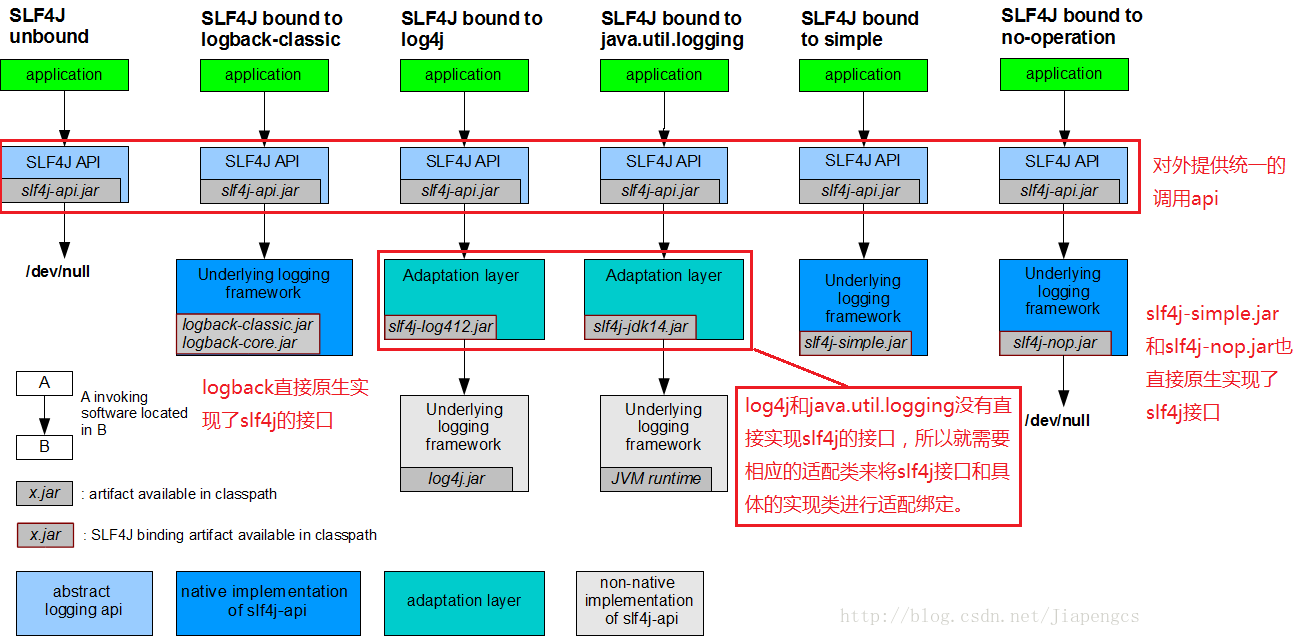
# SLF4J

什么是SLF4J?

意思为简单日志门面，它是把不同的日志系统的实现进行了具体的抽象化，只提供了统一的日志使用接口，使用时只需要按照其提供的接口方法进行调用即可，由于它只是一个接口，并不是一个具体的可以直接单独使用的日志框架，所以最终日志的格式、记录级别、输出方式等都要通过接口绑定的具体的日志系统来实现，这些具体的日志系统就有log4j,logback,java.util.logging等，它们才实现了具体的日志系统的功能。

如何使用SLF4J?

既然SLF4J只是一个接口，那么实际使用时必须要结合具体的日志系统来使用，我们首先来看SLF4J和各个具体的日志系统进行绑定时的框架原理图：



其实slf4j原理很简单，他只提供一个核心slf4j api(就是slf4j-api.jar包)，这个包只有日志的接口，并没有实现，所以如果要使用就得再给它提供一个实现了些接口的日志包，比 如：log4j,common logging,jdk log日志实现包等，但是这些日志实现又不能通过接口直接调用，实现上他们根本就和slf4j-api不一致，因此slf4j又增加了一层来转换各日志实现包的使用

slf4j-api(接口层)

|

各日志实现包的连接层( slf4j-jdk14, slf4j-log4j)

|

各日志实现包

为什么要使用SLF4J?

1、slf4j是一个日志接口，自己没有具体实现日志系统，只提供了一组标准的调用api,这样将调用和具体的日志实现分离，使用slf4j后有利于根据自己实际的需求更换具体的日志系统，比如:之前使用的具体的日志系统为log4j,想更换为logback时，只需要删除log4j相关的jar,然后加入logback相关的jar和日志配置文件即可，而不需要改动具体的日志输出方法，试想如果没有采用这种方式，当你的系统中日志输出有成千上万条时，你要更换日志系统将是多么庞大的一项工程。如果你开发的是一个面向公众使用的组件或公共服务模块，那么一定要使用slf4的这种形式，这有利于别人在调用你的模块时保持和他系统中使用统一的日志输出。

2、slf4j日志输出时可以使用{}占位符，如，logger.info("testlog: {}", "test")，而如果只使用log4j做日志输出时，只能以logger.info("testlog:"+"test")这种形式，前者要比后者在性能上更好，后者采用+连接字符串时就是new 一个String 字符串，在性能上就不如前者。

如果系统中之前已经使用了log4j做日志输出，想使用slf4j作为统一的日志输出，该怎么办呢？

如果之前系统中是单独使用log4j做为日志输出的，这时再想使用slf4j做为日志输出时，如果系统中日志比较多，此时更改日志输出方法肯定是不太现实的，这个时候就可以使用log4j-over-slf4j.jar将使用log4j日志框架输出的日志路由到slf4j上来统一采用slf4j来输出日志。

结合各日志实现包使用时提供的jar包情况为：

SLF4J和logback结合使用时需要提供jar:slf4j-api.jar,logback-classic.jar,logback-core.jar

SLF4J和log4j结合使用时需要提供jar:slf4j-api.jar,slf4j-log4j12.jar,log4j.jar

SLF4J和JDK中java.util.logging结合使用时需要提供jar:slf4j-api.jar,slf4j-jdk14.jar

SLF4J和simple(SLF4J本身提供的一个接口的简单实现)结合使用时需要提供的jar:slf4j-api.jar,slf4j-simple.jar

当然还有其他的日志实现包，以上是经常会使用到的一些。

**总结如下：**

1、slf4j是java的一个日志门面，实现了日志框架一些通用的api，log4j和logback是具体的日志框架。

2、他们可以单独的使用，也可以绑定slf4j一起使用。

单独使用，分别调用框架自己的方法来输出日志信息。绑定slf4j一起使用。调用slf4j的api来输入日志信息，具体使用与底层日志框架无关（需要底层框架的配置文件）。显然不推荐单独使用日志框架。假设项目中已经使用了log4j，而我们此时加载了一个类库，而这个类库依赖另一个日志框架。这个时候我们就需要维护两个日志框架，这是一个非常麻烦的事情。而使用了slf4j就不同了，由于应用调用的抽象层的api，与底层日志框架是无关的，因此可以任意更换日志框架

# Log4j

## 使用步骤

# logback

**logback的介绍**

logback同样是由log4j的作者设计完成的，拥有更好的特性，用来取代log4j的一个日志框架,是slf4j的原生实现(即直接实现了slf4j的接口，而log4j并没有直接实现，所以就需要一个适配器slf4j-log4j12.jar),logback一共有以下几个模块：

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slf4j API使你可很方便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

同样，单独使用它时，需要引入以上jar,然后进行配置文件的配置，最后就是在相关类中进行使用，使用时加入以下语句:

private final static Logger logger = LoggerFactory.getLogger(Test.class);

logger.info("打印日志");

logback取代log4j的理由：

1、更快的实现：Logback的内核重写了，在一些关键执行路径上性能提升10倍以上。而且logback不仅性能提升了，初始化内存加载也更小了。

2、非常充分的测试：Logback经过了几年，数不清小时的测试。Logback的测试完全不同级别的。

3、Logback-classic非常自然实现了SLF4j：Logback-classic实现SLF4j。在使用SLF4j中，你都感觉不到logback-classic。而且因为logback-classic非常自然地实现了slf4j ， 所以切换到log4j或者其他非常容易，只需要提供成另一个jar包就OK，根本不需要去动那些通过SLF4JAPI实现的代码。

4、非常充分的文档 官方网站有两百多页的文档。

5、自动重新加载配置文件，当配置文件修改了，Logback-classic能自动重新加载配置文件。扫描过程快且安全，它并不需要另外创建一个扫描线程。这个技术充分保证了应用程序能跑得很欢在JEE环境里面。

6、LIlith是log事件的观察者，和log4j的chainsaw类似。而lilith还能处理大数量的log数据 。

7、谨慎的模式和非常友好的恢复，在谨慎模式下，多个FileAppender实例跑在多个JVM下，能 够安全地写道同一个日志文件。RollingFileAppender会有些限制。Logback的FileAppender和它的子类包括 RollingFileAppender能够非常友好地从I/O异常中恢复。

8、配置文件可以处理不同的情况，开发人员经常需要判断不同的Logback配置文件在不同的环境下（开发，测试，生产）。而这些配置文件仅仅只有一些很小的不同，可以通过,和来实现，这样一个配置文件就可以适应多个环境。

9、Filters（过滤器）有些时候，需要诊断一个问题，需要打出日志。在log4j，只有降低日志级别，不过这样会打出大量的日志，会影响应用性能。在Logback，你可以继续 保持那个日志级别而除掉某种特殊情况，如alice这个用户登录，她的日志将打在DEBUG级别而其他用户可以继续打在WARN级别。要实现这个功能只需加4行XML配置。可以参考MDCFIlter 。

10、SiftingAppender（一个非常多功能的Appender）：它可以用来分割日志文件根据任何一个给定的运行参数。如，SiftingAppender能够区别日志事件跟进用户的Session，然后每个用户会有一个日志文件。

11、自动压缩已经打出来的log：RollingFileAppender在产生新文件的时候，会自动压缩已经打出来的日志文件。压缩是个异步过程，所以甚至对于大的日志文件，在压缩过程中应用不会受任何影响。

12、堆栈树带有包版本：Logback在打出堆栈树日志时，会带上包的数据。

13、自动去除旧的日志文件：通过设置TimeBasedRollingPolicy或者SizeAndTimeBasedFNATP的maxHistory属性，你可以控制已经产生日志文件的最大数量。如果设置maxHistory 12，那那些log文件超过12个月的都会被自动移除。

## 使用步骤