# 1.SLF4J

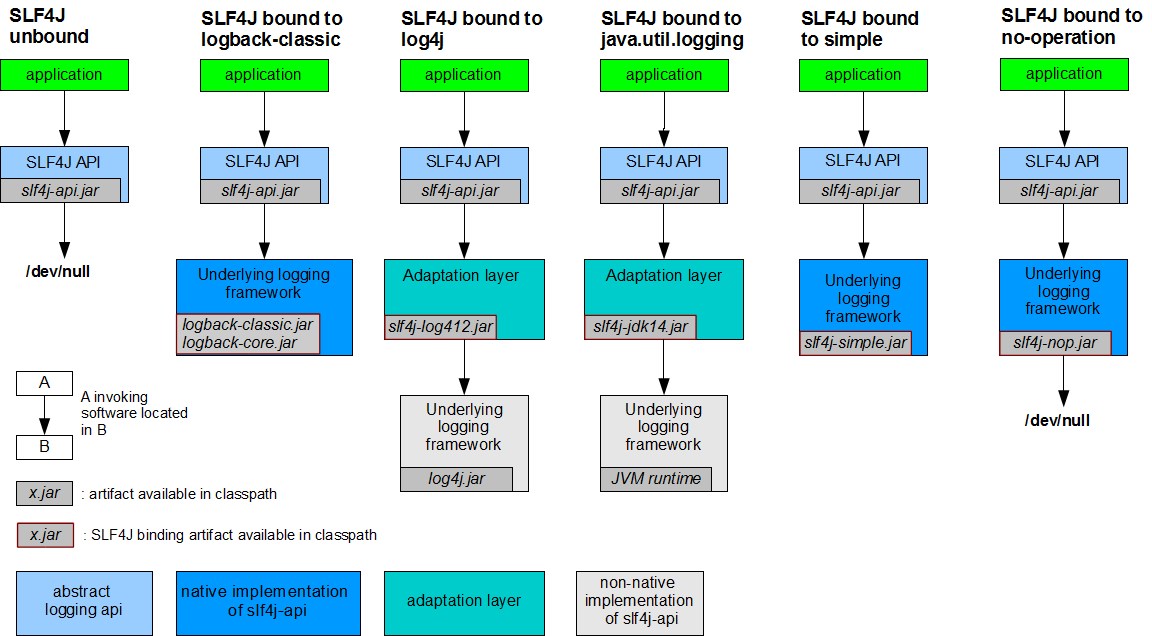
Slf4j（simple logging Façade for java）

Slf4j是日志系统的一个简单的外观，它允许终端用户在部署项目时插入合适的日志系统。

Slf4j支持一个高级特性叫做参数化日志，它可以显著提高禁用日志记录语句的日志记录性能。

Slf4j支持不同的日志框架，slf4j的每一个发行版都附带几个“SLF4J bings”的jar包，每个绑定对应一个支持的框架

Slf4j不依赖类转入机制，事实上，在编译时期每个slf4j都是硬绑定的，为了只使用一个特定的日志框架



# 2.Log4j

log4j主要有三个重要部分组成，日志记录器（Loggers），输出端（Appenders）和日志格式化器（Layout）。

## 2.1日志级别

**Trace（）**:是追踪，就是程序推进以下，你就可以写个trace输出，所以trace应该会特别多，不过没关系，我们可以设置最低日志级别不让他输出。

**Debug（7）**：调试么，我一般就只用这个作为最低级别，trace压根不用。是在没办法就用eclipse或者idea的debug功能就好了么。

**Info（6）**:输出一下你感兴趣的或者重要的信息，这个用的最多了。

**Warn（4）**:有些信息不是错误信息，但是也要给程序员的一些提示，类似于eclipse中代码的验证不是有error 和warn（不算错误但是也请注意，比如以下depressed的方法）。

**Error（3）**：错误信息。用的也比较多。

**Fatal（0）**：级别比较高了。重大错误，这种级别你可以直接停止程序了，是不应该出现的错误么！不用那么紧张，其实就是一个程度的问题

**通过控制日志级别可以控制相应级别的日志的输出**。

## 2.2输出端

org.apache.log4j.ConsoleAppender：将日志信息输出到控制台。

org.apache.log4j.FileAppender：将日志信息输出到一个文件。

org.apache.log4j.DailyRollingFileAppender：将日志信息输出到一个日志文件，并且每天输出到一个新的日志文件。

org.apache.log4j.RollingFileAppender：将日志信息输出到一个日志文件，并且指定文件的尺寸，当文件大小达到指定尺寸时，会自动把文件改名，同时产生一个新的文件。

org.apache.log4j.WriteAppender：将日志信息以流格式发送到任意指定地方。

org.apache.log4j.jdbc.JDBCAppender：通过JDBC把日志信息输出到数据库中。

## 2.3输出布局

org.apache.log4j.HTMLLayout：以HTML表格形式布局  
org.apache.log4j.PatternLayout：可以灵活地指定布局模式  
org.apache.log4j.SimpleLayout：包含日志信息的级别和信息字符串  
org.apache.log4j.TTCCLayout：包含日志产生的时间、线程、类别等等信息

## 2.4日志输出格式

**所用到的参数如下，按需添加：**

**%p:** 输出日志信息优先级，即DEBUG，INFO，WARN，ERROR，FATAL,

**%d:** 输出日志时间点的日期或时间，默认格式为ISO8601，也可以在其后指定格式，比如：%d{yyyy-MM-dd HH:mm:ss}。

**%r:** 输出自应用启动到输出该log信息耗费的毫秒数

**%c:** 输出日志信息所属的类目，通常就是所在类的全名

**%t:** 输出产生该日志事件的线程名

**%l:** 输出日志事件的发生位置，相当于%C.%M(%F:%L)的组合,包括类目名、发生的线程，以及在代码中的行数。举例：Testlog4.main(TestLog4.java:10)

**%x:** 输出和当前线程相关联的NDC(嵌套诊断环境),尤其用到像java servlets这样的多客户多线程的应用中。

**%%:** 输出一个”%”字符

**%F:** 输出日志消息产生时所在的文件名称

**%L:** 输出代码中的行号

**%m:** 输出代码中指定的消息,产生的日志具体信息

**%n:** 输出一个回车换行符，Windows平台为”\r\n”，Unix平台为”\n”输出日志信息换行

可以在%与模式字符之间加上修饰符来控制其最小宽度、最大宽度、和文本的对齐方式。如：

1)%20c：指定输出category的名称，最小的宽度是20，如果category的名称小于20的话，默认的情况下右对齐。

2)%-20c:指定输出category的名称，最小的宽度是20，如果category的名称小于20的话，”-”号指定左对齐。

3)%.30c:指定输出category的名称，最大的宽度是30，如果category的名称大于30的话，就会将左边多出的字符截掉，但小于30的话也不会有空格。

4)%20.30c:如果category的名称小于20就补空格，并且右对齐，如果其名称长于30字符，就从左边交远销出的字符截掉。

# 3.Logback

## 3.1 logback简介：

Logback是由log4j创始人设计的另一个开源日志组件

logback-core：其它两个模块的基础模块  
logback-classic：它是log4j的一个改良版本，同时它完整实现了slf4j API使你可以很方便地更换成其它日志系统如log4j或JDK14 Logging  
logback-access：访问模块与Servlet容器集成提供通过Http来访问日志的功能

## 3.2 logback配置说明

2.1 logger ,appender及layout

Logger作为日志的记录器，把它关联到应用的对应的context上后，主要用于存放日志对象，也可以定义日志类型、级别。  
**Appender：**主要用于指定日志输出的目的地，目的地可以是控制台、文件、远程套接字服务器、 MySQL、PostreSQL、 Oracle和其他数据库、 JMS和远程UNIX Syslog守护进程等。   
**Layout：** 负责把事件转换成字符串，格式化的日志信息的输出。

2.2 logger,context

各个logger 都被关联到一个 LoggerContext，LoggerContext负责制造logger，也负责以树结构排列各logger。其他所有logger也通过org.slf4j.LoggerFactory 类的静态方法getLogger取得。 getLogger方法以 logger名称为参数。用同一名字调用LoggerFactory.getLogger 方法所得到的永远都是同一个logger对象的引用。

2.3 有效级及级别的继承

Logger 可以被分配级别。级别包括：TRACE、DEBUG、INFO、WARN 和 ERROR，定义于ch.qos.logback.classic.Level类。如果 logger没有被分配级别，那么它将从有被分配级别的最近的祖先那里继承级别。root logger 默认级别是 DEBUG。

2.4 打印方式与基本的选择规则

打印方法决定记录请求的级别。例如，如果 L 是一个 logger 实例，那么，语句 L.info("..")是一条级别为 INFO的记录语句。记录请求的级别在高于或等于其 logger 的有效级别时被称为被启用，否则，称为被禁用。记录请求级别为 p，其 logger的有效级别为 q，只有则当 p>=q时，该请求才会被执行。  
该规则是 logback 的核心。级别排序为： TRACE < DEBUG < INFO < WARN < ERROR

## 3.3 logback默认配置

如果配置文件 logback-test.xml 和 logback.xml 都不存在，那么 logback 默认地会调用BasicConfigurator ，创建一个最小化配置。最小化配置由一个关联到根 logger 的ConsoleAppender 组成。输出用模式为%d{HH:mm:ss.SSS} [%thread] %-5level %logger{36} - %msg%n 的 PatternLayoutEncoder 进行格式化。root logger 默认级别是 DEBUG。

3.1 logback配置文件

Logback 配置文件的语法非常灵活。正因为灵活，所以无法用 DTD 或 XML schema 进行定义。尽管如此，可以这样描述配置文件的基本结构：以<configuration>开头，后面有零个或多个<appender>元素，有零个或多个<logger>元素，有最多一个<root>元素。

3.2 logback配置文件步骤

(1). 尝试在 classpath下查找文件logback-test.xml；

(2). 如果文件不存在，则查找文件logback.xml；  
(3). 如果两个文件都不存在，logback用BasicConfigurator自动对自己进行配置， 这会导致记录输出到控制台

## 3.4 logback.xml常用配置详解

### 4.1根节点<configuration>，包含下面三个属性： 　　　　scan: 当此属性设置为true时，配置文件如果发生改变，将会被重新加载，默认值为true。 　　　　scanPeriod: 设置监测配置文件是否有修改的时间间隔，如果没有给出时间单位，默认单位是毫秒。当scan为true时，此属性生效。默认的时间间隔为1分钟。 　　　　debug: 当此属性设置为true时，将打印出logback内部日志信息，实时查看logback运行状态。默认值为false。

### 4.2子节点<contextName>：用来设置上下文名称，每个logger都关联到logger上下文，默认上下文名称为default。但可以使用<contextName>设置成其他名字，用于区分不同应用程序的记录。一旦设置，不能修改。

### 4.3子节点<property> ：用来定义变量值，它有两个属性name和value，通过<property>定义的值会被插入到logger上下文中，可以使“${}”来使用变量。 name: 变量的名称 value: 的值时变量定义的值

### 4.4子节点<timestamp>：获取时间戳字符串，他有两个属性key和datePattern key: 标识此<timestamp> 的名字； datePattern: 设置将当前时间（解析配置文件的时间）转换为字符串的模式，遵循java.txt.SimpleDateFormat的格式。

#### 4.5 子节点<appender>：负责写日志的组件，它有两个必要属性name和class。name指定appender名称，class指定appender的全限定名

#### 4.5.1 ConsoleAppender： 把日志输出到控制台，有以下子节点： <encoder>：对日志进行格式化。（具体参数稍后讲解 ） <target>：字符串System.out(默认)或者System.err（区别不多说了）

#### 4.5.2 FileAppender：把日志添加到文件，有以下子节点： <file>：被写入的文件名，可以是相对目录，也可以是绝对目录，如果上级目录不存在会自动创建，没有默认值。 <append>：如果是 true，日志被追加到文件结尾，如果是 false，清空现存文件，默认是true。 <encoder>：对记录事件进行格式化。（具体参数稍后讲解 ） <prudent>：如果是 true，日志会被安全的写入文件，即使其他的FileAppender也在向此文件做写入操作，效率低，默认是 false。

#### 4.5.3 RollingFileAppender：滚动记录文件，先将日志记录到指定文件，当符合某个条件时，将日志记录到其他文件。有以下子节点： <file>：被写入的文件名，可以是相对目录，也可以是绝对目录，如果上级目录不存在会自动创建，没有默认值。 <append>：如果是 true，日志被追加到文件结尾，如果是 false，清空现存文件，默认是true。 <rollingPolicy>:当发生滚动时，决定RollingFileAppender的行为，涉及文件移动和重命名。属性class定义具体的滚动策略类 class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy"： 最常用的滚动策略，它根据时间来制定滚动策略，既负责滚动也负责出发滚动。

有以下子节点：  
**<fileNamePattern>**：必要节点，包含文件名及“%d”转换符，“%d”可以包含一个java.text.SimpleDateFormat指定的时间格式，如：%d{yyyy-MM}。  
如果直接使用 %d，默认格式是 yyyy-MM-dd。RollingFileAppender的file字节点可有可无，通过设置file，可以为活动文件和归档文件指定不同位置，当前日志总是记录到file指定的文件（活动文件），活动文件的名字不会改变；  
如果没设置file，活动文件的名字会根据fileNamePattern 的值，每隔一段时间改变一次。“/”或者“\”会被当做目录分隔符。   
**<maxHistory>:**可选节点，控制保留的归档文件的最大数量，超出数量就删除旧文件。假设设置每个月滚动，且<maxHistory>是6，则只保存最近6个月的文件，删除之前的旧文件。注意，删除旧文件是，那些为了归档而创建的目录也会被删除。

## 5. **常用loger配置**

<logger name="org.hibernate.type.descriptor.sql.BasicBinder" level="TRACE" />  
<logger name="org.hibernate.type.descriptor.sql.BasicExtractor" level="DEBUG" />  
<logger name="org.hibernate.SQL" level="DEBUG" />  
<logger name="org.hibernate.engine.QueryParameters" level="DEBUG" />  
<logger name="org.hibernate.engine.query.HQLQueryPlan" level="DEBUG" />  
<logger name="com.apache.ibatis" level="TRACE"/>  
<logger name="java.sql.Connection" level="DEBUG"/>  
<logger name="java.sql.Statement" level="DEBUG"/>  
<logger name="java.sql.PreparedStatement" level="DEBUG"/>