вашим пользователям в качестве зависимостей.

Для получения дополнительной информации о MDC см. [Главу о MDC](http://logback.qos.ch/manual/mdc.html) в руководстве по возврату.

Управляющее резюме

|  |  |
| --- | --- |
| **преимущество** | **Описание** |
| Выберите структуру ведения журнала во время развертывания | Желаемую структуру ведения журналов можно подключить во время развертывания, вставив соответствующий файл jar (связывание) в путь к классу. |
| Отказоустойчивая операция | Благодаря тому, что классы загружаются JVM, привязка фреймворка будет проверена автоматически очень рано. Если SLF4J не может найти привязку к пути к классу, он выдаст одно предупреждающее сообщение и по умолчанию не будет реализован. |
| Привязки для популярных каркасов | SLF4J поддерживает популярные каркасы журналирования, а именно log4j, java.util.logging, Simple logging и NOP. В [Logback](http://logback.qos.ch/) поддерживает проект SLF4J изначально. |
| Соединение устаревших API журналирования | Реализация JCL поверх SLF4J, то есть *jcl-over-slf4j.jar* , позволит вашему проекту перейти на SLF4J по частям, не нарушая совместимость с существующим программным обеспечением, использующим JCL. Аналогично, модули log4j-over-slf4j.jar и jul-to-slf4j позволят вам перенаправлять вызовы log4j и, соответственно, java.util.logging в SLF4J. Смотрите страницу на [Bridging унаследованных API](http://www.slf4j.org/legacy.html) для более подробной информации. |
| Перенос вашего исходного кода | [SLF4J-Migrator](http://www.slf4j.org/migrator.html) утилита может помочь вам перенести источник на использование SLF4J. |
| Поддержка параметризованных сообщений журнала | Все привязки SLF4J поддерживают параметризованные журнальные сообщения со значительно [улучшенными](http://www.slf4j.org/faq.html#logging_performance) результатами [производительности](http://www.slf4j.org/faq.html#logging_performance) . |