**Глава 4: Appenders**

Чтобы запустить примеры из этой главы, вам нужно убедиться, что в пути к классам присутствуют определенные файлы jar. Пожалуйста, обратитесь к [странице настройки](http://logback.qos.ch/setup.html) для получения дополнительной информации.

**Что такое Appender?**

Logback делегирует задачу записи события регистрации в компоненты, называемые appenders. Appenders должен реализовывать [ch.qos.logback.core.Appender](http://logback.qos.ch/xref/ch/qos/logback/core/Appender.html) интерфейс. Основные методы этого интерфейса приведены ниже:

package ch.qos.logback.core;  
    
import ch.qos.logback.core.spi.ContextAware;  
import ch.qos.logback.core.spi.FilterAttachable;  
import ch.qos.logback.core.spi.LifeCycle;  
    
  
public interface Appender<E> extends LifeCycle, ContextAware, FilterAttachable {  
  
  public String getName();  
  public void setName(String name);  
  **void doAppend(E event);**  
    
}

Большинство методов в Appender интерфейсе являются установщиками и получателями. Заметным исключением является doAppend()метод, принимающий экземпляр объекта типа *E в* качестве единственного параметра. Фактический тип *E* будет варьироваться в зависимости от модуля logback. В модуле logback-classic *E* будет иметь тип [ILoggingEvent,](http://logback.qos.ch/apidocs/ch/qos/logback/classic/spi/ILoggingEvent.html) а в модуле logback-access он будет иметь тип [AccessEvent](http://logback.qos.ch/apidocs/ch/qos/logback/access/spi/AccessEvent.html) . Этот doAppend()метод, пожалуй, самый важный в структуре logback. Он отвечает за вывод событий регистрации в подходящем формате на соответствующее устройство вывода.

Приложения являются именованными объектами. Это гарантирует, что на них можно ссылаться по имени, качество, которое подтверждается как полезное для сценариев конфигурации. AppenderИнтерфейс расширяет FilterAttachableинтерфейс. Из этого следует, что один или несколько фильтров могут быть присоединены к экземпляру приложения. Фильтры подробно обсуждаются в следующей главе.

Ответственность за вывод событий журнала несут конечные пользователи. Тем не менее, они могут передавать фактическое форматирование события к Layoutили к Encoderобъекту. Каждый макет / кодировщик связан с одним и только одним приложением, называемым владельцем приложения. Некоторые приложения имеют встроенный или фиксированный формат событий. Следовательно, они не требуют и не имеют макета / кодировщика. Например, SocketAppenderпросто сериализует события регистрации перед передачей их по проводам.

**AppenderBase**

Класс [ch.qos.logback.core.AppenderBase](http://logback.qos.ch/xref/ch/qos/logback/core/AppenderBase.html) является абстрактным классом реализации интерфейса. Он предоставляет базовые функциональные возможности, общие для всех приложений, такие как методы получения или установки их имени, их статуса активации, их макета и их фильтров. Это суперкласс всех дополнений, поставляемых с logback. Хотя абстрактный класс, AppenderBase фактически реализует doAppend() метод в Appender интерфейсе. Возможно, самый простой способ обсудить AppenderBase класс - это представить выдержку из исходного кода.

public synchronized void doAppend(E eventObject) {  
  
  // prevent re-entry.  
  if (guard) {  
    return;  
  }  
  
  try {  
    guard = true;  
  
    if (!this.started) {  
      if (statusRepeatCount++ < ALLOWED\_REPEATS) {  
        addStatus(new WarnStatus(  
            "Attempted to append to non started appender [" + name + "].",this));  
      }  
      return;  
    }  
  
    if (getFilterChainDecision(eventObject) == FilterReply.DENY) {  
      return;  
    }  
      
    // ok, we now invoke the derived class's implementation of append  
    this.append(eventObject);  
  
  } finally {  
    guard = false;  
  }  
}

Эта реализация doAppend()метода синхронизирована. Отсюда следует, что вход в систему одного и того же приложения из разных потоков безопасен. Пока поток, скажем, *T* , выполняет doAppend()метод, последующие вызовы других потоков помещаются в очередь, пока *T не* покинет doAppend()метод, обеспечивая эксклюзивный доступ *T* к appender.

Поскольку такая синхронизация не всегда уместна, вход в систему поставляется с [ch.qos.logback.core.UnsynchronizedAppenderBase](http://logback.qos.ch/xref/ch/qos/logback/core/UnsynchronizedAppenderBase.html) очень похожим [AppenderBase](http://logback.qos.ch/xref/ch/qos/logback/core/AppenderBase.html) классом. Для краткости, мы будем обсуждать UnsynchronizedAppenderBaseв оставшейся части этого документа.

Первое, что doAppend()делает метод, это проверяет, установлен ли для защиты значение true. Если это так, он немедленно выходит. Если защита не установлена, она устанавливается в true при следующей инструкции. Охранник гарантирует, что doAppend()метод не будет рекурсивно вызывать сам себя. Просто представьте, что компонент, вызванный где-то за пределами append()метода, хочет что-то зарегистрировать. Его вызов может быть направлен тому же самому приложению, которое только что вызвало его, что приводит к бесконечному циклу и переполнению стека.

В следующем утверждении мы проверяем, startedявляется ли поле истинным. Если это не так, doAppend()отправит предупреждающее сообщение и вернется. Другими словами, после закрытия приложения невозможно писать в него. Appenderобъекты реализации LifeCycleинтерфейса, который подразумевает , что они реализуют start(), stop()и isStarted()методы. После установки всех свойств приложения Joran, структура конфигурации logback, вызывает start()метод, чтобы дать сигнал приложению об активации его свойств. В зависимости от вида, приложение может не запуститься, если определенные свойства отсутствуют или из-за помех между различными свойствами. Например, учитывая, что создание файла зависит от режима усечения, FileAppenderне может влиять на значение егоFileпараметр до тех пор, пока значение параметра «Добавить» также точно не известно. Явный шаг активации гарантирует, что appender действует на его свойства *после того, как* их значения станут известны.

Если приложение не удалось запустить или оно было остановлено, через внутреннюю систему управления состоянием входа будет выдано предупреждение. После нескольких попыток, чтобы избежать переполнения внутренней системы статусов копиями одного и того же предупреждающего сообщения, doAppend()метод перестанет выдавать эти предупреждения.

Следующая if инструкция проверяет результат вложенных фильтров. В зависимости от решения, принятого в цепочке фильтров, события могут быть отклонены или приняты явно. При отсутствии решения по цепочке фильтров события принимаются по умолчанию.

Затем doAppend()метод вызывает реализацию append()метода производными классами . Этот метод выполняет фактическую работу по добавлению события на соответствующее устройство.

Наконец, охранник освобождается для последующего вызова doAppend()метода.

В оставшейся части этого руководства мы зарезервируем термин «опция» или, альтернативно, «свойство» для любого атрибута, который выводится динамически с помощью интроспекции JavaBeans через методы setter и getter.

**Logback-core**

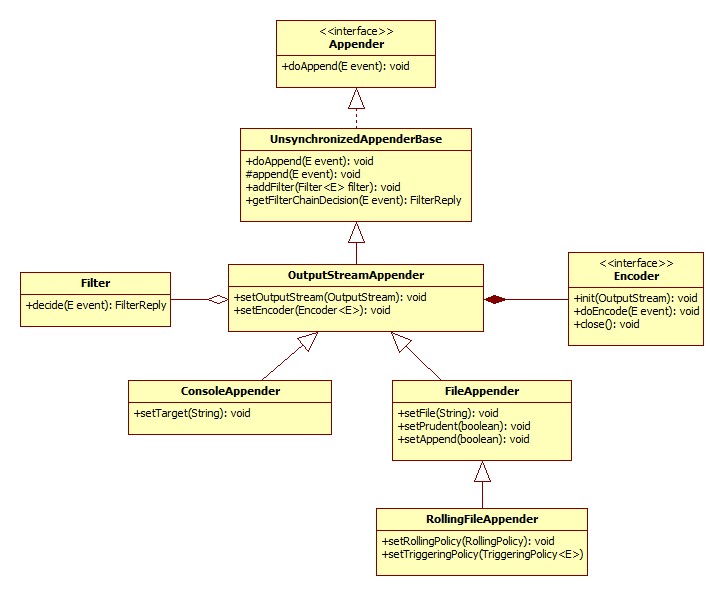
Logback-ядро закладывает фундамент, на котором строятся другие модули logback. В целом, компоненты в logback-core требуют некоторой, хотя и минимальной, настройки. Однако в следующих нескольких разделах мы опишем несколько приложений, которые готовы к использованию из коробки.

**OutputStreamAppender**

[OutputStreamAppender](http://logback.qos.ch/xref/ch/qos/logback/core/OutputStreamAppender.html) добавляет события в java.io.OutputStream. Этот класс предоставляет базовые сервисы, на которых основываются другие приложения. Пользователи обычно не создают экземпляры OutputStreamAppender объектов напрямую, поскольку в общем случае java.io.OutputStream тип не может быть удобно сопоставлен со строкой, поскольку нет способа указать целевой OutputStreamобъект в сценарии конфигурации. Проще говоря, вы не можете настроить OutputStreamAppender из файла конфигурации. Однако это не значит, что OutputStreamAppender не хватает настраиваемых свойств. Эти свойства описаны далее.

|  |  |  |
| --- | --- | --- |
| **Имя свойства** | **Тип** | **Описание** |
| **encoder** | [Encoder](http://logback.qos.ch/xref/ch/qos/logback/core/encoder/Encoder.html) | Определяет, каким образом событие записывается в базовый OutputStreamAppender. Кодеры описаны в отдельной [главе](http://logback.qos.ch/manual/encoders.html) . |
| **immediateFlush** | boolean | Значением по умолчанию для немедленной загрузки является «истина». Немедленная очистка выходного потока гарантирует, что события регистрации будут немедленно записаны и не будут потеряны в случае выхода из приложения без надлежащего закрытия аппендеров. С другой стороны, если установить для этого свойства значение false, вероятно, в четыре раза (ваш пробег может отличаться) пропускная способность журналирования. Опять же, если для значения InstantFlush задано значение «false», и приложения не закрываются должным образом при выходе из приложения, то запись событий, еще не записанных на диск, может быть потеряна. |

OutputStreamAppender является супер-классом для трех других appenders, а именно ConsoleAppender, FileAppender, который в свою очередь является супер классом для RollingFileAppender. На следующем рисунке показана диаграмма классов OutputStreamAppenderи ее подклассов.



**ConsoleAppender**

[ConsoleAppender](http://logback.qos.ch/xref/ch/qos/logback/core/ConsoleAppender.html), как видно из названия, он добавляется в консоль или, точнее, в *System.out* или *System.err*, причем первый является целью по умолчанию. ConsoleAppender форматирует события с помощью кодера, указанного пользователем. Кодеры будут обсуждаться в следующей главе. Оба *System.out* и *System.err* имеют тип java.io.PrintStream. Следовательно, они заключены в буфер OutputStreamWriter, который буферизует операции ввода-вывода.

|  |  |  |
| --- | --- | --- |
| **Имя свойства** | **Тип** | **Описание** |
| **encoder** | [Encoder](http://logback.qos.ch/xref/ch/qos/logback/core/encoder/Encoder.html) | Смотрите OutputStreamAppender свойства. |
| **target** | String | Одно из строковых значений *System.out* или *System.err* . По умолчанию целью является *System.out* . |
| **withJansi** | boolean | По умолчанию свойство withJansi установлено в false. Установка withJansi для trueактивации библиотеки [Jansi,](http://jansi.fusesource.org/) которая обеспечивает поддержку цветовых кодов ANSI на машинах Windows. На хосте Windows, если это свойство имеет значение true, вы должны поместить «org.fusesource.jansi: jansi: 1.17» в путь к классам. Обратите внимание, что операционные системы на основе Unix, такие как Linux и Mac OS X, по умолчанию поддерживают цветовые коды ANSI.  В Eclipse IDE вы можете попробовать [ANSI в](http://www.mihai-nita.net/eclipse/) плагине [Eclipse Console](http://www.mihai-nita.net/eclipse/) . |

Вот пример конфигурации, которая использует ConsoleAppender.

*Пример: конфигурация ConsoleAppender (logback-examples / src / main / resources / chapters / appenders / conf / logback-Console.xml)*

<configuration>  
  
  **<appender name="STDOUT" class="ch.qos.logback.core.ConsoleAppender">  
    <!-- encoders are assigned the type  
         ch.qos.logback.classic.encoder.PatternLayoutEncoder by default -->  
    <encoder>  
      <pattern>%-4relative [%thread] %-5level %logger{35} - %msg %n</pattern>  
    </encoder>  
  </appender>**  
  
  <root level="DEBUG">  
    <appender-ref ref="STDOUT" />  
  </root>  
</configuration>

После того, как вы установили свой текущий путь к *каталогу logback-examples* и [настроили свой путь к классу](http://logback.qos.ch/setup.html) , вы можете дать приведенный выше файл конфигурации, выполнив следующую команду:

java [chapters.appenders.ConfigurationTester](http://logback.qos.ch/xref/chapters/appenders/ConfigurationTester.html) src / main / java / chapters / appenders / conf / logback-Console.xml

**FileAppender**

[FileAppender](http://logback.qos.ch/xref/ch/qos/logback/core/FileAppender.html), Подкласс OutputStreamAppender, присоединяет журнала событий в файл. Целевой файл указывается в опции File . Если файл уже существует, он либо добавляется, либо усекается в зависимости от значения свойства append .

|  |  |  |
| --- | --- | --- |
| **Имя свойства** | **Тип** | **Описание** |
| **Append** | boolean | Если true, события добавляются в конец существующего файла. В противном случае, если append равен false, любой существующий файл усекается. Опция добавления по умолчанию имеет значение true. |
| **encoder** | [Encoder](http://logback.qos.ch/xref/ch/qos/logback/core/encoder/Encoder.html) | Смотрите OutputStreamAppender свойства. |
| **file** | String | Имя файла для записи. Если файл не существует, он создается. На платформе MS Windows пользователи часто забывают избежать обратной косой черты. Например, значение *c: \ temp \ test.log,* скорее всего, не будет правильно интерпретировано, поскольку *'\ t'* является escape-последовательностью, интерпретируемой как один символ табуляции *(\ u0009)* . Правильные значения могут быть указаны как *c: /temp/test.log* или как *c: \\ temp \\ test.log* . Параметр Файл не имеет значения по умолчанию.  Если родительский каталог файла не существует, он FileAppenderбудет автоматически создан , включая любые необходимые, но несуществующие родительские каталоги. |
| **prudent** | boolean | В разумном режиме FileAppender безопасно выполнит запись в указанный файл, даже в присутствии других FileAppender экземпляров, работающих в разных JVM, потенциально работающих на разных хостах. Значением по умолчанию для разумного режима является false.  Благоразумный режим может использоваться в сочетании с RollingFileAppender некоторыми [ограничениями](http://logback.qos.ch/manual/appenders.html#prudentWithRolling) .  Благоразумный режим подразумевает, что для свойства append автоматически устанавливается значение true.  Благоразумный больше полагается на эксклюзивные блокировки файлов. Эксперименты показывают, что файл блокирует примерно втрое (х3) стоимость написания события регистрации. На «среднем» ПК, выполняющем запись в файл, расположенный на **локальном** жестком диске, когда разумный режим отключен, для записи одного события регистрации требуется около 10 микросекунд. Когда включен разумный режим, для вывода одного события регистрации требуется около 30 микросекунд. Это приводит к пропускной способности регистрации 100 000 событий в секунду, когда разумный режим отключен, и приблизительно 33 000 событий в секунду в разумном режиме.  Режим Prudent эффективно сериализует операции ввода-вывода между всеми JVM, записывающими в один и тот же файл. Таким образом, с увеличением числа JVM, конкурирующих за доступ к файлу, увеличивается и задержка, возникающая при каждой операции ввода-вывода. Пока *общее* количество операций ввода-вывода составляет порядка 20 запросов журнала в секунду, влияние на производительность должно быть незначительным. Приложения, генерирующие 100 или более операций ввода-вывода в секунду, могут видеть влияние на производительность и должны избегать разумного режима.  **СЕТЕВЫЕ БЛОКИРОВКИ ФАЙЛОВ** Когда файл журнала находится в сетевой файловой системе, стоимость разумного режима еще выше. Не менее важно то, что блокировки файлов в сетевой файловой системе иногда могут быть сильно предвзятыми, так что процесс, владеющий блокировкой, немедленно повторно получает блокировку после ее освобождения. Таким образом, в то время как один процесс захватывает блокировку для файла журнала, другие процессы начинают голодать, ожидая, пока блокировка не окажется заблокированной.  Воздействие разумного режима в значительной степени зависит от скорости сети, а также от особенностей реализации ОС. Мы предоставляем очень маленькое приложение под названием [FileLockSimulator,](https://gist.github.com/2794241) которое может помочь вам моделировать поведение разумного режима в вашей среде. |

**НЕМЕДЛЕННАЯ ОЧИСТКА** По умолчанию каждое событие журнала немедленно сбрасывается в основной выходной поток. Этот подход по умолчанию является более безопасным в том смысле, что регистрация событий не теряется, если ваше приложение завершает работу без надлежащего закрытия приложений. Тем не менее, для значительного увеличения пропускной способности регистрации, вы можете установить **immediateFlush** свойство false.

Ниже приведен пример файла конфигурации для FileAppender:

*Пример: конфигурация FileAppender (logback-examples / src / main / resources / chapters / appenders / conf / logback-fileAppender.xml)*

<configuration>  
  
  **<appender name="FILE" class="ch.qos.logback.core.FileAppender">**

**<prudent>true</prudent>  
    <file>testFile.log</file>  
    <append>true</append>  
    <!-- установите immediateFlush в false для ускорения ведения журнала -->  
    <immediateFlush>true</immediateFlush>  
    <!--по умолчанию устанавливается тип  
         ch.qos.logback.classic.encoder.PatternLayoutEncoder-->  
    <encoder>  
      <pattern>%-4relative [%thread] %-5level %logger{35} - %msg%n</pattern>  
    </encoder>  
  </appender>**  
          
  <root level="DEBUG">  
    <appender-ref ref="FILE" />  
  </root>  
</configuration>

После изменения текущего каталога на *logback-examples* , запустите этот пример, запустив следующую команду:

java chapters.appenders.ConfigurationTester src / main / java / chapters / appenders / conf / logback-fileAppender.xml

**Файлы с уникальными именами (по метке времени)**

На этапе разработки приложений или в случае недолговечных приложений, например, пакетных приложений, желательно создавать новый файл журнала при каждом запуске нового приложения. Это довольно легко сделать с помощью <timestamp> элемента. Вот пример.

*Пример: Конфигурация FileAppender с уникальным именем по метке времени (logback-examples / src / main / resources / chapters / appenders / conf / logback-timestamp.xml)*

<configuration>  
  
  <!-- Вставьте текущее время, отформатированное как “yyyyMMdd'T'HHmmss" с ключом "bySecond", в контекст logger-а. Это значение будет доступно для всех последующих элементов конфигурации. -->  
  **<timestamp key="bySecond" datePattern="yyyyMMdd'T'HHmmss"/>**  
  
  <appender name="FILE" class="ch.qos.logback.core.FileAppender">  
    <!-- использовать ранее созданную временную метку для создания файла журнала с уникальным именем -->  
    <file>**log-${bySecond}.txt**</file>  
    <encoder>  
      <pattern>%logger{35} - %msg%n</pattern>  
    </encoder>  
  </appender>  
  
  <root level="DEBUG">  
    <appender-ref ref="FILE" />  
  </root>  
</configuration>

Элемент timestamp принимает два обязательных атрибута *key* и *datePattern* и необязательный атрибут *timeReference* . *Ключом* атрибута является имя ключа , при котором метка будет доступна для последующих элементов конфигурации в [качестве переменной](http://logback.qos.ch/manual/configuration.html#variableSubstitution). *DatePattern* атрибут обозначает шаблон даты , используемый для преобразования текущего времени (при котором файл конфигурации разобранные) в строку. Шаблон даты должен соответствовать соглашениям, определенным в [SimpleDateFormat](https://docs.oracle.com/javase/8/docs/api/java/text/SimpleDateFormat.html). *timeReference* атрибут обозначает временную привязку для отметки времени. По умолчанию используется время интерпретации / анализа файла конфигурации, т.е. текущее время.

Однако при определенных обстоятельствах может быть полезно использовать контекстное время рождения в качестве эталона времени. Это может быть достигнуто установкой атрибута *timeReference* в "contextBirth".

Поэкспериментируйте с <timestamp> элементом, выполнив команду:

java chapters.appenders.ConfigurationTester src / main / resources / chapters / appenders / conf / logback-timestamp.xml

Чтобы использовать дату рождения контекста регистратора в качестве временной привязки, вы должны установить для атрибута *timeReference* значение «contextBirth», как показано ниже.

*Пример: отметка времени с использованием даты рождения контекста в качестве эталона времени (logback-examples / src / main / resources / chapters / appenders / conf / logback-timestamp-contextBirth.xml)*

<configuration>  
  <timestamp key="bySecond" datePattern="yyyyMMdd'T'HHmmss"   
             **timeReference="contextBirth"**/>  
  ...  
</configuration>

**RollingFileAppender**

[RollingFileAppender](http://logback.qos.ch/xref/ch/qos/logback/core/rolling/RollingFileAppender.html) расширяет FileAppender возможностями ролловера файлов журнала (создания следующено лог-файла по времени, по достижении текущим файлом определенного размера + удаление старых лог-файлов если их больше определенного количества или если лог-файлы занимают объем больше установленного). Все это устанавливается в <rollingPolicy>.

Например, RollingFileAppender можете войти в файл с именем *log.txt* и, как только определенное условие выполнено, изменить его цель ведения журнала на другой файл.

Есть два важных подкомпонента, которые взаимодействуют с RollingFileAppender. Первый RollingFileAppender подкомпонент, а именно RollingPolicy( [см. Ниже](http://logback.qos.ch/manual/appenders.html" \l "onRollingPolicies) ), отвечает за выполнение действий, необходимых для опрокидывания. Второй подкомпонент RollingFileAppender, а именно TriggeringPolicy( [см. Ниже](http://logback.qos.ch/manual/appenders.html" \l "TriggeringPolicy) ), будет определять, когда именно произойдет опрокидывание. Таким образом, RollingPolicyнесет ответственность за то, *что* и TriggeringPolicyнесет ответственность за то, *когда* .

Чтобы иметь какое-либо применение, a RollingFileAppenderдолжен иметь как a, так RollingPolicyи TriggeringPolicyнастройку. Однако, если он RollingPolicyтакже реализует TriggeringPolicyинтерфейс, то только первый должен быть указан явно.

Вот доступные свойства для RollingFileAppender:

|  |  |  |
| --- | --- | --- |
| **Имя свойства** | **Тип** | **Описание** |
| **file** | String | Смотрите FileAppenderсвойства. |
| **Append** | boolean | Смотрите FileAppenderсвойства. |
| **encoder** | [Encoder](http://logback.qos.ch/xref/ch/qos/logback/core/encoder/Encoder.html) | Смотрите OutputStreamAppenderсвойства. |
| **rollingPolicy** | RollingPolicy | Эта опция является компонентом, который будет определять RollingFileAppender поведение при опрокидывании. Смотрите дополнительную информацию ниже. |
| **triggeringPolicy** | TriggeringPolicy | Эта опция является компонентом, который сообщит, RollingFileAppender когда активировать процедуру опрокидывания. Смотрите дополнительную информацию ниже. |
| **prudent** | boolean | [FixedWindowRollingPolicy](http://logback.qos.ch/manual/appenders.html#FixedWindowRollingPolicy) не поддерживается в разумном режиме.  RollingFileAppender поддерживает разумный режим в сочетании [TimeBasedRollingPolicy](http://logback.qos.ch/manual/appenders.html#TimeBasedRollingPolicy) с двумя ограничениями.   1. В разумном режиме сжатие файлов не поддерживается и не допускается. (У нас не может быть одной JVM, записывающей в файл, пока другая JVM сжимает его.) 2. Файл свойство FileAppenderне может быть установлено и должно быть оставлено пустым. Действительно, большинство операционных систем не позволяют переименовывать файл, пока другой процесс открывает его.   Смотрите также свойства для FileAppender. |

**Обзор скользящей политики**

[RollingPolicy](http://logback.qos.ch/xref/ch/qos/logback/core/rolling/RollingPolicy.html) отвечает за процедуру ролловера, которая включает перемещение и переименование файла.

RollingPolicy интерфейс представлен ниже:

package ch.qos.logback.core.rolling;    
  
import ch.qos.logback.core.FileAppender;  
import ch.qos.logback.core.spi.LifeCycle;  
  
public interface RollingPolicy extends LifeCycle {  
  
  **public void rollover() throws RolloverFailure;**  
  public String getActiveFileName();  
  public CompressionMode getCompressionMode();  
  public void setParent(FileAppender appender);  
}

rollover метод выполняет работу, участвующих в архивировании текущего файла журнала.  getActiveFileName()метод вызывается для вычисления имени текущего файла журнала (в который записываются текущие журналы). Как указано getCompressionMode методом, RollingPolicy также отвечает за определение режима сжатия. Наконец, a RollingPolicy дается ссылка на своего родителя через setParent метод.

**TimeBasedRollingPolicy (неполный фильтр, т.к. нет фильтра по макс.размеру каждого лог-файла)**

[TimeBasedRollingPolicy](http://logback.qos.ch/xref/ch/qos/logback/core/rolling/TimeBasedRollingPolicy.html) возможно самая популярная скользящая политика. Он определяет политику ролловера на основе времени, например, по дням или месяцам. TimeBasedRollingPolicy принимает на себя ответственность за опрокидывание, а также за запуск указанного опрокидывания. Действительно, TimeBasedTriggeringPolicy реализует *оба* RollingPolicy и TriggeringPolicy интерфейсы.

TimeBasedRollingPolicy для конфигурации требуется одно обязательное свойство fileNamePattern и несколько необязательных свойств.

|  |  |  |
| --- | --- | --- |
| **Имя свойства** | **Тип** | **Описание** |
| **fileNamePattern** | String | Обязательное свойство fileNamePattern определяет имя развернутых (архивированных) файлов журнала. Его значение должно состоять из имени файла и соответственно размещенного спецификатора преобразования *% d* . Спецификатор преобразования *% d* может содержать шаблон даты и времени, указанный java.text.SimpleDateFormat классом. Если шаблон даты и времени опущен, то шаблон по умолчанию *YYYY-MM-дд* предполагается. **Период пролонгации выводится из значения fileNamePattern .**  Обратите внимание, что свойство файла в RollingFileAppender (родительский элемент TimeBasedRollingPolicy) может быть установлено или опущено. Установив свойство файла содержимого FileAppender, вы можете отделить расположение активного файла журнала и расположение архивных файлов журнала. Текущие журналы всегда будут нацелены на файл, указанный свойством файла . Отсюда следует, что имя текущего активного файла журнала не будет меняться со временем. Однако, если вы решите опустить свойство файла , то активный файл будет вычисляться заново для каждого периода на основе значения fileNamePattern., Приведенные ниже примеры должны прояснить этот момент.  Шаблон даты и времени, найденный в наградах% d {}, следует соглашениям java.text.SimpleDateFormat. Символы прямой косой черты '/' или обратной косой черты '\' в любом месте свойства fileNamePattern или в шаблоне даты и времени будут интерпретироваться как разделители каталогов.  **Несколько% d спецификаторов**  Можно указать несколько спецификаторов % d, но только один из них может быть основным, т. е. использоваться для определения периода пролонгации. Все остальные токены *должны* быть помечены как вспомогательные путем передачи параметра “aux” (см. Примеры ниже).  Множество % d спецификаторов позволяют вам организовать архивные файлы в структуре папок, отличной от структуры периода пролонгации. Например, шаблон имени файла, показанный ниже, упорядочивает папки журналов по годам и месяцам, но каждый день в полночь обновляет файлы журналов.  /var / log / **%d{yyyy / MM, aux}** / myapplication.**%d{yyyy-MM-dd}**.log  **Часовой пояс**  При определенных обстоятельствах может потребоваться перенести файлы журнала в соответствии с часами в часовом поясе, отличном от часового пояса хоста. Можно передать аргумент часового пояса, следуя шаблону даты и времени в спецификаторе преобразования% d. Например:  aFolder / тест. **% d** {гггг-ММ-дд-ЧЧ, **UTC** } .log  Если указанный идентификатор часового пояса неизвестен или написан с ошибкой, часовой пояс GMT считается предполагаемым в соответствии со спецификацией метода [TimeZone.getTimeZone (String)](http://docs.oracle.com/javase/6/docs/api/java/util/TimeZone.html#getTimeZone(java.lang.String)) . |
| **maxHistory** | ИНТ | Необязательное свойство maxHistory контролирует максимальное количество сохраняемых архивных файлов, асинхронно удаляя старые файлы. Например, если вы укажете ежемесячный ролловер и зададите для maxHistory значение 6, архивные файлы на 6 месяцев будут сохранены, а файлы старше 6 месяцев будут удалены. Обратите внимание, что при удалении старых архивных файлов журнала все папки, созданные для архивирования файлов журнала, будут удаляться соответствующим образом. |
| **totalSizeCap** | ИНТ | Необязательное свойство totalSizeCap контролирует общий размер всех архивных файлов. Самые старые архивы удаляются асинхронно при превышении общего размера. Для свойства totalSizeCap также необходимо установить свойство maxHistory . Кроме того, ограничение «максимальная история» всегда применяется первым, а ограничение «общий размер ограничения» применяется вторым. |
| **cleanHistoryOnStart** | логический | Если установлено значение true, удаление архива будет выполняться при запуске приложения. По умолчанию это свойство имеет значение false.  Удаление архива обычно выполняется во время пролонгации. Тем не менее, некоторые приложения могут не функционировать достаточно долго, чтобы их можно было запустить. Отсюда следует, что для таких недолговечных приложений удаление архива может никогда не получить шанс выполнить. Если для параметра cleanHistoryOnStart установлено значение true, удаление архива выполняется при запуске приложения. |

Вот несколько fileNamePattern значений с объяснением их последствий.

|  |  |  |
| --- | --- | --- |
| **fileNamePattern** | **Расписание ролловера** | **пример** |
| */wombat/foo.%d*  *или*  */wombat/foo.%d***{**yyyy-MM-dd} | Ежедневный ролловер (в полночь). Из-за пропуска необязательного шаблона времени и даты для спецификатора токена *% d* предполагается шаблон по умолчанию *yyyy-MM-dd* , который соответствует ежедневному переключению. | свойство файла не задано: в течение 23 ноября 2006 года выходные данные журнала будут *отправлены* в файл */wombat/foo.2006-11-23* . В полночь и до 24 числа выходные данные журнала будут направлены в */wombat/foo.2006-11-24* .  Для свойства файла установлено значение */wombat/foo.txt* . В течение 23 ноября 2006 года выходные данные журналов перейдут в файл */wombat/foo.txt* . В полночь *файл foo.txt* будет переименован в */wombat/foo.2006-11-23* . Будет *создан* новый файл */wombat/foo.txt* , а оставшаяся часть записи журнала 24 ноября будет направлена ​​в *foo.txt* . |
| */wombat/%d{yyyy/MM}/foo.txt* | Ролловер в начале каждого месяца. | свойство файла не задано: в течение октября 2006 года выходные данные журнала будут *отправлены* в */wombat/2006/10/foo.txt* . После полуночи 31 октября и до конца ноября выходные данные журнала будут направлены в */wombat/2006/11/foo.txt* .  свойство файла установлено в */wombat/foo.txt* : активный файл журнала всегда будет */wombat/foo.txt* . В течение октября 2006 г. результаты журналирования будут идти в */wombat/foo.txt* . В полночь 31 октября */wombat/foo.txt* будет переименован в */wombat/2006/10/foo.txt* . Будет *создан* новый файл */wombat/foo.txt, в* который выходные данные журнала будут поступать до конца ноября. В полночь 30 ноября */wombat/foo.txt* будет переименован в */wombat/2006/11/foo.txt* и так далее. |
| */wombat/foo.%d{yyyy-ww}.log* | Ролловер в первый день каждой недели. Обратите внимание, что первый день недели зависит от локали. | Как и в предыдущих случаях, за исключением того, что опрокидывание будет происходить в начале каждой новой недели. |
| */wombat/foo%d{yyyy-MM-dd\_HH}.log* | Ролловер в начале каждого часа. | Как и в предыдущих случаях, за исключением того, что опрокидывание будет происходить в верхней части каждого часа. |
| */wombat/foo%d{yyyy-MM-dd\_HH-mm}.log* | Ролловер в начале каждой минуты. | Аналогичен предыдущим случаям, за исключением того, что опрокидывание будет происходить в начале каждой минуты. |
| /wombat/foo%d{yyyy-MM-dd\_HH-mm, UTC}.log | Ролловер в начале каждой минуты. | Аналогично предыдущим случаям, за исключением того, что имена файлов будут выражаться в UTC. |
| foo/%d{yyyy-MM,**aux**}/%d.log | Ролловер ежедневно. Архивы расположены в папке, содержащей год и месяц. | В этом примере первый маркер% d отмечен как **Окс** iliary. Второй токен% d с пропущенным шаблоном времени и даты считается первичным. Таким образом, ролловер будет происходить ежедневно (по умолчанию для% d), а имя папки будет зависеть от года и месяца. Например, в ноябре 2006 года все заархивированные файлы будут помещены в папку / foo / 2006-11 /, например */foo/2006-11/2006-11-14.log* . |

Любые символы прямой или обратной косой черты интерпретируются как разделители папок (каталогов). Любая необходимая папка будет создана по мере необходимости. Таким образом, вы можете легко разместить ваши файлы журналов в отдельных папках.

TimeBasedRollingPolicy поддерживает **автоматическое сжатие файлов**. Эта функция включена, если значение параметра fileNamePattern заканчивается на ***.gz*** или ***.zip*** .

|  |  |  |
| --- | --- | --- |
| **fileNamePattern** | **Расписание ролловера** | **пример** |
| */wombat/foo.%d.gz* | Ежедневный ролловер (в полночь) с автоматическим сжатием GZIP архивных файлов. | свойство файла не задано: в течение 23 ноября 2009 года выходные данные журнала будут *отправлены* в файл */wombat/foo.2009-11-23* . Однако в полночь этот файл будет сжат до уровня */wombat/foo.2009-11-23.gz* . В течение 24 ноября выходные данные журнала будут направлены в */wombat/folder/foo.2009-11-24* до *тех* пор, пока они не будут *перенесены* в начале следующего дня.  Для свойства файла установлено значение /wombat/foo.txt: 23 ноября 2009 года выходные данные журнала будут *переданы* в файл */wombat/foo.txt* . В полночь этот файл будет сжат и переименован в */wombat/foo.2009-11-23.gz* . Будет *создан* новый файл */wombat/foo.txt, в* который выходные данные журнала будут поступать до 24 ноября. В полночь 24 ноября */wombat/foo.txt* будет сжат и переименован в */wombat/foo.2009-11-24.gz* и так далее. |

FileNamePattern служит двойной цели. Сначала, изучая шаблон, logback вычисляет запрошенную периодичность опрокидывания. Во-вторых, он вычисляет имя каждого заархивированного файла. Обратите внимание, что две разные модели могут указывать одну и ту же периодичность. Шаблоны *yyyy-MM* и *yyyy @ MM* задают ежемесячный ролловер, хотя полученные архивные файлы будут иметь разные имена.

Установив свойство файла, вы можете отделить расположение активного файла журнала и расположение архивных файлов журнала. Выходные данные журнала будут направлены в файл, указанный в свойстве file . Отсюда следует, что имя активного файла журнала не будет меняться со временем. Однако, если вы решите опустить свойство файла , то активный файл будет вычисляться заново для каждого периода на основе значения fileNamePattern . Оставив параметр file не установленным, вы можете избежать [ошибок переименования](http://logback.qos.ch/codes.html#renamingError) файлов, которые возникают, когда существуют внешние файловые дескрипторы, ссылающиеся на файлы журналов во время пролонгации.

Свойство maxHistory контролирует максимальное количество сохраняемых архивных файлов, удаляя старые файлы. Например, если вы укажете ежемесячный ролловер и зададите для maxHistory значение 6, архивные файлы на 6 месяцев будут сохранены, а файлы старше 6 месяцев будут удалены. Обратите внимание, что при удалении старых архивных файлов журнала все папки, созданные для архивирования файлов журнала, будут удаляться соответствующим образом.

По различным техническим причинам опрокидывания не приводятся в движение часами, а зависят от прибытия событий регистрации. Например, 8 марта 2002 г. при условии, что для fileNamePattern задано значение *yyyy-MM-dd* (ежедневное переключение), прибытие первого события после полуночи приведет к переключению. Если в течение, скажем, 23 минут и 47 секунд после полуночи не будет событий регистрации, то ролловер действительно произойдет в 00:23:47 AM 9 марта, а не в 0:00 AM. Таким образом, в зависимости от скорости поступления событий, опрокидывание может быть инициировано с некоторой задержкой. Однако, независимо от задержки, алгоритм переноса, как известно, является правильным, в том смысле, что все события регистрации, генерируемые в течение определенного периода, будут выводиться в правильном файле, ограничивающем этот период.

Вот пример конфигурации для RollingFileAppender совместно с TimeBasedRollingPolicy.

*Пример: Пример конфигурации RollingFileAppenderиспользования TimeBasedRollingPolicy (logback-examples / src / main / resources / chapters / appenders / conf / logback-RollingTimeBased.xml)*

<configuration>  
  <appender name="FILE"

class="ch.qos.logback.core.rolling.RollingFileAppender">  
    <file>logFile.log</file>  
    **<rollingPolicy class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">  
      <!-- ежедневный ролловер -->  
      <fileNamePattern>logFile.%d{yyyy-MM-dd}.log</fileNamePattern>  
  
      <!-- сохраняйте историю за 30 дней до 3 ГБ -->  
      <maxHistory>30</maxHistory>  
      <totalSizeCap>3GB</totalSizeCap>  
  
    </rollingPolicy>**  
  
    <encoder>  
      <pattern>%-4relative [%thread] %-5level %logger{35} - %msg%n</pattern>  
    </encoder>  
  </appender>   
  
  <root level="DEBUG">  
    <appender-ref ref="FILE" />  
  </root>  
</configuration>

Следующий пример конфигурации иллюстрирует использование RollingFileAppender связанного с ним TimeBasedRollingPolicy в разумном режиме.

*Пример: Пример конфигурации RollingFileAppenderиспользования TimeBasedRollingPolicy (logback-examples / src / main / resources / chapters / appenders / conf / logback-PrudentTimeBasedRolling.xml)*

<configuration>  
  <appender name="FILE"

class="ch.qos.logback.core.rolling.RollingFileAppender">  
    **<!-- Поддержка множественной JVM записи в один и тот же файл журнала -->**  
    **<prudent>true</prudent>**  
    <rollingPolicy class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">  
      <fileNamePattern>logFile.%d{yyyy-MM-dd}.log</fileNamePattern>  
      <maxHistory>30</maxHistory>   
      <totalSizeCap>3GB</totalSizeCap>  
    </rollingPolicy>  
  
    <encoder>  
      <pattern>%-4relative [%thread] %-5level %logger{35} - %msg%n</pattern>  
    </encoder>  
  </appender>   
  
  <root level="DEBUG">  
    <appender-ref ref="FILE" />  
  </root>  
</configuration>

**Политика изменения размера и времени (основной режим)**

Тут три фильтра:

- по размеру лог-файла

- по количеству лог-файлов

- по общему объему всех лог-файлов

Иногда вы можете захотеть архивировать файлы по дате, но в то же время ограничить размер каждого файла журнала, особенно если инструменты постобработки накладывают ограничения на размер файлов журнала. Чтобы выполнить это требование, войдите в систему с SizeAndTimeBasedRollingPolicy.

Обратите внимание, что TimeBasedRollingPolicy уже позволяет ограничить общий размер архивированных файлов журнала. Если вы хотите ограничить только объединенный размер архивов журналов, то TimeBasedRollingPolicy описанного выше и установки свойства [totalSizeCap](http://logback.qos.ch/manual/appenders.html#tbrpTotalSizeCap) должно быть достаточно.

Вот пример файла конфигурации, демонстрирующий архивирование файла журнала на основе времени и размера.

*Пример: Пример конфигурации для SizeAndTimeBasedFNATP (logback-examples / src / main / resources / chapters / appenders / conf / logback-sizeAndTime.xml)*

Посмотреть как .groovy

<configuration>  
  <appender name="ROLLING" class="ch.qos.logback.core.rolling.RollingFileAppender">  
    <file>mylog.txt</file>  
    <rollingPolicy class="ch.qos.logback.core.rolling.SizeAndTimeBasedRollingPolicy">  
      <!-- rollover daily -->  
      <fileNamePattern>**mylog-%d{yyyy-MM-dd}.%i.txt**</fileNamePattern>  
       **<!-- each file should be at most 100MB, keep 60 days worth of history, but at most 20GB -->**  
       **<maxFileSize>100MB</maxFileSize>**      
       <maxHistory>60</maxHistory>  
       <totalSizeCap>20GB</totalSizeCap>  
    </rollingPolicy>  
    <encoder>  
      <pattern>%msg%n</pattern>  
    </encoder>  
  </appender>  
  
  
  <root level="DEBUG">  
    <appender-ref ref="ROLLING" />  
  </root>  
  
</configuration>

Обратите внимание на маркер преобразования «% i» в дополнение к «% d». **Оба токена% i и% d являются обязательными.**Каждый раз, когда текущий файл журнала достигает значения maxFileSize до окончания текущего периода времени, он будет архивироваться с возрастающим индексом, начиная с 0.

Архивирование на основе размера и времени поддерживает удаление старых архивных файлов. Вам необходимо указать количество периодов для сохранения с помощью свойства maxHistory . Когда ваше приложение остановлено и перезапущено, ведение журнала будет продолжено в правильном месте, то есть с наибольшим индексным номером за текущий период.

В версиях до 1.1.7 в этом документе упоминался компонент с именем SizeAndTimeBasedFNATP. Однако, учитывая, что это SizeAndTimeBasedFNATP предлагает более простую структуру конфигурации, мы больше не документируем SizeAndTimeBasedFNATP. Тем не менее, более раннее использование файлов конфигурации SizeAndTimeBasedFNATP будет продолжать работать нормально. На самом деле SizeAndTimeBasedRollingPolicy реализовано с SizeAndTimeBasedFNATP субкомпонентом.

**FixedWindowRollingPolicy () c SizeBasedTriggeringPolicy()**

Это ограничение по объему файла и количеству лог-файлов.

При пролонгации [FixedWindowRollingPolicy](http://logback.qos.ch/xref/ch/qos/logback/core/rolling/FixedWindowRollingPolicy.html) файлы переименовываются в соответствии с алгоритмом фиксированного окна, как описано ниже.

Опция fileNamePattern представляет шаблон имени файла для архивных ( пролонгированных ) файлов журнала. Эта опция обязательна и должна включать целочисленный токен *% i* где-то внутри шаблона.

Вот доступные свойства для FixedWindowRollingPolicy

|  |  |  |
| --- | --- | --- |
| **Имя свойства** | **Тип** | **Описание** |
| **minIndex** | int | Эта опция представляет нижнюю границу для индекса окна. |
| **maxIndex** | int | Эта опция представляет верхнюю границу для индекса окна. |
| **fileNamePattern** | String | Этот параметр представляет собой шаблон , который будет , за которым следует FixedWindowRollingPolicy при переименовании файлов журнал. Он должен содержать строку *% i* , которая будет указывать позицию, в которую будет вставлено значение текущего индекса окна.  Например, использование *MyLogFile% i.log,* связанного с минимальными и максимальными значениями *1* и *3,* приведет к созданию архивных файлов с именами *MyLogFile1.log* , *MyLogFile2.log* и *MyLogFile3.log* .  Обратите внимание, что сжатие файлов также указывается через это свойство. Например, fileNamePattern, установленный в *MyLogFile% i.log.zip,* означает, что архивные файлы должны быть сжаты с использованием формата *zip* ; Формат *GZ* также поддерживается. |

Принимая во внимание, что фиксированная политика смены окон требует столько операций переименования файлов, сколько размер окна, большие размеры окон настоятельно не рекомендуется. Когда пользователь указывает большие значения, текущая реализация автоматически уменьшает размер окна до 20.

Давайте рассмотрим более конкретный пример политики переноса фиксированных окон. Предположим, что для minIndex установлено значение *1* , для maxIndex установлено значение *3* , для свойства fileNamePattern установлено значение *foo% i.log* , а для свойства файла установлено значение *foo.log* .

|  |  |  |  |
| --- | --- | --- | --- |
| **Количество опрокидываний** | **Активная цель выхода** | **Архивные файлы журнала** | **Описание** |
| 0 | foo.log | - | Ролловер еще не произошел, logback входит в исходный файл. |
| 1 | foo.log | foo1.log | Первый ролловер. *foo.log* переименован в *foo1.log* . Новый файл *foo.log* создается и становится активным выходным объектом. |
| 2 | foo.log | foo1.log, foo2.log | Второй ролловер. *foo1.log* переименован в *foo2.log* . *foo.log* переименован в *foo1.log* . Новый файл *foo.log* создается и становится активным выходным объектом. |
| 3 | foo.log | foo1.log, foo2.log, foo3.log | Третий ролловер. *foo2.log* переименован в *foo3.log* . *foo1.log* переименован в *foo2.log* . *foo.log* переименован в *foo1.log* . Новый файл *foo.log* создается и становится активным выходным объектом. |
| 4 | foo.log | foo1.log, foo2.log, foo3.log | В этом и последующих раундах ролловер начинается с удаления *файла foo3.log* . Другие файлы переименовываются путем увеличения их индекса, как показано в предыдущих шагах. В этом и последующих ролловерах будет три архивных журнала и один активный файл журнала. |

В приведенном ниже файле конфигурации приведен пример настройки RollingFileAppender и FixedWindowRollingPolicy. Обратите внимание, что опция File является обязательной, даже если она содержит некоторую информацию, передаваемую с помощью опции fileNamePattern .

*Пример: Пример конфигурации RollingFileAppenderиспользования FixedWindowRollingPolicy(logback-examples / src / main / resources / chapters / appenders / conf / logback-RollingFixedWindow.xml)*

<configuration>  
  <appender name="FILE"

class="ch.qos.logback.core.rolling.RollingFileAppender">  
    **<file>test.log</file>**  
  
    **<rollingPolicy**

**class="ch.qos.logback.core.rolling.FixedWindowRollingPolicy">  
      <fileNamePattern>tests.%i.log.zip</fileNamePattern>  
      <minIndex>1</minIndex>  
      <maxIndex>3</maxIndex>  
    </rollingPolicy>**  
  
    <triggeringPolicy class="ch.qos.logback.core.rolling.SizeBasedTriggeringPolicy">  
      <maxFileSize>5MB</maxFileSize>  
    </triggeringPolicy>  
    <encoder>  
      <pattern>%-4relative [%thread] %-5level %logger{35} - %msg%n</pattern>  
    </encoder>  
  </appender>  
          
  <root level="DEBUG">  
    <appender-ref ref="FILE" />  
  </root>  
</configuration>

[**Обзор политики запуска**](http://logback.qos.ch/manual/appenders.html#TriggeringPolicy)

[TriggeringPolicy](http://logback.qos.ch/xref/ch/qos/logback/core/rolling/TriggeringPolicy.html)  реализация отвечает за указание, RollingFileAppender когда нужно выполнить опрокидывание.

TriggeringPolicy интерфейс содержит только один метод.

package ch.qos.logback.core.rolling;  
  
import java.io.File;  
import ch.qos.logback.core.spi.LifeCycle;  
  
public interface TriggeringPolicy<E> extends LifeCycle {  
  
  **public boolean isTriggeringEvent(final File activeFile, final <E> event);**  
}

Метод isTriggeringEvent () принимает в качестве параметров активный файл и событие регистрации, которое обрабатывается в данный момент. Конкретная реализация определяет, должно ли произойти опрокидывание, на основании этих параметров.

Наиболее широко используемая политика запуска TimeBasedRollingPolicy, которая уже [обсуждалась ранее](http://logback.qos.ch/manual/appenders.html#TimeBasedRollingPolicy) вместе с другими скользящими политиками.

**[SizeBasedTriggeringPolicy](http://logback.qos.ch/manual/appenders.html" \l "SizeBasedTriggeringPolicy)**

[SizeBasedTriggeringPolicy](http://logback.qos.ch/xref/ch/qos/logback/core/rolling/SizeBasedTriggeringPolicy.html) смотрит на размер текущего активного файла. Если он становится больше указанного размера, он будет сигнализировать владельцу, RollingFileAppender чтобы инициировать опрокидывание существующего активного файла.

SizeBasedTriggeringPolicy принимает только один параметр, а именно maxFileSize , со значением по умолчанию 10 МБ.

Параметр maxFileSize можно указать в байтах, килобайтах, мегабайтах или гигабайтах, добавив числовое значение с суффиксом *КБ* , *МБ* и соответственно *ГБ* . Например, *5000000* , *5000 КБ* , *5 МБ* и *2 ГБ* являются допустимыми значениями, причем первые три являются эквивалентными.

Вот пример конфигурации с RollingFileAppender в сочетании с SizeBasedTriggeringPolicy запуска опрокидывание , когда файл журнала достигает размера 5MB.

*Пример: Пример конфигурации RollingFileAppenderиспользования SizeBasedTriggeringPolicy (logback-examples / src / main / resources / chapters / appenders / conf / logback-RollingSizeBased.xml)*

<configuration>  
  <appender name="FILE" class="ch.qos.logback.core.rolling.RollingFileAppender">  
    <file>test.log</file>  
    <rollingPolicy class="ch.qos.logback.core.rolling.FixedWindowRollingPolicy">  
      <fileNamePattern>test.%i.log.zip</fileNamePattern>  
      <minIndex>1</minIndex>  
      <maxIndex>3</maxIndex>  
    </rollingPolicy>  
  
    **<triggeringPolicy class="ch.qos.logback.core.rolling.SizeBasedTriggeringPolicy">  
      <maxFileSize>5MB</maxFileSize>  
    </triggeringPolicy>**  
    <encoder>  
      <pattern>%-4relative [%thread] %-5level %logger{35} - %msg%n</pattern>  
    </encoder>  
  </appender>  
          
  <root level="DEBUG">  
    <appender-ref ref="FILE" />  
  </root>  
</configuration>

**Logback Classic**

В то время как события регистрации являются общими в logback-core, в logback-classic они всегда являются экземплярами ILoggingEvent. Logback-classic - это не что иное, как специализированный конвейер обработки экземпляров ILoggingEvent.

**SocketAppender и SSLSocketAppender**

Покрытые до настоящего времени аппендеры могут входить только в локальные ресурсы. Напротив,  SocketAppender предназначен для входа в удаленный объект путем передачи ILoggingEventSocketAppenderпо сети проводных экземпляров. При использовании SocketAppender регистрация событий происходит по сети в открытом виде. Однако при использовании SSLSocketAppender события регистрации доставляются по безопасному каналу.

Фактический тип сериализованного события - это то, [LoggingEventVO](http://logback.qos.ch/xref/ch/qos/logback/classic/spi/LoggingEventVO.html) что реализует ILoggingEvent интерфейс. Тем не менее, удаленное ведение журнала не является навязчивым в том, что касается события ведения журнала. На принимающей стороне после десериализации событие может быть зарегистрировано, как если бы оно было сгенерировано локально. Несколько SocketAppenderэкземпляров, запущенных на разных машинах, могут направлять свои выходные данные журналов на центральный сервер журналов, чей формат является фиксированным. SocketAppender не принимает связанный макет, потому что он отправляет сериализованные события на удаленный сервер. SocketAppenderработает выше *протокола управления передачей (TCP)*слой, который обеспечивает надежный, последовательный, контролируемый поток сквозного потока октетов. Следовательно, если удаленный сервер доступен, то события журнала будут в конечном итоге поступать туда. В противном случае, если удаленный сервер недоступен или недоступен, события регистрации будут просто отброшены. Если и когда сервер снова включится, то передача событий будет возобновлена ​​прозрачно. Это прозрачное переподключение выполняется потоком соединителя, который периодически пытается подключиться к серверу.

События регистрации автоматически буферизуются собственной реализацией TCP. Это означает, что если связь с сервером медленная, но все же быстрее, чем скорость создания событий клиентом, медленное сетевое соединение не повлияет на клиента. Однако, если сетевое соединение медленнее, чем скорость создания события, клиент может прогрессировать только с сетевой скоростью. В частности, в крайнем случае, когда сетевая связь с сервером не работает, клиент будет в конечном итоге заблокирован. В качестве альтернативы, если сетевое соединение работает, но сервер не работает, клиент не будет заблокирован, хотя события журнала будут потеряны из-за недоступности сервера.

Даже если a SocketAppenderбольше не подключен к какому-либо регистратору, он не будет собирать мусор при наличии потока соединителя. Поток соединителя существует, только если соединение с сервером не работает. Чтобы избежать этой проблемы сборки мусора, вы должны закрыть SocketAppenderявно. Долгоживущие приложения, которые создают / уничтожают много SocketAppenderэкземпляров, должны знать об этой проблеме сборки мусора. Большинство других приложений могут смело игнорировать это. Если JVM, в которой размещены SocketAppenderвыходы до SocketAppenderзакрытия, либо явным образом, либо после сборки мусора, то в канале могут быть непереданные данные, которые могут быть потеряны. Это распространенная проблема в системах на базе Windows. Чтобы избежать потери данных обычно бывает достаточно , чтобыclose()SocketAppenderлибо явно, либо путем вызова метода LoggerContexts stop()перед выходом из приложения.

Удаленный сервер идентифицируется по свойствам remoteHost и порта . SocketAppenderсвойства перечислены в следующей таблице. SSLSocketAppenderподдерживает множество дополнительных свойств конфигурации, которые подробно описаны в разделе « [Использование SSL»](http://logback.qos.ch/manual/usingSSL.html) .

|  |  |  |
| --- | --- | --- |
| **Имя свойства** | **Тип** | **Описание** |
| **includeCallerData** | boolean | Параметр includeCallerData принимает логическое значение. Если true, данные вызывающего абонента будут доступны для удаленного хоста. По умолчанию данные о вызывающем абоненте не отправляются на сервер. |
| **порт** | int | Номер порта удаленного сервера. |
| **reconnectionDelay** | [Duration](http://logback.qos.ch/apidocs/ch/qos/logback/core/util/Duration.html) | Параметр reconnectionDelay принимает строку продолжительности, например «10 секунд», представляющую время ожидания между каждой неудачной попыткой подключения к серверу. Значение по умолчанию для этой опции составляет 30 секунд. Установка этой опции в ноль отключает возможность переподключения. Обратите внимание, что в случае успешного подключения к серверу не будет никакого потока соединителя. |
| **Queuesize** | int | Свойство queueSize принимает целое число (больше нуля), представляющее количество событий регистрации, которые нужно сохранить для доставки удаленному получателю. Когда размер очереди равен единице, доставка событий на удаленный получатель происходит синхронно. Когда размер очереди больше единицы, новые события ставятся в очередь, предполагая, что в очереди есть свободное место. Использование длины очереди больше единицы может повысить производительность за счет устранения задержек, вызванных временными сетевыми задержками.  Смотрите также свойство eventDelayLimit . |
| **eventDelayLimit** | [Duration](http://logback.qos.ch/apidocs/ch/qos/logback/core/util/Duration.html) | Опция eventDelayLimit принимает строку продолжительности, например «10 секунд». Он представляет время ожидания перед удалением событий, если локальная очередь заполнена, то есть уже содержит события queueSize . Это может произойти, если удаленный хост постоянно медленно принимает события. Значение по умолчанию для этой опции составляет 100 миллисекунд. |
| **remoteHost** | String | Имя хоста сервера. |
| **SSL** | SSLConfiguration | Поддерживается только для SSLSocketAppender, это свойство предоставляет конфигурацию SSL, которая будет использоваться приложением, как описано в разделе [Использование SSL](http://logback.qos.ch/manual/usingSSL.html) . |

**Параметры сервера журналов**

Стандартный дистрибутив Logback Classic включает в себя две опции для серверов, которые можно использовать для получения событий регистрации от SocketAppenderили SSLSocketAppender.

* ServerSocketReceiverи его поддерживающий SSL аналог SSLServerSocketReceiver- это компоненты-получатели, которые можно настроить в *файле* конфигурации *logback.xml* приложения, чтобы получать события от удаленного приложения-сокета. См. [Приемники](http://logback.qos.ch/manual/receivers.html) для деталей конфигурации и примеров использования.
* SimpleSocketServerи его аналог с поддержкой SSL SimpleSSLSocketServerпредлагают простое в использовании автономное приложение Java, которое разработано для настройки и запуска из интерфейса командной строки вашей оболочки. Эти приложения просто ждать регистрации событий из SocketAppenderили SSLSocketAppender клиентов. Каждое полученное событие регистрируется в соответствии с политикой локального сервера. Примеры использования приведены ниже.

**Использование SimpleSocketServer**

SimpleSocketServerПриложение принимает два аргумента командной строки: *порт* и *CONFIGFILE* ; где *порт* - это порт для прослушивания, а *configFile* - это скрипт конфигурации в формате XML.

Предполагая, что вы находитесь в *каталоге logback-examples /* , начните SimpleSocketServerсо следующей команды:

java ch.qos.logback.classic.net.SimpleSocketServer 6000 \ src / main / java / chapters / appenders / socket / server1.xml

где 6000 - номер порта для прослушивания, а *server1.xml* - сценарий конфигурации, который добавляет a ConsoleAppenderи a RollingFileAppenderк корневому логгеру. После того, как вы начали SimpleSocketServer, вы можете отправлять в него журнал событий от нескольких клиентов, используя SocketAppender. Примеры, связанные с этим руководством, включают в себя два таких клиента: chapters.appenders.SocketClient1и chapters.appenders.SocketClient2Оба клиента ждут, пока пользователь наберет строку текста на консоли. Текст инкапсулируется в событие регистрации уровня отладки и затем отправляется на удаленный сервер. Два клиента отличаются по конфигурации SocketAppender. SocketClient1настраивает приложение автоматически, в то время как SocketClient2 требуется файл конфигурации.

Предполагая, SimpleSocketServerчто работает на локальном хосте, вы подключаетесь к нему с помощью следующей команды:

java chapters.appenders.socket.SocketClient1 localhost 6000

Каждая строка, которую вы вводите, должна появиться на консоли, SimpleSocketServerзапущенной на предыдущем шаге. Если вы остановите и перезапустите SimpleSocketServerклиент, он будет прозрачно переподключен к новому экземпляру сервера, хотя события, сгенерированные при отключении, будут просто (и безвозвратно) потеряны.

В отличие SocketClient1от примера приложения SocketClient2само по себе не настраивает logback. Требуется файл конфигурации в формате XML. Файл конфигурации *client1.xml,* показанный ниже, создает SocketAppender и присоединяет его к корневому логгеру.

*Пример: конфигурация SocketAppender (logback-examples / src / main / resources / chapters / appenders / socket / client1.xml)*

<configuration>  
            
  <appender name="SOCKET" class="ch.qos.logback.classic.net.SocketAppender">  
    <remoteHost>${host}</remoteHost>  
    <port>${port}</port>  
    <reconnectionDelay>10000</reconnectionDelay>  
    <includeCallerData>${includeCallerData}</includeCallerData>  
  </appender>  
  
  <root level="DEBUG">  
    <appender-ref ref="SOCKET" />  
  </root>    
  
</configuration>

Обратите внимание, что в приведенных выше конфигурационных сценариях значения для свойств remoteHost , port и includeCallerData задаются не напрямую, а в качестве замененных переменных ключей. Значения для переменных могут быть указаны как системные свойства:

java -Dhost = localhost -Dport = 6000 -DincludeCallerData = false \ chapters.appenders.socket.SocketClient2 src / main / java / chapters / appenders / socket / client1.xml

Эта команда должна дать результаты, аналогичные предыдущему SocketClient1 примеру.

Позвольте нам еще раз подчеркнуть, что сериализация событий регистрации не является навязчивой. Десериализованное событие несет ту же информацию, что и любое другое событие регистрации. Им можно манипулировать, как если бы он был создан локально; за исключением того, что сериализованные события регистрации по умолчанию не включают данные вызывающего абонента. Вот пример, чтобы проиллюстрировать это. Сначала начните SimpleSocketServerсо следующей команды:

java ch.qos.logback.classic.net.SimpleSocketServer 6000 \ src / main / java / chapters / appenders / socket / server2.xml

Файл конфигурации *server2.xml* создает ConsoleAppenderмакет, чей макет выводит имя файла вызывающего абонента и номер строки вместе с другой информацией. Если вы запустите SocketClient2файл конфигурации *client1.xml,* как и ранее, вы заметите, что вывод на стороне сервера будет содержать два вопросительных знака в скобках вместо имени файла и номера строки вызывающей стороны:

2006-11-06 17: 37: 30,968 DEBUG [Thread-0] [?:?] Chapters.appenders.socket.SocketClient2 - Привет

Результат можно легко изменить, указавSocketAppender включить данные вызывающего абонента, установив для параметра includeCallerData значение true. Используя следующую команду, добьемся цели:

java -Dhost = localhost -Dport = 6000 -DincludeCallerData = true \

chapters.appenders.socket.SocketClient2 src / main / java / chapters / appenders / socket / client1.xml

Поскольку десериализованные события могут обрабатываться так же, как локально сгенерированные события, они даже могут отправляться на второй сервер для дальнейшей обработки. В качестве упражнения вы можете настроить два сервера, где первый сервер туннелирует события, которые он получает от своих клиентов, на второй сервер.

**Использование SimpleSSLSocketServer**

SimpleSSLSocketServerТребует того же самого *порта* и *CONFIGFILE* аргументы командной строки , используемый SimpleSocketServer. Кроме того, необходимо указать местоположение и пароль для учетных данных X.509 сервера журналов, используя системные свойства, указанные в командной строке.

Предполагая, что вы находитесь в *каталоге logback-examples /* , начните SimpleSSLSocketServerсо следующей команды:

java -Djavax.net.ssl.keyStore = src / main / java / chapters / appenders / socket / ssl / keystore.jks \ -Djavax.net.ssl.keyStorePassword = changeit \ ch.qos.logback.classic.net.SimpleSSLSocketServer 6000 \ src / main / java / chapters / appenders / socket / ssl / server.xml

Этот пример выполняется SimpleSSLSocketServerс использованием учетных данных X.509, которые подходят только для тестирования и экспериментов. **Перед использованием SimpleSSLSocketServerв рабочих настройках вы должны получить соответствующие учетные данные X.509 для идентификации вашего сервера журналов** . См. [Использование SSL](http://logback.qos.ch/manual/usingSSL.html) для более подробной информации.

Поскольку конфигурация сервера debug="true" указана в корневом элементе, вы увидите в журнале запуска сервера конфигурацию SSL, которая будет использоваться. Это полезно для проверки правильности реализации локальных политик безопасности.

При SimpleSSLSocketServerзапуске вы можете подключиться к серверу с помощью SSLSocketAppender. В следующем примере показана необходимая конфигурация appender:

*Пример: конфигурация SSLSocketAppender (logback-examples / src / main / resources / chapters / appenders / socket / ssl / client.xml)*

<configuration debug="true">  
            
  <appender name="SOCKET" class="ch.qos.logback.classic.net.SSLSocketAppender">  
    <remoteHost>${host}</remoteHost>  
    <port>${port}</port>  
    <reconnectionDelay>10000</reconnectionDelay>  
    <ssl>  
      <trustStore>  
        <location>${truststore}</location>  
        <password>${password}</password>  
      </trustStore>  
    </ssl>  
  </appender>  
  
  <root level="DEBUG">  
    <appender-ref ref="SOCKET" />  
  </root>    
  
</configuration>

Следует отметить , что, так же , как и в предыдущем примере, значения для RemoteHost , порта задаются с помощью переменных ключей замещенных. Кроме того, обратите внимание на наличие свойства ssl и его вложенного свойства trustStore , которое указывает местоположение и пароль хранилища доверенных сертификатов с использованием подставляемых переменных. Эта конфигурация необходима, потому что наш пример сервера использует самозаверяющий сертификат. См. [Использование SSL](http://logback.qos.ch/manual/usingSSL.html) для получения дополнительной информации о свойствах конфигурации SSL для SSLSocketAppender.

Мы можем запустить клиентское приложение, используя эту конфигурацию, указав значения переменных замещения в командной строке в качестве системных свойств:

java -Dhost = localhost -Dport = 6000 \ -Dtruststore = file: src / main / java / chapters / appenders / socket / ssl / truststore.jks \ -Dpassword = changeit \ chapters.appenders.socket.SocketClient2 src / main / java /chapters/appenders/socket/ssl/client.xml

Как и в предыдущих примерах, вы можете ввести сообщение по запросу клиентского приложения, и оно будет доставлено на сервер регистрации (теперь по безопасному каналу), где оно будет отображаться на консоли.

Обратите внимание, что свойство *хранилища доверенных сертификатов,* указанное в командной строке, указывает URL-адрес файла, который определяет местоположение *хранилища доверенных сертификатов* . Вы также можете использовать URL пути к классу, как описано в [разделе Использование SSL](http://logback.qos.ch/manual/usingSSL.html) .

Как мы видели ранее при запуске сервера, поскольку конфигурация клиента debug="true"указана в корневом элементе, регистрация запуска клиента включает в себя сведения о конфигурации SSL в качестве помощи для аудита соответствия локальной политики.

**ServerSocketAppender и SSLServerSocketAppender**

SocketAppenderКомпонент (и его SSL-включен аналог) обсуждали ранее разработаны , чтобы позволить приложению подключаться к удаленному серверу регистрации по сети с целью доставки протоколирования событий на сервер. В некоторых ситуациях может быть неудобно или нецелесообразно, чтобы приложение инициировало соединение с удаленным сервером регистрации. Для этих ситуаций Logback предлагает . [ServerSocketAppender](http://logback.qos.ch/xref/ch/qos/logback/classic/net/server/ServerSocketAppender)

Вместо того чтобы инициировать соединение с удаленным сервером журналов, ServerSocketAppenderпассивно прослушивает сокет TCP в ожидании входящих соединений от удаленных клиентов. Журнальные события, которые доставляются аппендиру, распространяются на каждого подключенного клиента. События журнала, которые происходят, когда ни один клиент не подключен, *суммируются* .

В дополнение к основному ServerSocketAppenderпредложению Logback , которое распределяет события регистрации каждого подключенного клиента, используя безопасный, зашифрованный канал. Кроме того, приложение с поддержкой SSL полностью поддерживает взаимную аутентификацию на основе сертификатов, которую можно использовать для обеспечения того, чтобы только авторизованные клиенты могли подключаться к приложению для получения событий регистрации. [SSLServerSocketAppender](http://logback.qos.ch/xref/ch/qos/logback/classic/net/server/SSLServerSocketAppender)

Подход к кодированию событий регистрации для передачи по проводам идентичен тому, который используется SocketAppender; каждое событие является сериализованным экземпляром ILoggingEvent. Только направление инициации соединения меняется на противоположное. Хотя SocketAppender действует как активный одноранговый узел при установлении соединения с сервером журналов, ServerSocketAppenderявляется пассивным; он прослушивает входящие соединения от клиентов.

В ServerSocketAppenderподтипы предназначены для использования исключительно с Logback *приемника* компонентов. См. [Приемники](http://logback.qos.ch/manual/receivers.html) для получения дополнительной информации об этом типе компонента.

Следующие свойства конфигурации поддерживаются ServerSocketAppender:

|  |  |  |
| --- | --- | --- |
| **Имя свойства** | **Тип** | **Описание** |
| **адрес** | String | Адрес локального сетевого интерфейса, по которому будет слушать приложение. Если это свойство не указано, appender будет прослушивать все сетевые интерфейсы. |
| **includeCallerData** | boolean | Если true, данные вызывающего абонента будут доступны для удаленного хоста. По умолчанию данные о вызывающем абоненте не отправляются клиенту. |
| **port** | int | Номер порта, по которому будет слушать приложение. |
| **SSL** | SSLConfiguration | Поддерживается только для SSLServerSocketAppender, это свойство предоставляет конфигурацию SSL, которая будет использоваться приложением, как описано в разделе [Использование SSL](http://logback.qos.ch/manual/usingSSL.html) . |

В следующем примере показана конфигурация, которая использует ServerSocketAppender:

*Пример: базовая конфигурация ServerSocketAppender (logback-examples / src / main / resources / chapters / appenders / socket / server4.xml)*

<configuration debug="true">  
  <appender name="SERVER"   
    class="ch.qos.logback.classic.net.server.ServerSocketAppender">  
    <port>${port}</port>  
    <includeCallerData>${includeCallerData}</includeCallerData>  
  </appender>  
  
  <root level="debug">  
    <appender-ref ref="SERVER" />  
  </root>    
  
</configuration>

Обратите внимание, что эта конфигурация отличается от предыдущих примеров, использующих SocketAppenderтолько *класс,* указанный для appender, и в отсутствие свойства remoteHost - этот appender пассивно ожидает входящих подключений с удаленных хостов, а не открывает подключение к удаленному серверу журналирования.

Следующий пример иллюстрирует использование конфигурации SSLServerSocketAppender.

*Пример: базовая конфигурация SSLServerSocketAppender (logback-examples / src / main / resources / chapters / appenders / socket / ssl / server3.xml)*

<configuration debug="true">  
  <appender name="SERVER"   
    class="ch.qos.logback.classic.net.server.SSLServerSocketAppender">  
    <port>${port}</port>  
    <includeCallerData>${includeCallerData}</includeCallerData>  
    <ssl>  
      <keyStore>  
        <location>${keystore}</location>  
        <password>${password}</password>  
      </keyStore>  
    </ssl>  
  </appender>  
  
  <root level="debug">  
    <appender-ref ref="SERVER" />  
  </root>    
  
</configuration>

Принципиальные различия между этой конфигурацией и предыдущей конфигурацией состоят в том, что атрибут *класса* appender определяет SSLServerSocketAppenderтип и наличие вложенного элемента ssl, который указывает, в этом примере, конфигурацию хранилища ключей, содержащего учетные данные X.509 для appender. См. [Использование SSL](http://logback.qos.ch/manual/usingSSL.html) для получения информации о свойствах конфигурации SSL.

Поскольку ServerSocketAppenderподтипы предназначены для использования с компонентами приемника, мы отложим представление иллюстративных примеров к главе, озаглавленной « [Приемники»](http://logback.qos.ch/manual/receivers.html) .

**SMTPAppender**

[SMTPAppender](http://logback.qos.ch/xref/ch/qos/logback/classic/net/SMTPAppender.html) накапливает запись событий в одном или более фиксированного размере буферов и передает содержимое соответствующего буфера в сообщении электронной почты после того, как происходит заданный пользователь события. Передача (отправка) электронной почты SMTP выполняется асинхронно. По умолчанию передача электронной почты инициируется событием регистрации уровня ERROR. Более того, по умолчанию для всех событий используется один буфер.

Различные свойства для SMTPAppender суммированы в следующей таблице.

|  |  |  |
| --- | --- | --- |
| **Имя свойства** | **Тип** | **Описание** |
| **smtpHost** | String | Имя хоста SMTP-сервера. Этот параметр является обязательным. |
| **smtpPort** | int | Порт, на котором слушает SMTP-сервер. По умолчанию 25. |
| **to** | String | Адрес электронной почты получателя в виде *шаблона* . Шаблон оценивается заново с инициирующим событием в качестве входных данных для каждого исходящего электронного письма. Можно указать нескольких получателей, разделив адреса назначения запятыми. Кроме того, несколько получателей также могут быть указаны с использованием нескольких <to> элементов. |
| **from** | String | Отправитель сообщений электронной почты отправляется SMTPAppenderв [обычном формате адреса электронной почты](http://en.wikipedia.org/wiki/Email_address) . Если вы хотите указать имя отправителя, используйте формат «Adam Smith & lt; smith@moral.org>» так что сообщение появляется как исходящее от «Адама Смита <smith@moral.org>». |
| **subject** | String | Тема письма. Это может быть любое значение, принятое в качестве действительного шаблона преобразования в [PatternLayout](http://logback.qos.ch/manual/layouts.html#ClassicPatternLayout) . Макеты будут обсуждаться в следующей главе.  Исходящее сообщение электронной почты будет иметь строку темы, соответствующую применению шаблона в событии регистрации, которое инициировало сообщение электронной почты.  Предполагая, что для параметра « Тема» задано «Log:% logger -% msg», а регистратор инициирующего события называется «com.foo.Bar» и содержит сообщение «Hello world», тогда исходящее письмо будет иметь строку темы ». Журнал: com.foo.Bar - Hello World ".  По умолчанию для этого параметра установлено значение «% logger {20} -% m». |
| **discriminator** | [Discriminator](http://logback.qos.ch/xref/ch/qos/logback/core/sift/Discriminator.html) | С помощью дискриминатора , SMTPAppender может рассеивать входящие события в различные буфера в соответствии со значением возвращаемого дискриминатором. Дискриминатор по умолчанию всегда возвращает одно и то же значение, поэтому один и тот же буфер используется для всех событий.  Указав дискриминатор, отличный от значения по умолчанию, можно получать сообщения электронной почты, содержащие события, относящиеся к конкретному пользователю, сеансу пользователя или IP-адресу клиента. |
| **evaluator** | [IEvaluator](http://logback.qos.ch/xref/ch/qos/logback/classic/boolex/IEvaluator.html) | Эта опция объявляется путем создания нового <EventEvaluator/>элемента. Имя класса, которое пользователь желает использовать в качестве SMTPAppender«s», Evaluatorдолжно быть указано через атрибут *class* .  При отсутствии этой опции SMTPAppender назначается экземпляр [OnErrorEvaluator,](http://logback.qos.ch/xref/ch/qos/logback/classic/boolex/OnErrorEvaluator.html) который запускает передачу электронной почты, когда он встречает событие уровня *ERROR* или выше.  Logback поставляется с несколькими другими оценщиками, а именно [OnMarkerEvaluator](http://logback.qos.ch/xref/ch/qos/logback/classic/boolex/OnMarkerEvaluator.html) (обсуждается ниже) и с мощным оценщиком [JaninoEventEvaluator](http://logback.qos.ch/xref/ch/qos/logback/classic/boolex/JaninoEventEvaluator.html), который обсуждается в [другой главе](http://logback.qos.ch/manual/filters.html#evalutatorFilter) . Более свежие версии logback поставляются с еще более мощным средством оценки [GEventEvaluator](http://logback.qos.ch/manual/filters.html#GEventEvaluator). |
| **cyclicBufferTracker** | [CyclicBufferTracker](http://logback.qos.ch/xref/ch/qos/logback/core/spi/CyclicBufferTracker.html) | Как видно из названия, экземпляр CyclicBufferTrackerкласса отслеживает циклические буферы. Это делается на основе ключей, возвращаемых дискриминатором (см. Выше).  Если вы не укажете cyclicBufferTracker , экземпляр [CyclicBufferTracker](http://logback.qos.ch/xref/ch/qos/logback/core/spi/CyclicBufferTracker.html) будет создан автоматически. По умолчанию этот экземпляр будет хранить события в циклическом буфере размером 256. Вы можете изменить размер с помощью опции bufferSize (см. Ниже). |
| **username** | String | Значение имени пользователя для использования при простой аутентификации пользователя / пароля. По умолчанию этот параметр имеет значение null. |
| **password** | String | Значение пароля, используемое для простой аутентификации пользователя / пароля. По умолчанию этот параметр имеет значение null. |
| **STARTTLS** | boolean | Если для этого параметра задано значение true, этот аппендер будет запускать команду STARTTLS (если сервер ее поддерживает), в результате чего подключение переключится на SSL. Обратите внимание, что соединение изначально не зашифровано. По умолчанию этот параметр имеет значение false. |
| **SSL** | boolean | Если для этого параметра задано значение true, этот аппендер откроет SSL-соединение с сервером. По умолчанию этот параметр имеет значение false. |
| **charsetEncoding** | String | Исходящее сообщение электронной почты будет закодировано в назначенной [кодировке](https://docs.oracle.com/javase/8/docs/api/java/nio/charset/Charset.html) . Кодировка по умолчанию - «UTF-8», которая подходит для большинства целей. |
| **localhost** | String | Если имя хоста SMTP-клиента настроено неправильно, например, если имя хоста клиента не полностью определено, некоторые SMTP-серверы могут отклонять команды HELO / EHLO, отправленные клиентом. Чтобы преодолеть эту проблему, вы можете установить в качестве значения свойства localhost полное имя хоста клиента. См. Также свойство «mail.smtp.localhost» в документации по пакету [com.sun.mail.smtp](http://javamail.kenai.com/nonav/javadocs/com/sun/mail/smtp/package-summary.html) . |
| **asynchronousSending** | boolean | Это свойство определяет, выполняется ли передача электронной почты асинхронно или нет. По умолчанию свойство asynchronousSending имеет значение true. Однако при определенных обстоятельствах асинхронная отправка может быть неуместной. Например, если ваше приложение использует SMTPAppenderдля отправки оповещений в ответ на фатальную ошибку, а затем завершает работу, соответствующая ветка может не иметь времени для отправки оповещения по электронной почте. В этом случае установите для свойства asynchronousSending значение «false» для синхронной передачи электронной почты. |
| **includeCallerData** | boolean | По умолчанию includeCallerData имеет значение false. Вы должны установить includeCallerData в trueслучае asynchronousSending включен , и вы хотите , чтобы включить данные вызывающего абонента в журналах. |
| **sessionViaJNDI** | boolean | SMTPAppenderполагается на javax.mail.Sessionотправку сообщений электронной почты. По умолчанию для sessionViaJNDI задано значение, falseпоэтому javax.mail.Session экземпляр создаетсяSMTPAppender сам по себе со свойствами, указанными пользователем. Если свойство sessionViaJNDI установлено в значение true, javax.mail.Sessionобъект будет извлечен через JNDI. Смотрите также jndiLocation недвижимость.  Извлечение Sessionчерез JNDI может уменьшить количество мест, в которых вам нужно сконфигурировать / перенастроить одну и ту же информацию, что сделает ваше приложение более [сухим](http://en.wikipedia.org/wiki/Don't_repeat_yourself) . Для получения дополнительной информации о настройке ресурсов в Tomcat см. [JNDI Resources How-to](http://tomcat.apache.org/tomcat-6.0-doc/jndi-resources-howto.html#JavaMail_Sessions) . **ОСТЕРЕГАЙТЕСЬ** Как отмечается в этом документе, не забудьте удалить *mail.jar* и *activation.jar* из ваших веб-приложений *WEB-INF / Lib* папки при извлечении Sessionиз JNDI. |
| **jndiLocation** | String | Расположение, где javax.mail.Session находится в JNDI. По умолчанию для jndiLocation установлено значение "java: comp / env / mail / Session" . |

SMTPAppenderХранит только последние 256 записи событий в циклическом буфере, выбрасывая старые события , когда его буфер переполняется. Таким образом, число событий регистрации, доставленных в любом электронном письме, SMTPAppenderограничено верхним пределом 256. Это ограничивает требования к памяти, в то же время обеспечивая разумное количество контекста приложения.

SMTPAppenderОпирается на API JavaMail. Он был протестирован с JavaMail API версии 1.4. Для JavaMail API требуется пакет JavaBeans Activation Framework. Вы можете загрузить [JavaMail API](http://java.sun.com/products/javamail/) и [JavaBeans Activation Framework](http://java.sun.com/beans/glasgow/jaf.html) с их соответствующих веб-сайтов. Обязательно поместите эти два jar-файла в путь к классам, прежде чем пытаться использовать следующие примеры.

Пример приложения [chapters.appenders.mail.EMail](http://logback.qos.ch/xref/chapters/appenders/mail/EMail.html) генерирует несколько сообщений журнала, за которыми следует одно сообщение об ошибке. Требуется два параметра. Первый параметр - это целое число, соответствующее количеству событий регистрации, которые нужно сгенерировать. Второй параметр - это файл конфигурации logback. Последнее событие регистрации, генерируемое приложением *EMail* , ОШИБКА, инициирует передачу сообщения электронной почты.

Вот пример файла конфигурации, предназначенного для Emailприложения:

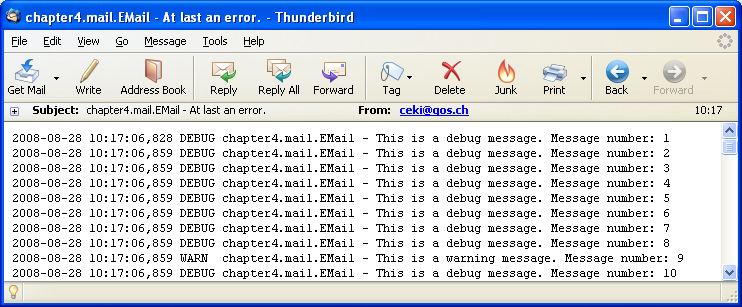
*Пример: пример SMTPAppenderконфигурации (logback-examples / src / main / resources / chapters / appenders / mail / mail1.xml)*

<configuration>     
  <appender name="EMAIL" class="ch.qos.logback.classic.net.SMTPAppender">  
    <smtpHost>ADDRESS-OF-YOUR-SMTP-HOST</smtpHost>  
    <to>EMAIL-DESTINATION</to>  
    <to>ANOTHER\_EMAIL\_DESTINATION</to> <!-- additional destinations are possible -->  
    <from>SENDER-EMAIL</from>  
    <subject>TESTING: %logger{20} - %m</subject>  
    <layout class="ch.qos.logback.classic.PatternLayout">  
      <pattern>%date %-5level %logger{35} - %message%n</pattern>  
    </layout>         
  </appender>  
  
  <root level="DEBUG">  
    <appender-ref ref="EMAIL" />  
  </root>    
</configuration>

Перед тем как опробовать chapters.appenders.mail.Emailприложение с указанным выше файлом конфигурации, необходимо установить smtpHost , чтобы и от свойств до значений , подходящего для вашей среды. После того, как вы установили правильные значения в файле конфигурации, выполните следующую команду:

java chapters.appenders.mail.EMail 100 src / main / java / chapters / appenders / mail / mail1.xml

Указанный вами получатель должен получить сообщение электронной почты, содержащее 100 событий журнала, отформатированных с помощью PatternLayoutприведенного ниже рисунка, который представляет собой полученное сообщение электронной почты, как показано в Mozilla Thunderbird.



В следующем примере файла конфигурации *mail2.xml* значения свойств smtpHost , to и from определяются путем подстановки переменных. Вот соответствующая часть *mail2.xml* .

<appender name="EMAIL" class="ch.qos.logback.classic.net.SMTPAppender">  
  <smtpHost>${smtpHost}</smtpHost>  
  <to>${to}</to>  
  <from>${from}</from>  
  <layout class="ch.qos.logback.classic.html.HTMLLayout"/>  
</appender>

Вы можете передать необходимые параметры в командной строке:

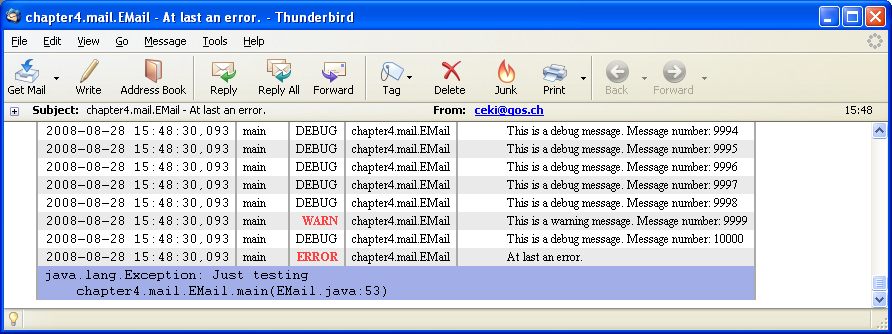
java -Dfrom=source@xyz.com -Dto=recipient@xyz.com -DsmtpHost = some\_smtp\_host \

chapters.appenders.mail.EMail 10000 src / main / java / chapters / appenders / mail / mail2.xml

Обязательно замените значения в соответствии с вашей средой.

Обратите внимание, что в этом последнем примере PatternLayout были заменены HTMLLayoutформаты журналов в виде таблицы HTML. Вы можете изменить список и порядок столбцов, а также CSS таблицы. Пожалуйста, обратитесь к документации [HTMLLayout](http://logback.qos.ch/manual/layouts.html#ClassicHTMLLayout) для получения дополнительной информации.

Учитывая, что размер циклического буфера равен 256, получатель должен увидеть сообщение электронной почты, содержащее 256 событий, удобно отформатированных в таблице HTML. Обратите внимание, что этот запуск chapters.appenders.mail.Emailприложения сгенерировал 10 000 событий, из которых только последние 256 были включены в исходящую электронную почту.



Почтовые клиенты, такие как Mozilla Thunderbird, Eudora или MS Outlook, предлагают достаточно хорошую поддержку CSS для электронной почты в формате HTML. Однако иногда они автоматически переводят HTML в открытый текст. Например, для просмотра электронной почты в формате HTML в Thunderbird необходимо установить параметр «Вид → Текст сообщения как → Исходный HTML». Yahoo! Поддержка почты для электронной почты в формате HTML, в частности, поддержка CSS очень хорошая. С другой стороны, Gmail, хотя и учитывает базовую структуру таблиц HTML, игнорирует внутреннее форматирование CSS. Gmail поддерживает встроенное форматирование CSS, но, поскольку встроенный CSS сделает вывод слишком объемным, HTMLLayoutон не использует встроенный CSS.

**Пользовательский размер буфера**

По умолчанию исходящее сообщение будет содержать последние 256 сообщений, просмотренных пользователем SMTPAppender. Если ваше сердце желает этого, вы можете установить другой размер буфера, как показано в следующем примере.

*Пример: SMTPAppenderконфигурация с пользовательским размером буфера (logback-examples / src / main / resources / chapters / appenders / mail / customBufferSize.xml)*

<configuration>     
  <appender name="EMAIL" class="ch.qos.logback.classic.net.SMTPAppender">  
    <smtpHost>${smtpHost}</smtpHost>  
    <to>${to}</to>  
    <from>${from}</from>  
    <subject>%logger{20} - %m</subject>  
    <layout class="ch.qos.logback.classic.html.HTMLLayout"/>  
  
    **<cyclicBufferTracker class="ch.qos.logback.core.spi.CyclicBufferTracker">**  
      **<!-- send just one log entry per email -->**  
      **<bufferSize>1</bufferSize>**  
    **</cyclicBufferTracker>**  
  </appender>  
  
  <root level="DEBUG">  
    <appender-ref ref="EMAIL" />  
  </root>    
</configuration>

**Инициирующее событие**

Если свойство Evaluator не установлено, по SMTPAppenderумолчанию используется экземпляр [OnErrorEvaluator,](http://logback.qos.ch/xref/ch/qos/logback/classic/boolex/OnErrorEvaluator.html) который запускает передачу электронной почты при обнаружении события уровня ERROR. Хотя инициирование исходящей электронной почты в ответ на ошибку является относительно разумным, можно переопределить это поведение по умолчанию, предоставив другую реализацию EventEvaluatorинтерфейса.

SMTPAppenderПредставляет каждое входящее событие в его оценщик, вызвав evaluate()метод для того , чтобы проверить , следует ли событие запуска по электронной почте или просто быть помещено в циклическом буфере. Когда оценщик дает положительный ответ на свою оценку, отправляется электронное письмо. SMTPAppenderСодержит один и только один объект оценщика. Этот объект может управлять своим внутренним состоянием. В иллюстративных целях CounterBasedEvaluator перечисленный ниже класс реализует оценщик событий, посредством которого каждое 1024-е событие вызывает сообщение электронной почты.

*Пример: EventEvaluatorреализация, которая оценивает trueкаждое 1024-е событие ( [logback-examples / src / main / java / chapters / appenders / mail / CounterBasedEvaluator.java](http://logback.qos.ch/xref/chapters/appenders/mail/CounterBasedEvaluator.html) )*

package chapters.appenders.mail;  
  
import ch.qos.logback.core.boolex.EvaluationException;  
import ch.qos.logback.core.boolex.EventEvaluator;  
import ch.qos.logback.core.spi.ContextAwareBase;  
  
public class CounterBasedEvaluator extends ContextAwareBase implements EventEvaluator {  
  
  static int LIMIT = 1024;  
  int counter = 0;  
  String name;  
  
  **public boolean evaluate(Object event) throws NullPointerException,  
      EvaluationException {  
    counter++;  
  
    if (counter == LIMIT) {  
      counter = 0;  
  
      return true;  
    } else {  
      return false;  
    }  
  }**  
  
  public String getName() {  
    return name;  
  }  
  
  public void setName(String name) {  
    this.name = name;  
  }  
}

Обратите внимание, что этот класс расширяет ContextAwareBaseи реализует EventEvaluator. Это позволяет пользователю сконцентрироваться на его основных функциях EventEvaluatorи позволить базовому классу обеспечить общую функциональность.

Установка опции EvaluatorSMTPAppender указывает ему использовать пользовательский оценщик. Следующий файл конфигурации присоединяет SMTPAppender к корневому логгеру. Этот appender использует CounterBasedEvaluatorэкземпляр в качестве оценщика событий.

*Пример: SMTPAppenderс пользовательским Evaluatorразмером и размером буфера (logback-examples / src / main / resources / chapters / appenders / mail / mail3.xml)*

<configuration>  
  <appender name="EMAIL" class="ch.qos.logback.classic.net.SMTPAppender">  
    **<evaluator class="chapters.appenders.mail.CounterBasedEvaluator" />**  
    <smtpHost>${smtpHost}</smtpHost>  
    <to>${to}</to>  
    <from>${from}</from>  
    <subject>%logger{20} - %m</subject>  
  
    <layout class="ch.qos.logback.classic.html.HTMLLayout"/>  
  </appender>  
  
  <root level="DEBUG">  
    <appender-ref ref="EMAIL" />  
  </root>    
</configuration>

**Маркерный запуск**

Хотя это разумно, политика запуска по умолчанию, при которой каждое событие уровня ERROR инициирует исходящую электронную почту, может привести к слишком большому количеству электронных писем, загромождая почтовый ящик целевого пользователя. Logback поставляется с другой политикой запуска, которая называется [OnMarkerEvaluator](http://logback.qos.ch/xref/ch/qos/logback/classic/boolex/OnMarkerEvaluator.html) . Он основан на маркерах. По сути, электронные письма запускаются, только если событие помечено указанным пользователем маркером. Следующий пример должен прояснить ситуацию.

Приложение [Marked\_EMail](http://logback.qos.ch/xref/chapters/appenders/mail/Marked_EMail.html) содержит несколько операторов регистрации, некоторые из которых имеют уровень ERROR. Одно заслуживающее внимания утверждение содержит маркер. Вот соответствующий код.

Marker notifyAdmin = MarkerFactory.getMarker("NOTIFY\_ADMIN");  
logger.error(**notifyAdmin**,  
  "This is a serious an error requiring the admin's attention",  
   new Exception("Just testing"));

Следующий файл конфигурации будет инициировать исходящие электронные письма только при наличии событий с маркерами NOTIFY\_ADMIN или TRANSACTION\_FAILURE.

*Пример: SMTPAppenderwith OnMarkerEvaluator(logback-examples / src / main / resources / chapters / appenders / mail / mailWithMarker.xml)*

Посмотреть как .groovy

<configuration>  
  <appender name="EMAIL" class="ch.qos.logback.classic.net.SMTPAppender">  
    **<evaluator class="ch.qos.logback.classic.boolex.OnMarkerEvaluator">  
      <marker>NOTIFY\_ADMIN</marker>  
      <!-- you specify add as many markers as you want -->  
      <marker>TRANSACTION\_FAILURE</marker>  
    </evaluator>**  
    <smtpHost>${smtpHost}</smtpHost>  
    <to>${to}</to>  
    <from>${from}</from>  
    <layout class="ch.qos.logback.classic.html.HTMLLayout"/>  
  </appender>  
  
  <root>  
    <level value ="debug"/>  
    <appender-ref ref="EMAIL" />  
  </root>    
</configuration>

Поверните его с помощью следующей команды:

java -Dfrom=source@xyz.com -Dto=recipient@xyz.com -DsmtpHost = some\_smtp\_host \

chapters.appenders.mail.Marked\_EMail src / main / java / chapters / appenders / mail / mailWithMarker.xml

**Маркерный запуск с помощью JaninoEventEvaluator**

Обратите внимание, что вместо использования маркера-ориентированного OnMarkerEvaluatorмы могли бы использовать гораздо более общего [JaninoEventEvaluator](http://logback.qos.ch/manual/filters.html#JaninoEventEvaluator) или даже более мощного кузена [GEventEvaluator](http://logback.qos.ch/manual/filters.html#GEventEvaluator). Например, следующий файл конфигурации использует JaninoEventEvaluatorвместо, OnMarkerEvaluatorно в остальном эквивалентен предыдущему файлу конфигурации.

*Пример: SMTPAppenderс JaninoEventEvaluator(logback-examples / src / main / resources / chapters / appenders / mail / mailWithMarker\_Janino.xml)*

Посмотреть как .groovy

<configuration>  
  <appender name="EMAIL" class="ch.qos.logback.classic.net.SMTPAppender">  
    <evaluator class="ch.qos.logback.classic.boolex.JaninoEventEvaluator">  
      <expression>  
        (marker != null) &&  
        (marker.contains("NOTIFY\_ADMIN") || marker.contains("TRANSACTION\_FAILURE"))  
      </expression>  
    </evaluator>      
    ... same as above  
  </appender>  
</configuration>

**Маркерный запуск с помощью GEventEvaluator**

Вот эквивалентный оценщик, использующий [GEventEvaluator](http://logback.qos.ch/manual/filters.html#GEventEvaluator) .

*Пример: то же самое с GEventEvaluator(logback-examples / src / main / resources / chapters / appenders / mail / mailWithMarker\_GEvent.xml)*

Посмотреть как .groovy

<configuration>  
  <appender name="EMAIL" class="ch.qos.logback.classic.net.SMTPAppender">  
    <evaluator class="ch.qos.logback.classic.boolex.GEventEvaluator">  
      <expression>  
        e.marker?.contains("NOTIFY\_ADMIN") || e.marker?.contains("TRANSACTION\_FAILURE")  
      </expression>  
    </evaluator>      
    ... same as above  
  </appender>  
</configuration>

Обратите внимание, что, поскольку у события может отсутствовать маркер, значение e.marker может быть нулевым. Отсюда использование [безопасного оператора разыменования](http://groovy.codehaus.org/Null+Object+Pattern) Groovy , то есть.? оператор.

**Аутентификация / STARTTLS / SSL**

SMTPAppenderподдерживает аутентификацию через простые пароли пользователей, а также протоколы STARTTLS и SSL. Обратите внимание, что STARTTLS отличается от SSL тем, что в STARTTLS соединение изначально не зашифровано и только после того, как клиентом будет выполнена команда STARTTLS (если сервер ее поддерживает), соединение переключается на SSL. В режиме SSL соединение шифруется с самого начала.

**Конфигурация SMTPAppender для Gmail (SSL)**

В следующем примере показано, как настроить SMTPAppenderGmail с протоколом SSL.

*Пример :: SMTPAppenderв Gmail с использованием SSL (logback-examples / src / main / resources / chapters / appenders / mail / gmailSSL.xml)*

Посмотреть как .groovy

<configuration>  
  <appender name="EMAIL" class="ch.qos.logback.classic.net.SMTPAppender">  
    **<smtpHost>smtp.gmail.com</smtpHost>**  
    **<smtpPort>465</smtpPort>**  
    **<SSL>true</SSL>**  
    **<username>YOUR\_USERNAME@gmail.com</username>**  
    **<password>YOUR\_GMAIL\_PASSWORD</password>**  
  
    <to>EMAIL-DESTINATION</to>  
    <to>ANOTHER\_EMAIL\_DESTINATION</to> <!-- additional destinations are possible -->  
    <from>YOUR\_USERNAME@gmail.com</from>  
    <subject>TESTING: %logger{20} - %m</subject>  
    <layout class="ch.qos.logback.classic.PatternLayout">  
      <pattern>%date %-5level %logger{35} - %message%n</pattern>  
    </layout>         
  </appender>  
  
  <root level="DEBUG">  
    <appender-ref ref="EMAIL" />  
  </root>

**SMTPAppender для Gmail (STARTTLS)**

В следующем примере показано, как настроить SMTPAppenderGmail для протокола STARTTLS.

*Пример: SMTPAppenderдля GMAIL с использованием STARTTLS (logback-examples / src / main / resources / chapters / appenders / mail / gmailSTARTTLS.xml)*

Посмотреть как .groovy

<configuration>     
  <appender name="EMAIL" class="ch.qos.logback.classic.net.SMTPAppender">  
    <smtpHost>smtp.gmail.com</smtpHost>  
    <smtpPort>587</smtpPort>  
    <STARTTLS>true</STARTTLS>  
    <username>YOUR\_USERNAME@gmail.com</username>  
    <password>YOUR\_GMAIL\_xPASSWORD</password>  
      
    <to>EMAIL-DESTINATION</to>  
    <to>ANOTHER\_EMAIL\_DESTINATION</to> <!-- additional destinations are possible -->  
    <from>YOUR\_USERNAME@gmail.com</from>  
    <subject>TESTING: %logger{20} - %m</subject>  
    <layout class="ch.qos.logback.classic.PatternLayout">  
      <pattern>%date %-5level %logger - %message%n</pattern>  
    </layout>         
  </appender>  
  
  <root level="DEBUG">  
    <appender-ref ref="EMAIL" />  
  </root>    
</configuration>

**SMTPAppender с MDCD-дискриминатором**

Как упоминалось ранее, при указании дискриминатора, отличного SMTPAppenderот заданного по умолчанию, будут генерироваться сообщения электронной почты, содержащие события, относящиеся к конкретному пользователю, сеансу пользователя или IP-адресу клиента, в зависимости от указанного дискриминатора.

Следующий пример иллюстрирует использование [MDCBasedDiscriminator](http://logback.qos.ch/xref/ch/qos/logback/classic/sift/MDCBasedDiscriminator.html) в сочетании с ключом MDC с именем «req.remoteHost», который предположительно содержит IP-адрес удаленного хоста, обращающегося к фиктивному приложению. В веб-приложении вы можете использовать [MDCInsertingServletFilter](http://logback.qos.ch/manual/mdc.html#mis) для заполнения значений MDC.

*Пример: SMTPAppenderс MDCBasedDsicriminator (logback-examples / src / main / resources / chapters / appenders / mail / mailWithMDCBasedDiscriminator.xml)*

<configuration>     
  <appender name="EMAIL" class="ch.qos.logback.classic.net.SMTPAppender">  
    <smtpHost>ADDRESS-OF-YOUR-SMTP-HOST</smtpHost>  
    <to>EMAIL-DESTINATION</to>  
    <from>SENDER-EMAIL</from>  
  
    **<discriminator class="ch.qos.logback.classic.sift.MDCBasedDiscriminator">**  
      **<key>req.remoteHost</key>**  
      **<defaultValue>default</defaultValue>**  
    **</discriminator>**  
  
    <subject>${HOSTNAME} -- %X{req.remoteHost} %msg"</subject>  
    <layout class="ch.qos.logback.classic.html.HTMLLayout">  
      <pattern>%date%level%thread%X{req.remoteHost}%X{req.requestURL}%logger%msg</pattern>  
    </layout>  
  </appender>  
  
  <root>  
    <level level="DEBUG"/>  
    <appender-ref ref="EMAIL" />  
  </root>    
</configuration>

Таким образом, каждое исходящее письмо, сгенерированное пользователем, SMTPAppender будет принадлежать *уникальному* удаленному хосту, что значительно облегчает диагностику проблемы.

**Управление буфером в очень загруженных системах**

Внутри каждое отдельное значение, возвращаемое дискриминатором, будет вызывать создание нового циклического буфера. Однако не более maxNumberOfBuffers (по умолчанию 64) будут сохранены. Всякий раз, когда число буферов превышает maxNumberOfBuffers , автоматически обновляется самый последний из обновленных буферов. В качестве второй меры безопасности любой буфер, который не обновлялся в течение последних 30 минут, также автоматически удаляется.

В системах, обслуживающих большое количество транзакций в минуту, разрешение только небольшого числа для maxNumberOfBuffers (по умолчанию 64) часто приводит к излишне малому количеству событий в исходящей электронной почте. Действительно, при наличии большого количества транзакций будет иметься более одного буфера, связанного с одной и той же транзакцией, поскольку буферы будут уничтожены и заново созданы последовательно для одного и того же значения дискриминатора (или транзакции). Обратите внимание, что даже в таких очень загруженных системах максимальное количество циклических буферов ограничено maxNumberOfBuffers .

Чтобы избежать таких эффектов йо-йо, SMTPAppenderосвободится буфер, связанный с данным ключом дискриминатора, как только он увидит событие, помеченное как «FINALIZE\_SESSION». Это приведет к удалению соответствующего буфера в конце каждой транзакции. Затем вы можете безопасно увеличить значение maxNumberOfBuffers до большего значения, например 512 или 1024, не рискуя исчерпать память.

Существует три различных, но дополняющих друг друга механизма, работающих вместе для управления циклическими буферами. Они гарантируют, что в любой момент времени поддерживаются только соответствующие буферы, даже в очень загруженных системах.

**DBAppender**

DBAppender вставляет события регистрации в три таблицы базы данных в формате, независимом от языка программирования Java.

Этими тремя таблицами являются *logging\_event*, *logging\_event\_property* и *logging\_event\_exception*. Они должны существовать, прежде чем их DBAppender можно будет использовать. Logback поставляется с SQL-скриптами, которые будут создавать таблицы. Их можно найти в *папке logback-classic / src / main / java / ch / qos / logback / classic / db / script* . Для каждой из самых популярных систем баз данных существует специальный скрипт. Если сценарий для вашего конкретного типа системы баз данных отсутствует, его будет довольно легко написать, на примере уже существующих сценариев. Если вы отправите их нам, мы с радостью добавим отсутствующие скрипты в будущие выпуски.

Если ваш драйвер JDBC поддерживает getGeneratedKeys метод, представленный в спецификации JDBC 3.0, при условии, что вы создали соответствующие таблицы базы данных, как указано выше, никаких дополнительных действий не требуется. В противном случае должна быть SQLDialect соответствующая вашей системе базы данных. В настоящее время logback имеет диалекты для H2, HSQL, MS SQL Server, MySQL, Oracle, PostgreSQL, SQLLite и Sybase.

В таблице ниже приведены типы баз данных и их поддержка getGeneratedKeys()метода.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **RDBMS** | **протестированная версия (ы)** | **протестированные версии драйверов JDBC** | **поддерживает getGeneratedKeys() метод** | **это диалект, предоставляемый logback** |
| DB2 | непроверенной | непроверенной | неизвестный | НЕТ |
| H2 | 1.2.132 | - | неизвестный | ДА |
| HSQL | 1.8.0.7 | - | НЕТ | ДА |
| Microsoft SQL Server | 2005 | 2.0.1008.2 (sqljdbc.jar) | ДА | ДА |
| MySQL | 5.0.22 | 5.0.8 (mysql-connector.jar) | ДА | ДА |
| PostgreSQL | 8.x | 8.4-701.jdbc4 | НЕТ | ДА |
| оракул | 10г | 10.2.0.1 (ojdbc14.jar) | ДА | ДА |
| SQLLite | 3.7.4 | - | неизвестный | ДА |
| Sybase SQLAnywhere | 10.0.1 | - | неизвестный | ДА |

Эксперименты показывают, что запись одного события в базу данных занимает примерно 10 миллисекунд на «стандартном» ПК. Если используются объединенные соединения, эта цифра падает примерно до 1 миллисекунды. Обратите внимание, что большинство драйверов JDBC уже поставляются с поддержкой пула соединений.

Настройка обратного входа в систему DBAppender может быть выполнена несколькими различными способами, в зависимости от инструментов, которые необходимо подключить к базе данных, и самой базы данных. Ключевой вопрос в настройке DBAppender заключается в настройке его ConnectionSource объекта.

После DBAppender настройки для вашей базы данных события регистрации отправляются в указанную базу данных. Как указывалось ранее, существует три таблицы, используемые logback для хранения данных о событиях регистрации.

Таблица *logging\_event* содержит следующие поля:

|  |  |  |
| --- | --- | --- |
| **поле** | **Тип** | **Описание** |
| **timestamp** | big int | Отметка времени, которая была действительной при создании события регистрации. |
| **formatted\_message** | text | Сообщение, которое было добавлено к событию регистрации после форматирования с помощью org.slf4j.impl.MessageFormatter, если объекты были переданы вместе с сообщением. |
| **logger\_name** | varchar | Имя регистратора, используемого для выдачи запроса на регистрацию. |
| **level\_string** | varchar | Уровень регистрации событий. |
| **reference\_flag** | smallint | Это поле используется logback для идентификации событий регистрации, с которыми связаны исключение или MDCзначения свойств.  Его значение рассчитывается с помощью ch.qos.logback.classic.db.DBHelper. Событие регистрации, которое содержит свойства MDCили Contextимеет номер флага *1* . Тот, который содержит исключение, имеет номер флага *2* . Событие регистрации, которое содержит оба элемента, имеет номер флага *3* . |
| **caller\_filename** | varchar | Имя файла, в который был отправлен запрос на регистрацию. |
| **caller\_class** | varchar | Класс, в котором был отправлен запрос на регистрацию. |
| **caller\_method** | varchar | Имя метода, в котором был отправлен запрос на регистрацию. |
| **caller\_line** | char | Номер строки, где был отправлен запрос на регистрацию. |
| **event\_id** | int | Идентификатор базы данных события регистрации. |

*Logging\_event\_property* используется для хранения ключей и значений , содержащихся в MDC или Context. Он содержит поля:

|  |  |  |
| --- | --- | --- |
| **поле** | **Тип** | **Описание** |
| **event\_id** | int | Идентификатор базы данных события регистрации. |
| **mapped\_key** | varchar | Ключ MDC свойства |
| **mapped\_value** | text | Значение MDC свойства |

Таблица *logging\_event\_exception* содержит следующие поля:

|  |  |  |
| --- | --- | --- |
| **поле** | **Тип** | **Описание** |
| **event\_id** | int | Идентификатор базы данных события регистрации. |
| **i** | smallint | Индекс строки в full stack trace. |
| **trace\_line** | varchar | Соответствующая строка |

Чтобы дать более наглядный пример работы, сделанной DBAppender, вот скриншот базы данных MySQL с контентом, предоставленным DBAppender.

Таблица *logging\_event* :

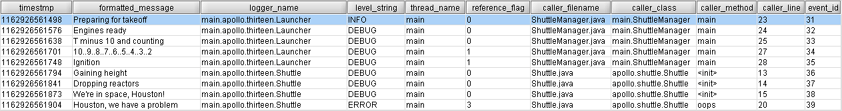


Таблица *logging\_event\_exception* :

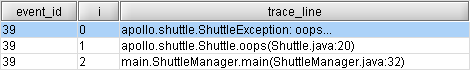


Таблица *logging\_event\_property* :



**ConnectionSource**

Интерфейс ConnectionSource предоставляет подключаемые средства для прозрачного получения JDBC-соединений для классов обратного входа, которые требуют использования java.sql.Connection. В настоящее время существует три реализации ConnectionSource, а именно DataSourceConnectionSource, DriverManagerConnectionSource и JNDIConnectionSource.

Первый пример, который мы рассмотрим, это использование конфигурации DriverManagerConnectionSource и базы данных MySQL.

*Пример: DBAppender конфигурация (logback-examples / src / main / resources / chapters / appenders / db / append-toMySQL-with-driverManager.xml)*

<configuration>  
  
  **<appender name="DB" class="ch.qos.logback.classic.db.DBAppender">  
    <connectionSource class="ch.qos.logback.core.db.DriverManagerConnectionSource">  
      <driverClass>com.mysql.jdbc.Driver</driverClass>  
      <url>jdbc:mysql://host\_name:3306/datebase\_name</url>  
      <user>username</user>  
      <password>password</password>  
    </connectionSource>  
  </appender>**  
    
  <root level="DEBUG" >  
    <appender-ref ref="DB" />  
  </root>  
</configuration>

Правильный драйвер должен быть объявлен. Здесь используется класс com.mysql.jdbc.Driver . URL должен начинаться с jdbc:mysql://.

[DriverManagerConnectionSource](http://logback.qos.ch/xref/ch/qos/logback/core/db/DriverManagerConnectionSource.html) это реализация ConnectionSource, которая получает соединение традиционным способом JDBC на основе URL-адреса соединения.

Обратите внимание, что этот класс будет устанавливать новый Connection для каждого вызова getConnection(). Рекомендуется либо использовать драйвер JDBC, который изначально поддерживает пул соединений, либо создать свою собственную реализацию, ConnectionSource которая подключается к любому механизму пулов, который вы уже используете. Если у вас есть доступ к реализации JNDI, которая поддерживается javax.sql.DataSource, например, на сервере приложений J2EE, см. [JNDIConnectionSource](http://logback.qos.ch/manual/appenders.html#JNDIConnectionSource) Ниже.

Подключение к базе данных с использованием источника данных довольно похоже. Конфигурация теперь использует DataSourceConnectionSource, который является реализацией ConnectionSource, который получает соединение рекомендованным способом JDBC на основе javax.sql.DataSource.

*Пример: DBAppenderконфигурация (logback-examples / src / main / resources / chapters / appenders / db / append-with-datasource.xml)*

<configuration  debug="true">  
  
  <appender name="DB" class="ch.qos.logback.classic.db.DBAppender">  
     **<connectionSource class="ch.qos.logback.core.db.DataSourceConnectionSource">  
         
       <dataSource class="${dataSourceClass}">**<!-- Joran cannot substitute variables  
         that are not attribute values. Therefore, we cannot  
         declare the next parameter like the others.   
         -->  
         **<param name="${url-key:-url}" value="${url\_value}"/>  
         <serverName>${serverName}</serverName>  
         <databaseName>${databaseName}</databaseName>  
       </dataSource>**  
         
       <user>${user}</user>  
       <password>${password}</password>  
     </connectionSource>  
  </appender>  
  
  <root level="INFO">  
    <appender-ref ref="DB" />  
  </root>    
</configuration>

Обратите внимание, что в этом примере конфигурации мы интенсивно используем переменные подстановки. Иногда они удобны, когда сведения о соединении должны быть централизованы в одном файле конфигурации и предоставлены для входа в систему и других сред.

**JNDIConnectionSource**

[JNDIConnectionSource](http://logback.qos.ch/xref/ch/qos/logback/core/db/JNDIConnectionSource.html) это еще одна ConnectionSource реализация доставки в logback. Как указывает его имя, он извлекает javax.sql.DataSource из JNDI, а затем использует его для получения java.sql.Connection экземпляра. JNDIConnectionSource в первую очередь предназначен для использования внутри серверов приложений J2EE или клиентами серверов приложений, при условии, что сервер приложений поддерживает удаленный доступ javax.sql.DataSource. Таким образом, можно воспользоваться пулами соединений и любыми другими преимуществами, которые предоставляет сервер приложений. Что еще более важно, ваше приложение будет более “сухим”, так как больше не нужно будет определять источник данных в logback.xml.

Например, вот фрагмент конфигурации для Tomcat. Он предполагает PostgreSQL в качестве базы данных, хотя любая из поддерживаемых систем баз данных (перечисленных выше) будет работать.

<Context docBase="/path/to/app.war" path="/myapp">  
  ...  
  <Resource **name="jdbc/logging"**  
               auth="Container"  
               type="javax.sql.DataSource"  
               username="..."  
               password="..."  
               driverClassName="org.postgresql.Driver"  
               url="jdbc:postgresql://localhost/..."  
               maxActive="8"  
               maxIdle="4"/>  
  ...  
</Context>

Как только a DataSourceопределено на сервере J2EE, на него можно легко ссылаться в файле конфигурации обратного входа, как показано в следующем примере.

*Пример: DBAppenderконфигурация по JNDIConnectionSource (logback-examples / src / main / resources / chapters / appenders / db / append-via-jndi.xml)*

Посмотреть как .groovy

<configuration debug="true">  
  <appender name="DB" class="ch.qos.logback.classic.db.DBAppender">  
    <connectionSource class="ch.qos.logback.core.db.JNDIConnectionSource">  
      **<!-- please note the "java:comp/env/" prefix -->**  
      **<jndiLocation>java:comp/env/jdbc/logging</jndiLocation>**  
    </connectionSource>  
  </appender>  
  <root level="INFO">  
    <appender-ref ref="DB" />  
  </root>    
</configuration>

Обратите внимание, что этот класс получит javax.naming.InitialContext использование конструктора без аргументов. Обычно это работает при выполнении в среде J2EE. Вне среды J2EE убедитесь, что вы предоставляете файл *jndi.properties,* как описано в документации вашего провайдера JNDI.

**Пул подключений**

Регистрация событий может быть создана в довольно быстром темпе. Чтобы не отставать от потока событий, которые необходимо вставить в базу данных, рекомендуется использовать пул соединений с DBAppender.

Эксперимент показывает, что использование пула соединений с DBAppender большим приростом производительности. С помощью следующего файла конфигурации события регистрации отправляются в базу данных MySQL без какого-либо пула.

*Пример: DBAppenderконфигурация без пула (logback-examples / src / main / resources / chapters / appenders / db / append-toMySQL-with-datasource.xml)*

<configuration>  
  
  <appender name="DB" class="ch.qos.logback.classic.db.DBAppender">  
    <connectionSource class="ch.qos.logback.core.db.DataSourceConnectionSource">  
      <dataSource class="com.mysql.jdbc.jdbc2.optional.MysqlDataSource">  
        <serverName>${serverName}</serverName>  
        <port>${port$</port>  
        <databaseName>${dbName}</databaseName>  
        <user>${user}</user>  
        <password>${pass}</password>  
      </dataSource>  
    </connectionSource>  
  </appender>  
      
  <root level="DEBUG">  
    <appender-ref ref="DB" />  
  </root>  
</configuration>

С этим файлом конфигурации отправка 500 событий регистрации в базу данных MySQL занимает колоссальные 5 секунд, то есть 10 миллисекунд на запрос. Эта цифра недопустима при работе с большими приложениями.

Выделенная внешняя библиотека необходима для использования пула соединений с DBAppender. В следующем примере используется [c3p0](http://sourceforge.net/projects/c3p0) . Чтобы иметь возможность использовать c3p0, необходимо скачать его и поместить *c3p0-VERSION.jar* в *путь* к классам.

*Пример: DBAppenderконфигурация с пулами (logback-examples / src / main / resources / chapters / appenders / db / append-toMySQL-with-datasource-and-pooling.xml)*

<configuration>  
  
  <appender name="DB" class="ch.qos.logback.classic.db.DBAppender">  
    <connectionSource  
      class="ch.qos.logback.core.db.DataSourceConnectionSource">  
      **<dataSource  
        class="com.mchange.v2.c3p0.ComboPooledDataSource">  
        <driverClass>com.mysql.jdbc.Driver</driverClass>  
        <jdbcUrl>jdbc:mysql://${serverName}:${port}/${dbName}</jdbcUrl>  
        <user>${user}</user>  
        <password>${password}</password>  
      </dataSource>**  
    </connectionSource>  
  </appender>  
  
  <root level="DEBUG">  
    <appender-ref ref="DB" />  
  </root>  
</configuration>

В этой новой конфигурации отправка 500 запросов на регистрацию в вышеупомянутую базу данных MySQL занимает около 0,5 секунды, в среднем на 1 миллисекунду на запрос, что в десять раз повышает производительность.

**SyslogAppender**

Протокол системного журнала является очень простым протоколом: отправитель системного журнала отправляет небольшое сообщение получателю системного журнала. Получатель обычно называется *syslog daemon* или *syslog server* . Logback может отправлять сообщения удаленному демону системного журнала. Это достигается с помощью [SyslogAppender](http://logback.qos.ch/xref/ch/qos/logback/classic/net/SyslogAppender.html).

Вот свойства, которые вы можете передать SyslogAppender.

|  |  |  |
| --- | --- | --- |
| **Имя свойства** | **Тип** | **Описание** |
| **syslogHost** | String | Имя хоста сервера системного журнала. |
| **port** | String | Номер порта на сервере системного журнала для подключения. Обычно никто не хотел бы изменить значение по умолчанию *514* . |
| **facility** | String | Объект предназначен для идентификации источника сообщения.  Опция средства должна быть установлена ​​на одну из строк: *KERN, USER, MAIL, DAEMON, AUTH, SYSLOG, LPR, NEWS, UUCP, CRON, AUTHPRIV, FTP, NTP, AUDIT, ALERT, CLOCK, LOCAL0, LOCAL1, LOCAL2, LOCAL3 , LOCAL4, LOCAL5, LOCAL6, LOCAL7* . Дело не важно. |
| **suffixPattern** | String | Параметр suffixPattern указывает формат нестандартизированной части сообщения, отправляемого на сервер системного журнала. По умолчанию его значение равно *[% thread]% logger% msg* . Любое значение, которое PatternLayoutможет использовать, является правильным значением suffixPattern . |
| **stackTracePattern** | String | Свойство stackTracePattern позволяет настраивать строку, появляющуюся непосредственно перед каждой строкой трассировки стека. Значением по умолчанию для этого свойства является "\ t", т.е. символ табуляции. Любое принятое значение PatternLayoutявляется допустимым значением для stackTracePattern . |
| **throwableExcluded** | boolean | Установка throwableExcluded to trueприведет к тому, что данные трассировки стека, связанные с Throwable, будут опущены. По умолчанию для throwableExcluded установлено значение falseтак, что данные трассировки стека отправляются на сервер системного журнала. |

Важность системного журнала события регистрации преобразуется из уровня события регистрации. *DEBUG* преобразуется в *7* , *INFO* преобразуется в *6* , *WARN* преобразуется в *4,* а *ERROR* преобразуется в *3* .

Поскольку формат запроса системного журнала следует довольно строгим правилам, макета для использования не существует SyslogAppender. Однако использование параметра suffixPattern позволяет пользователю отображать любую информацию, которую он пожелает.

Вот пример конфигурации с использованием SyslogAppender.

*Пример: SyslogAppenderконфигурация (logback-examples / src / main / resources / chapters / appenders / conf / logback-syslog.xml)*

<configuration>  
  
  <appender name="SYSLOG" class="ch.qos.logback.classic.net.SyslogAppender">  
    <syslogHost>remote\_home</syslogHost>  
    <facility>AUTH</facility>  
    <suffixPattern>[%thread] %logger %msg</suffixPattern>  
  </appender>  
  
  <root level="DEBUG">  
    <appender-ref ref="SYSLOG" />  
  </root>  
</configuration>

При тестировании этой конфигурации вы должны убедиться, что демон удаленного системного журнала принимает запросы от внешнего источника. Опыт показывает, что по умолчанию демоны системного журнала обычно отклоняют запросы, поступающие через сетевое соединение.

**SiftingAppender**

Как следует из его названия, SiftingAppender можно использовать для разделения (или просеивания) ведения журнала в соответствии с заданным атрибутом времени выполнения. Например, SiftingAppender может разделять события регистрации в соответствии с пользовательскими сеансами, так что журналы, созданные разными пользователями, попадают в отдельные файлы журналов, по одному файлу журнала на пользователя.

|  |  |  |
| --- | --- | --- |
| **Имя свойства** | **Тип** | **Описание** |
| **timeout** | [Duration](http://logback.qos.ch/apidocs/ch/qos/logback/core/util/Duration.html) | Вложенный appender, к которому не было доступа после истечения времени ожидания, считается устаревшим. Несвежий аппликатор закрыт и на него не ссылаются SiftingAppender. Значение по умолчанию для тайм-аута составляет 30 минут. |
| **maxAppenderCount** | integer | Максимальное количество вложенных приложений, которые SiftingAppender может создавать и отслеживать. Значением по умолчанию для maxAppenderCount является Integer.MAX\_VALUE. |

SiftingAppender достигает этого, создавая вложенные дополнения на лету. Вложенные дополнения создаются на основе шаблона, указанного в самой конфигурации SiftingAppender (заключенного в <sift>элемент, см. Пример ниже).

SiftingAppender отвечает за управление жизненным циклом детей-аппендеров. Например, SiftingAppender автоматически закроет и удалит все устаревшие приложения.

Вложенный аппендер считается устаревшим, если к нему не обращаются по истечении времени, указанного параметром timeout .

При обработке события регистрации SiftingAppender выберет дочерний аппендер для делегирования. Критерии выбора вычисляются во время выполнения discriminator-ом. Пользователь может указать критерии выбора с помощью [Discriminator](http://logback.qos.ch/xref/ch/qos/logback/core/sift/Discriminator.html).

**пример**

Приложение [SiftExample](http://logback.qos.ch/xref/chapters/appenders/sift/SiftExample.html) регистрирует сообщение о том, что приложение запущено. Затем он устанавливает ключ MDC "userid" в значение " Alice " и регистрирует сообщение:

logger.debug("Application started");

MDC.put("userid", "Alice");

logger.debug("Alice says hello");

Шаблон для файла конфигурации иллюстрирует использование SiftingAppender.

*Пример: SiftingAppender конфигурация (logback-examples / src / main / resources / chapters / appenders / sift / byUserid.xml)*

<configuration>  
  
  **<appender name="SIFT" class="ch.qos.logback.classic.sift.SiftingAppender">**  
    <!-- в отсутствие атрибута класса предполагается, что желаемый тип дискриминатора равен ch.qos.logback.classic.sift.MDCBasedDiscriminator -->  
    **<discriminator>**  
      **<key>userid</key>**  
      **<defaultValue>unknown</defaultValue>**  
    **</discriminator>**

    **<sift>**  
      **<appender name="FILE-${userid}" class="ch.qos.logback.core.FileAppender">**  
        **<file>${userid}.log</file>**  
        **<append>false</append>**  
        **<layout class="ch.qos.logback.classic.PatternLayout">**  
          **<pattern>%d [%thread] %level %mdc %logger{35} - %msg%n</pattern>**  
        **</layout>**  
      **</appender>**  
    **</sift>**  
  </appender>  
  
  <root level="DEBUG">  
    <appender-ref ref="SIFT" />  
  </root>  
</configuration>

При отсутствии атрибута класса предполагается, что типом дискриминатора является [MDCBasedDiscriminator](http://logback.qos.ch/xref/ch/qos/logback/classic/sift/MDCBasedDiscriminator.html) . Различающим значением является значение MDC, связанное с ключом, данным свойством ключа . Однако если MDC равно нулю, тогда в качестве различающего значения используется defaultValue.

В приведенном выше примере SiftingAppender создаст несколько экземпляров FileAppender, причем каждый экземпляр FileAppender определяется значением, связанным с ключом MDC «userid». Каждый раз, когда ключу MDC «userid» назначается новое значение, будет создаваться с нуля новый экземпляр FileAppender.

SiftingAppender отслеживает создаваемые им приложения. Аппендеры, неиспользованные в течение 30 минут, автоматически закрываются и удаляются.

**ЭКСПОРТ ПЕРЕМЕННЫХ** Недостаточно иметь разные экземпляры appender; каждый экземпляр должен выводиться на отдельный целевой ресурс. Чтобы разрешить такое разграничение, в шаблоне appender ключ, передаваемый дискриминатору «userid» в приведенном выше примере, экспортируется и становится [переменной](http://logback.qos.ch/manual/configuration.html#variableSubstitution) . Следовательно, эту переменную можно использовать для дифференциации фактического ресурса, используемого данным дочерним аппендиатом.

Запуск SiftExample приложения с файлом конфигурации «byUserid.xml», показанным выше, приведет к двум отдельным файлам журнала, «unknown.log» и «Alice.log».

**ПЕРЕМЕННЫЕ** **локальной области** Начиная с версии 1.0.12, свойства, определенные в локальной области в файле конфигурации, будут доступны для вложенных дополнений. Кроме того, вы можете определять переменные или динамически вычислять переменные из элемента <sift>. Также поддерживается объединение переменных из частей, определенных снаружи и внутри элемента <sift>.

**Правильное получение тайм-аута**

Для определенных типов приложений может быть трудно правильно настроить параметр timeout . Если тайм-аут слишком мал, вложенный аппендер может быть удален, чтобы его можно было создать заново через несколько секунд. Это явление называется *разгромом*. Если время ожидания слишком велико и добавочные элементы создаются в быстрой последовательности, возможно, у вас закончились ресурсы. Аналогично, установка maxAppenderCount слишком низкой может также привести к сбою.

Когда вложенный аппендер становиться дальше не нужен, в этом месте кода нужно написать «logger.info(FINALIZE\_SESSION\_MARKER, "About to end the job");», используя маркер [FINALIZE\_SESSION](http://logback.qos.ch/apidocs/ch/qos/logback/classic/ClassicConstants.html#FINALIZE_SESSION_MARKER). Всякий раз, когда SiftingAppender видит событие регистрации, помеченное как FINALIZE\_SESSION, он завершает работу связанного вложенного аппендера. По окончании срока службы вложенный аппендер задерживается на несколько секунд, чтобы обработать любые поздние события (если они есть), а затем будет закрыт.

import org.slf4j.Logger;  
import static ch.qos.logback.classic.ClassicConstants.FINALIZE\_SESSION\_MARKER;  
  
  void job(String jobId) {  
     
    MDC.put("jobId", jobId);  
    logger.info("Starting job.");  
  
    ... do whather the job needs to do  
      
    // приведет к тому, что вложенный аппендер завершит работу, задержавшись на несколько секунд

    logger.info(FINALIZE\_SESSION\_MARKER, "About to end the job");  
  
    try {  
      .. perform clean up  
    } catch(Exception e);    
      // Эта запись журнала будет обработана appender-ом задержаным на несколько секунд.   
      // Никакой новый appender не будет создан.  
      logger.error("unexpected error while cleaning up", e);  
    }  
  }

**AsyncAppender**

AsyncAppender регистрирует [ILoggingEvent](http://logback.qos.ch/apidocs/ch/qos/logback/classic/spi/ILoggingEvent.html) s асинхронно. Этот апендер является оберткой над другими апендерами.

**отброс по умолчанию, при заполнении на 80%** AsyncAppender буферизует события в BlockingQueue. Рабочий поток, созданный AsyncAppender, получает события из заголовка очереди и отправляет их одному приложению, подключенному к AsyncAppender.

Обратите внимание, что по умолчанию AsyncAppender отбрасываются события уровня TRACE, DEBUG и INFO, если его очередь заполнена на 80%. Эта стратегия оказывает удивительно благоприятное влияние на производительность за счет потери события.

**ОСТАНОВКА / ПОВТОРНОЕ** **РАЗВЕРТЫВАНИе** После завершения или повторного развертывания работы приложения AsyncAppender должен быть остановлен, чтобы остановить и вернуть рабочий поток, а также удалить события регистрации из очереди. Это может быть достигнуто путем [остановки LoggerContext,](http://logback.qos.ch/manual/configuration.html#stopContext) который закроет все приложения, включая любые AsyncAppender экземпляры. AsyncAppender будет ожидать сброса рабочего потока до времени ожидания, указанного в maxFlushTime . Если вы обнаружите, что события из очереди отбрасываются во время закрытия LoggerContext, вам может потребоваться увеличить время ожидания. Если указать значение 0 для maxFlushTime, AsyncAppender будет ожидать сброса всех событий в очереди, прежде чем вернуться из метода stop.

**ОЧИСТКА ПОСЛЕ ЗАВЕРШЕНИЯ РАБОТЫ** В зависимости от режима завершения работы JVM рабочий поток, обрабатывающий события в очереди, может быть прерван, в результате чего события окажутся в очереди. Это обычно происходит, когда LoggerContextне останавливается полностью или когда JVM завершается вне типичного потока управления. Чтобы не прерывать рабочий поток в этих условиях, в среду выполнения JVM может быть вставлен хук отключения, [который правильно останавливает LoggerContext.](http://logback.qos.ch/manual/configuration.html#stopContext) после завершения JVM. Хук отключения может также быть предпочтительным методом для чистого отключения Logback, когда другие хуки отключения пытаются регистрировать события.

Вот список свойств, допущенных AsyncAppender:

|  |  |  |
| --- | --- | --- |
| **Имя свойства** | **Тип** | **Описание** |
| **Queuesize** | int | Максимальная емкость очереди блокировки. По умолчанию для queueSize установлено значение 256. |
| **discardingThreshold** | int | По умолчанию, когда в очереди блокировки остается 20% оставшейся емкости, она отбрасывает события уровня TRACE, DEBUG и INFO, сохраняя только события уровня WARN и ERROR. Чтобы сохранить все события, установите DiscardingThreshold в 0. |
| **includeCallerData** | boolean | Извлечение данных вызывающего абонента может быть довольно дорогим. Для повышения производительности по умолчанию данные вызывающего абонента, связанные с событием, не извлекаются, когда событие добавляется в очередь событий. По умолчанию копируются только «дешевые» данные, такие как имя потока и [MDC](http://logback.qos.ch/manual/mdc.html) . Вы можете указать этому приложению включить данные вызывающего абонента, установив для свойства includeCallerData значение true. |
| **maxFlushTime** | int | В зависимости от глубины очереди и времени задержки для указанного аппендера, AsyncAppender может потребоваться недопустимое количество времени для полной очистки очереди. Когда метод LoggerContext остановлен, AsyncAppender stop метод ожидает до этого времени ожидания завершения рабочего потока. Используйте maxFlushTime, чтобы указать максимальное время ожидания сброса очереди в миллисекундах. События, которые не могут быть обработаны в этом окне, отбрасываются. Семантика этого значения идентична [семантике Thread.join (long)](http://docs.oracle.com/javase/7/docs/api/java/lang/Thread.html#join(long)) . |
| **neverBlock** | boolean | Если false(по умолчанию), аппендер будет блокировать добавление в полную очередь, а не потерять сообщение. Установите значение, trueи appender просто отбросит сообщение и не заблокирует ваше приложение. |

По умолчанию очередь событий настроена с максимальной емкостью 256 событий. Если очередь заполнена, потоки приложений блокируются для регистрации новых событий, пока рабочий поток не получит возможность отправить одно или несколько событий. Когда очередь больше не достигает максимальной емкости, потоки приложения могут снова начать регистрировать события. Следовательно, асинхронное ведение журнала становится псевдосинхронным, когда приложение работает на уровне или около емкости своего буфера событий. Это не обязательно плохо. Приложение предназначено для того, чтобы приложение продолжало работать, хотя и заняло немного больше времени для регистрации событий, пока давление в буфере приложения не уменьшится.

Оптимальная настройка размера очереди событий приложения для максимальной пропускной способности приложения зависит от нескольких факторов. Любой или все из следующих факторов могут вызвать проявление псевдосинхронного поведения:

* Большое количество потоков приложений
* Большое количество событий регистрации за вызов приложения
* Большие объемы данных на событие регистрации
* Высокая латентность у детей "аппендеров"

Для продолжения работы обычно помогает увеличение размера очереди за счет кучи, доступной приложению.

**ПОВЕДЕНИЕ С ПОТЕРЯМИ** В свете вышеприведенного обсуждения и для уменьшения блокировки по умолчанию, когда остается менее 20% емкости очереди, AsyncAppender будут отбрасываться события уровня TRACE, DEBUG и INFO, сохраняющие только события уровня WARN и ERROR. Эта стратегия обеспечивает неблокирующую обработку событий регистрации (следовательно, отличную производительность) при потере стоимости событий уровня TRACE, DEBUG и INFO, когда очередь имеет емкость менее 20%. Потери событий можно предотвратить, установив для свойства discardingThreshold значение 0 (ноль).

*Пример: AsyncAppender конфигурация (logback-examples / src / main / resources / chapters / appenders / conc / logback-async.xml)*

<configuration>  
  <appender name="**FILE**" class="ch.qos.logback.core.FileAppender">  
    <file>myapp.log</file>  
    <encoder>  
      <pattern>%logger{35} - %msg%n</pattern>  
    </encoder>  
  </appender>  
  
  **<appender name="ASYNC" class="ch.qos.logback.classic.AsyncAppender">**  
    **<appender-ref ref="FILE" />**  
  **</appender>**  
  
  <root level="DEBUG">  
    <appender-ref ref="**ASYNC**" />  
  </root>  
</configuration>

**Написание своего собственного Appender**

Вы можете легко написать свой аппендер путем создания подклассов AppenderBase. Он обрабатывает поддержку фильтров, сообщений о состоянии и других функций, используемых большинством пользователей. Производный класс должен реализовывать только один метод, а именно append(Object eventObject).

Те CountingConsoleAppender, которые мы перечислим далее, добавляют на консоль ограниченное количество входящих событий. Отключается после достижения лимита. Он использует PatternLayoutEncoderдля форматирования событий и принимает параметр с именем limit. Следовательно, требуется еще несколько методов append(Object eventObject). Как показано ниже, эти параметры автоматически обрабатываются различными механизмами конфигурации logback.

*Пример 4 .: CountingConsoleAppender (logback-examples / src / main / java / chapters / appenders / CountingConsoleAppender.java)*

package chapters.appenders;  
  
import java.io.IOException;  
  
import ch.qos.logback.classic.encoder.PatternLayoutEncoder;  
import ch.qos.logback.classic.spi.ILoggingEvent;  
import ch.qos.logback.core.AppenderBase;  
  
  
public class CountingConsoleAppender extends AppenderBase<ILoggingEvent> {  
  static int DEFAULT\_LIMIT = 10;  
  int counter = 0;  
  int limit = DEFAULT\_LIMIT;  
    
  PatternLayoutEncoder encoder;  
    
  public void setLimit(int limit) {  
    this.limit = limit;  
  }  
  
  public int getLimit() {  
    return limit;  
  }  
    
  @Override  
  public void start() {  
    if (this.encoder == null) {  
      addError("No encoder set for the appender named ["+ name +"].");  
      return;  
    }  
      
    try {  
      encoder.init(System.out);  
    } catch (IOException e) {  
    }  
    super.start();  
  }  
  
  public void append(ILoggingEvent event) {  
    if (counter >= limit) {  
      return;  
    }  
    // output the events as formatted by our layout  
    try {  
      this.encoder.doEncode(event);  
    } catch (IOException e) {  
    }  
  
    // prepare for next event  
    counter++;  
  }  
  
  public PatternLayoutEncoder getEncoder() {  
    return encoder;  
  }  
  
  public void setEncoder(PatternLayoutEncoder encoder) {  
    this.encoder = encoder;  
  }  
}

В start()методе проверка на присутствие PatternLayoutEncoder. Если кодировщик не установлен, приложение не запускается и выдает сообщение об ошибке.

Этот пользовательский appender иллюстрирует два момента:

* Все свойства, которые следуют соглашениям JavaBeans-установщика / получателя, прозрачно обрабатываются конфигураторами обратного входа. start()Метод, который вызывается автоматически при конфигурировании Logback, несет ответственность за проверки того, что различные свойства Appender устанавливаются и когерентны.
* AppenderBase.doAppend()Метод вызывает метод Append () из его производных классов. Фактические операции вывода выполняются в appendметоде (). В частности, именно в этом методе appenders форматируют события, вызывая их макеты.

[CountingConsoleAppender](http://logback.qos.ch/xref/chapters/appenders/CountingConsoleAppender.html) Может быть сконфигурирован как и любой другой Appender. См. Пример файла конфигурации *logback-examples / src / main / resources / chapters / appenders / countingConsole.xml* для примера.

## Logback Access

Большинство приложений, найденных в logback-classic, имеют аналог в logback-access. Они работают в основном так же, как и их классические аналоги. В следующем разделе мы рассмотрим их использование.

**SocketAppender и SSLSocketAppender**

Он предназначен для входа в удаленный объект путем передачи сериализованных объектов по проводам. Удаленное ведение журнала не является навязчивым в том, что касается события доступа. На принимающей стороне после десериализации событие может быть зарегистрировано, как если бы оно было сгенерировано локально. [SocketAppender](http://logback.qos.ch/xref/ch/qos/logback/access/net/SocketAppender.html)AccessEvent

Расширяет базовое , позволяющее протоколирование к удаленному объекту через Secure Sockets Layer (SSL). [SSLSocketAppender](http://logback.qos.ch/xref/ch/qos/logback/access/net/SSLSocketAppender.html)SocketAppender

Свойства доступа SocketAppenderтакие же, как у классических SocketAppender.

**ServerSocketAppender и SSLServerSocketAppender**

Например SocketAppender, он предназначен для входа в удаленный объект путем передачи сериализованных объектов по проводам. Однако при использовании appender выступает в роли сервера, пассивно прослушивающего сокет TCP и ожидающего входящих подключений от заинтересованных клиентов. Регистрация событий, доставленных аппендеру, распространяется на все подключенные клиенты. [ServerSocketAppender](http://logback.qos.ch/xref/ch/qos/logback/access/net/server/ServerSocketAppender.html)AccessEventServerSocketAppender

Расширяет базовое , позволяющее протоколирование к удаленному объекту через Secure Sockets Layer (SSL). [SSLSocketAppender](http://logback.qos.ch/xref/ch/qos/logback/access/net/server/SSLServerSocketAppender.html)ServerSocketAppender

Свойства доступа ServerSocketAppenderтакие же, как у классических ServerSocketAppender.

**SMTPAppender**

Access ' работает так же, как и его классический аналог. Тем не менее, опция оценки довольно отличается. По умолчанию объект используется . Этот оценщик содержит список URL-адресов, которые сравниваются с текущим URL-адресом запроса. Когда запрашивается одна из страниц, предоставленных для , отправляет электронное письмо. [SMTPAppender](http://logback.qos.ch/xref/ch/qos/logback/access/net/SMTPAppender.html)URLEvaluatorSMTPAppenderURLEvaluatorSMTPAppender

Вот пример конфигурации SMTPAppenderв среде доступа.

*Пример: SMTPAppender конфигурация (logback-examples / src / main / resources / chapters / appenders / conf / access / logback-smtp.xml)*

<appender name="SMTP"  
  class="ch.qos.logback.access.net.SMTPAppender">  
  <layout class="ch.qos.logback.access.html.HTMLLayout">  
    <pattern>%h%l%u%t%r%s%b</pattern>  
  </layout>  
      
  **<Evaluator class="ch.qos.logback.access.net.URLEvaluator">  
    <URL>url1.jsp</URL>  
    <URL>directory/url2.html</URL>  
  </Evaluator>**  
  <from>sender\_email@host.com</from>  
  <smtpHost>mail.domain.com</smtpHost>  
  <to>recipient\_email@host.com</to>  
</appender>

Этот способ запуска электронной почты позволяет пользователям выбирать страницы, которые являются важными шагами в определенном процессе, например. Когда к такой странице обращаются, электронное письмо отправляется со страницами, к которым ранее обращались, и любой информацией, которую пользователь хочет включить в электронное письмо.

**DBAppender**

[DBAppender](http://logback.qos.ch/xref/ch/qos/logback/access/db/DBAppender.html) используется для вставки событий доступа в базу данных.

Две таблицы используются DBAppender: *access\_event* и *access\_event\_header* . Они оба должны существовать, прежде чем их DBAppenderможно будет использовать. Logback поставляется с SQL-скриптами, которые будут создавать таблицы. Их можно найти в *каталоге logback-access / src / main / java / ch / qos / logback / access / db / script* . Для каждой из самых популярных систем баз данных существует специальный скрипт. Если сценарий для вашего конкретного типа системы баз данных отсутствует, его должно быть довольно легко написать, например, один из существующих сценариев. Вам предлагается внести такие недостающие скрипты обратно в проект.

В *access\_event* полей таблицы описаны ниже:

|  |  |  |
| --- | --- | --- |
| **поле** | **Тип** | **Описание** |
| **timestamp** | big int | Отметка времени, которая была действительной при создании события доступа. |
| **requestURI** | varchar | URI, который был запрошен. |
| **requestURL** | varchar | URL, который был запрошен. Это строка, состоящая из метода запроса, URI запроса и протокола запроса. |
| **remoteHost** | varchar | Имя удаленного хоста. |
| **RemoteUSER** | varchar | Имя удаленного пользователя. |
| **remoteAddr** | varchar | Удаленный IP-адрес. |
| **protocol** | varchar | Протокол запроса, например *HTTP* или *HTTPS* . |
| **method** | varchar | Метод запроса, обычно *GET* или *POST* . |
| **serverName** | varchar | Имя сервера, который выдал запрос. |
| **event\_id** | int | Идентификатор базы данных события доступа. |

Таблица *access\_event\_header* содержит заголовок каждого запроса. Информация организована так, как показано ниже:

|  |  |  |
| --- | --- | --- |
| **поле** | **Тип** | **Описание** |
| **event\_id** | int | Идентификатор базы данных соответствующего события доступа. |
| **header\_key** | varchar | Имя заголовка, например, *User-Agent* . |
| **header\_value** | varchar | Значение заголовка, например *Mozilla / 5.0 (Windows; U; Windows NT 5.1; fr; rv: 1.8.1) Gecko / 20061010 Firefox / 2.0* |

Все свойства классики DBAppenderдоступны в доступе DBAppender. Последний предлагает еще один вариант, описанный ниже.

|  |  |  |
| --- | --- | --- |
| **Имя свойства** | **Тип** | **Описание** |
| **insertHeaders** | boolean | Указывает DBAppenderзаполнить базу данных информацией заголовка всех входящих запросов. |

Вот пример конфигурации, которая использует DBAppender.

*Пример: конфигурация DBAppender (logback-examples / src / main / resources / chapters / appenders / conf / access / logback-DB.xml)*

<configuration>  
  
  <appender name="DB" class="ch.qos.logback.access.db.DBAppender">  
    <connectionSource class="ch.qos.logback.core.db.DriverManagerConnectionSource">  
      <driverClass>com.mysql.jdbc.Driver</driverClass>  
      <url>jdbc:mysql://localhost:3306/logbackdb</url>  
      <user>logback</user>  
      <password>logback</password>  
    </connectionSource>  
    <insertHeaders>true</insertHeaders>  
  </appender>  
  
  <appender-ref ref="DB" />  
</configuration>

**SiftingAppender**

SiftingAppender в logback-доступе очень похож на своего классического аналога logback. Основное различие заключается в том, что при доступе по протоколу назад дискриминатор по умолчанию, а именно [AccessEventDiscriminator](http://logback.qos.ch/xref/ch/qos/logback/access/sift/AccessEventDiscriminator.html) , не основан на MDC. Как следует из его названия, AccessEventDiscriminator использует назначенное поле в AccessEvent в качестве основы для выбора вложенного приложения. Если значение указанного поля равно нулю, используется значение, указанное в свойстве defaultValue .

Назначенное поле AccessEvent может быть одним из COOKIE, REQUEST\_ATTRIBUTE, SESSION\_ATTRIBUTE, REMOTE\_ADDRESS, LOCAL\_PORT, REQUEST\_URI. Обратите внимание, что первые три поля требуют, чтобы свойство AdditionalKey также было указано.

Ниже приведен пример файла конфигурации.

*Пример: конфигурация SiftingAppender (logback-examples / src / main / resources / chapters / appenders / conf / sift / access-siftingFile.xml)*

<configuration>  
  <appender name="SIFTING" class="ch.qos.logback.access.sift.SiftingAppender">  
    <Discriminator class="ch.qos.logback.access.sift.AccessEventDiscriminator">  
      <Key>id</Key>  
      <FieldName>SESSION\_ATTRIBUTE</FieldName>  
      <AdditionalKey>username</AdditionalKey>  
      <defaultValue>NA</defaultValue>  
    </Discriminator>  
    <sift>  
       <appender name="ch.qos.logback:logback-site:jar:1.3.0-alpha5" class="ch.qos.logback.core.FileAppender">  
        <file>byUser/ch.qos.logback:logback-site:jar:1.3.0-alpha5.log</file>  
        <layout class="ch.qos.logback.access.PatternLayout">  
          <pattern>%h %l %u %t \"%r\" %s %b</pattern>  
        </layout>  
      </appender>  
    </sift>  
  </appender>  
  <appender-ref ref="SIFTING" />  
</configuration>

В приведенном выше файле конфигурации SiftingAppender вложенные FileAppender экземпляры. Ключ «id» обозначен как переменная, которая будет доступна для вложенных FileAppenderэкземпляров. Дискриминатор по умолчанию, а именно AccessEventDiscriminator, будет искать атрибут сеанса "username" в каждом AccessEvent. Если такой атрибут недоступен, будет использовано значение по умолчанию «NA». Таким образом, предполагая, что атрибут сеанса с именем «username» содержит имя пользователя каждого вошедшего в систему пользователя, в папке *byUser /* (текущей папки) будет *указан* файл журнала, названный в честь каждого пользователя, содержащего журналы доступа для этого пользователя.