**logback和slf4j的使用之logger使用**

一、logger标签描述：（了解logger标签之前先看看两个重要概念）

1.主要概念：★★**级别继承**（level inheritance）

1）root标签是所有logger的祖先元素，所有的logger都继承与root元素，相当于java中的object对象。

2）每一个logger标签都可以指定一个级别（TRACE，DEBUG，INFO，WARN，ERROR五种，他们是在ch.qos.logback.classic.Levelclass中定义的）。

3）如果一个指定的logger标签没有指定规则的话，就会继承离他最近的祖先的级别。

4）因为所有的logger都是继承root标签，所以为了确保所有的logger都有一个级别，root标签有一个默认的级别，默认值为DEBUG

5）看看下面四个例子：

2.重要概念：★★**基本选择规则**（base selection rule）:

1）任何一个logger对象都有五种级别，所以就对应有五种打印日志方法：logger.info(字符串)，logger.debug(字符串)

2）如果使用logger.info（字符串）的话，那么logger打印级别就是INFO。

3）看看一个logger能不能打印出日志主要看两个因素：①打印方法时使用的级别（logger.info()），②logger对象的**有效级别**。第一个

重要概念所讲。

4）如何判断是否打印出日志：规则如下：

A log request of level p issued to a logger having an effective level q, is enabled if p >= q.

一个logger打印级别为P，而他的有效级别为q。如果p>=q的话，p对应的信息能打印出来。否则就不能打印出对应的信息。

记住：有效级别是最低级别，高于有效级别的信息都能打印出来，低于有效级别都不能打印出来。

5）级别大小规则：TRACE<DEBUG<INFO<WARN<ERROR。

6）例如：

3.**logger结构**：

1）一个logger标签包含：一个**强制的name**属性，一个**可选的level**属性和一个**可选的additivity**属性（additivity的值有true和false）

2）**level属性的值**：可以是**TRACE，DEBUG，INFO，WARN，ERROR，OFF，ALL**中任何一个，并且**不区分大小写**。如果没有写

的话就会继承离他最近的祖先的级别，就是第一个概念的内容。

3）其中**name**的值，**很重要**，并且是**必须的**。它主要是指定**包名或者类名的全路径**（包含包名）。所有的logger对象都是通过这个来确定

**祖先关系**的。

4）logger标签可以包含0个或者多个**appender-ref**标签，所以每一个**appender标签**都可以关联到**指定的logger**中。

5）**level**默认是以**覆盖的**方式继承的。而**appender-ref**是**附加的**形式继承的。

1）**覆盖方式**：如果父亲的级别是DEBUG，如果在这个logger中设置了INFO，最终这个logger的级别就是INFO

2）**附加方式**：如果在父亲中有一个appender-ref（关联一个appender标签），在当前这个logger中也设置一个appender-ref，

那么当前的logger就有两个appender-ref的关联。如果想要改变默认的附加形式，那就要使用addtivity属性值。

二、logger中appender附加方式：

1.在默认情况下：appender是**累加的方式**进行操作的。即：一个logger对象将会记录：①与这个**logger对象**直接关联的**appender**日志格式，

②是当前**logger对象**的**祖先对象**（所有祖先）的logger相关联的**appender**日志格式。因此：同一个appender对象绑定到多个logger

对象上，就会造成日志输出多份。

2.例如：

<configuration>

<appender name="STDOUT" class="ch.qos.logback.core.ConsoleAppender">

<encoder>

<pattern>%d{HH:mm:ss.SSS} [%thread] %-5level %logger{36} - %msg%n</pattern>

</encoder>

</appender>

<logger name="chapters.configuration">

<appender-ref ref="STDOUT" /><!--STDOUT绑定到chapters.configuration上，这个logger的祖先是root-->

</logger>

<root level="debug"><!--在root上也绑定STDOUT的appender对象到root上。-->

<appender-ref ref="STDOUT" />

</root>

</configuration>

如果在charpters.configuration的类的日志打印的话，就会**打印两份**。因为一份是charpters.configuration

的，一份是root的。所以就会打印两次：如下所示：

14:25:36.343 [main] INFO chapters.configuration.MyApp3 - Entering application.

14:25:36.343 [main] INFO chapters.configuration.MyApp3 - Entering application.

14:25:36.359 [main] DEBUG chapters.configuration.Foo - Did it again!

14:25:36.359 [main] DEBUG chapters.configuration.Foo - Did it again!

14:25:36.359 [main] INFO chapters.configuration.MyApp3 - Exiting application.

14:25:36.359 [main] INFO chapters.configuration.MyApp3 - Exiting application.

这个名字为STDOUT附加到两个logger对象上，root是任何logger对象的祖先。而chapters.configuration

又是MyApp3和Foo的祖先，所有打印两次。

3.appender的累加功能不想想对新用户造成困扰。而是对于日志记录非常的灵活。

1）例如：如果你想要配置日志文件以便形成：①对于所有的logger对象的日志信息都打印到控制台上。

②并且只有一部分的logger对象打印到日志文件中。那就如下配置logback文件

<configuration>

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

<file>myApp.log</file>

<encoder>

<pattern>%date %level [%thread] %logger{10} [%file:%line] %msg%n</pattern>

</encoder>

</appender>

<appender name="STDOUT" class="ch.qos.logback.core.ConsoleAppender">

<encoder>

<pattern>%msg%n</pattern>

</encoder>

</appender>

<logger name="chapters.configuration">

<appender-ref ref="FILE" />

</logger>

<root level="debug">

<appender-ref ref="STDOUT" />

</root>

</configuration>

对于所有的日志对象都会打印日志信息到控制台上，但是对于chapters.configuration的日志对象还会记录到

文件中。

4.覆盖默认的appender的累加行为：

如果默认的累加行为不符合你的要求， 你可以通过设置【activity=false】来取消默认的行为（累加）。

1）例如：

<configuration>

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

<file>foo.log</file>

<encoder>

<pattern>%date %level [%thread] %logger{10} [%file : %line] %msg%n</pattern>

</encoder>

</appender>

<appender name="STDOUT" class="ch.qos.logback.core.ConsoleAppender">

<encoder>

<pattern>%msg%n</pattern>

</encoder>

</appender>

<logger name="chapters.configuration.Foo" **additivity="false"**>

<appender-ref ref="FILE" />

</logger>

<root level="debug">

<appender-ref ref="STDOUT" />

</root>

</configuration>

上面那个例子中：一个名字叫FILE的appender附加到chapters.configuration.Foo的logger对象上，

并且这个logger对象的activity的属性为false。**这就意味着**：Foo的日志信息将会按照名字叫FILE的

appender的样式输出，而不会附加任何一个祖先上的appender对象。**但是**，其他日志对象（不属于

chapters.configuration.Foo及其子类的logger）仍然以附加的形式继承appender对象。